



HAL
open science

Complete Subhedge Projection for Stepwise Hedge Automata

Antonio Al Serhali, Joachim Niehren

► **To cite this version:**

Antonio Al Serhali, Joachim Niehren. Complete Subhedge Projection for Stepwise Hedge Automata. Algorithms, 2024, Selected Algorithmic Papers From FCT 2023, 17 (8), 10.3390/a17080339 . hal-04421323v3

HAL Id: hal-04421323

<https://inria.hal.science/hal-04421323v3>

Submitted on 2 Aug 2024

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.



Distributed under a Creative Commons Attribution 4.0 International License

Complete Subhedge Projection for Stepwise Hedge Automata

Antonio Al Serhali and Joachim Niehren

Inria Center of the University of Lille, France

Abstract: We show how to evaluate stepwise hedge automata (SHAs) with subhedge projection, while completely projecting irrelevant subhedges. Since this requires passing finite state information top-down, we introduce the notion of downward stepwise hedge automata. We use them to define in-memory and streaming evaluators with complete subhedge projection for SHAs. We then tune the evaluators so that they can decide membership at the earliest time point. We apply our algorithms to the problem of answering regular XPath queries on XML streams. Our experiments show that complete subhedge projection of SHAs can indeed speed up earliest query answering on XML streams, so that it becomes competitive with the best existing streaming tools for XPath queries.

Contents

1. Introduction	1
2. Preliminaries	4
3. Problem	6
3.1. Hedge Pattern Matching	6
3.2. Irrelevant Subhedges	7
3.3. Basic Properties	8
4. Hedge Automata	10
4.1. Stepwise Hedge Automata	10
4.2. Downward Stepwise Hedge Automata	13
4.3. Schema Completeness	16
5. Subhedge Projection	17
5.1. Subhedge Projection States	17
5.2. Completeness for Subhedge Projection	19
5.3. In-Memory Evaluator with Subhedge Projection	20
6. Safe-No-Change Projection	21
6.1. Algorithm	22
6.2. Soundness	23
6.3. Incompleteness	30
7. Congruence Projection	31
7.1. Ideas	31
7.2. Algorithm	33
7.3. Soundness	38
7.4. Completeness	45
7.5. In-Memory Complexity	48
8. Earliest Membership with Subhedge Projection	50
8.1. Earliest Membership	50
8.2. Adding Subhedge Projection	51
8.3. Soundness and Completeness	52

Citation: Al Serhali, A.; Niehren, J.
Complete Subhedge Projection. *Journal
Not Specified* 2024, 1, 0.
<https://doi.org/>

Received:
Revised:
Accepted:
Published:

Copyright: © 2024 by the author.
possible open access publication
under the terms and conditions
of the Creative Commons Attri-
bution (CC BY) license ([https://
creativecommons.org/licenses/by/
4.0/](https://creativecommons.org/licenses/by/4.0/)).

8.4. In-Memory Complexity	53
9. Streaming Algorithms	54
9.1. Streaming Evaluators for Downward Hedge Automata	54
9.2. Adding Subhedge Projection	55
9.3. Streaming Complexity of Earliest Membership with Subhedge Projection	57
10. Monadic Regular Queries and XPath	58
10.1. Monadic Regular Queries	58
10.2. Earliest Query Answering with Subhedge Projection	59
11. Experiments with Streaming XPath Evaluation	60
11.1. Benchmark dSHA for XPath Queries	60
11.2. Streaming Evaluation Tool: AStream	61
11.3. Evaluation Measures	62
11.4. Experiments	63
12. Related Automata Models	65
12.1. SHAs versus Tree, Forest, and Hedge Automata	66
12.2. SHA^\downarrow s versus SHAs	66
12.3. SHA^\downarrow s versus NWAAs	67
13. References	69
N. Earliest Membership	71
N.1. Schema Safety is an Inclusion Problem	71
N.2. Algorithm	72
N.3. Soundness and Completeness	73

1. Introduction

Hedges are sequences of letters from some alphabet and trees $\langle h \rangle$ where h is again a hedge. Hedges abstract from the syntactical details of XML or JSON documents while still being able to represent them. The linearization of a hedge is a nested word that can be written to a file. In this article, we study the problem of hedge pattern matching, so whether a given hedge h belongs to a given regular hedge language L . Regular hedge languages can be defined in XPath or JSONPath in particular.

Projection is necessary for the efficiency of many algorithms on words, trees, hedges, or nested words. Intuitively, an algorithm is projecting if it visits only a fragment of the input structure, in the best case, the part that is relevant to the problem under consideration. The relevance of projection for XML processing was already noticed by [1–4]. Saxon’s in-memory evaluator, for instance, projects input XML document relative to an XSLT program, which contains a collection of XPath queries to be answered simultaneously [5]. The QuiXPath tool [4] evaluates XPath queries in streaming mode with subtree and descendant projection. Projection during the evaluation JSONPath queries on JSON documents in streaming mode is called fast-forwarding [6].

Projecting in-memory evaluation assumes that the full graph of the input hedge is constructed beforehand. Nevertheless, projection may still save time if one has to match several patterns on the same input hedge, or, if the graph was constructed for different reasons anyway. In streaming mode, the situation is similar: the whole input hedge on the stream needs to be parsed, but the evaluators need to inspect only nodes that are not projected away. Given that pure parsing is by two or three orders of magnitude faster than pattern evaluation, the time gain of projection may be considerable.

The starting point of this article is the notion of subhedge irrelevance of positions of a hedge h with respect to a hedge language L that we introduce. These are positions of h where, for all possible continuations, inserting any subhedge does not affect membership to L . We contribute an algorithm for hedge pattern matching with complete subhedge

projection. Our algorithm decides hedge pattern matching while completely projecting away the subhedges located at subhedge irrelevant positions. In other words, it decides whether a hedge h matches a pattern L without visiting any subhedge of h that is located at some position that is subhedge irrelevant with respect to L .

Regular hedge languages can be represented by nested regular expressions, which can be derived from regular XPath queries [7], or else by some kind of hedge automata, which can be compiled from nested regular expressions. We use stepwise hedge automata (SHAs) [8], a recent variant of standard hedge automata, which in turn go back to the sixties [9,10]. SHAs mix up bottom-up processing of standard tree automata with the left-to-right processing of finite word automata (DFAs). They neither support top-down processing nor have an explicit stack, in contrast to nested word automata NWA [11–15]. SHAs have a good notion of bottom-up and left-to-right determinism, avoiding the problematic notion of determinism for standard hedge automata, and the too-costly notion of determinism for NWA. Furthermore, compilers from nested regular expressions to SHAs are available [8], and determinization algorithms [16] producing small SHAs for all regular XPath queries in the existing practical benchmarks [7,17].

The motivation for the present work is to add subhedge projection to a recent streaming algorithm for deterministic SHAs that can evaluate regular XPath queries in an earliest manner. Most alternative streaming approaches avoid earliest query answering, by considering sublanguages of regular queries without delays, so that node selection depends only on the past of the node but not on the future [18,19], or by admitting late selection [20]. In particular, it could be shown recently that earliest query answering for regular monadic queries defined by deterministic SHAs [21] has a lower worst case complexity than for deterministic NWA [22]. Thereby, earliest query answering for regular queries became feasible in practice for the first time. On the other hand side, it is still slower experimentally than the best existing non-earliest approaches for general regular monadic queries on XML streams (see [23] for a comparison) since the latter supports projection. How projection could be done for SHAs has not been studied before the conference version of the present article [24].

Consider the example of the XPath filter `[self::list/child::item]` that is satisfied by an XML document if its root is a `list` element that has some `item` child. With the encoding of XML document as hedges chosen in the present article, the satisfying hedges have the form $\langle \text{list} \cdot h_1 \cdot \langle \text{item} \cdot h_2 \rangle \cdot h_3 \rangle \cdot h_4$ for some hedges h_1, h_2, h_3, h_4 . When evaluating this filter on some hedge, it is sufficient to inspect the first subtree for having the root label `list` and then all its children until one with root label `item` is found. The subhedges of these children, i.e., the hedge h_3 and the subhedges of the top-level trees in h_1 and h_3 , can be projected away, as well as the hedge h_4 : They don't have to be inspected for evaluating this filter. However, one must memoize whether the level of the current node is 0, 1, or greater. This level information can be naturally updated in a top-down manner. The evaluators of SHAs, however, operate bottom-up and left-to-right exclusively. Therefore, projecting evaluators for SHAs need to be based on more general machines. We propose *downward stepwise hedge automata* (SHA^\downarrow s), a variant of SHAs that supports top-down processing in addition. They are basically Neumann and Seidl's pushdown forest automata [25], except that they apply to unlabeled hedges instead of labeled forests. Furthermore, NWA are known to operate in basically the same manner on nested words [26], while allowing for more slightly more general visible pushdowns. We then distinguish subhedge projection states for SHA^\downarrow s, and show how to use them to evaluate SHAs with subhedge projection both in-memory and in streaming mode.

As a first contribution, we present the safe-no-change compiler from SHAs to SHA^\downarrow s that can distinguish appropriate subhedge projection states. The idea is that the safe-no-change SHA^\downarrow can distinguish contexts in which the states of the SHA will safely not change. For instance, the XPath filter `[self::list/child::item]` can be defined by the deterministic SHA in Fig. 4, which our compiler maps to the SHA^\downarrow in Fig. 7. The context information made explicit is about the levels of the states. This permits us to distinguish projection

states from level 2 on, and in which subhedges can be ignored. We prove the soundness of our compiler based on a nontrivial invariant that we establish¹. We also present a counter example showing that the safe-no-change projection is not complete for subhedge projection. It shows that a subhedge may be irrelevant even though its state may still be changing.

As a second contribution, we propose the congruence projection algorithm. It again compiles SHAS to SHA^\downarrow s but relies on the congruence relations on automata states. We then prove that congruence projection yields not only a sound algorithm, but that this algorithm is also complete for subhedge projection, i.e., all strongly irrelevant subhedges (see Definition 23) are evaluated into some looping state. Congruence projection starts on top-level of hedges with the Myhill-Nerode congruence that is well-known from automata minimization. For languages on words L , this congruence identifies prefixes v that have the same residual language $v^{-1}(L) = \{w \mid v \cdot w \in L\}$. A prefix v is then suffix irrelevant, if its residual language $v^{-1}(L)$ is either universal or empty. In the case of hedges – which extend on words by adding hierarchical nesting via a tree constructor – the congruence must be adapted when moving down into subtrees.

We show that both compilers may increase the size of the automata exponentially, since they must be able to convert bottom-up deterministic automata on monadic trees into top-down deterministic automata. That is the same as inverting deterministic automata on words, which is well known to raise an exponential blow up in the worst case². The exponential explosion can be avoided when interested in the evaluation of hedges by automaton with subhedge projection, by not constructing the complete SHA^\downarrow s statically. Instead, only the needed part of the SHA^\downarrow s may be constructed on the fly when evaluating some hedge.

Our third contribution is a refinement of the two previous compilers so that they can also distinguish safe states for selection at the earliest position. For this, we combine these compilers with a variant of the earliest membership tester of [21] that operates in-memory by compiling to SHA^\downarrow s instead of NWAs. Furthermore, membership failure is detected at the earliest position too. In this way, we obtain an earliest in-memory membership tester for deterministic SHAS.

Our fourth contribution is to show how to run the previous in-memory evaluators in streaming mode while reading the linearization of the hedge into a nested word as an input. Thereby, we can improve on the recent earliest streaming membership tester behind the AStream tool [21] by adding subhedge projection to it.

Our fifth and last contribution is an implementation and experimental evaluation of the streaming earliest query answering algorithm for dSHAS by introducing subhedge projection into the AStream tool [21]. We implemented both, safe-no-change and congruence projection. For this, we have to lift our earliest membership tester with subhedge projection to an earliest query answering algorithm for monadic queries. For the experimentation, we start with the deterministic SHAS constructed with the compiler from [8] for the forward regular XPath queries of the XPathMark benchmark [27] and real-world XPath queries [7]. It turns out that congruence projection projects much more strongly than safe-no-change projection for at least half of the benchmark queries. It reduces the running time for all regular XPath queries considerably since large parts of the input hedges can be projected away. In our benchmark the projected percentage ranges from 75.7% to 100% of the input stream. For XPath queries that contain only child axes, the earliest query answering algorithm of AStream with congruence projection becomes competitive in efficiency with the best existing streaming tool called QuiXPath [23], which however is not always earliest for some queries. Our current implementation of earliest congruence projection in AStream by a factor from 1.3 to 2.9 slower than QuiXPath on the benchmark queries. The improvement is smaller for XPath queries with the descendant axis, where less subhedge projection is

¹ We note that the proof required an adaptation of the original compiler from FCT [24].

² For the family of languages $(a + b)^* \cdot a \cdot (a + b)^n$ where $n \in \mathbb{N}$ the minimal deterministic left-to-left automaton has 2^{n+1} states while the minimal deterministic right-to-left automaton has $n + 1$ states.

possible. Instead, some kind of descendant projection would be needed. Still, even in the worst case in our benchmark, earliest congruence projection with AStream is currently only by a factor of 13.8 slower than QuiXPath.

Outline. We start with preliminaries on hedges and nested words in Section 2. In Section 3, we recall the problem of hedge pattern matching and formally define irrelevant subhedges. In Section 4, we recall the definition of SHAs, introduce SHA^\downarrow s, discuss their in-memory evaluators. In Section 5, we introduce the notion of subhedge projection states, define when an SHA^\downarrow is complete for subhedge projection, and present an in-memory evaluator for SHA^\downarrow s with subhedge projection. In Section 6 we introduce safe-no-change projection and prove its soundness and show it incompleteness. In Section 7, we introduce congruence projection, and prove it to be sound and complete for subhedge projection. Earliest in-memory evaluators for SHA^\downarrow s with subhedge projection follow in Section 8. Streaming variants of our in-memory evaluators are derived systematically in Section 9. Section 10 discusses how to lift our algorithms from membership testing to monadic queries. Section 11 discusses our practical experiments. Section 12 discusses further related work on automata notions related to SHAs and SHA^\downarrow s. Appendix N presents the reformulation of the earliest query answering algorithm from [21] based on SHA^\downarrow s and with schema restrictions.

Publication Comments. This original version of journal article was published at the FCT conference [24]. Compared to there, the following contributions are new:

- The definition of subhedge irrelevant prefixes of nested words and the definition of completeness for subhedge projection (Section 3).
- The addition of schema restrictions throughout the paper and the notion of schema-completeness (Section 4.3).
- The notion of completeness for subhedge projection (Section 5.2).
- The soundness proof of safe-no-change projection algorithm (Section 6.2).
- The congruence projection algorithm and the proof of its soundness and completeness for subhedge projection (Section 7).
- A systematic method to add subhedge projection to an earliest SHA^\downarrow (Section 8).
- A systematic method to reduce the correctness of streaming automata evaluators to the corresponding in-memory evaluators (Section 9).
- A discussion of how to deal with monadic queries while exploiting the availability of schema restrictions (Section 10).
- An implementation of congruence projection with experimental results for XPath evaluation on XML streams (Section 11).
- A longer discussion on related automaton notions (Section 12).
- In Appendix N we also show how to obtain earliest dSHA^\downarrow s from dSHAs by adapting the compiler from [21] mapping dSHAs to earliest dNWAs .

2. Preliminaries

Let A and B be sets. If A is a subset of B then the complement of A in B is denoted by $\overline{A} = B \setminus A$ while keeping the set B relative to which the complement is taken implicit in the notation. The *domain* of a binary relation $r \subseteq A \times B$ is $\text{dom}(r) = \{a \in A \mid \exists b \in B. (a, b) \in r\}$. A *partial function* $f : A \hookrightarrow B$ is a binary relation $f \subseteq A \times B$ that is functional. A *total function* $f : A \rightarrow B$ is a partial function $f : A \hookrightarrow B$ with $\text{dom}(f) = A$.

Words. Let \mathbb{N} be the set of natural numbers including 0. Let the alphabet Σ be a set. The set of words over Σ is $\Sigma^* = \bigcup_{n \in \mathbb{N}} \Sigma^n$. A word $(a_1, \dots, a_n) \in \Sigma^n$ where $n \in \mathbb{N}$ is written as $a_1 \dots a_n$. We denote the empty word of length 0 by $\varepsilon \in \Sigma^0$ and by $v_1 \cdot v_2 \in \Sigma^*$ the concatenation of two words $v_1, v_2 \in \Sigma^*$.

If $v = u \cdot v' \cdot w$ is a word, then we call u a prefix of v and v' a factor of v and w a suffix of v . Given any subset $L \subseteq \Sigma^*$, we denote the set of prefixes of words in L by $\text{prefs}(L)$ and the set of suffixes of words in L by $\text{suffs}(L)$.

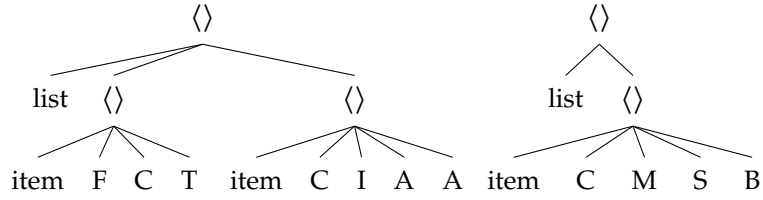


Figure 1. The graph of the hedge $\langle list \cdot \langle item \cdot F \cdot C \cdot T \rangle \cdot \langle item \cdot C \cdot I \cdot A \cdot A \rangle \rangle \cdot \langle list \cdot \langle item \cdot C \cdot M \cdot S \cdot B \rangle \rangle$ is a sequence of trees.

```

<list>
  <item>FCT</item>
  <item>CIAA</item>
</list>
<list>
  <item>CMSB</item>
</list>

```

Figure 2. The sequence of XML documents encoded by the hedge in Fig. 1.

For any subalphabet $\Sigma' \subseteq \Sigma$, the projection of a word $v \in \Sigma^*$ to Σ' is the word $proj_{\Sigma'}(v)$ in $(\Sigma')^*$ that is obtained from v by removing all letters from $\Sigma \setminus \Sigma'$.

Hedges. Hedges are sequences of letters and trees $\langle h \rangle$ with some hedge h . More formally, a hedge $h \in \mathcal{H}_{\Sigma}$ has the following abstract syntax:

$$h, h' \in \mathcal{H}_{\Sigma} ::= \varepsilon \mid a \mid \langle h \rangle \mid h \cdot h' \quad \text{where } a \in \Sigma$$

We assume $\varepsilon \cdot h = h \cdot \varepsilon = h$ and $(h \cdot h_1) \cdot h_2 = h \cdot (h_1 \cdot h_2)$. Therefore, we consider any word in Σ^* as a hedge in \mathcal{H}_{Σ} , i.e., $\Sigma^* \ni aab = a \cdot a \cdot b \in \mathcal{H}_{\Sigma}$. Any hedge can be converted or stored as a graph. For the signature $\Sigma = \{list, item, A, \dots, Z\}$, the graph of an example hedge in \mathcal{H}_{Σ} is shown in Fig. 1. It encodes the sequence of XML documents in Fig. 2:

Nested Words. A nested word over Σ is a word over the alphabet $\hat{\Sigma} = \Sigma \cup \{\langle, \rangle\}$ in which all parentheses are well-nested. Intuitively, this means that for any opening parenthesis there is a matching closing parenthesis and vice versa.

The set of nested words over Σ can be defined as the subsets of words over $\hat{\Sigma}$ that are a linearization of some hedge, where the linearization function $nw : \mathcal{H}_{\Sigma} \rightarrow (\Sigma \cup \{\langle, \rangle\})^*$ is defined as follows:

$$nw(\varepsilon) = \varepsilon, \quad nw(\langle h \rangle) = \langle \cdot nw(h) \cdot \rangle, \quad nw(a) = a, \quad nw(h \cdot h') = nw(h) \cdot nw(h').$$

So the set of all nested words over Σ is $nw(\mathcal{H}_{\Sigma})$. Let hdg be the inverse of the injective function nw restricted to its image, so the mapping from nested words to hedges with $hdg(nw(h)) = h$ for all $h \in \mathcal{H}_{\Sigma}$.

Nested Word Prefixes. Prefixes, suffixes, and factors of nested words may not be nested words themselves. For instance, the hedge $h = a \cdot \langle b \cdot c \rangle$ has the linearization $nw(h) = a\langle bc \rangle$. Its prefix $a\langle b$ is not well-nested since it has a dangling opening parenthesis. Neither its suffix $c \rangle$ is well-nested since it has a dangling closing parenthesis.

Any algorithm that traverses a hedge h in a top-down and left-to-right manner inspects all the prefixes of $nw(h)$. Any prefix v of $nw(h)$ not only distinguishes some position of the hedge h but also specifies the part of hedge that is located before this position in the linearization $nw(h)$. An example is illustrated graphically in Fig. 3. This particularly holds for streaming algorithms that receive the nested word $nw(h)$ as an input on a stream, and may be inspected only once from the left to the right. But it holds equally for in-memory algorithms that receive the input hedge h as a hierarchical structure whose graph is stored

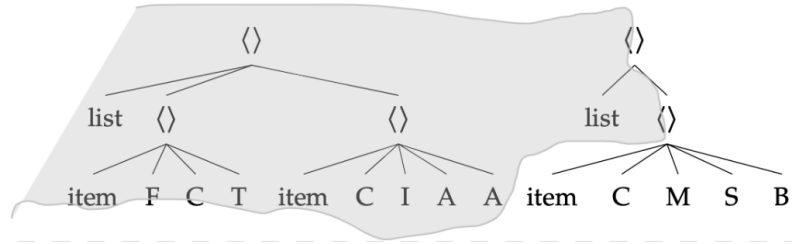


Figure 3. The part of the hedge distinguished by the nested word prefix $\langle list \cdot \langle item \cdot F \cdot C \cdot T \cdot \langle item \cdot C \cdot I \cdot A \cdot A \rangle \rangle \cdot \langle list \cdot \langle$.

in-memory, and then traverse the graph top-down and left-to-right. Any $\langle \rangle$ node will then be visited twice, once when reading an opening parenthesis \langle and going down into a subhedge, and another time when going up to the closing parenthesis \rangle after having processed the subhedge.

3. Problem

We formally define the pattern matching problem for hedges and the notion of subhedge irrelevance which allows us to characterize the parts of the input hedge that don't need to be visited during pattern-matching.

3.1. Hedge Pattern Matching

We are interested in algorithms that match a hedge pattern against a hedge. We start with algorithms that test whether such a matching exists. This is sometimes called filtering, or alternatively, Boolean query answering. In Section 10, we consider the more general problem to return the set of matchings. This problem is sometimes called monadic query answering. Our experiments in Section 11 will apply to monadic query answering for regular XPath queries.

We start from hedge patterns that are nested regular expressions with letters in some signature Σ . For these, we fix a set V of recursion variables and assume that it is disjoint from Σ .

$$e, e' \in nRegExp_{\Sigma} ::= \varepsilon \mid a \mid z \mid e \cdot e' \mid e + e' \mid e^* \mid \langle e \rangle \mid \mu z.e \quad \text{where } a \in \Sigma, z \in V$$

Each nested regular expression $e \in nRegExp_{\Sigma}$ is satisfied by a subset of hedges in $\llbracket e \rrbracket \in \mathcal{H}_{\Sigma \cup V}$. We recall the definition of this set only informally. The expression ε is satisfied only by the empty word, the expression $a \in \Sigma$ only by the hedge a only, and the expression $z \in V$ only by the one letter hedge z . An expression $e \cdot e'$ with the concatenation operator and $e, e' \in nRegExp_{\Sigma}$ is satisfied by the concatenations of hedges satisfying e and e' respectively. An expression e^* with Kleene's star operator and $e \in nRegExp_{\Sigma}$ is satisfied by repetitions of words satisfying e . The Kleene star provides for horizontal recursion on hedges. The μ -expression $\mu z.e$ where $z \in V$ and $e \in nRegExp_{\Sigma}$ is satisfied by all hedges without recursion variable x in which z is repeatedly replaced by a hedge satisfying e until it disappears. We adopt the usual restriction [28] that in any nested regular expression $\mu z.e$, all occurrences of z in e are guarded by some tree constructor $\langle \cdot \rangle$. In this way, the μ -operator provides proper vertical recursion. Also note that, for defining sublanguages of \mathcal{H}_{Σ} , we only need such expressions in $nRegExp_{\Sigma}$ in which all occurrences of recursion variables z are bound in the scope of some binder μz .

It is well known that nested regular expressions in $nRegExp_{\Sigma}$ can define all regular sublanguages of hedges in \mathcal{H}_{Σ} , i.e., they have the same expressiveness as hedge automata with alphabet Σ . Analogous results for tree regular expressions and tree automata are folklore [29].

Example 1 (Nested regular expressions of regular XPath filters). *For instance, we can define subsets of hedges encodings of XML documents with the two kinds of elements in $\Sigma = \{\text{list}, \text{item}\}$*

by expressions in $n\text{RegExp}_\Sigma$. For this article, we will use an encoding of XML document which produces hedges satisfies the following regular expression $\text{doc} \in n\text{RegExp}_\Sigma$ where $z \in V$:

$$\text{doc} =_{\text{def}} \text{tree}^* \quad \text{where} \quad \text{tree} =_{\text{def}} \mu z. \langle (\text{list} + \text{item}) \cdot z^* \rangle$$

In our application, the encodings are a little richer in order to distinguish different types of XML nodes such as elements, attributes, comments, and texts. An XPath filter such as for instance $[\text{self}::\text{list}/\text{child}::\text{item}]$ can then be expressed by the nested regular expression:

$$\text{filter} =_{\text{def}} \langle \text{list} \cdot \text{doc} \cdot \langle \text{item} \cdot \text{doc} \rangle \cdot \text{doc} \rangle \cdot \text{doc}$$

General compilers from regular downward XPath queries to nested regular expressions were proposed in [8]. An implementation was developed in the context of the AStream tool [21].

We will use schemas $\mathbf{S} \subseteq \mathcal{H}_\Sigma$ to restrict the inputs of our pattern matching problem. While mostly interested in regular schemas we do not assume regularity globally, since safe-no-change projection does not rely on it. The pattern matching problem for nested regular expressions with respect to a schema $\mathbf{S} \subseteq \mathcal{H}_\Sigma$ receives the following inputs:

- a nested regular expression $e \in n\text{RegExp}_\Sigma$ and
- a hedge $h \in \mathbf{S}$.

It then returns the truth value of the judgment $h \in \llbracket e \rrbracket$. The input hedge $h \in \mathbf{S}$ may either be given by a graph that resides in memory or by a stream containing the linearization $nw(h)$ of the input hedge and which can be read only once from left-to-right.

3.2. Irrelevant Subhedges

We define the concept of irrelevant occurrences of subhedges with respect to a given hedge pattern. What this means depends on the kind of algorithm that we will use for pattern matching. We will use algorithms that operate on the input hedge top-down, left-to-right, and bottom-up. This holds for streaming algorithms in particular.

Intuitively, when the pattern matching algorithm reaches a node top-down or left-to-right whose subsequent subhedge is irrelevant, then it can jump over it without inspecting its structure. What jumping means should be clear if the hedge is given by a graph that is stored in memory. Notice, that the full graph is to be constructed at beforehand even though some parts of it may turn out irrelevant. Still one may win lots of time by jumping over irrelevant parts if either the graph was already constructed for other reasons, or if many pattern matching problems are to be solved on the same hedge.

In the case of streams, the irrelevant subhedge still needs to be parsed, but it will not be analyzed otherwise. Most typically, the possible analysis is done by automata, which may take 2 orders of magnitudes more time than needed for the parsing. So therefore not having to do any analysis may considerably speed up a streaming algorithm.

Definition 2. Let $\mathbf{S} \subseteq \mathcal{H}_\Sigma$ be a schema and $L \subseteq \mathbf{S}$ a language satisfying this schema. We define $\text{diff}_\mathbf{S}^L$ as the least symmetric relation on prefixes of nested words $u, u' \in \text{prefs}(\mathcal{N}_\Sigma)$ such that:

$$\text{diff}_\mathbf{S}^L(u, u') \Leftrightarrow \exists w \in \text{suffs}(\mathcal{N}_\Sigma). u \cdot w \in nw(L) \wedge u' \cdot w \in nw(\mathbf{S} \setminus L)$$

A nested word prefix u is called subhedge relevant for L with schema \mathbf{S} if there exists nested words $v, v' \in \mathcal{N}_\Sigma$ such that $\text{diff}_\mathbf{S}^L(u \cdot v, u \cdot v')$. Otherwise, the prefix u is called subhedge irrelevant for L with schema \mathbf{S} .

So $\text{diff}_\mathbf{S}^L(u, u')$ states that u and u' have continuations that behave differently with respect to L and \mathbf{S} . Furthermore, a prefix u is subhedge irrelevant if language membership does not depend on hedge insertion at u under the condition that schema membership is guaranteed. The case without schema restrictions is subsumed by the above definition by

Hence, the nested word prefix $u \cdot v$ is subhedge irrelevant for L with schema \mathbf{S} too. \square

Definition 6. We call a binary relation D on prefixes of nested words a difference relation on prefixes if for all prefixes of nested words $u, u' \in \text{prefs}(\mathcal{N}_\Sigma)$ and nested words $v \in \mathcal{N}_\Sigma$:

$$(u \cdot v, u' \cdot v) \in D \Rightarrow (u, u') \in D$$

Lemma 7. $\text{diff}_\mathbf{S}^L$ is a difference relation.

Proof. We have to show for all prefixes $u, u' \in \text{prefs}(\mathcal{N}_\Sigma)$ and nested words $v \in \mathcal{N}_\Sigma$ that $(u \cdot v, u' \cdot v) \in \text{diff}_\mathbf{S}^L$ implies $(u, u') \in \text{diff}_\mathbf{S}^L$. So let u, u' be prefixes of nested words and v a nested word such that $(u \cdot v, u' \cdot v) \in \text{diff}_\mathbf{S}^L$. Then there exists a nested suffix w such that

$$u \cdot v \cdot w \in \text{nw}(L) \wedge u' \cdot v \cdot w \in \text{nw}(\mathbf{S} \setminus L)$$

The suffix $\tilde{w} = v \cdot w$ then satisfies $u \cdot \tilde{w} \in \text{nw}(L) \wedge u' \cdot \tilde{w} \in \text{nw}(\mathbf{S} \setminus L)$. Hence $(u, u') \in \text{diff}_\mathbf{S}^L$ as required. \square

In the schema-free case on words, the complement $\overline{\text{diff}_{\Sigma^*}^L}$ is an equivalence relation that is known as the Myhill-Nerode congruence. In the case of schemas, however, the complement $\overline{\text{diff}_\mathbf{S}^L}$ may not be transitive, and thus, it may even not be an equivalence relation. This might be surprising at first sight.

Example 8. Let $\Sigma = \{a, b\}$, $L = \{\varepsilon, b, aa\}$ and $\mathbf{S} = L \cup \{aab\}$. Then $(\varepsilon, a) \in \overline{\text{diff}_\mathbf{S}^L}$ since there is no continuation, that extends both ε and a into the schema. For the same reason, we have $(a, aa) \in \overline{\text{diff}_\mathbf{S}^L}$. However, $\text{diff}_\mathbf{S}^L(\varepsilon, aa)$ since $\varepsilon \cdot b \in L$ and $aa \cdot b \in \mathbf{S} \setminus L$. Thus, $(\varepsilon, aa) \notin \overline{\text{diff}_\mathbf{S}^L}$, so the complement $\overline{\text{diff}_\mathbf{S}^L}$ is not transitive.

It should be noted for all languages L that $\overline{\text{diff}_{\mathcal{H}_\Sigma}^L}$ is reflexive and transitive, so it is an equivalence relation and thus a congruence. For general schemas \mathbf{S} , however, $\overline{\text{diff}_\mathbf{S}^L}$ may not be transitive as shown by Example 8, and thus not be a congruence.

This indicates that schemas may make it more difficult to decide subhedge irrelevance. And indeed, the natural manner that one might expect to eliminate schemas based on complements does not work, as confirmed by following lemma (in particular the counter example in the proof 1. $\not\Rightarrow$ 2.):

Lemma 9. For any nested word prefix $u \in \text{suffs}(\mathcal{N}_\Sigma)$, and languages $L, \mathbf{S} \subseteq \mathcal{H}_\Sigma$ consider the following two properties:

1. u is irrelevant for L with schema \mathbf{S}
2. u is irrelevant for $L \cup \bar{\mathbf{S}}$ with schema \mathcal{H}_Σ .

Then property 2. implies property 1., but not always vice versa.

Proof. 1. $\not\Rightarrow$ 2.: Here comes a counter-example. Let $L = \{\varepsilon\}$ and $\mathbf{S} = \{\varepsilon, a\}$. Then prefix a is irrelevant for L with schema \mathbf{S} . However, prefix a is relevant for $L \cup \bar{\mathbf{S}} = \mathcal{H}_\Sigma \setminus \{a\}$ with schema \mathcal{H}_Σ , since for the nested words $v = \varepsilon$ and $v' = a$ we have $a \cdot v \notin L \cup \bar{\mathbf{S}}$ and $a \cdot v' \in L \cup \bar{\mathbf{S}}$.

1. \Leftarrow 2.: We assume property 2. Consider arbitrary nested words $v, v' \in \mathcal{N}_\Sigma$ and a nested word suffix $w \in \text{suffs}(\mathcal{N}_\Sigma)$ such that $u \cdot v \cdot w, u \cdot v' \cdot w \in \mathbf{S}$. It is sufficient to show that $u \cdot v \cdot w \in \mathbf{S}$ and $u \cdot v' \cdot w \in \mathbf{S}$ implies $u \cdot v \cdot w \in L \Leftrightarrow u \cdot v' \cdot w \in L$. This implication holds trivially if $u \cdot v \cdot w \in \bar{\mathbf{S}}$ or $u \cdot v' \cdot w \in \bar{\mathbf{S}}$. Otherwise, we have $u \cdot v \cdot w \in \mathbf{S}$ and $u \cdot v' \cdot w \in \mathbf{S}$. Property 2. as assumed then implies $u \cdot v \cdot w \in L \Leftrightarrow u \cdot v' \cdot w \in L$. So property 1. holds.

\square

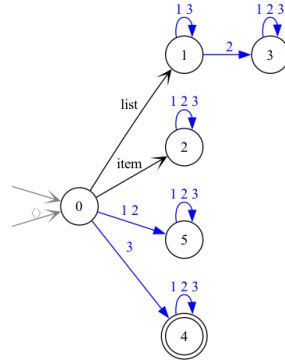


Figure 4. A dSHA for the XPath filter $[\text{self}::\text{list}/\text{child}::\text{item}]$ that is schema-complete for schema $[\text{doc}]$.

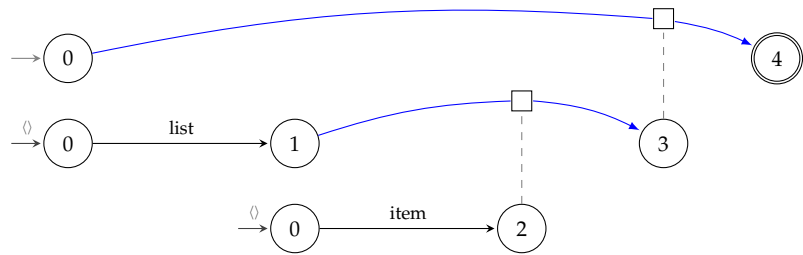


Figure 5. The unique successful run of the dSHA in Fig. 4 on the hedge $\langle \text{list} \cdot \langle \text{item} \rangle \rangle$ is the hedge $0 \cdot \langle 0 \cdot \text{list} \cdot 1 \cdot \langle 0 \cdot \text{item} \cdot 2 \rangle \cdot 3 \rangle \cdot 4$ computed as illustrated above.

4. Hedge Automata

We recall the notion of stepwise hedge automata (SHAs), introduce their downward extension (SHA^\downarrow), discuss schema-completeness for both kinds of automata, define subhedge projection states for SHA^\downarrow s, and show how to use them for evaluating SHA^\downarrow s with subhedge projection. We limit ourselves to in-memory evaluation in this section, but will discuss streaming evaluation in Section 9. The relation to several closely related notions of automata on hedges, forests or nested words are discussed in some more detail in Section 12.

4.1. Stepwise Hedge Automata

SHAs are a recent notion of automata for hedges [8] that mix up bottom-up tree and left-to-right word automata in a natural manner. They extend on stepwise tree automata [31] in that, they operate on hedges rather than unranked trees, i.e., on sequences of letters and trees containing hedges. They differ from nested word automata (NWA) [11,15] in that, they process hedges directly rather than their linearizations to nested words.

Definition 10. A stepwise hedge automaton (SHA) is a tuple $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ where Σ and \mathcal{Q} are finite sets, $I, F \subseteq \mathcal{Q}$, and $\Delta = ((a^\Delta)_{a \in \Sigma}, \langle \rangle^\Delta, @^\Delta)$ where: $a^\Delta \subseteq \mathcal{Q} \times \mathcal{Q}$, $\langle \rangle^\Delta \subseteq \mathcal{Q}$, and $@^\Delta \subseteq (\mathcal{Q} \times \mathcal{Q}) \times \mathcal{Q}$. A SHA is deterministic or equivalently a dSHA if I and $\langle \rangle^\Delta$ contain at most one element, and all relations $(a^\Delta)_{a \in \Sigma}$ and $@^\Delta$ are partial functions.

The set of states \mathcal{Q} subsumes a subset I of initial state, a subset F of final states, and a subset $\langle \rangle^\Delta$ of tree initial states. The transition rules in Δ have three forms: If $(q, q') \in a^\Delta$ then we have a letter rule that we write as $q \xrightarrow{a} q'$ in Δ . If $(q, p, q') \in @^\Delta$ then we have an apply rule that we write as: $q@p \rightarrow q'$ in Δ . And if $q \in \langle \rangle^\Delta \in \mathcal{Q}$ then we have a tree initial rule that we denote as $\langle \rangle \xrightarrow{} q$ in Δ .

We illustrate in Fig. 4 the graph of a dSHA for the XPath filter $[\text{self}::\text{list}/\text{child}::\text{item}]$. The graph of SHA is obtained and drawn as usual for state automaton on words. The states of the automaton are the nodes of the graph, in the example $\mathcal{Q} = \{0, 1, 2, 3, 4, 5\}$. The transition rules define the edges. The edge of a letter rule $q \xrightarrow{a} q'$ in Δ is annotated by the

letter $a \in \Sigma$. In the example, $\Sigma = \{\text{list}, \text{item}\}$. Any apply rule $q@p \rightarrow q'$ in Δ is drawn as an edge $q \xrightarrow{p} q'$ annotated by state $p \in \mathcal{Q}$. The initial state $0 \in I$ is indicated by an ingoing gray arrow and the tree initial state $0 \in \langle \rangle^\Delta$ by an ingoing gray arrow that is annotated by symbol $\langle \rangle$. Final states such as $4 \in F$ are indicated by a double circle.

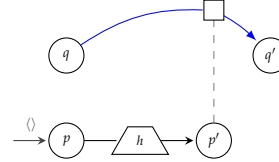
4.1.1. Semantics

We next define the semantics of SHAs, i.e., the transitions on hedges that it defines and the hedge language that it accepts. For any hedge $h \in \mathcal{H}_\Sigma$ we define the transition steps $q \xrightarrow{h} p$ wrt Δ such that for all $q, q', p, p' \in \mathcal{Q}$, $a \in \Sigma$, and $h, h' \in \mathcal{H}_\Sigma$:

$$\frac{q \xrightarrow{a} q' \text{ in } \Delta}{q \xrightarrow{a} q' \text{ wrt } \Delta} \quad \frac{q \xrightarrow{h} q' \text{ wrt } \Delta \quad q' \xrightarrow{h'} q'' \text{ wrt } \Delta}{q \xrightarrow{h \cdot h'} q'' \text{ wrt } \Delta}$$

$$\frac{q \in \mathcal{Q}}{q \xrightarrow{\varepsilon} q \text{ wrt } \Delta} \quad \frac{\langle \rangle \rightarrow p \text{ in } \Delta \quad p \xrightarrow{h} p' \quad q@p' \rightarrow q' \text{ in } \Delta}{q \xrightarrow{\langle h \rangle} q' \text{ wrt } \Delta}$$

The transitions can be used to evaluate hedges nondeterministically bottom-up and left-to-right by SHAs. The first three rules state how to evaluate sequences of trees and letters such as a usual finite state automaton, while assuming that the trees were already evaluated to states. The last rule states how to evaluate a tree $\langle h \rangle$ from some state q to some state q' . This can be visualized as follows:



For any tree initial state $p \in \langle \rangle^\Delta$, one has to evaluate the subhedge h to some state p' nondeterministically. For each p' obtained, one then returns some state q' such that $q@p' \rightarrow q'$ in Δ nondeterministically.

A hedge is accepted by A if it permits a transition from some initial state to some final state. The language $\mathcal{L}(A)$ is the set of all accepted hedges:

$$\mathcal{L}(A) = \{h \in \mathcal{H}_\Sigma \mid q \xrightarrow{h} q' \text{ wrt } \Delta, q \in I, q' \in F\}$$

4.1.2. Runs

Runs can represent the whole history of a single choice of the nondeterministic evaluator on a hedge. For instance, the unique successful run of the dSHA in Fig. 4 on the hedge $h = \langle h' \rangle$ with subhedge $h' = \text{list} \cdot \langle \text{item} \rangle$ is the hedge $0 \cdot \langle 0 \cdot \text{list} \cdot 1 \cdot \langle 0 \cdot \text{item} \cdot 2 \rangle \cdot 3 \rangle \cdot 4$ whose computation is illustrated in Fig. 5. On the top level, the run justifies the transition $0 \xrightarrow{h} 4$ wrt Δ from the initial state 0 to a final state 4. When started in state 0, the subhedge h' needs to be evaluated first, so the run on the upper level needs to suspend until this is done. The subrun on h' eventually justifies the transition $0 \xrightarrow{h'} 3$ wrt Δ . At this time point the run of the upper level can be resumed and continue in state 4 since $0@3 \rightarrow 4$ in Δ .

When drawing runs, we illustrate any suspension/resumption mechanism by a box, for instance by the box in the edge $0 \text{---}\square \text{---} 4$. The box stands for a future value computed by the lower level, causing the upper level to suspend until its arrival. Eventually, the box is filled by state 3 as illustrated by $3 \text{---}\square$, so that the computation can be resumed on the upper level.

$$\begin{array}{c}
\frac{Q \subseteq \mathcal{Q} \quad a \in \Sigma \quad Q' = \{q' \in \mathcal{Q} \mid q \xrightarrow{a} q' \in \Delta, q \in Q\}}{Q \xrightarrow{a} Q' \in \Delta^{det}} \\
\frac{\langle \rangle^\Delta \neq \emptyset \quad Q_1, Q_2 \subseteq \mathcal{Q} \quad Q' = \{q' \in \mathcal{Q} \mid q_1 @ q_2 \rightarrow q' \in \Delta, q_1 \in Q_1, q_2 \in Q_2\}}{\langle \rangle^\Delta \in \Delta^{det} \quad Q_1 @ Q_2 \rightarrow Q' \in \Delta^{det}}
\end{array}$$

Figure 6. The determinisation Δ^{det} of the transition rules Δ of a SHA with state set \mathcal{Q} .

We next define runs of SHAs on hedges formally. Whether a hedge with letters and states $R \in \mathcal{H}_{\Sigma \cup \mathcal{Q}}$ is a Δ -run or simply a run on a hedge $h \in \mathcal{H}_\Sigma$ – written $R \in \text{run}^\Delta(h)$ – is defined by the following rules:

$$\begin{array}{c}
\frac{q \xrightarrow{a} q' \text{ in } \Delta}{q \cdot a \cdot q' \in \text{run}^\Delta(a)} \quad \frac{q \cdot R \cdot q' \in \text{run}^\Delta(h) \quad q' \cdot R' \cdot q'' \in \text{run}^\Delta(h')}{q \cdot R \cdot q' \cdot R' \cdot q'' \in \text{run}^\Delta(h \cdot h')} \\
\frac{\text{true}}{q \in \text{run}^\Delta(\varepsilon)} \quad \frac{\langle \rangle \xrightarrow{} p \text{ in } \Delta \quad p \cdot R \cdot p' \in \text{run}^\Delta(h) \quad q @ p' \rightarrow q' \text{ in } \Delta}{q \cdot \langle R \rangle \cdot q' \in \text{run}^\Delta(\langle h \rangle)}
\end{array}$$

Note that if $R \in \text{run}^\Delta(h)$ then h can be obtained by removing all states from R , i.e., $\text{proj}_\Sigma(R) = h$. Any run $R \in \text{run}^\Delta(h)$ can be identified with a mapping of prefixes of the nested word $nw(h)$ to states in \mathcal{Q} . Note that $nw(h) = \text{proj}_\Sigma(nw(R))$. For any prefix v of $nw(h)$ therefore there exists a unique prefix r of the nested word $nw(R)$ that ends with some state $q \in \mathcal{Q}$ and satisfies $\text{proj}_\Sigma(r) = v$. We then call q the state of R at prefix v . The run $0 \cdot \langle 0 \cdot \text{list} \cdot 1 \cdot \langle 0 \cdot \text{item} \cdot 2 \rangle \cdot 3 \rangle \cdot 4$ on hedge $\langle \text{list} \langle \text{item} \rangle \rangle$ from Fig. 5, for instance, assigns state 0 to the nested word prefixes ε and \langle and state 1 to the nested word prefix $\langle \cdot \text{list}$, etc.

A Δ -run is called a run of $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ if it starts with some state in I . A run of A is called successful if it ends with some state in F .

Lemma 11. $h \in \mathcal{L}(A)$ iff there exists a successful run $R \in \text{run}^\Delta(h)$ of A .

Proof. Straightforward by induction on h . \square

4.1.3. Determinization

SHAs can always be determinized by using the subset construction known from the determinization of finite state automata on words or of tree automata.

Given a SHA $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ the state set of its determinization is $\mathcal{Q}^{det} = 2^{\mathcal{Q}}$. The transition rules Δ^{det} of the determinization are given in Fig. 6.

4.1.4. In-Memory Membership Testing

Testing membership $h \in \mathcal{L}(A)$ for a hedge h , whose graph is stored in-memory, can be done by computing the unique run of the determinization of A in a bottom-up and left-to-right manner, and testing whether it is successful. Alternatively, one can compute the unique set $P \subseteq \mathcal{Q}$ such that $I \xrightarrow{h} P$ with respect to the determinization of A and test whether $P \cap F \neq \emptyset$. The computations are essentially the same, but the full history of the computation is lost when returning only the set P of states reached and not the run.

For determinizing SHAs we can rely on a variant of the usual subset construction that is well-known from finite state automata on words or standard tree automata (see e.g. [8,16]). When it comes to membership testing, on-the-fly determinization is sufficient, which creates only the part of the deterministic automaton that is used by the run on h . This part is of linear size $O(|h|)$ while the full determinization of A may be of exponential size $O(2^{|\mathcal{Q}|})$. In this way, membership $h \in \mathcal{L}(A)$ can be tested in time $O(|h||A|)$.

4.1.5. Hedge Accessibility

For any set $P \subseteq \mathcal{Q}$ we define the set of states accessible from P via some hedge by:

$$\text{acc}^\Delta(P) = \{q' \in \mathcal{Q} \mid \exists q \in P. \exists h \in \mathcal{H}_\Sigma. q \xrightarrow{h} q' \text{ wrt } \Delta\}$$

We note that $\text{acc}^\Delta(P)$ can be computed from Δ and P in linear time in the size of Δ . since a state is hedge accessible from P if and only if it is accessible for P in the graph of Δ . Or by computing the least fixed point of the following ground Datalog program depending on P and Δ , that is of linear size in A :

$$\frac{q \in P}{\text{acc}(p)}. \quad \frac{q \xrightarrow{a} q' \text{ in } \Delta}{\text{acc}(q') :- \text{acc}(q)}. \quad \frac{q@p \rightarrow q' \text{ in } \Delta}{\text{acc}(q') :- \text{acc}(q), \text{acc}(p)}.$$

Clearly $\mathcal{L}(A) = \emptyset$ if and only if $F \cap \text{acc}^\Delta(I) = \emptyset$. Therefore, emptiness can be decided in linear time for SHAs. This is in contrast to the more general notion of SHA^\downarrow s that we introduce next.

4.2. Downward Stepwise Hedge Automata

The evaluation of SHAs operates in a bottom-up and left-to-right manner, but cannot pass any information top-down. We next propose an extension of SHAs to SHA^\downarrow s, adding the ability to pass finite state information top-down.

SHA^\downarrow s are basically equal to Neumann and Seidl's pushdown forest automata [32] but lifted from (labeled) forests to (unlabeled) hedges. See Section 12 for a more detailed discussion on the relationship of SHA^\downarrow s to other automata notions and NWA's in particular.

Definition 12. A downward stepwise hedge automaton (SHA^\downarrow) is a tuple $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ where Σ and \mathcal{Q} are finite sets, $I, F \subseteq \mathcal{Q}$, and $\Delta = ((a^\Delta)_{a \in \Sigma}, \langle \rangle^\Delta, @^\Delta)$. Furthermore, $a^\Delta \subseteq \mathcal{Q} \times \mathcal{Q}$, $\langle \rangle^\Delta \subseteq \mathcal{Q} \times \mathcal{Q}$, and $@^\Delta \subseteq (\mathcal{Q} \times \mathcal{Q}) \times \mathcal{Q}$. A SHA^\downarrow is called deterministic or equivalently a $d\text{SHA}^\downarrow$ if I contains at most one element, and all relations $\langle \rangle^\Delta$, a^Δ , and $@^\Delta$ are partial functions.

The only difference to SHAs is the form of the tree opening rules. If $(q, q') \in \langle \rangle^\Delta \in \mathcal{Q}$ then we have a tree initial rule that we denote as: $q \xrightarrow{\langle \rangle} q' \text{ in } \Delta$.

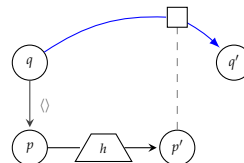
So here the state q' where the evaluation of a subhedge starts depends on the state q of the parent.

4.2.1. Semantics

The definition of transition steps for SHA^\downarrow s differs from that of SHAs only in the usage of opening rules in the following inference rule:

$$\frac{q \xrightarrow{\langle \rangle} p \text{ in } \Delta \quad p \xrightarrow{h} p' \text{ wrt } \Delta \quad q@p' \rightarrow q' \text{ in } \Delta}{q \xrightarrow{\langle h \rangle} q' \text{ wrt } \Delta}$$

This means that the evaluation of the subhedge h starts in some state p such that $q \xrightarrow{\langle \rangle} p \text{ in } \Delta$.



So the restart state p that is chosen may now depend on the state q above. This is how finite state information is passed top-down by SHA^\downarrow s. SHAs in contrast, operate purely bottom-up and left-to-right.

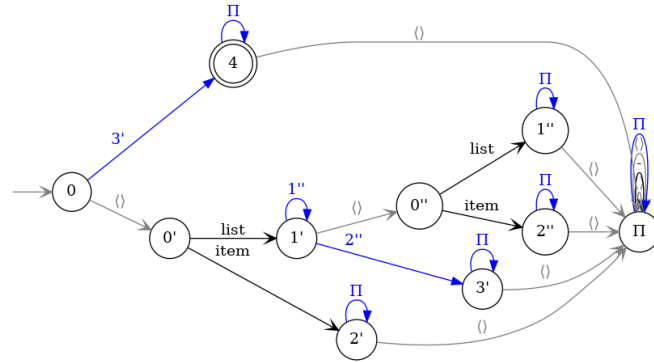


Figure 7. The deterministic SHA^\downarrow with subhedge projection state Π obtained by safe-no-change projection.

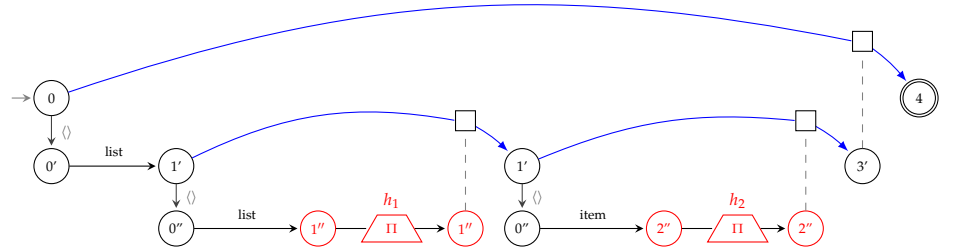


Figure 8. A successful run of the SHA^\downarrow in Fig. 7 on the hedge $\langle \text{list} \cdot \langle \text{list} \cdot h_1 \rangle \cdot \langle \text{item} \cdot h_2 \rangle \rangle$.

4.2.2. Runs

The notion of runs can be adapted straightforwardly from SHAs to SHA^\downarrow s. When in state q , it is sufficient to restart the computation in subhedges on the state p such that $q \overset{\diamond}{\rightarrow} p \in \Delta$. In this way, finite state information is passed down (while for SHAs some tree initial is to be chosen that is independent of q). The only rule of runs to be changed is the following:

$$\frac{q \overset{\diamond}{\rightarrow} p \text{ in } \Delta \quad p \cdot R \cdot p' \in \text{run}^\Delta(h) \quad q@p' \rightarrow q' \text{ in } \Delta}{q \cdot \langle R \rangle \cdot q' \in \text{run}^\Delta(\langle h \rangle)}$$

An example of a run of the dSHA^\downarrow in Fig. 7 is shown in Fig. 8. Here, the information on the level of nodes is passed down. It is represented by the number of primes. For instance, when opening the first subtree labeled with *item* we use the transition rule $1' \overset{\diamond}{\rightarrow} 0''$ stating that one moves from state 1 on the first level to state 0 on the second level.

One can see that all nodes of the subtrees h_1 and h_2 are evaluated to the projection state Π . So one can jump over these subtrees without reading them. This is justified by the fact that they are on level 2, the information that the evaluator of this SHA^\downarrow was passing down.

4.2.3. In-Memory Membership Testing

Testing membership $h \in \mathcal{L}(A)$ for a SHA^\downarrow s in memory can again be done by computing the unique run of the determinization of A . As before, the membership tester can be based on on-the-fly determinization. However, the determinization procedure for SHA^\downarrow s is more complicated than for SHAs, since SHA^\downarrow s can pass information top-down and not only bottom-up and left-to-right. Determinization therefore does not only rely on the pure subset construction, but also uses pairs of states in the subsets, basically in the same way as for nested word automata (NWA) [14,15]. This is needed to deal with the stack of states that were seen above. Therefore, each determinization step may cost time $O(|A|^5)$ as stated for instance in [33]. The upper time bound for membership testing for SHA^\downarrow s is thus in time $O(|h||A|^5)$, and no longer combined linear as it is for SHAs.

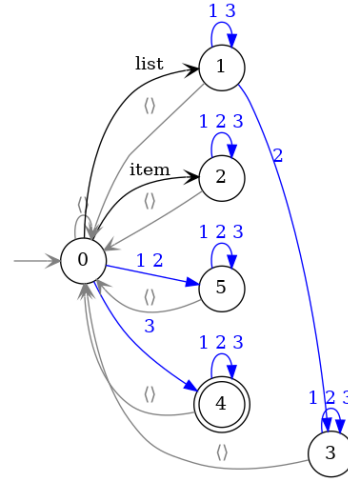


Figure 9. The $\text{SHA}^\downarrow A^{\text{down}}$ for the dSHA A for the filter $[\text{self}::\text{list}/\text{child}::\text{item}]$ in Fig. 4.

4.2.4. Relationship to SHAS

Any SHA $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ can be mapped to a $\text{SHA}^\downarrow A^{\text{down}} = (\Sigma, \mathcal{Q}, \Delta^{\text{down}}, I, F)$ while preserving the runs and determinism. The only change of Δ^{down} compared to Δ is described by the following rule:

$$\frac{\langle \rangle \xrightarrow{\Delta} p \text{ in } \Delta \quad q \in \mathcal{Q}}{q \xrightarrow{\Delta^{\text{down}}} p \text{ in } \Delta^{\text{down}}}$$

Independently of the current state $q \in \mathcal{Q}$, the $\text{SHA}^\downarrow A^{\text{down}}$ can start the evaluation of the subhedge of a subtree in any open tree state $p \in \langle \rangle^\Delta$. For instance, the $\text{dSHA}^\downarrow A^{\text{down}}$ for dSHA A from Fig. 4 from the introduction is given in Fig. 9. Conversely, one can convert any $\text{dSHA}^\downarrow A$ into an equivalent SHA by introducing nondeterminism to eliminate top-down processing. See Section 12.2 for the construction.

4.2.5. Hedge Accessibility

Hedge accessibility can be defined for SHA^\downarrow s identically as for SHAs. However, the set $\text{acc}^\Delta(P)$ can no longer be computed in linear time in the size of Δ – in contrast to SHAs. It can still be done in cubic time, similarly as for NWA [22]. The problem is that hedge accessibility for SHA^\downarrow s does no more coincide with the accessibility notion of the automaton's graph.

For instance, in the SHA^\downarrow from Fig. 7, we have $\text{acc}^\Delta(\{0\}) = \{4\}$. Note that $0'$ does not belong to this set, even though there is a transition rule $0 \xrightarrow{\langle \rangle} 0'$ in Δ , making node $0'$ accessible from node 0 in the graph of the automaton. This is because $0'$ is not accessible over any hedge from 0, i.e., there is no hedge h such that $0 \xrightarrow{h} 0'$ wrt Δ . Note that $0 \cdot \langle \cdot 0' \rangle$ is a factor of the nested word of some run of Δ . This shows that $0'$ is accessible from 0 over a nested word prefix nevertheless. For any $\text{SHA}^\downarrow A$, hedge accessibility can be computed by the following ground Datalog program of cubic size depending on the number of states of A :

$$\frac{p@q' \rightarrow q \text{ in } \Delta \quad p \xrightarrow{\langle \rangle} p' \text{ in } \Delta \quad p \in \mathcal{Q}}{\text{acc}(p, q) :- \text{acc}(p', q').} \quad \frac{p \in \mathcal{Q}}{\text{acc}(p, p).} \quad \frac{p \xrightarrow{a} q \text{ in } \Delta}{\text{acc}(p, q).}$$

$$\frac{p, q, r \in \mathcal{Q}}{\text{acc}(p, q) :- \text{acc}(p, r), \text{acc}(r, q).}$$

Due to the cubic time of hedge accessibility for SHA^\downarrow s, we will use hedge accessibility only for SHAs where it is in linear time.

4.3. Schema Completeness

Schemas are inputs of the membership problem that we consider, while completeness is a fundamental property of automata. To combine both aspects appropriately, we propose the notion of schema completeness, that we define uniformly for both SHAs and SHA^\downarrow s.

We call a set Δ of transition rules of a SHA^\downarrow complete if all its relations $(a^\Delta)_{a \in \Sigma}$, $\langle \rangle^\Delta$ and $@^\Delta$ are total. For SHAs we require that $\langle \rangle^\Delta$ is a nonempty set (instead of a total relation). A SHA or $\text{SHA}^\downarrow A$ is called complete if Δ is complete and $I \neq \emptyset$. We can complete any SHA or SHA^\downarrow by adding a fresh sink state and also transition rules into the sink state for all left-hand sides, for which no other transition rule exists. The sink state is made initial if and only if there was no initial state before. It is not final though. We denote the completion of A by $\text{compl}(A)$.

Definition 13. A partial run of A on a hedge h is a prefix r of some run of $\text{compl}(A)$ on h such that:

- r ends with some state of \mathcal{Q} , and
- r does not contain the sink state of $\text{compl}(A)$.

A partial run r of A on h is called blocking if there does not exist any partial run r' of A on h such that r is a strict prefix of r' .

For instance, consider the hedge $h = \langle \text{list} \cdot \langle \text{item} \rangle \rangle$. The dSHA in Fig. 4 has the unique run $R = 0 \cdot \langle \cdot 0 \cdot \text{list} \cdot 1 \cdot \langle \cdot 0 \cdot \text{item} \cdot 2 \rangle \cdot 3 \cdot \rangle \cdot 4$ on h that is illustrated in Fig. 5. The partial runs of h are thus all prefixes of $nw(R)$ that end with some state, i.e., 0 , $0 \cdot \langle \cdot 0$, $0 \cdot \langle \cdot 0 \cdot \text{list} \cdot 1$, etc. None of these partial runs is blocking.

Definition 14. A schema for an automaton A is a hedge language $\mathbf{S} \subseteq \mathcal{H}_\Sigma$ such that $\mathcal{L}(A) \subseteq \mathbf{S}$. We call the automaton A schema-complete for schema \mathbf{S} if no hedge $h \in \mathbf{S}$ has some blocking partial run of A .

Schemas \mathbf{S} are usually used to restrict the type of hedges that some problem may accept as input. The dSHA pattern matching problem for a schema \mathbf{S} takes two inputs: a hedge $h \in \mathbf{S}$ and an automaton A – rather than an nested regular expression e . The automaton A then selects those input hedges $h \in \mathbf{S}$ that match the pattern, i.e., such that $h \in \mathcal{L}(A)$. For this reason, we assume that \mathbf{S} is a schema for A , i.e., $\mathcal{L}(A) \subseteq \mathbf{S}$. We will often assume that A is schema-complete for \mathbf{S} , so that no partial run of A on any input hedge $h \in \mathbf{S}$ may ever block. This can always be achieved based on the next Lemma 16.

Example 15. The dSHA in Fig. 4 for the XPath filter $[\text{self}::\text{list}/\text{child}::\text{item}]$ has signature $\Sigma = \{\text{item}, \text{list}\}$. It is schema-complete for schema $\llbracket \text{doc} \rrbracket$. To make this happen, we added state 5 to this automaton, which is not used in any successful run. But still, this SHA is not complete. For completing it, we would have to run 5 into a sink states by adding many transitions into it, such as $0@4 \rightarrow 5$ and $0@5 \rightarrow 5$, but also for all other states $q \in \{1, 2, 3, 4, 5\}$ the transition rules $q \xrightarrow{\text{list}} 5$, $q \xrightarrow{\text{item}} 5$, $q@4 \rightarrow 5$ and $q@5 \rightarrow 5$.

Automaton completion may raise problems to safe-no-change projection: it may render it incomplete for subhedge projection in many examples, for which it was complete otherwise. This happens for instance for the dSHA in Fig. 4 for the XPath query $[\text{self}::\text{list}/\text{child}::\text{item}]$, completed in Example 15: the states 2, 3, 5, 6, that otherwise can no more be changed before completion, can be changed after completion into the sink.

Without schema-completeness, however, safe-no-change projection may be unsound: it might overlook the rejection of hedges inside the schema. This is why we have to assume schema-completeness for the input automata of safe-no-change projection, and do not what

to assume full completeness there. For congruence projection, schema-completeness will be assumed implicitly in the setting taken there. Further automata completion will not affect there completeness for subhedge projection there.

We note that schema-completeness is well-defined even for nonregular schemas. For safe-no-change projection, we indeed don't have to restrict schemas to be regular, while for congruence projection only regular schemas are considered. Furthermore, if A is schema-complete for the universal schema $\mathbf{S} = \mathcal{H}_\Sigma$ and does not have inaccessible states then A is complete.

Schema-completeness of deterministic automata for regular schemas can always be made to hold. In order to show this, we define that A is *compatible with schema \mathbf{S}* if for any hedge $h \in \mathbf{S}$ there exists a run of A in $run^\Delta(h)$. Schema-completeness implies compatibility, as shown by the next lemma. The converse is true only when assuming determinism.

Lemma 16. *Let A be a deterministic SHA with schema \mathbf{S} . Then A is compatible with \mathbf{S} iff A is schema-complete for \mathbf{S} .*

Proof. Suppose that A is schema-complete for \mathbf{S} . Wlog., we can assume $\mathbf{S} \neq \emptyset$. Note that if $I = \emptyset$, then ε would be a blocking partial run for any hedge in $\mathbf{S} \setminus \{\varepsilon\}$. Since $\mathbf{S} \neq \emptyset$ it follows that there exists $q_0 \in I$. So q_0 is a partial run for any hedge $h \in \mathbf{S} \setminus \{\varepsilon\}$. Since there exist no blocking partial runs by schema-completeness, this run can be extended step by step to a run on any $h \in \mathbf{S}$. So for any hedge $h \in \mathbf{S}$ there exists some run by A , showing compatibility.

For the converse, let A be compatible with \mathbf{S} and deterministic. We have to show that A is schema-complete for \mathbf{S} . Let v be a partial run of A on $h \in \mathbf{S}$. By compatibility, there exists a run $R \in run^\Delta(h)$ that starts in some initial state of A . By determinism, v must be a prefix of $nw(R)$. Thus, v is not blocking. \square

Proposition 17. *Any SHA A with regular schema \mathbf{S} can be made schema-complete for \mathbf{S} .*

Proof. By Lemma 16 it is sufficient to make A compatible with \mathbf{S} . Let B be a dSHA with $\mathbf{S} = \mathcal{L}(B)$. Automaton B is compatible with \mathbf{S} . Since B is deterministic, Lemma 16 implies that B is schema-complete for \mathbf{S} . Since \mathbf{S} is a schema for A with $\mathcal{L}(A) \subseteq \mathcal{L}(B)$. We then compute the completion of $compl(A)$. The product $C = B \times compl(A)$ with final states $F^C = F^B \times F^A$ is schema-complete for schema \mathbf{S} . Furthermore, since $\mathcal{L}(A) \subseteq \mathbf{S} = \mathcal{L}(B)$ it follows that $\mathcal{L}(C) = \mathcal{L}(B) \cap \mathcal{L}(A) = \mathcal{L}(A)$. \square

Finally note that the schema \mathbf{S} can be recognized by the same product $C = B \times compl(A)$ except for replacing the set of final states F^C by the set of schema final states $F^S = F^B \times (\mathcal{Q}^A \cup \{sink\})$. We have $\mathbf{S} = \mathcal{L}(C[F^C/F^S])$ where $C[F^C/F^S]$ is the dSHA obtained from C by replacing the set of final states F^C by F^S .

5. Subhedge Projection

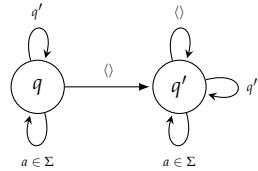
We next introduce the concept of subhedge projection states for SHA^\downarrow s in order to distinguish prefixes of hedges that are subhedge irrelevant, and to evaluate SHA^\downarrow s with subhedge projection.

5.1. Subhedge Projection States

Intuitively, a subhedge projection state can only loop in itself until a hedge closes (and then it is applied). When going down into a subtree, the subhedge projection state may still be changed into some other looping state. When leaving the subtree, however, one must come back to the original subhedge projection state. Clearly, any sink is a subhedge projection state. The even more interesting cases, however, are subhedge projection states that are not sinks.

We start with SHA^\downarrow s without any restrictions but will impose schema-completeness and determinism later on.

Definition 18. We call a state $q \in \mathcal{Q}$ a subhedge projection state of Δ if there exists $q' \in \mathcal{Q}$ called the witness of q such that the set of transition rules of Δ containing q' or q on the leftmost position is included in the following set:

$$\begin{aligned} & \{q \xrightarrow{\langle \rangle} q', q@q' \rightarrow q, q' \xrightarrow{\langle \rangle} q', q'@q' \rightarrow q'\} \\ & \cup \{q' \xrightarrow{a} q', q \xrightarrow{a} q \mid a \in \Sigma\} \end{aligned}$$


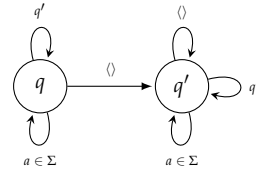
The set of all subhedge projection states of Δ is denoted by \mathcal{Q}_{shp}^Δ .

For any complete SHA^\downarrow , the above set of transition rules must be equal to the set of all transition rules of Δ with q or q' on the leftmost position. But if the SHA^\downarrow is only schema-complete for some schema \mathbf{S} then not all these transitions must be present. Note also that a subhedge projection state q may be equal to its witness q' . Therefore any witness q' of some subhedge projection state is itself a subhedge projection state with witness q' .

In the example $\text{dSHA}^\downarrow A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ in Fig. 7, we have $\mathcal{Q}_{shp}^\Delta = \{\Pi, 4, 2', 3', 1'', 2''\}$. The witness of all these subhedge projection states is Π . Note that not all possible transitions are present for the states in $\mathcal{Q}_{shp}^\Delta \setminus \{\Pi\}$, given that this automaton is not complete (but still schema-complete for schema $\llbracket \text{doc} \rrbracket$).

Lemma 19. If $q \in \mathcal{Q}_{shp}^\Delta$ is a subhedge projection state and $q \xrightarrow{h} p$ wrt Δ then $q = p$.

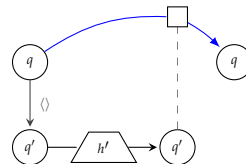
Proof. Suppose that $q \xrightarrow{h} p$ wrt Δ and that q is a subhedge projection state of Δ with witness q' . So the only possible transitions with q or q' on the left-hand side are the following:



Suppose that $h = t_1 \cdot \dots \cdot t_n$ is a sequence of trees or letters $t_i \in \langle \mathcal{H}_\Sigma \rangle \cup \Sigma$ where $1 \leq i \leq n$. Then there exists runs $R_i \in \text{run}^\Delta(t_i)$ and a run $R \in \text{run}^\Delta(h)$ that has the form $h = q_0 \cdot R_1 \cdot q_1 \cdot \dots \cdot R_n \cdot q_n$ where $q_0 = q$ and $q_n = p$. We prove for all $0 \leq i \leq n$ that $q_i = q$ by induction on i . For $i = 0$, this is trivial. For $i > 0$, the induction hypothesis shows that $q_{i-1} = q$

Case $t_i = a \in \Sigma$. Then $q_{i-1} \xrightarrow{a} q_i$ in Δ . Since $q_{i-1} = q$ is a subhedge projection state of Δ it follows that $q_i = q$ too.

Case $t_i = \langle h' \rangle$ for some $h' \in \mathcal{H}_\Sigma$. Since $q_{i-1} = q$ is a subhedge projection state of Δ with witness q' , the subrun of R recognizing t_i must be justified by the following diagram:



So the subrun of R of h' may only contain the witness q' and furthermore, $q_i = q$.

□

5.2. Completeness for Subhedge Projection

Before defining when a SHA^\downarrow is complete for subhedge projection, we show that subhedge projection states in runs of deterministic SHAs may only occur at irrelevant prefixes.

Proposition 20. *Let $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ be a dSHA^\downarrow with schema \mathbf{S} , $R \in \text{run}^\Delta(h)$ the run of $\text{compl}(A)$ on some hedge $h \in \mathbf{S}$, and q a subhedge projection state for Δ . Then for any prefix $r \cdot q$ of $\text{nw}(R)$, the prefix $\text{proj}_\Sigma(r)$ of $\text{nw}(h)$ is subhedge irrelevant for $\mathcal{L}(A)$ for schema \mathbf{S} .*

Proof. Let $R \in \text{run}^\Delta(h)$ be the unique run of $\text{compl}(A)$ on some hedge $h \in \mathbf{S}$. Suppose that $r \cdot q \cdot s = \text{nw}(R)$ for some subhedge projection state q of Δ and let $v = \text{proj}_\Sigma(r)$ and $w = \text{proj}_\Sigma(s)$. Since $v \cdot w = \text{nw}(h)$ and $h \in \mathbf{S}$ it follows that $v \cdot w \in \text{nw}(\mathbf{S})$. We fix some hedge $h' \in \mathcal{H}_\Sigma$ such that $v \cdot \text{nw}(h') \cdot w \in \text{nw}(\mathbf{S})$ arbitrarily. Let $\tilde{h} \in \mathcal{H}_\Sigma$ such that $\text{nw}(\tilde{h}) = v \cdot \text{nw}(h') \cdot w$. For the subhedge irrelevance of v , we have to show that:

$$h \in \mathcal{L}(A) \Leftrightarrow \tilde{h} \in \mathcal{L}(A)$$

“ \Rightarrow ” We suppose that $h \in \mathcal{L}(A)$ and have to show that $\tilde{h} \in \mathcal{L}(A)$. For the partial run $r \cdot q$ of A on \tilde{h} there must exist R' and p such that $q \cdot R' \cdot p \in \text{run}^\Delta(h')$. Since q is a subhedge projection state of Δ , it follows by Lemma 19 that $q = p$. Hence $r \cdot q \cdot \text{nw}(R') \cdot q \cdot s$ is a run of A on \tilde{h} . Since R was successful for A , s must end in F , and thus the above run on \tilde{h} is successful for A too. Hence $\tilde{h} \in \mathcal{L}(A)$ as we had to show.

“ \Leftarrow ” We suppose that $\tilde{h} \in \mathcal{L}(A)$ and have to show that $h \in \mathcal{L}(A)$. Let \tilde{R} be the unique run of $\text{compl}(A)$ on \tilde{h} . By determinism, \tilde{R} must have the prefix $r \cdot q$. So there exist R', p', s such that $p \cdot R' \cdot q \in \text{run}^\Delta(h')$ and $r \cdot q \cdot R' \cdot p \cdot s = \tilde{R}$. Lemma 19 shows that $p = q$. Hence, $r \cdot q \cdot s$ is the unique run of $\text{compl}(A)$ on h and this run is accepting. So $h \in \mathcal{L}(A)$.

□

We note that Proposition 20 would not hold without assuming determinism. To see this, we can add some sink to any dSHA as a second initial state. One can then always go to this sink, which is a subhedge projection state. Nevertheless, no prefix going to the sink may be subhedge irrelevant.

Proposition 20 shows that subhedge projection states permit distinguishing prefixes that are subhedge irrelevant for deterministic SHA^\downarrow s. An interesting question is whether all subhedge irrelevant prefixes can be found in this way. A closely related question is whether there for all regular patterns there exist dSHAs that project all irrelevant subhedges.

Example 21. *Reconsider Example 4 with $\Sigma = \{a\}$, regular pattern $L = \{\langle a \rangle\}$ and regular schema $\mathbf{S} = L \cup \{\langle \rangle \cdot a\}$. The prefix $\langle \rangle$ is irrelevant for L and \mathbf{S} , since all its continuations ending in the schema are rejected: there is only one such continuation which is $\langle \rangle \cdot a$. However, a dSHA for L and \mathbf{S} with maximal subhedge projection – as given in Fig. 17 – will not go into a subhedge projection state at this prefix. The reason is that it will go into the same state $\{3, 4\}D0$ for any prefix in $\langle \mathcal{N}_\Sigma \rangle$, since ignoring the nested word of the irrelevant subhedge. So this state must accept $\langle a \rangle$. But when reading an a in this state, the dSHA goes into the rejection state $\{5\}D0$, showing that the hedge from the schema $\langle \rangle \cdot a \in \mathbf{S}$ is rejected.*

The example shows that we have eventually to reason about with sets of prefixes. We first lift the notion of subhedge irrelevance to prefix sets.

Definition 22. *Let $\mathbf{S} \subseteq \mathcal{H}_\Sigma$ be a schema and $L \subseteq \mathbf{S}$ a language satisfying this schema. A set of nested word prefixes $U \subseteq \text{prefs}(\mathcal{N}_\Sigma)$ is called subhedge relevant for L with schema \mathbf{S} if there exist nested words $v', v'' \in \mathcal{N}_\Sigma$ prefixes $u', u'' \in U$ such that $\text{diff}_\Sigma^L(u' \cdot v', u'' \cdot v'')$. Otherwise, the set U is called subhedge irrelevant for L wrt \mathbf{S} .*

In order to explain the problem of Example 21, we define for each nested word prefix $w \in \text{prefs}(\mathcal{N}_\Sigma)$ a class $\text{class}_S^L(w)$ of similar prefixes up replacing the nested word of any irrelevant subhedge by \mathcal{N}_Σ from the left to the right. For all prefixes $w \in \text{prefs}(\mathcal{N}_\Sigma)$, nested words $v \in \mathcal{N}_\Sigma$, letters $a \in \Sigma$, subsets of prefixes $U \subseteq \text{prefs}(\mathcal{N}_\Sigma)$, and $l \in \hat{\Sigma}$ letters or parenthesis:

$$\begin{aligned} \text{class}_S^L(w) &= \text{class_pr}_{\{\varepsilon\}}(w) \\ \text{class_pr}_U(\varepsilon) &= \begin{cases} U \cdot \mathcal{N}_\Sigma & \text{if } U \text{ irrelevant for } L \text{ wrt } \mathbf{S} \\ U & \text{else} \end{cases} \\ \text{class_pr}_U(w \cdot \langle \cdot v \rangle) &= \text{class_ins}_{U'}(\langle \cdot v \rangle) \text{ where } U' = \text{class_pr}_U(w) \\ \text{class_pr}_U(a \cdot v) &= \text{class_ins}_{U'}(a, v) \text{ where } U' = U \cdot a \\ \text{class_pr}_U(\langle v' \rangle \cdot v) &= \text{class_ins}_{U'}(\langle \cdot v' \rangle) \text{ where } U' = \text{class_pr}_U(\langle \cdot v' \rangle) \\ \text{class_ins}_U(l, v) &= \begin{cases} U \cdot \mathcal{N}_\Sigma & \text{if } U \text{ irrelevant for } L \text{ wrt } \mathbf{S} \\ \text{class_pr}_{U.l}(v) & \text{else} \end{cases} \end{aligned}$$

Definition 23. A prefix $u \in \text{prefs}(\mathcal{N}_\Sigma)$ is called strongly subhedge irrelevant for L wrt \mathbf{S} if the set $\text{class}_S^L(u)$ is subhedge irrelevant for L wrt \mathbf{S} .

Example 24. In our running Example 21 where $L = \{\langle a \rangle\}$, $\mathbf{S} = L \cup \{\langle \cdot \rangle \cdot a\}$ and $\Sigma = \{a\}$ we have $\text{class}_S^L(\langle \cdot \rangle) = \langle \cdot \mathcal{N}_\Sigma$, so the prefix $\langle \cdot \rangle$ is strongly subhedge irrelevant. The set $\text{class}_S^L(\langle a \rangle) = \langle \mathcal{N}_\Sigma \rangle$ is subhedge relevant, so the prefix $\langle a \rangle$ is not strongly subhedge irrelevant. The set $\text{class}_S^L(\langle \cdot \rangle \cdot a) = \langle \mathcal{N}_\Sigma \rangle \cdot a \cdot \mathcal{N}_\Sigma$ is subhedge irrelevant, so the prefix $\langle \cdot \rangle \cdot a$ is strongly subhedge irrelevant.

Definition 25. A $d\text{SHA}^\downarrow A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ is called complete for subhedge projection for schema \mathbf{S} if A is schema-complete for \mathbf{S} and for all hedges $h \in \mathbf{S}$ and all prefixes v of $\text{nw}(h)$ that are strongly subhedge irrelevant for Δ , the unique run of A on h goes to some subhedge projection state of Δ at v .

Example 26. The $d\text{SHA}$ in Fig. 17 is complete for subhedge projection for our running Example 21 where $L = \{\langle a \rangle\}$ and $\mathbf{S} = L \cup \{\langle \cdot \rangle \cdot a\}$. It was obtained by congruence projection. On all prefixes of the subhedge irrelevant class $\text{class}_S^L(\langle \cdot \rangle)$ it goes to the subhedge projection state $\{0\}D1$. On all prefixes of the class $\text{class}_S^L(\langle a \rangle)$ which is not subhedge irrelevant, it goes to the state $\{3, 4\}D0$ which is not a subhedge projection state. And on all prefixes of the subhedge irrelevant class $\text{class}_S^L(\langle \cdot \rangle \cdot a)$ it goes to the subhedge projection state $\{5\}D0$.

Example 27. The $d\text{SHA}^\downarrow$ from Fig. 7 is complete for subhedge projection with schema $\llbracket \text{doc} \rrbracket$. It can be obtained by safe-no-change projection. In general, however, $d\text{SHA}^\downarrow$ s obtained by safe-no-change projection may not be complete for subhedge projection, as we will discuss in Section 6.3.

5.3. In-Memory Evaluator with Subhedge Projection

We next show how to refine the transitions for SHA^\downarrow s by subhedge projection. We define the transition relation with subhedge projection $\xrightarrow{h}_{shp} \subseteq \mathcal{Q} \times \mathcal{Q}$ with respect to Δ such that for all hedges $h, h' \in \mathcal{H}_\Sigma$ and letters $a \in \Sigma$:

$$\begin{array}{c} \frac{q \in \mathcal{Q}_{shp}^\Delta \quad h \in \mathcal{H}_\Sigma}{q \xrightarrow{h}_{shp} q \text{ wrt } \Delta} \quad \frac{q \notin \mathcal{Q}_{shp}^\Delta \quad q \xrightarrow{a} q' \text{ in } \Delta}{q \xrightarrow{a}_{shp} q' \text{ wrt } \Delta} \quad \frac{q \notin \mathcal{Q}_{shp}^\Delta}{q \xrightarrow{\varepsilon}_{shp} q \text{ wrt } \Delta} \\ \frac{q \notin \mathcal{Q}_{shp}^\Delta \quad q \xrightarrow{h}_{shp} q' \text{ wrt } \Delta \quad q' \xrightarrow{h'}_{shp} q'' \text{ wrt } \Delta}{q \xrightarrow{h \cdot h'}_{shp} q'' \text{ wrt } \Delta} \end{array}$$

$$\frac{q \notin \mathcal{Q}_{shp}^\Delta \quad q \xrightarrow{\langle h \rangle} p \text{ in } \Delta \quad p \xrightarrow{h}_{shp} p' \quad q@p' \rightarrow q' \text{ in } \Delta}{q \xrightarrow{\langle h \rangle}_{shp} q' \text{ wrt } \Delta}$$

The first rule says that subhedge projecting transitions stay in subhedge projection states until the end of the current subhedge is reached. This is correct by Lemma 19 under the condition that there are no blocking runs. The other rules state that the evaluator behaves as without subhedge projection in other state.

Proposition 28. *Let $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ be a SHA^\downarrow that is schema-complete for \mathbf{S} . Then for all hedges $h \in \mathbf{S}$ and states $q, p \in \mathcal{Q}$:*

$$q \xrightarrow{h} p \text{ wrt } \Delta \text{ iff } q \xrightarrow{h}_{shp} p \text{ wrt } \Delta$$

Proof. We distinguish two cases:

Case $q \notin \mathcal{Q}_{shp}^\Delta$. Then for any $p \in \mathcal{Q}$, $q \xrightarrow{h} p \text{ wrt } \Delta$ iff $q \xrightarrow{h}_{shp} p \text{ wrt } \Delta$ by definition of the projecting transitions.

Case $q \in \mathcal{Q}_{shp}^\Delta$. Then $q \xrightarrow{h}_{shp} q \text{ wrt } \Delta$. Since $h \in \mathbf{S}$ and A is schema-complete for \mathbf{S} there exists some run of A on h , and thus some state q' such that $q \xrightarrow{h} q' \text{ wrt } \Delta$. Lemma 19 shows that $q' = q$ and thus $q \xrightarrow{h} q \text{ wrt } \Delta$.

□

For evaluating a nondeterministic $\text{SHA}^\downarrow A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ with subhedge projection on some hedge h whose graph is stored in memory, we can compute the unique subset $P \subseteq \mathcal{Q}$ such that $I \xrightarrow{h}_{shp} P$ with respect to the transition relation Δ^{det} of determinization of A and test whether $P \cap F \neq \emptyset$. When reaching any subhedge projection state P' while doing so, the evaluator can directly jump to the end of the current subhedge, by applying the “stay” rule of the determinization of A :

$$\frac{P' \in \mathcal{Q}_{shp}^{\Delta^{det}} \quad h \in \mathcal{H}_\Sigma}{P' \xrightarrow{h}_{shp} P' \text{ wrt } \Delta^{det}}$$

Thereby it will avoid visiting any subhedge in a subhedge projection state of the determinization of A . The running time will thus depend linearly on the size of the non-projected part of h , under the assumption that the determinization of A is available.

As before, only the needed part of the determinization of A for evaluating h is to be produced on-the-fly. Overall, this costs polynomial time in the size of the needed part of the determinization (but not necessarily linear time since determinization of SHA^\downarrow s is more costly than for SHAs). Note also that whenever discovering a new state P' of the determinization of A , we have to test whether it is a subhedge projection state. For this, we have to compute the state P'' such that $P' \xrightarrow{\langle h \rangle} P'' \text{ wrt } \Delta^{det}$ and test whether only the transition rules allowed for subhedge projection states for P' with witness P'' are available wrt Δ^{det} .

6. Safe-No-Change Projection

We want to solve the regular pattern matching problem for hedges with subhedge projection. We assume that the regular pattern is given by a dSHA while the schema may be arbitrary, except that the dSHA must be schema-complete for it (see Section 4.3). We then present the safe-no-change projection algorithm which compiles the dSHA into some dSHA^\downarrow while introducing subhedge projection states.

The idea of the safe-no-change projection is to push information top-down by which to distinguish looping states that will safely no more be changed. Thereby, subhedge

projection states are produced as we will illustrate by examples. The soundness proof of safe-no-change projection is nontrivial and instructive for the soundness proof of congruence projection in Section 7.3. Completeness for subhedge projection cannot be expected, since the safe-no-change projection does only take simple loops into account. More general loops cannot be treated and it is not clear how that could be done. The congruence projection algorithm in Section 7.2 will eventually give an answer to this.

We keep the safe-no-change compiler independent of the schema and therefore have not even to assume its regularity. But we have to assume the schema-completeness of the input dSHA, in order to show that the output dSHA[↓] recognizes the same language within the schema. The regular pattern matching problem can then be solved in memory and with subhedge projection, by evaluating the dSHA[↓] on the input hedge, in memory and with subhedge projection as discussed in Section 5.3.

6.1. Algorithm

We now describe the safe-no-change projection algorithm. Let $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ be a SHA with schema \mathbf{S} . The algorithm takes the dSHA A as input and compiles it to some dSHA[↓] A^{snc} . We will state why A^{snc} is sound for subhedge projection under the condition that A is schema-complete for schema \mathbf{S} . We do *not* even have to assume that the schema \mathbf{S} is regular. The proof is 8 pages long and therefore delegated to the appendix.

For any set $P \subseteq \mathcal{Q}$ we define the set of states that safely lead to P :

$$safe^\Delta(P) = \{q \in \mathcal{Q} \mid acc^\Delta(\{q\}) \subseteq P\}$$

So a state q is safe for P if any hedge read from q on which the run with Δ does not block must reach some state in P . We note that $safe^\Delta(P)$ can be computed in linear time in the size of Δ , by using inverse hedge accessibility from P . We define the set of states that will no longer change:

$$no-change^\Delta = \{q \mid q \in safe^\Delta(\{q\})\}$$

So $q \in no-change^\Delta$ if and only if $acc^\Delta(\{q\}) \subseteq \{q\}$. This means that all transition rules starting in q must loop in q . In the example automaton from Fig. 4, the self-looping states are those in $no-change^\Delta = \{2, 3, 4, 5\}$.

Lemma 29. *Let $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ be a SHA that is schema-complete for schema \mathbf{S} . For any hedge $h \in \mathbf{S}$ and state $q \in I \cap no-change^\Delta$ it then holds that $q \xrightarrow{h} q$ wrt Δ .*

Proof. The schema-completeness of A for \mathbf{S} applied to $h \in \mathbf{S}$ and $q \in I$ yields the existence of some state $q' \in \mathcal{Q}$ such that $q \xrightarrow{h} q'$ wrt Δ . Let q' be any such state. Note that $q' \in acc^\Delta(q)$. Since $q \in no-change^\Delta$, we have $q \in safe^\Delta(\{q\})$ so that $q' \in acc^\Delta(\{q\})$ implies $q' = q$. This proves $q \xrightarrow{h} q$ wrt Δ . \square

For any state $q \in \mathcal{Q}$ and subset of states $Q \subseteq \mathcal{Q}$ we define:

$$\begin{aligned} s-down^\Delta(q, Q) &= safe^\Delta(\{p \in \mathcal{Q} \mid q @^\Delta p \subseteq Q\}) \\ s-no-change^\Delta(q) &= s-down^\Delta(q, \{q\}) \end{aligned}$$

A state belongs to $s-down^\Delta(q, Q)$ if all states $p \in acc^\Delta(q)$ satisfy $q @^\Delta p \subseteq Q$. So p is a state down that will safely go up to some state in Q . A state p belongs to $s-no-change^\Delta(q)$ if it safely does not change q , i.e., if $\{q\} @^\Delta acc^\Delta(\{p\}) \subseteq \{q\}$.

We next compile the SHA A to a SHA[↓] $A^{snc} = (\Sigma, \mathcal{Q}^{snc}, \Delta^{snc}, I^{snc}, F^{snc})$. For this let Π be a fresh symbol and consider the state set:

$$\mathcal{Q}^{snc} = \{\Pi\} \uplus (\mathcal{Q} \times 2^{\mathcal{Q}})$$

$$\begin{array}{c}
\frac{q \xrightarrow{a} q' \text{ in } \Delta \quad q \notin P \cup \text{no-change}^\Delta}{(q, P) \xrightarrow{a} (q', P) \text{ in } \Delta^{snc}} \quad \frac{a \in \Sigma \quad q \in P \cup \text{no-change}^\Delta}{(q, P) \xrightarrow{a} (q, P) \text{ in } \Delta^{snc}} \\
\frac{\Downarrow q' \text{ in } \Delta \quad q \notin P \cup \text{no-change}^\Delta}{(q, P) \Downarrow (q', s\text{-no-change}^\Delta(q)) \text{ in } \Delta^{snc}} \quad \frac{q \in P \cup \text{no-change}^\Delta}{(q, P) \Downarrow \Pi \text{ in } \Delta^{snc}} \\
\frac{q@p \rightarrow q' \text{ in } \Delta \quad q \notin P \cup \text{no-change}^\Delta}{(q, P)@(p, s\text{-no-change}^\Delta(q)) \rightarrow (q', P) \text{ in } \Delta^{snc}} \quad \frac{q \in P \cup \text{no-change}^\Delta}{(q, P)@\Pi \rightarrow (q, P) \text{ in } \Delta^{snc}} \\
\frac{a \in \Sigma}{\Pi \xrightarrow{a} \Pi \text{ in } \Delta^{snc}} \quad \frac{\text{true}}{\Pi@\Pi \rightarrow \Pi \text{ in } \Delta^{snc}} \quad \frac{\text{true}}{\Pi \Downarrow \Pi \text{ in } \Delta^{snc}}
\end{array}$$

Figure 10. The transition rules of the $\text{SHA}^\downarrow A^{snc}$ inferred from those of the $\text{SHA} A$.

In practice we restrict the state space to the those states that are accessible, or clean the dSHAs keeping only those states that are used in some successful run. But in the worst case, the construction may indeed be exponential. Example 51 can be adapted to show this.

A pair (q, P) means the state q may will safely no more change in the current subhedge if $q \in P \cup \text{no-change}^\Delta$. In this case, the subhedge can be projected. The sets of initial and final states are defined as follows:

$$I^{snc} = I \times \{\emptyset\} \quad F^{snc} = F \times \{\emptyset\}$$

How to generate the transition rules of A^{snc} from those of A is described in Fig. 10. On states assigned on top-level (q, P) , the set P is empty, so that only the states in no-change^Δ are safe for no change. This is why the definition of I^{snc} and F^{snc} use $P = \emptyset$. When going down from some state (q, P) for which q is safe to not change, i.e., $q \in P \cup \text{no-change}^\Delta$, then the following rule is applied:

$$\frac{q \in P \cup \text{no-change}^\Delta}{(q, P) \Downarrow \Pi \text{ in } \Delta^{snc}}$$

The evaluation on the lower level goes to the extra state Π , where it then loops until going back to q on the upper level. When going down from some state (q, P) such that $q \notin P \cup \text{no-change}^\Delta$, then the following rule is applied:

$$\frac{\Downarrow q' \text{ in } \Delta \quad q \notin P \cup \text{no-change}^\Delta}{(q, P) \Downarrow (q', s\text{-no-change}^\Delta(q)) \text{ in } \Delta^{snc}}$$

The states in the set $s\text{-no-change}^\Delta(q)$ on the lower level safely will not make q change on the upper level for any subhedge to come⁵.

When applied to the SHA in Fig. 4 for $[\text{self}::\text{list}/\text{child}::\text{item}]$, the construction yields the SHA^\downarrow in Fig. 11 which is indeed equal to the SHA^\downarrow from Fig. 7 up to state renaming. When run on the hedge $\langle \text{list} \cdot \langle \text{list} \cdot h_1 \rangle \cdot \langle \text{item} \cdot h_2 \rangle \rangle$ as shown in Fig. 8, it does not have to visit the subhedges h_1 nor h_2 , since all of them will be reached starting from the projection state Π .

6.2. Soundness

We next state and prove a soundness result for safe-no-change projection.

⁵ We could detect more irrelevant subhedges by allowing q to change but only in a unique manner. This can be obtained with the alternative definition: $s\text{-no-change}^\Delta(q) = \cup_{r \in \mathcal{Q}} s\text{-down}^\Delta(q, \{r\})$. Computing this set would require quadratic time $O(n |A|)$, while computing $s\text{-no-change}^\Delta(q)$ can be done in linear time $O(|A|)$.

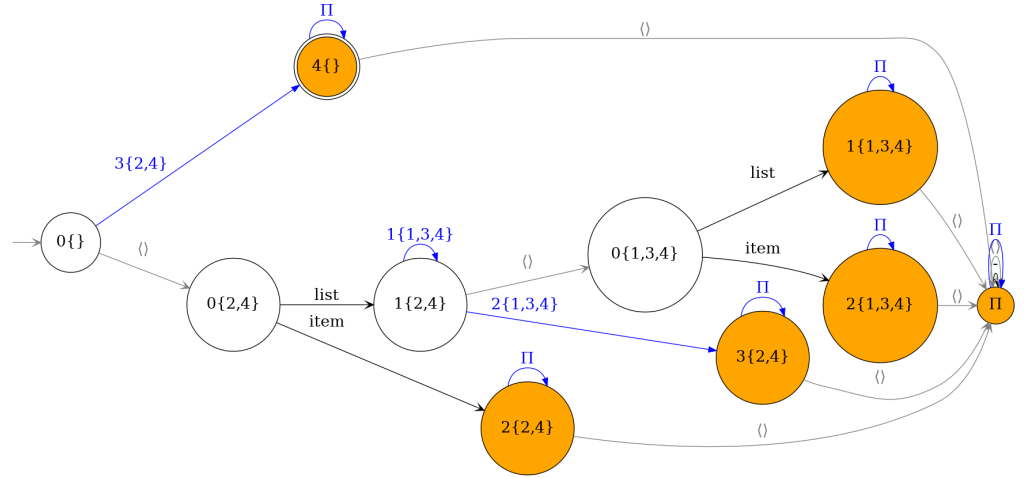


Figure 11. The safe-no-change projection $dSHA^\downarrow A^{snc}$ constructed from the dSHA A for query $[self::list/child::item]$ in Fig. 4. Subhedge projection states are colored in orange. Useless states and transitions leading out of schema $[[doc]]$ are omitted. State Π has an else transition labeled by wildcard letter “-”, which stands for either letter `list` or `item`. This $dSHA^\downarrow$ is equal to the $dSHA^\downarrow$ in Fig. 7 up to the state renaming $0 = (0, \{\})$, $0' = (0, \{2,4\})$, $0'' = (0, \{1,3,4\})$, $1' = (1, \{2,4\})$, $1'' = (1, \{1,3,4\})$, $2' = (2, \{2,4\})$, $2'' = (2, \{1,3,4\})$, $3' = (3, \{2,4\})$, $4 = (4, \{\})$. Recall that $no-change^\Delta = \{2,3,4,5\}$.

Theorem 1 (Soundness of Safe-No-Change Projection). *If a SHA A is schema-complete for some schema S , then safe-no-change projection for A preserves the language within this schema: $\mathcal{L}(A^{snc}) \cap S = \mathcal{L}(A)$.*

Proof. We have to prove that no more changing states $q \in P \cup no-change^\Delta$ is sound. If $q \in no-change^\Delta$ this follows from the schema-completeness of Δ , so that one can neither block on any hedge from the schema nor change the state. In the case $q \in P$, the intuition is that the state on level above - say r - can neither change, since $P = s-no-change^\Delta(r)$, nor can the automaton block on any hedge from the schema due to schema-completeness.

We first prove the inclusion $\mathcal{L}(A) \subseteq \mathcal{L}(A^{snc})$. Since $\mathcal{L}(A) \subseteq S$ by definition of schemas, this implies $\mathcal{L}(A) \subseteq \mathcal{L}(A^{snc}) \cap S$. The proof will be based on the following three Claims 1.1a, 1.2a, and 1.3a. Note that schema-completeness is not needed for this direction.

Claim 1.1a. $\Pi \xrightarrow{h} \Pi$ wrt Δ^{snc} for all hedges $h \in \mathcal{H}_\Sigma$.

The proof is straightforward by induction on the structure of h . It uses the last three transition rules of Δ^{snc} in Fig. 10 permitting to always stay in Π for whatever hedge follows.

Claim 1.2a. For all $h \in \mathcal{H}_\Sigma$, $q \in Q$, and $P \subseteq Q$ such that $q \in P \cup no-change^\Delta$:

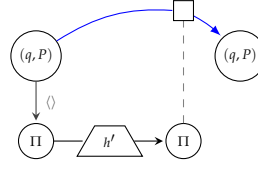
$$(q, P) \xrightarrow{h} (q, P) \text{ wrt } \Delta^{snc}$$

We prove Claim 1.2a by induction on the structure of h .

Case $h = \langle h' \rangle$. In this case, we can use Claim 1.1a to show $\Pi \xrightarrow{h'} \Pi$ wrt Δ^{snc} and the inference rules

$$\frac{q \in P \cup no-change^\Delta}{(q, P) \xrightarrow{\langle \rangle} \Pi \text{ in } \Delta^{snc}} \quad \frac{q \in P \cup no-change^\Delta}{(q, P) @ \Pi \rightarrow (q, P) \text{ in } \Delta^{snc}}$$

in order to close the following diagram with respect to Δ^{snc} :



This proves $(q, P) \xrightarrow{h} (q, P)$ wrt Δ^{snc} as required by the claim.

Case $h = a$. Since $q \in P \cup \text{no-change}^\Delta$ we can apply the inference rule:

$$\frac{a \in \Sigma \quad q \in P \cup \text{no-change}^\Delta}{(q, P) \xrightarrow{a} (q, P) \text{ in } \Delta^{snc}}$$

This proves this case of the claim.

Case $h = \varepsilon$. We trivially have $(q, P) \xrightarrow{\varepsilon} (q, P)$ wrt Δ^{snc} .

Case $h = h' \cdot h''$. By induction hypothesis applied to h' and h'' we have: $(q, P) \xrightarrow{h'} (q, P)$ and $(q, P) \xrightarrow{h''} (q, P)$ wrt Δ^{snc} . Hence $(q, P) \xrightarrow{h' \cdot h''} (q, P)$ wrt Δ^{snc} .

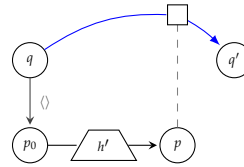
This ends the proof of Claim 1.2a. The next claim, in which the induction step is a little more tedious to prove, is the key of the soundness proof. We define the following predicate for all $q', q'' \in \mathcal{Q}$ and $P \subseteq \mathcal{Q}$:

$$q' \sim_P q'' \text{ iff } (q' = q'' \vee q', q'' \in P)$$

Claim 1.3a. Let $h \in \mathcal{H}_\Sigma$ a hedge, $q, q' \in \mathcal{Q}$ states and $P \subseteq \mathcal{Q}$ a subset of states such that $\text{acc}^\Delta(P) \subseteq P$ and $q \notin P \cup \text{no-change}^\Delta$. If $q \xrightarrow{h} q'$ wrt Δ then there exists q'' such that $(q, P) \xrightarrow{h} (q'', P)$ wrt Δ^{snc} and $q' \sim_P q''$.

Proof. By induction on the structure of h .

Case $h = \langle h' \rangle$. The assumption $q \xrightarrow{h} q'$ wrt Δ shows that there exists states $p_0 \in \langle \rangle^\Delta$ and $p \in \mathcal{Q}$ closing the following diagram:

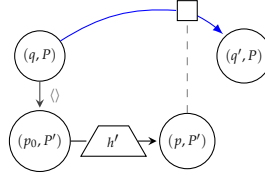


Let $P' = s\text{-no-change}^\Delta(q)$ and note that $\text{acc}^\Delta(P') \subseteq P'$. Since $q \notin P \cup \text{no-change}^\Delta$ we can infer:

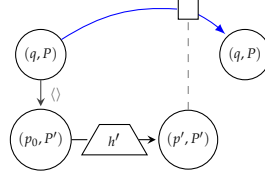
$$\frac{\langle \rangle p_0 \text{ in } \Delta \quad q \notin P \cup \text{no-change}^\Delta}{(q, P) \xrightarrow{\langle \rangle} (p_0, P') \text{ in } \Delta^{snc}} \quad \frac{q@p \rightarrow q' \text{ in } \Delta \quad q \notin P \cup \text{no-change}^\Delta}{(q, P)@(p, P') \rightarrow (q', P) \text{ in } \Delta^{snc}}$$

Subcase $p_0 \notin P' \cup \text{no-change}^\Delta$. The induction hypothesis applies to h' shows that there exists p' such that $(p_0, P') \xrightarrow{h'} (p', P')$ wrt Δ^{snc} and $p \sim_P p'$. We distinguish the two cases justifying the latter predicate:

Subsubcase $p' = p$. Hence $(p_0, P') \xrightarrow{h'} (p, P')$ wrt Δ^{snc} , so we can close the diagram as follows:



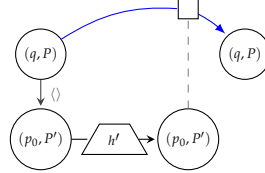
This shows that $(q, P) \xrightarrow{h} (q', P)$ wrt Δ^{snc} , and thus the claim since $q' \sim_P q'$.
Subsubcase $p, p' \in P'$. Since $p' \in P'$ and $P' = s\text{-no-change}^\Delta(q)$ we have $q@p' \rightarrow q$ in Δ . Hence we can close the diagram as follows:



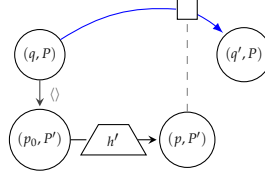
Since $p \in P'$ and $q@p \rightarrow q'$ in Δ we have $q = q'$ by definition of $P' = s\text{-no-change}^\Delta(q)$. This shows that $(q, P) \xrightarrow{h} (q, P)$ wrt Δ^{snc} . Since $q \sim_P q$ the claim follows.

Subcase $p_0 \in P' \cup \text{no-change}^\Delta$. Claim 1.2a then shows that $(p_0, P') \xrightarrow{h'} (p_0, P')$ wrt Δ^{snc} .

Subsubcase $p_0 \in P'$. Since $p \in \text{acc}^\Delta(p_0)$ and $\text{acc}^\Delta(P') \subseteq P'$ it follows that $p \in P'$ too. By definition $P' = s\text{-no-change}^\Delta(q)$ and the completeness of Δ , the memberships $p_0 \in P'$ and $p \in P'$ imply that $q@^\Delta p_0 = \{q\} = q@^\Delta p$. We can now close the diagram below as follows:



Subsubcase $p_0 \in \text{no-change}^\Delta$. In this case $p_0 = p$ so that $q' \in q@^\Delta p_0$. Hence:



Case $h = a$. Since $q \notin P \cup \text{no-change}^\Delta$ we can apply the inference rule:

$$\frac{q \xrightarrow{a} q' \text{ in } \Delta \quad q \notin P \cup \text{no-change}^\Delta}{(q, P) \xrightarrow{a} (q', P) \text{ in } \Delta^{snc}}$$

This shows that $(q, P) \xrightarrow{h} (q', P)$ validating the claim since $q' \sim_P q'$.

Case $h = \varepsilon$. In this case we have $q = q'$ and $(q, P) \xrightarrow{\varepsilon} (q, P)$, so the claim holds.

Case $h = h_1 \cdot h_2$. Since $q \xrightarrow{h} q'$ wrt Δ , there exists $q_1 \in \mathcal{Q}$ such that $q \xrightarrow{h_1} q_1$ wrt Δ and $q_1 \xrightarrow{h_2} q'$ wrt Δ . Since $q \notin P \cup \text{no-change}^\Delta$, we apply the induction hypothesis on h_1 . This implies that there exists q'_1 such that:

$$(q, P) \xrightarrow{h_1} (q'_1, P) \text{ wrt } \Delta^{snc} \text{ and } q_1 \sim_P q'_1$$

We distinguish the two cases of $q_1 \sim_P q'_1$:

Subcase $q_1 = q'_1$. We also distinguish two subcases here:

Subsubcase $q_1 \notin P$. The induction hypothesis applied to h_2 yields:

$$\exists q'' . (q_1, P) \xrightarrow{h_2} (q'', P) \text{ wrt } \Delta^{snc} \wedge q' \sim_P q''$$

Hence

$$\exists q'' . (q, P) \xrightarrow{h} (q'', P) \text{ wrt } \Delta^{snc} \wedge q' \sim_P q''$$

Subsubcase $q_1 \in P$. By Claim 1.2a, we have $(q_1, P) \xrightarrow{h_2} (q_1, P)$. We also have $q' \in \text{acc}(\{q_1\})$ and since we assume $\text{acc}(P) \subseteq P$, this implies $q' \in P$. Hence $(q, P) \xrightarrow{h} (q_1, P)$ and $q', q_1 \in P$ implying the claim with $q' \sim_P q_1$.

Subcase $q_1, q'_1 \in P$. Since $q'_1 \in P$, Claim 1.2a, implies $(q'_1, P) \xrightarrow{h_2} (q'_1, P) \text{ wrt } \Delta^{snc}$. Thus $(q, P) \xrightarrow{h} (q'_1, P) \text{ wrt } \Delta^{snc}$. Since $q' \in \text{acc}^\Delta(\{q_1\})$ and $q_1 \in P$ it follows that $q' \in \text{acc}^\Delta(P) \subseteq P$. Here we used as in the previous subsubcase that $\text{acc}(P) \subseteq P$ is assumed by the claim. Let $q'' = q'_1$. Then we have $(q, P) \xrightarrow{h} (q'', P) \text{ wrt } \Delta^{snc}$ and $q', q'' \in P$ showing the claim.

This ends the proof of Claim 1.3a.

Proof of inclusion $\mathcal{L}(A) \subseteq \mathcal{L}(A^{snc})$. Let $h \in \mathcal{L}(A)$. Then there exists $q_0 \in I$ and $q \in F$ such that $q_0 \xrightarrow{h} q$. We distinguish two cases:

Case $q_0 \in \text{no-change}^\Delta$. By definition of no-change^Δ and since $q \in \text{acc}^\Delta(q_0)$ we have $q_0 = q$. Claim 1.2a shows that $(q_0, \emptyset) \xrightarrow{h} (q_0, \emptyset) \text{ wrt } \Delta^{snc}$ and thus $(q_0, \emptyset) \xrightarrow{h} (q, \emptyset)$ so that $h \in \mathcal{L}(A^{snc})$.

Case $q_0 \notin \text{no-change}^\Delta$. Claim 1.3a with $P = \emptyset$ shows that $(q_0, \emptyset) \xrightarrow{h} (q, \emptyset) \text{ wrt } \Delta^{snc}$ and hence $h \in \mathcal{L}(A^{snc})$.

This ends the proof of the first inclusion. We next want to show the inverse inclusion $\mathcal{L}(A^{snc}) \cap \mathbf{S} \subseteq \mathcal{L}(A)$. It will eventually follow from the following three Claims 1.1b, 1.2b, and 1.3b.

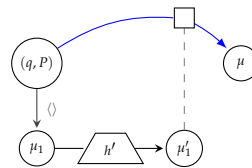
Claim 1.1b. For any hedge h and state $\mu \in \mathcal{Q}^{snc}$, if $\Pi \xrightarrow{h} \mu \text{ wrt } \Delta^{snc}$ then $\mu = \Pi$.

The proof is straightforward by induction on the structure of h : the only transitions rules of Δ^{snc} with Π on the left hand side are inferred by the last three rules in Fig. 10. These require to stay in Π whatever hedge follows.

Claim 1.2b. For any hedge h , set $P \subseteq \mathcal{Q}$, state $q \in P \cup \text{no-change}^\Delta$, and state $\mu \in \mathcal{Q}^{snc}$: if $(q, P) \xrightarrow{h} \mu \text{ wrt } \Delta^{snc}$ then $\mu = (q, P)$.

Proof. By induction on the structure of h . Suppose that $(q, P) \xrightarrow{h} \mu \text{ wrt } \Delta^{snc}$.

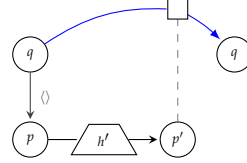
Case $h = \langle h' \rangle$. There must exist states $\mu_1, \mu'_1 \in \mathcal{Q}^{snc}$ closing the following diagram:



Since $q \in P \cup \text{no-change}^\Delta$, the following rule must have been applied to infer $(q, P) \xrightarrow{\diamond} \mu_1 \text{ wrt } \Delta^{snc}$:

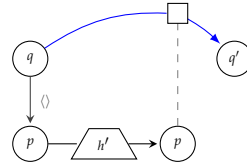
$$\frac{q \in P \cup \text{no-change}^\Delta}{(q, P) \xrightarrow{\diamond} \mu_1 \text{ in } \Delta^{snc}}$$

This shows that $\mu = (q, P)$. Let $q' = q$ so that $\mu = (q', P)$. Since $p \xrightarrow{h'} p'$ wrt Δ we have $p' \in acc^\Delta(\{p\})$ so that $q@p' \rightarrow q$ wrt Δ by definition of $s\text{-no-change}^\Delta(\{q\})$. Hence, we can close the following diagram:



Let $q'' = q$, so that $q' = q''$. It then holds that $q' \sim_P q''$ and $q \xrightarrow{\langle h' \rangle} q'$ wrt Δ , as required by the claim.

Subcase $p \in no\text{-change}^\Delta$. In this case $p \xrightarrow{h'} p'$ wrt Δ implies that $p' = p$. Hence, we can close the following diagram:



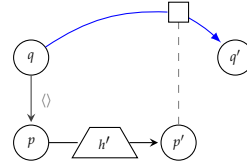
This shows that $q \xrightarrow{\langle h' \rangle} q'$ wrt Δ .

Subcase $p \notin P' \cup no\text{-change}^\Delta$. By induction hypothesis applied to (p, P') $\xrightarrow{h'} \mu'_1$ wrt Δ there exists p'' such that:

$$\mu'_1 = (p', P'), \quad p \xrightarrow{h'} p'' \text{ wrt } \Delta, \text{ and } p' \sim_{P'} p''.$$

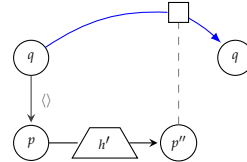
Since Δ does not have blocking runs on h starting with q there exist q'' such that $q@p'' \rightarrow q''$ wrt Δ . There are two ways to satisfy $p' \sim_{P'} p''$:

Subsubcase $p' = p''$. We then have:



This shows $q \xrightarrow{\langle h' \rangle} q'$ wrt Δ .

Subsubcase $p', p'' \in P$. By definition of P' it follows that $q' = q = q''$. Hence:



This shows $q \xrightarrow{\langle h' \rangle} q$ wrt Δ .

Case $h = a$. Since $q \notin P \cup no\text{-change}^\Delta$, the following inference rule must be used:

$$\frac{q \xrightarrow{a} q' \text{ in } \Delta \quad q \notin P \cup no\text{-change}^\Delta}{(q, P) \xrightarrow{a} (q', P) \text{ in } \Delta^{snc}}$$

So $\mu = (q', P)$ and $q \xrightarrow{a} q'$ wrt Δ .

Case $h = \varepsilon$. Obvious.

Case $h = h_1 \cdot h_2$. The judgement $(q, P) \xrightarrow{h} \mu$ wrt Δ^{snc} shows that there exists μ_1 such that $(q, P) \xrightarrow{h_1} \mu_1 \xrightarrow{h_2} \mu$ wrt Δ^{snc} . Since $q \notin P \cup no\text{-change}^\Delta$, we can apply the induction

hypothesis to h_1 . It shows that there exists q'_1 and q''_1 such that $\mu_1 = (q'_1, P)$, $q \xrightarrow{h_1} q'_1$ wrt Δ and $q'_1 \sim_P q''_1$.

Subcase $q'_1 = q''_1$. Hence $(q_1, P) \xrightarrow{h_2} \mu$ wrt Δ^{snc} .

Subsubcase $q'_1 \notin P \cup \text{no-change}^\Delta$. In this case, we can apply the induction hypothesis to $(q'_1, P) \xrightarrow{h_2} \mu$ wrt Δ^{snc} showing the existence of q' such that $\mu = (q', P)$ and a state q'' such that $q'_1 \xrightarrow{h_2} q''$ and $q' \sim_P q''$. Hence $\exists q'' . q \xrightarrow{h} q''$ wrt Δ and $q' \sim_P q''$, so the claim holds.

Subsubcase $q'_1 \in P \cup \text{no-change}^\Delta$. Claim 1.2b applied to $(q'_1, P) \xrightarrow{h_2} \mu$ wrt Δ^{snc} shows that $\mu = (q'_1, P)$.

Subcase $q'_1, q''_1 \in P$. Recall that $(q''_1, P) \xrightarrow{h_2} \mu$ wrt Δ^{snc} and $q''_1 \in P$. Claim 1.2b shows that $\mu = (q''_1, P)$ wrt Δ^{snc} . We also have $q \xrightarrow{h_1} q''_1$ wrt Δ . Since there are no blocking partial runs on h starting from q there exist a state q'' such that $q''_1 \xrightarrow{h_2} q''$ wrt Δ . Since $q''_1 \in P$ and P is closed by accessibility, we have $q'' \in \text{acc}(\{q''_1\}) \subseteq \text{acc}(P) \subseteq P$. From $q \xrightarrow{h_1} q''_1$ wrt Δ , we get $q \xrightarrow{h} q''$ wrt Δ . Since $q''_1, q'' \in P$ it follows that $q''_1 \sim_P q''$ and thus the claim holds.

This ends the proof of Claim 1.3b.

Proof of inclusion $\mathcal{L}(A^{snc}) \cap \mathbf{S} \subseteq \mathcal{L}(A)$. Let $h \in \mathcal{L}(A^{snc}) \cap \mathbf{S}$. Since $h \in \mathcal{L}(A^{snc})$ then there exists $q_0 \in I$ and $q \in F$ such that $(q_0, \emptyset) \xrightarrow{h} (q, \emptyset)$ wrt Δ^{snc} . By Lemma 29 we have that $q_0 \xrightarrow{h} q$ wrt Δ .

We distinguish two cases:

Case $q_0 \in \text{no-change}^\Delta$. Claim 1.2b shows that $q = q_0$. Since A is schema-complete for \mathbf{S} , $h \in \mathbf{S}$, and $q_0 \in I \cap \text{no-change}^\Delta$, Lemma 29 shows that $q_0 \xrightarrow{h} q_0$ wrt Δ . Since $q = q_0$ this yields $h \in \mathcal{L}(A)$.

Case $q_0 \notin \text{no-change}^\Delta$. Since A is schema-complete for \mathbf{S} and $h \in \mathbf{S}$ there exist no blocking runs on h that start in q_0 . Therefore, we can apply Claim 1.3b with $P = \emptyset$ to $(q_0, \emptyset) \xrightarrow{h} (q, \emptyset)$ wrt Δ^{snc} . This shows that $q_0 \xrightarrow{h} q$ wrt Δ and hence $h \in \mathcal{L}(A)$.

This ends the proof of the inverse inclusion, and thus of $\mathcal{L}(A) = \mathcal{L}(A^{snc})$. \square

The projecting in-memory evaluator of A^{snc} will be more efficient than that non-projecting evaluator of A . Note, however, that the size of A^{snc} may be exponentially bigger than that of A . Therefore, for evaluating a dSHA A with subhedge projection on a given hedge h , we may prefer to only the needed part of A^{snc} on the fly. This part has size $O(|h|)$ and can be computed in time $O(|A| |h|)$, so the exponential preprocessing time is avoided.

6.3. Incompleteness

Safe-no-change projection may be incomplete for subhedge projection, so that not all prefixes that are strongly subhedge irrelevant are mapped to subhedge projection states. This is shown by the counter example dSHA A for the XPath filter `[child::item]` in Fig. 12. Its safe-no-change projection A^{snc} is given in Fig. 13. Note that the prefix $u = \langle \text{item} . \langle \text{item} \rangle \rangle$ has the class $\text{class}_{\mathbf{S}}^{\mathcal{L}(A)}(u) = \langle \text{item} . \langle \text{item} . \mathcal{N}_{\Sigma} \rangle \rangle$, which is subhedge irrelevant, so u is strongly subhedge irrelevant for the XPath filter `[child::item]`. Nevertheless it leads to the state $(2, \{1, 3, 5, 6\})$ which is not a subhedge projection state since $2 \notin \{1, 3, 5, 6\}$. The problem is that this state can still be changed to $(4, \{1, 3, 5, 6\})$. This state is somehow equivalent with respect to the filter but not equal to $(2, \{1, 3, 5, 6\})$.

Another incompleteness problem should be mentioned: Safe-no-change projection is sensitive to automata completion as noticed already earlier in Example 15. This is because

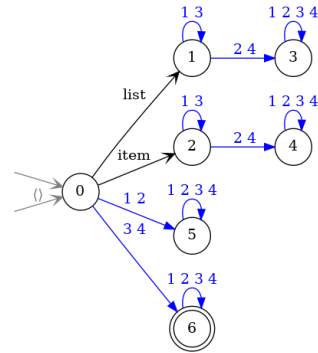


Figure 12. A schema-complete dSHA for the XPath filter $[child::item]$. It is a counter example for the completeness of safe-no-change projection, see Fig. 13.

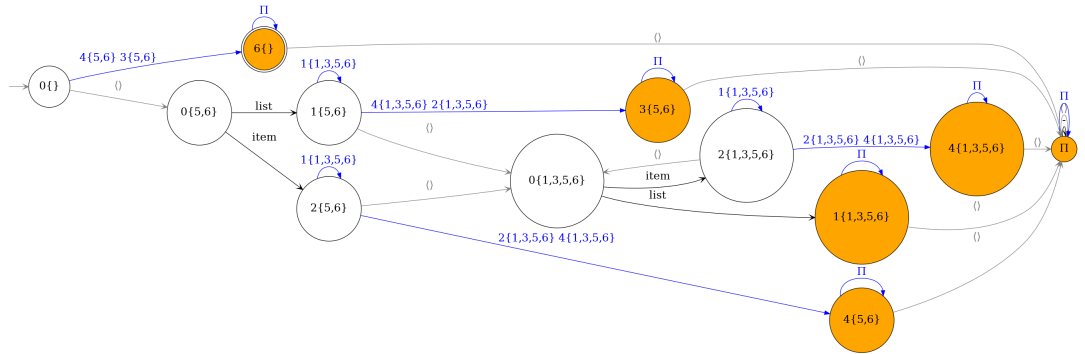


Figure 13. The safe-no-change projection A^{SMC} of the dSHA A in Fig. 12. It is incomplete for subhedge projection with with schema $\llbracket doc \rrbracket$ at the state $(2, \{1, 3, 5, 6\})$: this state is not a subhedge projection state, even though the prefixes $\langle list \cdot \langle item$ and $\langle item \cdot \langle item$ leading to it are subhedge irrelevant.

a state may belong to $no-change^\Delta$ before completion but no more after adding a sink. Nevertheless, such states never change on any tree satisfying the schema. This problem applies, for instance, to example dSHA in Fig. 4 for XPath query $[self::list/child::item]$. Therefore, it was important to assume only schema-completeness for safe-no-change projection and not to impose full completeness.

7. Congruence Projection

We present the congruence projection algorithm for detecting irrelevant subhedges for regular hedge patterns with regular schema restrictions. We prove that congruence projection is complete for subhedge projection, so resolving the incompleteness of safe-no-change projection. For this, congruence projection may no more ignore the schema, so we have to assume that the input schema is regular too.

7.1. Ideas

The starting idea of congruence projection is to resolve the counter example for the completeness of safe-no-change projection in Fig. 13. There, a state is changed when reading an irrelevant subhedge hedge. But the state change moves to a somehow equivalent state, so it is not really relevant. Therefore, the idea is to detect when a state always remains equivalent rather than unchanged when reading an arbitrary subhedge. This is eventually done by the congruence projection in Fig. 14 that we are going to construct.

The obvious question is then: which equivalence relation to choose? Suppose that we want to test whether some hedge satisfying a regular schema S matches a regular pattern L . In the restricted case of words without schema restrictions, the idea could be to use Myhill-Nerode's congruence $diff_{\Sigma^*}^L$ but mapped to states. In the general case, however, the situation becomes more complex, given that in the general case $diff_S^L$ may fail to be an equivalence relation. So it may not be a congruence, as already illustrated in Example 8.

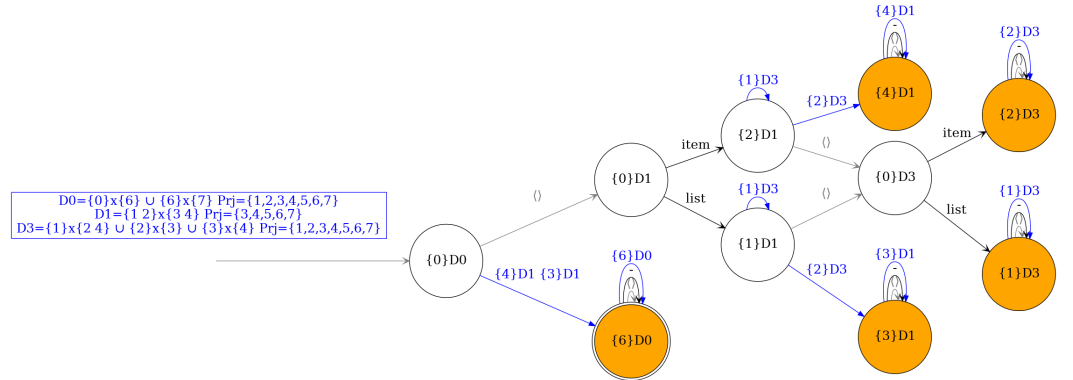


Figure 14. The congruence projection $d\text{SHA}^{\downarrow} A^{cgr}(\llbracket \text{doc} \rrbracket)$ for the counter example of the completeness of safe-no-change projection, i.e., the dSHA A for the filter $[\text{child}::\text{item}]$ in Fig. 12 with the schema final states $F_S = \{0, 5, 6\}$. Note that state $\{1\}D3$ that is reached over the prefix $\langle \text{list} \cdot \text{item} \rangle$ and the state $\{2\}D3$ that is reached over the prefix $\langle \text{item} \cdot \text{item} \rangle$ are subhedger irrelevant and thus subhedger projection states.

Furthermore, the treatment of the nesting of hedges will require to update the considered relation when moving down a hedge.

7.1.1. Difference Relations on States

The congruence projection algorithm will maintain a difference relation on states of the dSHA which at the beginning will be induced by the difference relation on hedges diff_S^L and updated whenever moving down a hedge.

Definition 30. Let $(\Sigma, \mathcal{Q}, \Delta, _ , _)$ be a dSHA. A difference relation for Δ is a symmetric relation on states $D \subseteq \mathcal{Q} \times \mathcal{Q}$ such that for all $h \in \mathcal{H}_\Sigma$:

$$(q \xrightarrow{h} q' \text{ wrt } \Delta \wedge p \xrightarrow{h} p' \text{ wrt } \Delta \wedge (q', p') \in D) \Rightarrow (q, p) \in D$$

The set of all difference relations for Δ is denoted by \mathcal{D}^Δ .

We call a subset $Q \subseteq \mathcal{Q}$ compatible with a difference relation $D \in \mathcal{D}^\Delta$ if $Q^2 \cap D = \emptyset$. This means that no two states of Q may be different with respect to D .

Definition 31. Let $(\Sigma, \mathcal{Q}, \Delta, _ , _)$ be a SHA and D a difference relation for Δ . A subset of states $Q \subseteq \mathcal{Q}$ is called subhedger irrelevant for D wrt Δ if $\text{acc}^\Delta(Q)$ is compatible with D . The set of all subhedger irrelevant subsets of states for D wrt. Δ thus is:

$$d\text{Prj}^\Delta(D) = \{Q \subseteq \mathcal{Q} \mid \text{acc}^\Delta(Q)^2 \cap D = \emptyset\}$$

A state $q \in \mathcal{Q}$ is called subhedger irrelevant for D if the singleton $\{q\}$ is subhedger irrelevant for D . The set of all subhedger irrelevant states of \mathcal{Q} is denoted by $\text{Prj}^\Delta(D)$.

We consider subhedger irrelevance for subsets of states since the congruence projection algorithm has to eliminate the nondeterminism that it introduces by a kind of SHA determinization. Determinization is necessary in order to recognize subhedger irrelevant prefixes properly: all single states in a subset may be subhedger irrelevant while the whole subset is not.

7.1.2. Least Difference Relations

For any binary relation $R \subseteq \mathcal{Q} \times \mathcal{Q}$ let $\text{ldr}^\Delta(R)$ be the least difference relation on states that contains R .

Lemma 32. $(p_1, p_2) \in \text{ldr}^\Delta(R) \Leftrightarrow \exists(q_1, q_2) \in R. \exists h \in \mathcal{H}_\Sigma. p_1 \xrightarrow{h} q_1 \text{ wrt } \Delta \wedge p_2 \xrightarrow{h} q_2 \text{ wrt } \Delta.$

Proof. The set $\{(p_1, p_2) \in \mathcal{Q}^2 \mid \exists(q_1, q_2) \in R. \exists h \in \mathcal{H}_\Sigma. p_1 \xrightarrow{h} q_1 \text{ wrt } \Delta \wedge p_2 \xrightarrow{h} q_2 \text{ wrt } \Delta\}$ clearly is a difference relation that contains R and thus contains the least such difference relation $\text{ldr}^\Delta(R)$. Conversely, each pair in the above set must be contained in any difference relation containing R and thus in $\text{ldr}^\Delta(R)$. \square

Lemma 33. For any $R \subseteq \mathcal{Q} \times \mathcal{Q}$, the difference relation $\text{ldr}^\Delta(R)$ is the value of predicate D in the least fixed point of the ground datalog program generated by the following inference rules:

$$\frac{p, q \in \mathcal{Q}}{D(p, q) :- R(p, q)}. \quad \frac{p_1 \xrightarrow{a} p_2 \text{ wrt } \Delta \quad q_1 \xrightarrow{a} q_2 \text{ wrt } \Delta}{D(p_1, q_1) :- D(p_2, q_2)}.$$

$$\frac{p_1 @ p \rightarrow p_2 \text{ wrt } \Delta \quad q_1 @ q \rightarrow q_2 \text{ wrt } \Delta}{D(p_1, q_1) :- D(p_2, q_2)}. \quad \frac{p, q \in \mathcal{Q}}{D(q, p) :- D(p, q)}.$$

Proof. The first inference rule guarantees that $R \subseteq D$. The three later inference rules characterize difference relations $D \in \mathcal{D}^\Delta$: The second and third rules state that differences in D are propagated backwards over hedges h . This is done recursively by treating letter hedges $h = a$ by the second rule and tree hedges $h = \langle h' \rangle$ by the third rule. The fourth rule expresses the symmetry of difference relations D . So the least fixed point of the datalog program generated by the above rules contains R and is the least relation that satisfies all axioms of difference relations, so it is $\text{ldr}^\Delta(R)$. \square

Proposition 34. For any $R \subseteq \mathcal{Q} \times \mathcal{Q}$ the least difference relation $\text{ldr}^\Delta(R)$ can be computed in time $O(|A|^2)$.

Proof. The size of the ground datalog program computing $\text{ldr}^\Delta(R)$ from Lemma 33 is at most quadratic in the size of A , so its least fixed point can be computed in time $O(|A|^2)$. \square

7.2. Algorithm

We now define the congruence projection algorithm by a compiler from dSHAs and a set of schema final states to SHA^\downarrow s, which when run on a hedge can detect all subhedge irrelevant prefixes.

7.2.1. Inputs and Outputs

As inputs, the congruence projection algorithm is given a dSHA $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ for the regular pattern and a set F_S with $F \subseteq F_S \subseteq \mathcal{Q}$. The dSHA defines the regular language $L = \mathcal{L}(A)$, while the regular schema \mathbf{S} is recognized by the dSHA $A[F/F_S] = (\Sigma, \mathcal{Q}, \Delta, I, F_S)$. So the same base automaton defines the regular language and regular schema by choosing different sets of final states.

Example 35. In the example dSHA in Fig. 4 for the XPath filter `[self::list/child::item]` with schema `[[doc]]`, we have $F = \{4\}$ and $F_S = \{4, 5\}$. In the automaton for the counter example `[child::item]` for safe-no-change projection in Fig. 12, we have $F = \{6\}$ and $F_S = \{5, 6\}$.

We note that if L and \mathbf{S} were given by independent SHAs, then we can obtain a common dSHA A and a set F_S as above by computing the product of the dSHAs for L and \mathbf{S} and completing it. As noticed in [16], it may be more efficient to determinize the SHA for \mathbf{S} in a first step, build the product of the SHA for L with the dSHA for \mathbf{S} in a second, and determinize this product in the third step.

The congruence projection of A wrt. F_S will maintain in its current configuration a subset of states $Q \subseteq \mathcal{Q}$ and a difference relation on states in $D \in \mathcal{D}^\Delta$. Thereby the congruence projection dSHA^\downarrow can always detect whether the current prefix is subhedge

irrelevant for L wrt. schema \mathbf{S} , by testing whether the current set of states Q is subhedge irrelevant for the current difference relation D .

7.2.2. Initial Difference Relation

The initial difference relation on states $D_{init}^{A, F_S} \in \mathcal{D}^\Delta$ is induced from the difference relation on prefixes $diff_S^L$ as follows:

$$D_{init}^{A, F_S} = \{(q', q'') \mid \exists (v', v'') \in diff_S^L. \exists q_0 \in I. q_0 \xrightarrow{hdg(v')} q' \text{ wrt } \Delta \wedge q_0 \xrightarrow{hdg(v'')} q'' \text{ wrt } \Delta\}$$

The next lemma indicates how D_{init}^{A, F_S} can be defined from A and F_S without reference to the languages $L = \mathcal{L}(A)$ and $\mathbf{S} = \mathcal{L}(A[F/F_S])$.

Lemma 36. $D_{init}^{A, F_S} = ldr^\Delta(F \times (F_S \setminus F)) \cap acc^\Delta(I)^2$.

Proof. For the one direction let $(p', p'') \in D_{init}^{A, F_S}$. Then there exist nested words $(v', v'') \in diff_S^L$ and an initial state $q_0 \in I$ such that $q_0 \xrightarrow{hdg(v')} p' \text{ wrt } \Delta$ and $q_0 \xrightarrow{hdg(v'')} p'' \text{ wrt } \Delta$. Since v', v'' are nested words, hedge accessibility $(p', p'') \in acc^\Delta(I)^2$ follows. Furthermore, $(v', v'') \in diff_S^L$ requires the existence of a hedge $h \in \mathcal{H}_\Sigma$ such that $hdg(v') \cdot h \in L$ and $hdg(v'') \cdot h \in \mathbf{S} \setminus L$. Hence there are states $q' \in F$ and $q'' \in F_S \setminus F$ such that $p' \xrightarrow{h} q'$ and $p'' \xrightarrow{h} q''$. By Lemma 32 this implies that $(p', p'') \in ldr^\Delta(F \times (F_S \setminus F))$.

For the other direction let $(p', p'') \in ldr^\Delta(F \times (F_S \setminus F)) \cap acc^\Delta(I)^2$. By Lemma 32, property $(p', p'') \in ldr^\Delta(F \times (F_S \setminus F))$ shows that there exist states $q' \in F$ and $q'' \in F_S \setminus F$ and $h \in \mathcal{H}_\Sigma$ such that $p' \xrightarrow{h} q' \text{ wrt } \Delta$ and $p'' \xrightarrow{h} q'' \text{ wrt } \Delta$. From $(p', p'') \in acc^\Delta(I)^2$ it follows that there are nested words $v', v'' \in \mathcal{N}_\Sigma$ and an initial state $q_0 \in I$ such that $q_0 \xrightarrow{hdg(v')} p' \text{ wrt } \Delta$ and $q_0 \xrightarrow{hdg(v'')} p'' \text{ wrt } \Delta$. Hence $(v', v'') \in diff_S^L$, so that $(q', q'') \in D_{init}^{A, F_S}$. \square

As a consequence of Lemma 36 and Proposition 34, the initial difference relation D_{init}^{A, F_S} can be computed in time $O(|A|^2)$ from A and F_S .

Proposition 37 (Soundness of the initial difference relation). *Let $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ be a complete dSHA, $F \subseteq F_S \subseteq \mathcal{Q}$, $L = \mathcal{L}(A)$ and $\mathbf{S} = \mathcal{L}(A[F/F_S])$. For any hedge $h \in \mathcal{H}_\Sigma$ and state $q \in \mathcal{Q}$ such that $q_0 \xrightarrow{h} q \text{ wrt } \Delta$ for some $q_0 \in I$, the nested word $nw(h)$ is subhedge irrelevant for L and \mathbf{S} if and only if q is subhedge irrelevant for D_{init}^{A, F_S} wrt. Δ .*

Proof. We show that $nw(h)$ is subhedge relevant for L and \mathbf{S} if and only if q is subhedge relevant for D_{init}^{A, F_S} wrt. Δ .

“ \Rightarrow ” Let $nw(h)$ be subhedge relevant for L and \mathbf{S} . Then there exist hedges $h', h'' \in \mathcal{H}_\Sigma$ and a nested word $w \in \mathcal{N}_\Sigma$ such that:

$$nw(h) \cdot nw(h') \cdot w \in L \wedge nw(h) \cdot nw(h'') \cdot w \in \mathbf{S} \setminus L$$

Since A is deterministic and $q_0 \xrightarrow{h} q \text{ wrt } \Delta$ for the unique $q_0 \in I$ it follows that there exist states $p', p'' \in \mathcal{Q}$, $q' \in F$ and $q'' \in F_S \setminus F$ such that:

$$q \xrightarrow{h'} p' \xrightarrow{hdg(w)} q' \text{ wrt } \Delta \wedge q \xrightarrow{h''} p'' \xrightarrow{hdg(w)} q'' \text{ wrt } \Delta$$

From $q' \in F$ and $q'' \in F_S \setminus F$ it follows that $(q', q'') \in D_{init}^{A, F_S}$. Since D_{init}^{A, F_S} is a difference relation, $p' \xrightarrow{hdg(w)} q' \text{ wrt } \Delta$ and $p'' \xrightarrow{hdg(w)} q'' \text{ wrt } \Delta$, this implies that $(p', p'') \in D_{init}^{A, F_S}$ too. Hence, $(p', p'') \in acc^\Delta(\{q\})^2 \cap D_{init}^{A, F_S}$, i.e., q is relevant for D_{init}^{A, F_S} wrt. Δ .

“ \Leftarrow ” Let q be subhedge relevant for D_{init}^{A, F_S} wrt. Δ . Then there exist some pair $(p', p'') \in acc^\Delta(\{q\})^2 \cap D_{init}^{A, F_S}$. So there are hedges h' and h'' such that:

$$q \xrightarrow{h'} p' \text{ wrt } \Delta \wedge q \xrightarrow{h''} p'' \text{ wrt } \Delta$$

Since $D_{init}^{A, F_S} = ldr^\Delta(F \times (F_S \setminus F))$ by Lemma 36, there exist a nested word w and a pair $(q', q'') \in \mathcal{Q}^2$ so that either (q', q'') or (q'', q') belongs to $F \times (F_S \setminus F)$ and:

$$p' \xrightarrow{hdg(w)} q' \text{ wrt } \Delta \wedge p'' \xrightarrow{hdg(w)} q'' \text{ wrt } \Delta$$

Hence $nw(h) \cdot nw(h') \cdot w$ in L and $nw(h) \cdot nw(h'') \cdot w \in \mathbf{S} \setminus L$ or vice versa. This shows that $nw(h)$ is relevant for L and \mathbf{S} .

□

7.2.3. Updating the Difference Relation

For any difference relation $D \subseteq \mathcal{D}^\Delta$ and subset of states $Q \subseteq \mathcal{Q}$, we define a binary relation $down_Q^\Delta(D) \subseteq \mathcal{Q} \times \mathcal{Q}$ such that for all states $p_1, p_2 \in \mathcal{Q}$:

$$(p_1, p_2) \in down_Q^\Delta(D) \text{ iff } \exists q_1, q_2 \in Q. (q_1 @^\Delta p_1, q_2 @^\Delta p_2) \in D$$

For any subset of states $Q \subseteq \mathcal{Q}$ and difference relation $D \in \mathcal{D}^\Delta$ let $D_Q \in \mathcal{D}^\Delta$ be the least difference relation that contains $down_Q^\Delta(D)$:

$$D_Q = ldr^\Delta(down_Q^\Delta(D))$$

Lemma 38. *The difference relation D_Q can be computed in time $O(|A|^2)$ from Q, Δ , and D .*

Proof. The binary relation $down_Q^\Delta(D)$ can be computed in time $O(|\Delta|^2)$ from Q, D , and Δ . And the least difference relation $ldr^\Delta(down_Q^\Delta(D))$ can be computed by ground datalog in time $O(|A|^2)$ from $down_Q^\Delta(D)$ by Proposition 34. □

Example 39. *Reconsider the dSHA for the XPath filter $[self::list/child::item]$ in Fig. 4 with the set of schema-final states $F_S = \{4, 5\}$. Since $F = \{4\}$, we have $F_S \setminus F = \{5\}$. The initial difference relation is $D_0 = D_{init}^{A, F_S}$ is the symmetric closure of $(\{0\} \times \{1, 4, 5\}) \cup \{(1, 4), (4, 5)\}$. The subhedge irrelevant states are $Prj^\Delta(D_0) = \{1, 2, 3, 4, 5\}$ since only state 0 can access two states in the difference relation D_0 , the final state in $F = \{4\}$ and a nonfinal state that is schema-final in $F_S \setminus F = \{5\}$ of some hedges. The difference relation $down_{\{0\}}^\Delta(D_{init}^{A, F_S})$ is the symmetric closure of $\{1, 2\} \times \{3\}$ which is equal to the difference relation $D_1 = D_{0\{0\}} = ldr^\Delta(down_{\{0\}}^\Delta(D_0))$. Hence, the projection states are $Prj^\Delta(D_1) = \{2, 3, 4, 5\}$ since from states 0 and 1 one can still reach both 1 and 3 while $(1, 3) \in D_1$. The difference relation $D_2 = down_{\{1\}}^\Delta(D_1)$ is the symmetric closure of $\{(1, 2), (2, 3)\}$. Hence, the projection states are $Prj^\Delta(D_2) = \{1, 2, 3, 4, 5\}$. From state 1, in particular one can only reach the states 1 and 3 which are not different for D_2 .*

Projection states for the initial difference relation contain all states that are safe for selection or safe for rejection with respect to the schema:

Lemma 40. *All states in $safe^\Delta(F)$ and $safe^\Delta(F_S \setminus F)$ are subhedge irrelevant for D_{init}^{A, F_S} .*

We omit the proof since the result is not used later on.

$$\begin{array}{c}
\frac{a \in \Sigma \quad Q \in dPrj^\Delta(D)}{(Q, D) \xrightarrow{a} (Q, D) \text{ in } \Delta^{cgr}} \quad \frac{Q \in dPrj^\Delta(D)}{(Q, D) \xrightarrow{\langle \rangle} (Q, D) \text{ in } \Delta^{cgr}} \\
\\
\frac{Q@P \rightarrow Q' \text{ in } \Delta^{det} \quad Q \in dPrj^\Delta(D)}{(Q, D)@(Q, D) \rightarrow (Q, D) \text{ in } \Delta^{cgr}} \\
\\
\frac{Q \xrightarrow{a} Q' \text{ in } \Delta^{det} \quad Q \notin dPrj^\Delta(D)}{(Q, D) \xrightarrow{a} (Q', D) \text{ in } \Delta^{cgr}} \quad \frac{Q \notin dPrj^\Delta(D)}{(Q, D) \xrightarrow{\langle \rangle} (\langle \rangle^\Delta, D_Q) \text{ in } \Delta^{cgr}} \\
\\
\frac{Q@acc^\Delta(P) \rightarrow Q' \text{ in } \Delta^{det} \quad Q \notin dPrj^\Delta(D) \quad P \in dPrj^\Delta(D_Q)}{(Q, D)@(P, D_Q) \rightarrow (Q', D) \text{ in } \Delta^{cgr}} \\
\\
\frac{Q@P \rightarrow Q' \text{ in } \Delta^{det} \quad Q \notin dPrj^\Delta(D) \quad P \notin dPrj^\Delta(D_Q)}{(Q, D)@(P, D_Q) \rightarrow (Q', D) \text{ in } \Delta^{cgr}}
\end{array}$$

Figure 15. The transitions rules Δ^{cgr} of the congruence projection $A^{cgr}(\mathbf{S})$.

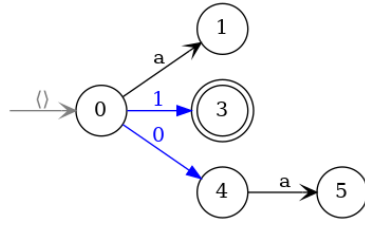


Figure 16. A dSHA A for $\mathcal{L}(A) = \{\langle a \rangle\}$ and schema-final states $F_S = \{3, 5\}$ for the schema $\mathbf{S} = \mathcal{L}(A) \cup \{\langle \rangle \cdot a\}$.

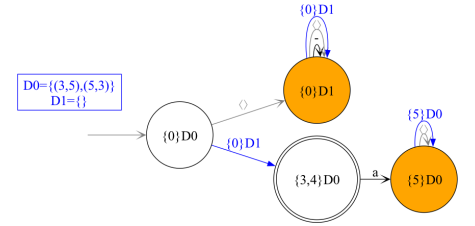


Figure 17. The congruence projection $A^{cgr}(\mathbf{S})$ for the dSHA A and the schema-final states F_S in Fig. 16.

Given a dSHA $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ and a set $F \subseteq F_S \subseteq \mathcal{Q}$, we now construct the congruence projection $A^{cgr}(\mathbf{S})$ as the following dSHA † :

$$A^{cgr}(\mathbf{S}) = (\Sigma, \mathcal{Q}^{cgr}, \Delta^{cgr}, I^{cgr}(\mathbf{S}), F^{cgr}(\mathbf{S}))$$

where the set of states, initial states, and final states are:

$$\begin{aligned}
\mathcal{Q}^{cgr} &\subseteq 2^{\mathcal{Q}} \times \mathcal{D}^\Delta \\
I^{cgr}(\mathbf{S}) &= \{(I, D_{init}^{A, F_S})\} \\
F^{cgr}(\mathbf{S}) &= \left\{ (Q, D_{init}^{A, F_S}) \mid \left(\begin{array}{l} Q \notin dPrj^\Delta(D_{init}^{A, F_S}) \Rightarrow Q \cap F \neq \emptyset \quad \wedge \\ Q \in dPrj^\Delta(D_{init}^{A, F_S}) \Rightarrow acc^\Delta(Q) \cap F \neq \emptyset \end{array} \right) \right\}
\end{aligned}$$

The transition rules in Δ^{cgr} are given by the inference rules in Fig. 15. For illustrating the construction and why determinization is needed, we reconsider Example 4, where schema restrictions have consequences that at first sight may be counter intuitive.

Example 41 (Determinization during congruence projection is needed). *Reconsider Example 4 with signature $\Sigma = \{a\}$, pattern $L = \{\langle a \rangle\}$ and schema $\mathbf{S} = L \cup \{\langle \rangle \cdot a\}$. This language can be defined by the dSHA A in Fig. 16. The schema \mathbf{S} can be defined by the same automaton but with the schema-final states $F_S = \{3, 5\}$ and $F = \{3\}$. The dSHA $A^{cgr}(\mathbf{S})$ produced by congruence projection is shown in Fig. 17. The unique hedge $\langle a \rangle$ of the language $L = \mathcal{L}(A)$ is accepted in state $(\{3, 4\}, D0)$, where $D0 = \{(3, 5), (5, 3)\}$. The unique hedge $\langle \rangle \cdot a$ in $\mathbf{S} \setminus L$ is rejected: after reading the tree $\langle \rangle$, the automaton goes to state $(\{3, 4\}, D0)$ where it goes to a projecting sink*

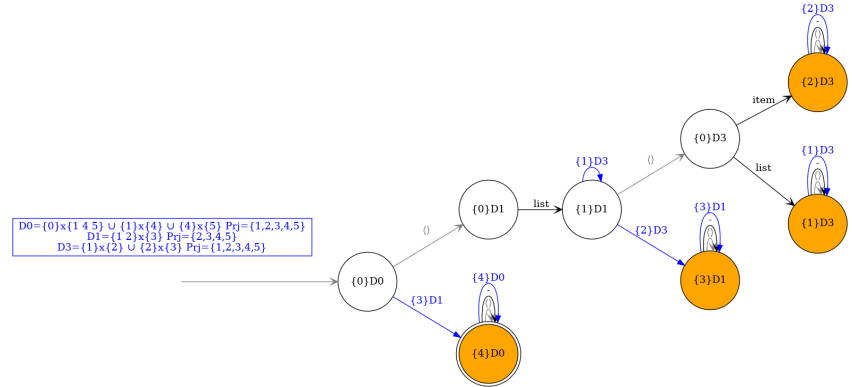


Figure 18. The congruence projection $A^{cgr}(\llbracket doc \rrbracket)$ of the dSHA A in Fig. 4 for the XPath filter $[self::list/child::item]$ with schema $\llbracket doc \rrbracket$. Subhedger irrelevant states are colored in orange. The underscore stands for any label, either `list` or `item`.

$(\{5\}, D0)$ when reading the subsequent letter a . We note that any hedge in $\langle \Sigma^* \rangle$ is accepted in state $(\{3, 4\}, D0)$ by $A^{cgr(S)}$, even though only the single hedge $\langle a \rangle$ belongs to L . This is sound given that the hedges in $\langle (\varepsilon + a \cdot a \cdot a^*) \rangle$ do not belong to schema S anyway. We notice that $(\{3, 4\}, D0)$ cannot be a projection state, since the run on hedge $\langle \rangle \cdot a$ must continue to the sink $(\{5\}, D0)$, so state $(\{3, 4\}, D0)$ is subhedger relevant.

Note also that both individual states 3 and 4 are subhedger irrelevant for $D0$ while the subset with both states $\{3, 4\}$ is subhedger relevant for $D0$, since $(3, 5) \in acc^\Delta(\{3, 4\})^2 \cap D0$. This shows that the determinizing construction is indeed needed to decide subhedger irrelevance for cases such as in this example. Also notice that state 0 is subhedger irrelevant for $D1 = \emptyset$. This is fine, since the acceptance of hedges of the form $\langle h \rangle \cdot h'$ by $A^{cgr(S)}$ does indeed not depend on h . In contrast, it depends on h' , so that the subset of states $\{3, 4\}$ cannot be subhedger irrelevant for $D0$.

Finally, let us discuss how the transition rules of $dSHA^\downarrow A^{cgr(S)}$ are inferred. The most interesting transition rule is $(\{0\}, D0) @ (\{0\}, D1) \rightarrow (\{3, 4\}, D0)$ in Δ^{cgr} . It is produced by the following inference rule where $Q = P = \{0\}$, $acc^\Delta(P) = \{0, 1, 3, 4, 5\}$, $Q' = \{3, 4\}$, $D = D0$, and $D_Q = D1$:

$$\frac{Q @ acc^\Delta(P) \rightarrow Q' \text{ in } \Delta^{det} \quad Q \notin dPrj^\Delta(D) \quad P \in dPrj^\Delta(D_Q)}{(Q, D) @ (P, D_Q) \rightarrow (Q', D) \text{ in } \Delta^{cgr}}$$

This rule states that if P is subhedger irrelevant for D_Q then all states accessible from P must be tried out, since all of them could be reached by some different subhedger that got projected away. So one cannot know due to subhedger projection, into which state of Q' one should go. In order to stay deterministic, we thus go into all possible states, i.e., into the subset Q' . In the example, these are all the states that can be reached when reading some hedge in the pattern $\{\langle h \rangle \mid h \in \mathcal{H}_\Sigma\}$, given that the subhedges $h \in \Sigma$ are not inspected with subhedger projection.

Example 42 (XPath filter $[self::list/child::item]$ with schema $\llbracket doc \rrbracket$). The congruence projection of the example dSHA in Fig. 4 is given in Fig. 18. This dSHA is similar to the safe-no-change projection in Fig. 7, except that the useless state $2'$ is removed and that the four projection states are now looping in themselves, rather than going to a shared looping state Π . It should also be noticed that only singleton state sets are used there, so no determinization is needed there. As we will see, this is typical for experiments on regular XPath queries where larger subsets do rarely occur.

Example 43 (Counter example for completeness of safe-no-change projection). Reconsider the counter example for the completeness of safe-no-change projection, i.e., the dSHA in Fig. 12 for the XPath query $[child::item]$ with schema final states $F_S = \{0, 5, 6\}$. Its congruence projection is shown in Fig. 14. We note that the prefix $\langle item \rangle$ leads there to the state $\{2\}D3$, which

is a subhedge projection state since 2 is subhedge irrelevant for $D3$. In particular, note that $\text{acc}^\Delta(\{2\}) = \{2, 4\}$, so no two states accessible from 2 are different in $D3$. This means that the state 2 may still be changed to 4 but then it does not become different with respect to $D3$. This resolves the incompleteness issue with the safe-no-change projection on this example.

The first property for states (Q, D) assigned by the congruence projection is that Q is compatible with D . This means that no two states in Q are different with respect to D . Intuitively, each state of Q is as good as each other except for leading out of the schema. So if (Q, D) is assigned to some prefix, and some suffix leads from some state in Q to F , then it cannot lead to $F_S \setminus F$ from some other state in Q .

Lemma 44. *If some partial run of $A^{\text{cgr}(S)}$ assigns a state (Q, D) to some prefix then Q is compatible with D .*

We omit the proof since this instructive result will not be used directly later on. Still compatibility will play an important role in the soundness proof.

7.3. Soundness

We next adapt the soundness result and proof from safe-no-change projection to congruence projection.

Theorem 2 (Soundness of Congruence Projection). *For any dSHA $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ and set $F \subseteq F_S \subseteq \mathcal{Q}$ the congruence projection of A with respect to F_S preserves the language of A within schema $\mathbf{S} = \mathcal{L}(A[F/F_S])$, i.e.:*

$$\mathcal{L}(A^{\text{cgr}(S)}) \cap \mathbf{S} = \mathcal{L}(A)$$

We note that \mathbf{S} is a schema for A since $F \subseteq F_S$. Furthermore, A is schema-complete for \mathbf{S} since A is deterministic and $\mathbf{S} = \mathcal{L}(A[F/F_S])$.

Proof. The proof of the inclusion $\mathcal{L}(A) \subseteq \mathcal{L}(A^{\text{cgr}(S)}) \cap \mathbf{S}$ will be based on the following two claims:

Claim 2.1a. For all $h \in \mathcal{H}_\Sigma$, $D \in \mathcal{D}^\Delta$, and $Q \in dPrj^\Delta(D)$: $(Q, D) \xrightarrow{h} (Q, D)$ wrt Δ^{cgr} .

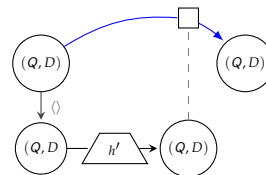
We prove Claim 2.1a by induction on the structure of h . Suppose that $Q \in dPrj^\Delta(D)$.

Case $h = \langle h' \rangle$. The induction hypothesis applied to h' yields $(Q, D) \xrightarrow{h'} (Q, D)$ wrt Δ^{cgr} .

We can thus apply the following two inference rules:

$$\frac{Q \in dPrj^\Delta(D)}{(Q, D) \xrightarrow{\langle \rangle} (Q, D) \text{ in } \Delta^{\text{cgr}}} \quad \frac{Q \in dPrj^\Delta(D)}{(Q, D) @ (Q, D) \rightarrow (Q, D) \text{ in } \Delta^{\text{cgr}}}$$

in order to close the following diagram with respect to Δ^{cgr} :



This proves $(Q, D) \xrightarrow{h} (Q, D)$ wrt Δ^{cgr} as required by the claim.

Case $h = a$. Since $q \in dPrj^\Delta(D)$ we can apply the inference rule:

$$\frac{a \in \Sigma \quad Q \in dPrj^\Delta(D)}{(Q, D) \xrightarrow{a} (Q, D) \text{ in } \Delta^{\text{cgr}}}$$

This proves this case of the claim.

Case $h = \varepsilon$. We trivially have $(Q, D) \xrightarrow{\varepsilon} (Q, D)$ wrt Δ^{cgr} .

Case $h = h' \cdot h''$. By induction hypothesis applied to h' and h'' we have: $(Q, D) \xrightarrow{h'} (Q, D)$ and $(Q, D) \xrightarrow{h''} (Q, D)$ wrt Δ^{cgr} . Hence $(Q, D) \xrightarrow{h' \cdot h''} (Q, D)$ wrt Δ^{cgr} .

This ends the proof of Claim 2.1a. The next claim is the key of the soundness proof.

For any difference relation D we define a binary relation $dep^a(D) \subseteq 2^{\mathcal{Q}} \times 2^{\mathcal{Q}}$ such that for any two subsets of states $Q', Q'' \subseteq \mathcal{Q}$:

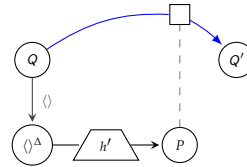
$$(Q', Q'') \in dep^a(D) \Leftrightarrow \begin{cases} Q' \notin dPrj^\Delta(D) \Rightarrow Q' \subseteq Q'' & \wedge & (1^a) \\ Q' \in dPrj^\Delta(D) \Rightarrow (Q' \subseteq acc^\Delta(Q'') \wedge Q'' \in dPrj^\Delta(D)) & (2^a) \end{cases}$$

Furthermore, note that $Q'' \in dPrj^\Delta(D) \wedge Q' \subseteq acc^\Delta(Q'')$ implies $Q' \in dPrj^\Delta(D)$ and thus $(Q', Q'') \in dep^a(D)$.

Claim 2.2a. Let $h \in \mathcal{H}_\Sigma$ a hedge, $Q, Q' \subseteq \mathcal{Q}$ subsets of states, and $D \in \mathcal{D}^\Delta$ a difference relation. If $Q \xrightarrow{h} Q'$ wrt Δ^{det} then there exists $Q'' \subseteq \mathcal{Q}$ such that $(Q, D) \xrightarrow{h} (Q'', D)$ wrt Δ^{cgr} and $(Q', Q'') \in dep^a(D)$.

Proof. If $Q \in dPrj^\Delta(D)$ then $(Q, D) \xrightarrow{h} (Q, D)$ wrt Δ^{cgr} by Claim 2.1a. Let $Q'' = Q$. We then have $Q' \subseteq acc^\Delta(Q'')$ and $Q'' \in dPrj^\Delta(D)$ and thus $Q' \in dPrj^\Delta(D)$, so that (1^a) and (2^a) of $(Q', Q'') \in dep^a(D)$. So it is sufficient to prove the claim under the assumption that $Q \notin dPrj^\Delta(D)$. The proof is by induction on the structure of h .

Case $h = \langle h' \rangle$. The assumption $Q \xrightarrow{h} Q'$ wrt Δ^{det} shows that there exists a subset of states $P \subseteq \mathcal{Q}$ closing the following diagram:



In particular, we have $Q@P \rightarrow Q'$ in Δ^{det} . Let $D' = D_Q$. Since $Q \notin dPrj^\Delta(D)$ we can infer:

$$\frac{Q \notin dPrj^\Delta(D)}{(Q, D) \xrightarrow{\langle \rangle^\Delta} (\langle \rangle^\Delta, D') \text{ in } \Delta^{cgr}}$$

We have to distinguish two cases depending on whether $\langle \rangle^\Delta$ belongs to $dPrj^\Delta(D')$ or not.

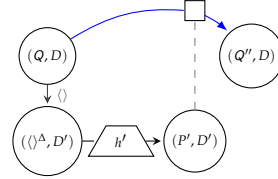
Subcase $\langle \rangle^\Delta \notin dPrj^\Delta(D')$. The induction hypothesis applied to h' yields the existence of $P' \subseteq \mathcal{Q}$ such that $(P, P') \in dep^a(D')$ and:

$$(\langle \rangle^\Delta, D') \xrightarrow{h'} (P', D') \text{ wrt } \Delta^{cgr}$$

Subsubcase $P \notin dPrj^\Delta(D')$. Since $(P, P') \in dep^a(D')$ it follows that $P \subseteq P'$. Hence, $P' \notin dPrj^\Delta(D')$. Let $Q'' \subseteq \mathcal{Q}$ be such that $Q@P' \rightarrow Q''$ in Δ^{det} . We can then apply the following inference rule:

$$\frac{Q@P' \rightarrow Q'' \text{ in } \Delta^{det} \quad Q \notin dPrj^\Delta(D) \quad P' \notin dPrj^\Delta(D')}{(Q, D)@(P', D') \rightarrow (Q'', D) \text{ in } \Delta^{cgr}}$$

Hence we can close the diagram as follows:

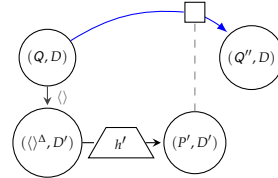


This shows that $(Q, D) \xrightarrow{h} (Q'', D)$ wrt Δ^{cgr} . Since $P \subseteq P'$, $Q@P \rightarrow Q'$ and $Q@P' \rightarrow Q''$ in Δ^{det} it follows that $Q' \subseteq Q''$ and thus $(Q', Q'') \in dep^a(D)$. This shows the claim in this case.

Subsubcase $P \in dPrj^\Delta(D')$. Since $(P, P') \in dep^a(D')$ it follows that $P \subseteq acc^\Delta(P')$ and $P' \in dPrj^\Delta(D')$. Hence, we can apply the following inference rule for some $Q'' \subseteq \mathcal{H}_\Sigma$:

$$\frac{Q@acc^\Delta(P') \rightarrow Q'' \text{ in } \Delta^{det} \quad Q \notin dPrj^\Delta(D) \quad P' \in dPrj^\Delta(D')}{(Q, D)@(P', D') \rightarrow (Q'', D) \text{ in } \Delta^{cgr}}$$

Hence we can close the diagram as follows:

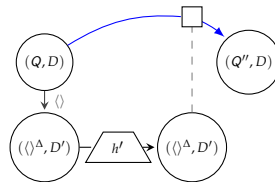


This shows that $(Q, D) \xrightarrow{h} (Q'', D)$ wrt Δ^{cgr} . Since $P \subseteq acc^\Delta(P')$, it follows from $Q@P \rightarrow Q'$ and $Q@acc^\Delta(P') \rightarrow Q''$ in Δ^{det} , that $Q' \subseteq Q''$. Thus $(Q', Q'') \in dep^a(D)$, so the claim holds in this case too.

Subcase $\langle \rangle^\Delta \in Prj^\Delta(D')$. Claim 2.1a shows that $(\langle \rangle^\Delta, D') \xrightarrow{h'} (\langle \rangle^\Delta, D')$ wrt Δ^{cgr} . Let Q'' be such that $Q@acc^\Delta(\langle \rangle^\Delta) \rightarrow Q''$ in Δ^{det} . We can apply the following inference rule:

$$\frac{Q@acc^\Delta(\langle \rangle^\Delta) \rightarrow Q'' \text{ in } \Delta^{det} \quad Q \notin dPrj^\Delta(D) \quad \langle \rangle^\Delta \in Prj^\Delta(D')}{(Q, D)@(\langle \rangle^\Delta, D') \rightarrow (Q'', D) \text{ in } \Delta^{cgr}}$$

We can thus close the diagram below as follows with respect to Δ^{cgr} :



This shows that $(Q, D) \xrightarrow{h} (Q'', D)$ wrt Δ^{cgr} . Since $P \subseteq acc^\Delta(\langle \rangle^\Delta)$, it follows from $Q@P \rightarrow Q'$ and $Q@acc^\Delta(\langle \rangle^\Delta) \rightarrow Q''$ in Δ^{det} that $Q' \subseteq Q''$ and thus $(Q', Q'') \in dep^a(D)$.

Case $h = a$. Since $Q \notin dPrj^\Delta(D)$ we can apply the inference rule:

$$\frac{Q \xrightarrow{a} Q' \text{ in } \Delta^{det} \quad Q \notin dPrj^\Delta(D)}{(Q, D) \xrightarrow{a} (Q', D) \text{ in } \Delta^{cgr}}$$

Hence $(Q, D) \xrightarrow{h} (Q', D)$ wrt Δ^{cgr} . With $Q'' = Q'$, the claim follows since $(Q', Q'') \in dep^a(D)$.

Case $h = \varepsilon$. In this case we have $Q = Q''$ and $(Q, D) \xrightarrow{\varepsilon} (Q, D)$ wrt Δ^{cgr} , so the claim holds.

Case $h = h_1 \cdot h_2$. Since $Q \xrightarrow{h} Q'$ wrt Δ^{det} there exists $Q_1 \subseteq Q$ such that $Q \xrightarrow{h_1} Q_1$ and $Q_1 \xrightarrow{h_2} Q'$ wrt Δ^{det} . By induction hypothesis applied to h_1 there exists $Q'_1 \subseteq Q$ such that $(Q, D) \xrightarrow{h_1} (Q'_1, D)$ wrt Δ^{cgr} and $(Q_1, Q'_1) \in dep^a(D)$.

Subcase $Q_1 \in dPrj^\Delta(D)$. Since $(Q_1, Q'_1) \in dep^a(D)$ it follows that $Q_1 \subseteq acc^\Delta(Q'_1)$ and $Q'_1 \in dPrj^\Delta(D)$. Furthermore, Claim 2.1a shows that $(Q'_1, D) \xrightarrow{h_2} (Q'_1, D)$ and hence $(Q, D) \xrightarrow{h} (Q'_1, D)$ wrt Δ^{cgr} . Since $Q' \subseteq acc^\Delta(Q_1)$ and $Q_1 \subseteq acc^\Delta(Q'_1)$ it follows that $Q' \subseteq acc^\Delta(Q'_1)$. Hence, $(Q'_1, Q') \in dep^a(D)$ and $(Q, D) \xrightarrow{h} (Q'_1, D)$.

Subcase $Q_1 \notin dPrj^\Delta(D)$. In this case, we can apply the induction hypothesis to $Q_1 \xrightarrow{h_2} Q'$ wrt Δ^{det} showing that there exists $Q'' \subseteq Q$ such that $(Q'_1, D) \xrightarrow{h_2} (Q'', D)$ wrt Δ^{cgr} and $(Q', Q'') \in dep^a(D)$. Hence, $(Q, D) \xrightarrow{h} (Q'', D)$ wrt Δ^{cgr} and $(Q', Q'') \in dep^a(D)$.

This ends the proof of Claim 2.2a.

Proof of inclusion $\mathcal{L}(A) \subseteq \mathcal{L}(A^{cgr(\mathbf{S})}) \cap \mathbf{S}$. Since \mathbf{S} is a schema for A , we have $\mathcal{L}(A) \subseteq \mathbf{S}$ so that it is sufficient to show $\mathcal{L}(A) \subseteq \mathcal{L}(A^{cgr(\mathbf{S})})$. Let $h \in \mathcal{L}(A)$. Then there exist $q_0 \in I$ and $q \in F$ such that $q_0 \xrightarrow{h} q$ wrt Δ . Let $Q = \{q\}$. Since A is deterministic, it follows that $I \xrightarrow{h} Q$ wrt Δ^{det} . By Claim 2.2a this implies the existence of a subset of states $Q' \in \mathcal{Q}$ such that $(Q, Q') \in dep^a(D)$ and $(I, D_{init}^{A, Fs}) \xrightarrow{h} (Q', D_{init}^{A, Fs})$ wrt Δ^{cgr} . Furthermore, $(I, D_{init}^{A, Fs}) \in I^{cgr(\mathbf{S})}$. In order to prove $h \in \mathcal{L}(A^{cgr(\mathbf{S})})$ it is thus sufficient to show that $(Q', D_{init}^{A, Fs}) \in F^{cgr(\mathbf{S})}$. We distinguish two cases:

Case $Q' \notin dPrj^\Delta(D_{init}^{A, Fs})$. Condition (1^a) of $(Q, Q') \in dep^a(D)$, shows that $Q \subseteq Q'$ so that $q \in Q'$. Since also $q \in F$, it follows that $Q' \cap F \neq \emptyset$. Thus, $(Q', D_{init}^{A, Fs}) \in F^{cgr(\mathbf{S})}$, so that $h \in \mathcal{L}(A^{cgr(\mathbf{S})})$.

Case $Q' \in dPrj^\Delta(D_{init}^{A, Fs})$. Condition (2^a) of $(Q, Q') \in dep^a(D)$ yields $Q \subseteq acc^\Delta(Q')$. Hence, $q \in acc^\Delta(Q') \cap F$, so that $(Q', D_{init}^{A, Fs}) \in F^{cgr(\mathbf{S})}$, and thus $h \in \mathcal{L}(A^{cgr(\mathbf{S})})$.

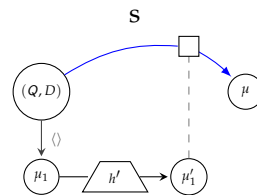
This ends the proof of the first inclusion.

We next want to show the inverse inclusion $\mathcal{L}(A^{cgr(\mathbf{S})}) \cap \mathbf{S} \subseteq \mathcal{L}(A)$. It will eventually follow from the next two claims.

Claim 2.1b. For any hedge $h \in \mathcal{H}_\Sigma$, difference relation $D \subseteq \mathcal{D}^\Delta$, projection state $Q \in dPrj^\Delta(D)$ and state $\mu \in \mathcal{Q}^{cgr}$: if $(Q, D) \xrightarrow{h} \mu$ wrt Δ^{cgr} then $\mu = (Q, D)$.

Proof. By induction on the structure of $h \in \mathcal{H}_\Sigma$. Suppose that $(Q, D) \xrightarrow{h} \mu$ wrt Δ^{cgr} .

Case $h = \langle h' \rangle$. There must exist states $\mu_1, \mu'_1 \in \mathcal{Q}^{cgr}$ closing the following diagram:



Since $Q \in dPrj^\Delta(D)$, the following rule must have been applied to infer $(Q, D) \xrightarrow{\langle h' \rangle} \mu_1$ wrt Δ^{cgr} :

$$\frac{Q \in dPrj^\Delta(D)}{(Q, D) \xrightarrow{\langle h' \rangle} (Q, D) \text{ in } \Delta^{cgr}}$$

Therefore $\mu_1 = (Q, D)$. The induction hypothesis applied to $(Q, D) \xrightarrow{h'} \mu'_1$ wrt Δ^{cgr} shows that $\mu'_1 = (Q, D)$ too. So μ must have been inferred by applying the rule:

$$\frac{Q \in dPrj^\Delta(D)}{(Q, D)@(Q, D) \rightarrow (Q, D) \text{ in } \Delta^{cgr}}$$

Hence, $\mu = (Q, D)$ as required.

Case $h = a$. The following rule must have been applied:

$$\frac{a \in \Sigma \quad Q \in dPrj^\Delta(D)}{(Q, D) \xrightarrow{a} (Q, D) \text{ in } \Delta^{cgr}}$$

Hence, $\mu = (Q, D)$.

Case $h = \varepsilon$. Obvious.

Case $h = h_1 \cdot h_2$. There must exist some μ_1 such that $(Q, D) \xrightarrow{h_1} \mu_1 \xrightarrow{h_2} \mu$ wrt Δ^{cgr} . By induction hypothesis applied to h_1 , we have $\mu_1 = (Q, D)$. We can thus apply the induction hypothesis to h_2 to obtain $\mu_2 = (Q, D)$.

This ends the proof of Claim 2.1b.

We next need an inverse of Claim 2.2a. For relating runs A^{cgr} back to runs of A , we define for any difference relation D another binary relation $dep^b(D) \subseteq 2^Q \times 2^Q$ such that for any two subsets of states $Q', Q'' \subseteq Q$:

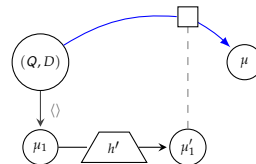
$$(Q', Q'') \in dep^b(D) \Leftrightarrow \begin{cases} (Q' \times Q'') \cap D = \emptyset & \wedge & (0^b) \\ Q' \notin dPrj^\Delta(D) \Rightarrow Q'' \subseteq Q' & \wedge & (1^b) \\ Q' \in dPrj^\Delta(D) \Rightarrow Q'' \subseteq acc^\Delta(Q') & & (2^b) \end{cases}$$

Claim 2.2b. Let $Q \in \mathcal{Q}$ be a subset of states that is compatible with a difference relation $D \in \mathcal{D}^\Delta$. For any hedge $h \in \mathcal{H}_\Sigma$ and state $\mu \in \mathcal{Q}^{cgr}$ with $(Q, D) \xrightarrow{h} \mu$ wrt Δ^{cgr} there exist a pair of subsets of states $(Q', Q'') \in dep^b(D)$ such that $\mu = (Q', D)$ and $Q \xrightarrow{h} Q''$ wrt Δ^{det} .

Proof. If $Q \in dPrj^\Delta(D)$ then Claim 2.1b shows that $\mu = (Q, D)$. Let $Q' = Q$ and Q'' be the unique subset of states such that $Q \xrightarrow{h} Q''$ wrt Δ^{det} . We have to show that $(Q', Q'') \in dep^b(D)$. Since $Q \xrightarrow{h} Q''$, we have $Q'' \subseteq acc^\Delta(Q')$, so condition (2^b) holds. Condition (1^b) holds trivially since $Q \in dPrj^\Delta(D)$. For condition (0^b), note that $Q \in dPrj^\Delta(D)$ implies $acc^\Delta(Q)^2 \cap D = \emptyset$. Furthermore, $Q' \times Q'' \subseteq acc^\Delta(Q)^2$, so that $(Q' \times Q'') \cap D = \emptyset$ as required.

We can thus assume that $Q \notin dPrj^\Delta(D)$. The proof is by induction on the structure of $h \in \mathcal{H}_\Sigma$. We distinguish all the possible forms of hedges $h \in \mathcal{H}_\Sigma$:

Case $h = \langle h' \rangle$. By definition of $(Q, P) \xrightarrow{h} \mu$ wrt Δ^{cgr} there must exist $\mu_1, \mu'_1 \in \mathcal{Q}^{cgr}$ such that the following diagram can be closed:



Since $Q \notin dPrj^\Delta(D)$, the following inference rule got applied to infer $(Q, D) \xrightarrow{\langle \rangle} \mu_1$ wrt Δ^{cgr} where $D' = D_Q$:

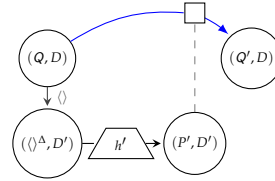
$$\frac{Q \notin dPrj^\Delta(D)}{(Q, D) \xrightarrow{\langle \rangle} (\langle \rangle^\Delta, D') \text{ in } \Delta^{cgr}}$$

Hence $\mu_1 = (\langle \rangle^\Delta, D')$. If $\langle \rangle^\Delta \notin dPrj^\Delta(D)$ then the induction hypothesis applied to $(\langle \rangle^\Delta, D') \xrightarrow{h'} \mu'_1$ wrt Δ^{cgr} show that there exists $(P', P'') \in dep^b(D')$ such that $\mu'_1 = (P', D')$ and $\langle \rangle^\Delta \xrightarrow{h'} P''$ wrt Δ^{det} . Otherwise, the same can be concluded as we argued at the beginning.

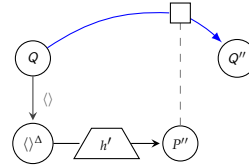
Subcase $P' \in dPrj^\Delta(D')$. The transition rule $(Q, D)@_{\mu_1} \rightarrow \mu$ must thus be inferred as follows for some $Q' \subseteq Q$:

$$\frac{Q@acc^\Delta(P') \rightarrow Q' \text{ in } \Delta^{det} \quad Q \notin dPrj^\Delta(D) \quad P' \in dPrj^\Delta(D)}{(Q, D)@(P', D') \rightarrow (Q', D) \text{ in } \Delta^{cgr}}$$

This shows that $\mu = (Q', D)$. So we have the following diagram:



Let Q'' be the unique subset of states such that $Q@P'' \rightarrow Q''$ wrt Δ^{det} . We can then close the following diagram:

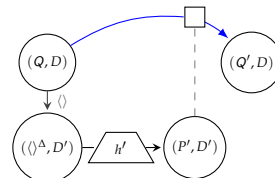


From $(P', P'') \in dep^b(D')$ it follows $(P' \times P'') \cap D' = \emptyset$, and thus $(Q' \times Q'') \cap D = \emptyset$, i.e., condition (0^b) of $(Q', Q'') \in dep^b(D)$. Since $P' \in dPrj^\Delta(D')$, condition (2^b) of $(P', P'') \in dep^b(D')$ yields $P'' \subseteq acc^\Delta(P')$. Since furthermore $Q@P'' \rightarrow Q''$ and $Q@acc^\Delta(P') \rightarrow Q'$ in Δ^{det} , this yields $Q'' \subseteq Q'$, so that conditions (1^b) and (2^b) of $(Q', Q'') \in dep^b(D)$ follow. Hence, $(Q', Q'') \in dep^b(D)$. In summary, we have show that $\mu = (Q', D)$, $Q \xrightarrow{\langle h' \rangle} Q''$ wrt Δ^{det} and $(Q', Q'') \in dep^b(D)$ as required by the claim.

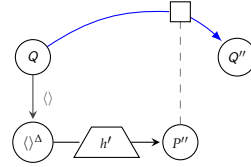
Subcase $P' \notin dPrj^\Delta(D')$. The transition rule $(Q, D)@_{\mu_1} \rightarrow \mu$ must thus be inferred as follows for some $Q' \subseteq Q$:

$$\frac{Q@P' \rightarrow Q' \text{ in } \Delta^{det} \quad Q \notin dPrj^\Delta(D) \quad P' \notin dPrj^\Delta(D)}{(Q, D)@(P', D') \rightarrow (Q', D) \text{ in } \Delta^{cgr}}$$

This shows that $\mu = (Q', D)$ and that we can close the following diagram:



Condition (0^b) of $(P', P'') \in \text{dep}^b(D')$ yields $(P' \times P'') \cap D' = \emptyset$. Let Q'' be the unique subset of states such that $Q@P'' \rightarrow Q''$ wrt. Δ^{det} . Since $D' \subseteq \text{down}_Q^\Delta(D)$ and $(P' \times P'') \cap D' = \emptyset$, it follows from $Q@P' \rightarrow Q'$ and $Q@P'' \rightarrow Q''$ wrt. Δ^{det} that $(Q' \cap Q'') \cap D = \emptyset$. That is condition (0^b) of $(Q', Q'') \in \text{dep}^b(D)$. Since $P' \notin \text{dPrj}^\Delta(D')$, condition (1^b) of $(P', P'') \in \text{dep}^b(D')$ is $P'' \subseteq P'$. From $Q@P'' \rightarrow Q''$ and $Q@P' \rightarrow Q'$ it thus follows that $Q'' \subseteq Q'$. Hence, conditions (1^b) and (2^b) of $(Q', Q'') \in \text{dep}^b(D)$ are valid too, so that $(Q', Q'') \in \text{dep}^b(D)$ holds. Furthermore:



This shows that $Q \xrightarrow{\langle h' \rangle} Q''$ wrt Δ . The other two requirements of the claim, $\mu = (Q', D)$ and $(Q', Q'') \in \text{dep}^b(D)$, were shown earlier.

Case $h = a$. Since $Q \notin \text{dPrj}^\Delta(D)$, the following inference rule must be used:

$$\frac{Q \xrightarrow{a} Q' \text{ in } \Delta \quad Q \notin \text{dPrj}^\Delta(D)}{(Q, D) \xrightarrow{a} (Q', D) \text{ in } \Delta^{\text{cgr}}}$$

So $\mu = (Q', D)$, $Q \xrightarrow{a} Q'$ wrt Δ^{det} . Furthermore, since Q is compatible with D , and D is a difference relation, Q' is compatible with D too. Hence $(Q' \times Q') \cap D = \emptyset$ showing condition (0^b) of $(Q', Q') \in \text{dep}^b(D)$. Trivially, $Q' \subseteq Q' \subseteq \text{acc}^\Delta(Q')$ so conditions (1^b) and (2^b) of $(Q', Q') \in \text{dep}^b(D)$ hold too. Hence $(Q', Q') \in \text{dep}^b(D)$.

Case $h = \varepsilon$. Obvious.

Case $h = h_1 \cdot h_2$. Since $(Q, D) \xrightarrow{h} \mu$ wrt. Δ^{cgr} there exists some $\mu_1 \in \mathcal{Q}^{\text{cgr}}$ such that $(Q, D) \xrightarrow{h_1} \mu_1$ and $\mu_1 \xrightarrow{h_2} \mu$ wrt Δ^{cgr} . We can apply the induction hypothesis to h_1 . Hence there exist subsets of states $Q_1, Q'_1 \subseteq Q$ such that $\mu_1 = (Q_1, D)$, $Q \xrightarrow{h_1} Q'_1$ wrt Δ , and $(Q_1, Q'_1) \in \text{dep}^b(D)$. We distinguish two cases:

Subcase $Q_1 \in \text{dPrj}^\Delta(D)$. Since $\mu_1 = (Q_1, D) \xrightarrow{h_2} \mu$ wrt Δ^{cgr} , Claim 2.1b shows in this case that $\mu = (Q_1, D)$ so that $(Q_1, D) \xrightarrow{h_2} (Q_1, D)$ wrt. Δ^{cgr} . Condition (2^b) of $(Q_1, Q'_1) \in \text{dep}^b(D)$ and $Q_1 \in \text{dPrj}^\Delta(D)$ imply that $Q'_1 \in \text{acc}^\Delta(Q_1)$. Let Q_2 be the unique subset of states such that $Q'_1 \xrightarrow{h_2} Q_2$ wrt. Δ^{det} . Then $Q_2 \subseteq \text{acc}^\Delta(Q'_1)$ so that $Q_2 \subseteq \text{acc}^\Delta(Q_1)$ and thus condition (2^b) of $(Q_1, Q_2) \in \text{dep}^b(D)$ holds. Condition (1^b) of $(Q_1, Q_2) \in \text{dep}^b(D)$ is trivial since $Q_1 \in \text{dPrj}^\Delta(D)$. Since $Q_1 \in \text{dPrj}^\Delta(D)$ and $Q_1 \times Q_2 \in \text{acc}^\Delta(Q)^2$ it follows that $Q_1 \times Q_2 \cap D = \emptyset$, so condition (0^b) of $(Q_1, Q_2) \in \text{dep}^b(D)$ holds too. Hence $Q \xrightarrow{h} Q_2$ wrt Δ^{det} and $(Q_1, Q_2) \in \text{dep}^b(D)$.

Subcase $Q_1 \notin \text{dPrj}^\Delta(D)$. Since $Q_1 \xrightarrow{h_1} Q'_1$ and Q_1 is compatible with difference relation D , it follows that Q'_1 is compatible with D too. We can thus apply the induction hypothesis to $(Q_1, D) \xrightarrow{h_2} \mu$ showing the existence of subsets of states $(Q_2, Q'_2) \in \text{dep}^b(D)$ such that $\mu = (Q_2, D)$ and $Q_1 \xrightarrow{h_2} Q'_2$ wrt Δ . So we have $Q \xrightarrow{h} Q'_2$ wrt. Δ and $(Q_2, Q'_2) \in \text{dep}^b(D)$ as required.

This ends the proof of Claim 2.2b.

Proof of inclusion $\mathcal{L}(A^{\text{cgr}}(\mathbf{S})) \cap \mathbf{S} \subseteq \mathcal{L}(A)$. Let $h \in \mathcal{L}(A^{\text{cgr}}(\mathbf{S})) \cap \mathbf{S}$. Then there exists a final subset of states $Q \in F^{\text{cgr}}(\mathbf{S})$ such that $(I, D_{\text{init}}^{A, F_S}) \xrightarrow{h} (Q, D_{\text{init}}^{A, F_S})$ wrt Δ^{cgr} . Since I is a

singleton or empty, it is compatible with D_{init}^{A, F_S} . Claim 2.2b thus shows that there exists a subset of states $Q' \subseteq Q$ such that $I \xrightarrow{h} Q'$ wrt. Δ^{det} and $(Q, Q') \in dep^b(D_{init}^{A, F_S})$. Condition (0^b) of $(Q, Q') \in dep^b(D_{init}^{A, F_S})$ shows that $(Q, Q') \cap D_{init}^{A, F_S} = \emptyset$. Since $h \in S$ it follows that $Q' \cap F_S \neq \emptyset$. The determinism of A shows that Q' is a singleton. So there exists a state $q' \in F_S$ such that $Q' = \{q'\}$.

Case $Q \notin dPrj^\Delta(D_{init}^{A, F_S})$. Condition (1^b) shows that $Q' \subseteq Q$, so that $q' \in Q$. Since $Q \in F_S^{cgr}$ we have $Q \cap F \neq \emptyset$. Since Q is compatible with D_{init}^{A, F_S} , $q' \in Q$ and $Q \cap F \neq \emptyset$, it follows that $q' \notin F_S \setminus F$. Since $q' \in F_S$ this implies $q' \in F$ and thus $h \in \mathcal{L}(A)$.

Case $Q \in dPrj^\Delta(D_{init}^{A, F_S})$. Condition (2^b) shows that $Q' \subseteq acc^\Delta(Q)$, so that $q' \in acc^\Delta(Q)$. Since $Q \in F_S^{cgr}$ we have $acc^\Delta(Q) \cap F \neq \emptyset$. Let $q \in acc^\Delta(Q) \cap F$ be arbitrary. Since $Q \in dPrj^\Delta(D_{init}^{A, F_S})$, and $(q', q) \in acc^\Delta(Q)^2$ it follows that $(q, q') \notin D_{init}^{A, F_S}$. Furthermore, $q \in F$ so that $q' \notin F_S \setminus F$. In combination with $q' \in F_S$ this implies $q' \in F_S$ so $h \in \mathcal{L}(A)$.

This ends the proof of the inverse inclusion, and thus of $\mathcal{L}(A) = \mathcal{L}(A^{cgr(S)}) \cap S$. \square

7.4. Completeness

We next show the completeness of congruence projection for subhedge projection according to Definition 25. Let $A = (\Sigma, Q, \Delta, I, F)$ be a complete dSHA and $F \subseteq F_S \subseteq Q$. Automaton A defines the regular pattern $L = \mathcal{L}(A)$ and with the schema-final states in F_S the regular schema $S = \mathcal{L}(A[F/F_S])$. We first show that all subhedge irrelevant states of difference relations are subhedge projection states $A^{cgr(S)}$ according to Definition 18.

Lemma 45. *If $Q \in dPrj^\Delta(D)$ then (Q, D) is a subhedge projection state of Δ^{cgr} with witness (Q, D) .*

Proof. We assume that $q \in Prj^\Delta(D)$ and have to show that (Q, D) is a subhedge projection state of Δ^{cgr} . We have to show that all transition rules starting with (Q, D) are permitted for a subhedge projection state (Q, D) with witness (Q, D) . They are generated by the following rules, all of which are looping:

$$\frac{a \in \Sigma \quad Q \in dPrj^\Delta(D)}{(Q, D) \xrightarrow{a} (Q, D) \text{ in } \Delta^{cgr}} \quad \frac{Q \in dPrj^\Delta(D)}{(Q, D)@(Q, D) \rightarrow (Q, D) \text{ in } \Delta^{cgr}} \quad \frac{Q \in dPrj^\Delta(D)}{(Q, D) \xrightarrow{\langle \rangle} (Q, D) \text{ in } \Delta^{cgr}}$$

\square

So if a partial run of $A^{cgr(S)}$ on a prefix u assigns some state (P, D) with $P \in dPrj^\Delta(D)$, then (P, D) is a subhedge projection state by Lemma 45, and thus subhedge irrelevant by Proposition 20.

Lemma 46. *let $A' = A[I/\langle \rangle^\Delta]$, $L' = \mathcal{L}(A')$ and $S' = \mathcal{L}(A'[F/F_S])$. If then $(\langle \rangle^\Delta, D) \xrightarrow{hdg(u)} (Q, D)$ wrt $A^{cgr(S)}$ for some nested word $u \in \mathcal{N}_\Sigma$ and either*

- $Q \notin dPrj^\Delta(D)$ and $q \in Q$, or
- $Q \in dPrj^\Delta(D)$ and $q \in acc^\Delta(Q)$,

then there exists a nested word $u' \in class_{S'}^{L'}(u)$ and $q_0 \in \langle \rangle^\Delta$ such that $q_0 \xrightarrow{hdg(u')} q$ wrt Δ .

Proof. By induction on the length of u . Suppose that $(\langle \rangle^\Delta, D) \xrightarrow{hdg(u)} (Q, D)$ wrt $A^{cgr(S)}$. In the base case, we have $u = \varepsilon$. Hence $Q = \langle \rangle^\Delta$.

Case $Q \notin dPrj^\Delta(D)$ and $q \in Q$. Then $\langle \rangle^\Delta = \{q\}$. The unique run of A' on $u = \varepsilon$ starts and ends in the tree initial state $q \in \langle \rangle^\Delta$.

Case $Q \in dPrj^\Delta(D)$ and $q \in acc^\Delta(Q)$. Since $Q \in dPrj^\Delta(D)$, Lemma 45 shows that (Q, D) is a subhedged projection state, so that ε is subhedged irrelevant for L' and S' by Proposition 20. Hence, $class_{S'}^{L'}(\varepsilon) = \mathcal{N}_\Sigma$. Furthermore, since $q \in acc^\Delta(Q)$, there exists a hedge h and $q_0 \in I$ such that $q_0 \xrightarrow{h} q$ wrt Δ . Let $u' = nw(h)$. The run of A on u' then ends in q and $u' \in class_{S'}^{L'}(u)$.

For the induction step, we distinguish the possible forms of the nested word u . So there exist $\tilde{u}, v \in \mathcal{N}_\Sigma$ and $a \in \Sigma$ such that either of the following cases holds:

Case $u = \tilde{u} \cdot \langle \cdot v \cdot \rangle$. Let (\tilde{Q}, D) be the state in which the run of $(A')^{cgr(S)}$ on \tilde{u} ends.

Subcase $\tilde{Q} \notin dPrj^\Delta(D)$. Let $D' = D_{\tilde{Q}}$. Then there exist $P \subseteq Q$ such that $(\langle \rangle^\Delta, D') \xrightarrow{v} (P, D')$ wrt $A^{cgr(S)}$. So the run of $(A')^{cgr(S)}$ on v goes to state (P, D') .

Subsubcase $P \in dPrj^\Delta(D')$. By construction of $A^{cgr(S)}$, we then have $Q = \tilde{Q} @^\Delta acc^\Delta(P)$. Since $q \in Q$ there exist $\tilde{q} \in \tilde{Q}$ and $p \in acc^\Delta(P)$ such that $\tilde{q} @ p \rightarrow q$ in Δ . By induction hypothesis, there exists $v' \in class_{S'}^{L'}(v)$ such that the run of A' on v' ends in p . Hence the run of A on $u' = \tilde{u} \cdot \langle \cdot v' \cdot \rangle$ ends in q , and furthermore, $u' \in class_S^L(u)$.

Subsubcase $P \notin dPrj^\Delta(D')$. By construction of $A^{cgr(S)}$, we then have $Q = \tilde{Q} @^\Delta P$. Since $q \in Q$ there exist $\tilde{q} \in \tilde{Q}$ and $p \in P$ such that $\tilde{q} @ p \rightarrow q$ in Δ . By induction hypothesis, there exists $v' \in class_{S'}^{L'}(v)$ such that the run of A' on v' ends in p . Hence the run of A on $u' = \tilde{u} \cdot \langle \cdot v' \cdot \rangle$ ends in q , and furthermore, $u' \in class_S^L(u)$.

Subcase $\tilde{Q} \in dPrj^\Delta(D)$. Then $Q = \tilde{Q}$ and $q \in acc^\Delta(\tilde{Q})$. By induction hypothesis applied to \tilde{u} there exists $\tilde{u}' \in class_S^L(\tilde{u})$ such that $\langle \rangle^\Delta \xrightarrow{hdg(\tilde{u}')} q$. Since \tilde{u} is irrelevant for L and S' we have that $\tilde{u}' \in class_S^L(u)$.

Case $u = \tilde{u} \cdot a$. Easier than the previous cases and thus omitted.

□

Lemma 47. *If a partial run of $A^{cgr(S)}$ assign some state (Q, D) to a prefix $u \in prefs(\mathcal{N}_\Sigma)$. If then $Q \notin dPrj^\Delta(D)$ and $q \in Q$ then there exists a prefix $u' \in class_S^L(u)$ such that a partial run of A assigns state q to u' .*

Proof. By induction on the length of u . In the base case, we have $u = \varepsilon$. The partial run of $A^{cgr(S)}$ on u assigns state (Q, D) where $Q = I$ and $D = D_{init}^{A, Fs}$. Suppose that $Q \notin dPrj^\Delta(D)$ and $q \in Q$. Then $I = \{q\}$. The unique partial run of A on $u = \varepsilon$ ends in the initial state $q \in I$. For the induction step, let $A' = A[I / \langle \rangle^\Delta]$, $L' = \mathcal{L}(A')$ and $S' = \mathcal{L}(A'[F / Fs])$. We distinguish the possible forms of u . So there exist $\tilde{u} \in prefs(\mathcal{N}_\Sigma)$, $v \in \mathcal{N}_\Sigma$, and $a \in \Sigma$ such that either of the following hold:

Case $u = \tilde{u} \cdot \langle \cdot v \cdot \rangle$. Let (\tilde{Q}, \tilde{D}) be the state in which the partial run of $A^{cgr(S)}$ on \tilde{u} ends. From $Q \notin dPrj^\Delta(D)$ it follows that $\tilde{Q} \notin dPrj^\Delta(\tilde{D})$. Furthermore, $(\langle \rangle^\Delta, \tilde{D}) \xrightarrow{v} (Q, D)$ wrt $A^{cgr(S)}$. So the partial run of $(A')^{cgr(S)}$ on v goes to state (Q, D) . By induction hypothesis, there exists $v' \in class_{S'}^{L'}(v)$ such that the partial run of A' on v' ends in q . Hence the partial run of A on $u' = \tilde{u} \cdot \langle \cdot v' \cdot \rangle$ ends in q , and furthermore, $u' \in class_S^L(u)$.

Case $u = \tilde{u} \cdot \langle \cdot v \cdot \rangle$. Let (\tilde{Q}, \tilde{D}) be the state in which the partial run of $A^{cgr(S)}$ on \tilde{u} ends. From $Q \notin dPrj^\Delta(D)$ it follows that $\tilde{Q} \notin dPrj^\Delta(\tilde{D})$. Furthermore, $(\langle \rangle^\Delta, \tilde{D}) \xrightarrow{v} (P, D)$ wrt $A^{cgr(S)}$ for some $P \subseteq Q$.

Subcase $P \in dPrj^\Delta(D)$. Then $Q = \tilde{Q} @^\Delta acc^\Delta(P)$. So there exists $\tilde{q} \in Q$ and $p \in acc^\Delta(P)$ such that $\tilde{q} @ p \rightarrow q$ in Δ . By Lemma 46 there exists $v' \in class_{S'}^{L'}(v)$ such that $\langle \rangle^\Delta \xrightarrow{hdg(v')} p$ wrt Δ . By induction hypothesis applied to \tilde{u} there exists an

initial state $q_0 \in I$ and a prefix $\tilde{u}' \in \text{class}_S^L(\tilde{u})$ such that the partial run of A on \tilde{u}' goes to state \tilde{q} . Hence the partial run of A on $u' = \tilde{u}' \cdot \langle \cdot v' \cdot \rangle$ goes to q and $u' \in \text{class}_S^L(u)$.

Subcase $P \notin dPrj^\Delta(D)$. Then $Q = \tilde{Q} @^\Delta P$. So there exists $\tilde{q} \in Q$ and $p \in P$ such that $\tilde{q} @ p \rightarrow q$ in Δ . By Lemma 46 there exists $v' \in \text{class}_S^{L'}(v)$ such that $\langle \rangle^\Delta \xrightarrow{\text{hdg}(v')} p$ wrt Δ . By induction hypothesis applied to \tilde{u} there exists an initial state $q_0 \in I$ and a prefix $\tilde{u}' \in \text{class}_S^L(\tilde{u})$ such that the partial run of A on \tilde{u}' goes to state \tilde{q} . Hence the partial run of A on $u' = \tilde{u}' \cdot \langle \cdot v' \cdot \rangle$ goes to q and $u' \in \text{class}_S^L(u)$.

Case $u = \tilde{u} \cdot a$. Easier than the previous case and thus omitted.

□

The next lemma states the key invariant of congruence projection, which eventually proves it completeness for subhedge projection. For this we define for any $F \subseteq F_S \subseteq F$, the binary relation $\approx_{F_S}^F$ as the symmetric closure of $F \times (F_S \setminus F)$, i.e., for all $(q', q'') \in Q^2$:

$$q' \approx_{F_S}^F q'' \Leftrightarrow \left\{ \begin{array}{l} (q' \in F \quad \wedge \quad q'' \in F_S \setminus F) \quad \vee \\ (q' \in F_S \setminus F \quad \wedge \quad q'' \in F) \end{array} \right.$$

Lemma 48 (Key Invariant). *Let $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ a dSHA, $F \subseteq F_S \subseteq \mathcal{Q}$, $L = \mathcal{L}(A)$ and $S = \mathcal{L}(A[F/F_S])$. If $A^{cgr(S)}$ has a partial run on prefix $u \in \text{prefs}(\mathcal{N}_\Sigma)$ to state $(P, D) \in 2^{\mathcal{Q}} \times \mathcal{D}^\Delta$ then for all $(p', p'') \in P^2$, $(r', r'') \in D$, and $(v', v'') \in \mathcal{N}_\Sigma^2$ such that:*

$$p' \xrightarrow{\text{hdg}(v')} r' \text{ wrt } \Delta \quad \wedge \quad p'' \xrightarrow{\text{hdg}(v'')} r'' \text{ wrt } \Delta$$

there exist $(u', u'') \in \text{class}_S^L(u)$, $w \in \text{suffs}(\mathcal{N}_\Sigma)$, $q' \approx_{F_S}^F q''$, and $q_0 \in I$ such that:

$$q_0 \xrightarrow{\text{hdg}(u' \cdot v' \cdot w)} q' \text{ wrt } \Delta \quad \wedge \quad q_0 \xrightarrow{\text{hdg}(u'' \cdot v'' \cdot w)} q'' \text{ wrt } \Delta$$

Proof. By induction on the number of dangling opening parenthesis of u .

In the base case, u does not have any dangling opening parenthesis, so $u \in \mathcal{N}_\Sigma$ is a nested word. In this case, $D = D_{init}^{A, F_S}$. So $A^{cgr(S)}$ has a partial run on nested word $u \in \mathcal{N}_\Sigma$ to (P, D_{init}^{A, F_S}) where $P \subseteq \mathcal{Q}$. Let $(p', p'') \in P^2$, $(r', r'') \in D_{init}^{A, F_S}$, and $(v', v'') \in \mathcal{N}_\Sigma^2$ such that:

$$p' \xrightarrow{\text{hdg}(v')} r' \text{ wrt } \Delta \quad \wedge \quad p'' \xrightarrow{\text{hdg}(v'')} r'' \text{ wrt } \Delta$$

It then follows that $P \notin dPrj^\Delta(D_{init}^{A, F_S})$. Therefore we can apply Lemma 47. It shows that there exist nested words $(u', u'') \in (\text{class}_S^L(u))^2$ and $q_0 \in I$ such that:

$$q_0 \xrightarrow{\text{hdg}(u')} p' \text{ wrt } \Delta \quad \wedge \quad q_0 \xrightarrow{\text{hdg}(u'')} p'' \text{ wrt } \Delta$$

Since $D_{init}^{A, F_S} \subseteq \text{ldr}^\Delta(F \times (F_S \setminus F))$ by Lemma 36, there exist a nested word $w \in \mathcal{N}_\Sigma$ and states $q' \approx_{F_S}^F q''$ such that:

$$r' \xrightarrow{\text{hdg}(w)} q' \text{ wrt } \Delta \quad \wedge \quad r'' \xrightarrow{\text{hdg}(w)} q'' \text{ wrt } \Delta$$

Then we have:

$$q_0 \xrightarrow{\text{hdg}(u' \cdot v' \cdot w)} q' \text{ wrt } \Delta \quad \wedge \quad q_0 \xrightarrow{\text{hdg}(u'' \cdot v'' \cdot w)} q'' \text{ wrt } \Delta$$

For the induction step let $u = u_1 \cdot \langle \cdot u_2$ for some prefix $u_1 \in \text{prefs}(\mathcal{N}_\Sigma)$ and nested word $u_2 \in \mathcal{N}_\Sigma$, so that u_1 has one open dangling bracket less than u . Let $A^{cgr(S)}$ have a

partial run on nested word $u \in \mathcal{N}_\Sigma$ to (P, D) . Let (P_1, D_1) be the state that this run assigns to prefix u_1 . Then $D = (D_1)_{P_1}$. Let $(p', p'') \in P^2$, $(r', r'') \in D$, and $(v', v'') \in \mathcal{N}_\Sigma^2$ such that:

$$p' \xrightarrow{\text{hdg}(v')} r' \text{ wrt } \Delta \wedge p'' \xrightarrow{\text{hdg}(v'')} r'' \text{ wrt } \Delta$$

Since $D = (D_1)_{P_1}$ there exist $(p'_1, p''_1) \in (P_1)^2$ and states $(r'_1, r''_1) \in D_1$ such that $p'_1 @ q' \rightarrow r'_1$ and $p''_1 @ q'' \rightarrow r''_1$. In particular, $P_1 \notin dPrj^\Delta(D_1)$ so that we can apply Lemma 47. It shows that there exist $(u'_2, u''_2) \in \text{class}_{\mathbf{S}}^L(u_2)$ such that $\langle \rangle^\Delta \xrightarrow{u'_2} p'$ and $\langle \rangle^\Delta \xrightarrow{u''_2} p''$. Let $v'_1 = \langle \cdot u'_2 \cdot v' \cdot \rangle$ and $v''_1 = \langle \cdot u''_2 \cdot v'' \cdot \rangle$. Then $p'_1 \xrightarrow{\text{hdg}(v'_1)} r'_1$, $p''_1 \xrightarrow{\text{hdg}(v''_1)} r''_1$ wrt Δ . By induction hypothesis applied to u_1 – on which $A^{\text{cgr}(\mathbf{S})}$ has a partial run to state (P_1, D_1) such that $(p'_1, p''_1) \in (P_1)^2$, $p'_1 \xrightarrow{\text{hdg}(v'_1)} r'_1$, $p''_1 \xrightarrow{\text{hdg}(v''_1)} r''_1$, and $(r'_1, r''_1) \in D$ – there exist $(u'_1, u''_1) \in \text{class}_{\mathbf{S}}^L(u_1)$, $w_1 \in \text{suffs}(\mathcal{N}_\Sigma)$, $q' \not\approx_{F_{\mathbf{S}}}^F q''$ and $q_0 \in I$ such that:

$$q_0 \xrightarrow{\text{hdg}(u'_1 \cdot v'_1 \cdot w_1)} q' \text{ wrt } \Delta \wedge q_0 \xrightarrow{\text{hdg}(u''_1 \cdot v''_1 \cdot w_1)} q'' \text{ wrt } \Delta$$

Let $u' = u'_1 \cdot \langle \cdot u'_2$ and $u'' = u''_1 \cdot \langle \cdot u''_2$ and $w = \rangle \cdot w_1$. The above then yields:

$$q_0 \xrightarrow{\text{hdg}(u' \cdot v' \cdot w)} q' \text{ wrt } \Delta \wedge q_0 \xrightarrow{\text{hdg}(u'' \cdot v'' \cdot w)} q'' \text{ wrt } \Delta$$

Furthermore, $(u', u'') \in (\text{class}_{\mathbf{S}}^L(u))^2$, so this was to be shown. \square

Proposition 49. *Let $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ a dSHA, $F \subseteq F_{\mathbf{S}} \subseteq \mathcal{Q}$, $L = \mathcal{L}(A)$ and $\mathbf{S} = \mathcal{L}(A[F/F_{\mathbf{S}}])$. If $A^{\text{cgr}(\mathbf{S})}$ has a partial run on prefix $u \in \text{pref}(\mathcal{N}_\Sigma)$ to some state $(P, D) \in 2^{\mathcal{Q}} \times \mathcal{D}^\Delta$ so that $P \notin dPrj^\Delta(D)$, then the set $\text{class}_{\mathbf{S}}^L(u)$ is subhedge relevant for L wrt \mathbf{S} .*

Proof. Suppose that $A^{\text{cgr}(\mathbf{S})}$ has a partial run on prefix $u \in \text{pref}(\mathcal{N}_\Sigma)$ to some state $(P, D) \in 2^{\mathcal{Q}} \times \mathcal{D}^\Delta$ so that $P \notin dPrj^\Delta(D)$. Since $P \notin dPrj^\Delta(D)$ there exist $\text{acc}^\Delta(P)^2 \cap D \neq \emptyset$. So there exist $(p', p'') \in P^2$, $(r', r'') \in D$, and $(v', v'') \in \mathcal{N}_\Sigma^2$ such that:

$$p' \xrightarrow{\text{hdg}(v')} r' \text{ wrt } \Delta \wedge p'' \xrightarrow{\text{hdg}(v'')} r'' \text{ wrt } \Delta$$

By Lemma 48 there exist $(u', u'') \in \text{class}_{\mathbf{S}}^L(u)$, $w \in \text{suffs}(\mathcal{N}_\Sigma)$, $q' \not\approx_{F_{\mathbf{S}}}^F q''$ and $q_0 \in I$ such that:

$$q_0 \xrightarrow{\text{hdg}(u' \cdot v' \cdot w)} q' \text{ wrt } \Delta \wedge q_0 \xrightarrow{\text{hdg}(u'' \cdot v'' \cdot w)} q'' \text{ wrt } \Delta$$

Hence, the set $\text{class}_{\mathbf{S}}^L(u)$ is relevant for L wrt \mathbf{S} . \square

Theorem 3 (Completeness of congruence projection). *For any complete dSHA $(\Sigma, \mathcal{Q}, \Delta, I, F)$ and set $F \subseteq F_{\mathbf{S}} \subseteq \mathcal{Q}$, the congruence projection $A^{\text{cgr}(\mathbf{S})}$ is sound and complete for subhedge projection for the regular pattern $\mathcal{L}(A)$ wrt the regular schema $\mathbf{S} = \mathcal{L}(A[F_{\mathbf{S}}/F])$.*

Proof. The soundness of congruence projection was shown in Theorem 2. For proving the completeness, let $u \in \text{pref}(\mathcal{N}_\Sigma)$ be a nested word prefix that is strongly subhedge irrelevant for L wrt \mathbf{S} . By definition of strong irrelevance, the class $\text{class}_{\mathbf{S}}^L(u)$ is irrelevant for L and \mathbf{S} . Let (P, D) be the unique state assigned by the partial run of $A^{\text{cgr}(\mathbf{S})}$ to prefix u . Since $\text{class}_{\mathbf{S}}^L(u)$ is irrelevant for L and \mathbf{S} , Proposition 49 shows that $P \in dPrj^\Delta(D)$. By Lemma 45, (P, D) then is a subhedge projection state of $A^{\text{cgr}(\mathbf{S})}$. \square

7.5. In-Memory Complexity

We next discuss the complexity of membership testing with complete subhedge projection based on the in-memory evaluation of the input hedge by the congruence projection of the input automaton.

Lemma 50. *The number of states $|Q^{cgr}|$ is in $O(2^{n^2+n})$ where n is the number of states of A .*

Proof. With the deterministic construction the states of $A^{cgr(S)}$ are pairs in $2^Q \times \mathcal{D}^\Delta$. So, the maximal number of states of the congruence projection is $|Q^{cgr(S)}| = 2^{|Q|} |\mathcal{D}^\Delta| \leq 2^n 2^{n^2} = 2^{n^2+n}$. \square

Let $diff-rel(Q^{cgr(S)})$ be the set of difference relations used in the states of $A^{cgr(S)}$.

Corollary 4. *Let the signature Σ be fixed. For any complete dSHA A with schema $S \subseteq \mathcal{H}_\Sigma$, the membership of any hedge $h \in S$ can be tested in-memory in time $O(|1|)$ per nonprojected node of h after a preprocessing time of $O(|A^{cgr(S)}| + n^3d)$ where d the cardinality of $diff-rel(Q^{cgr(S)})$ and n is the number of states of A .*

Since $|Q^{cgr(S)}|$ is in $O(2^{n^2+n})$ by Lemma 50, the preprocessing time $O(|A^{cgr(S)}| + n^3d)$ is in $O(2^{2n^2+2n})$ too.

Proof. Since A is complete, Theorem 3 shows that $A^{cgr(S)}$ is complete for subhedge projection for schema S . The in-memory evaluator of $A^{cgr(S)}$ can be run on any input hedge $h \in S$ in time $O(|1|)$ per nonprojected node of h after a preprocessing time of $O(|A^{cgr(S)}| + n^3d)$ where d is the cardinality of $diff-rel(A^{cgr(S)})$. Since the signature Σ is fixed and $A^{cgr(S)}$ deterministic, $|A^{cgr(S)}|$ is in $O(|Q^{cgr(S)}|^2)$. The dominating part $|A^{cgr(S)}|$ is in $O(|Q^{cgr}|^2)$. \square

When applied to regular XPath query answering as in our experiments in Section 11, we have $n \leq 80$ and $d \leq 25$ so the cubic part $O(n^3d)$ is not limiting. The sizes of the SHA $^\downarrow$ s there are below 2600 counting both states and rules. The upper bound $O(|A^{cgr(S)}| + n^3d)$ thus suggests that the preprocessing time may be feasible in practice, even though exponential in the worst case.

We must be careful here, since the sizes of the SHA $^\downarrow$ s reported in Section 11 do not refer to the pure congruence projection $A^{cgr(S)}$ but to the earliest congruence projection $A_{e(S)}^{cgr(S)}$ that we will introduce only in Section 8. The noncreation of transition rules to states that are safe for rejection may lead to a radical size reduction. Alternatively, an important size reduction of $A^{cgr(S)}$ could be obtained by removing all useless states and transition rules. But this would not reduce the precomputation time, just the opposite: it would increase the preprocessing time, since cleaning automata a posteriori also needs time, which for SHA $^\downarrow$ s may be nonlinear.

Instead of precomputing $A^{cgr(S)}$ from the inputs A and F_S statically, we can compute only the part needed of $A^{cgr(S)}$ for evaluating the input hedge h on-the-fly. We note that the number of transition rules of the needed part is bounded by $O(|h|)$ but may be way smaller due to projection. The preprocessing time then goes down to $O(|A|)$. Since the full automaton $A^{cgr(S)}$ may be exponential in n in the worst case, this may reduce the overall processing time.

On the positive side, on-the-fly evaluation reduces the overall computation time to $O(|h| + n^3d(h))$, where $d(h)$ is the number of difference relations in the part of $A^{cgr(S)}$ needed for the evaluation of h , so $d(h) \leq \min(|h|, d)$. On the down side, $d(h)$ steps will no more be in constant but cubic time in $O(n^3)$. It should also be noted that the overall time of the safe-no-change projection on-the-fly evaluation is in $O(|A| + |h| + n s(h))$, where $s(h)$ is the number of subsets of safe states of A in states of Q^{cgr} , ie., $s(h) \leq |h|$. This is since each subsets of safe states can be computed in linear time in $O(n)$. As argued above, the overall time of the congruence projection obtained by on-the-fly evaluation is in $O(|A| + |h| + n^3 d(h))$. This is because the computation of a difference relation is in $O(n^3)$ by using ground Datalog programs of cubic size. The additional cubic factor in $n^3 d(h)$ instead of a linear factor in $n s(h)$ is the price to be paid for achieving complete subhedge projection with on-the-fly evaluation. Note that this price is largely acceptable in cases where $n \leq 80$, $d(h) \leq 15$, and $s(h) \leq 15$.

Example 51 (Exponential blow-up). *In order to see how the exponential worst case may happen, we consider a family of regular languages, for which the minimal left-to-right DFA is exponentially bigger than the minimal right-to-left DFA. The classical example languages with this property are $L_n = \Sigma^*.a.\Sigma^n$ where $n \in \mathbb{N}$ and $\Sigma = \{a, b\}$. Intuitively, a word in Σ^* belongs to L_n if and only its $n + 1$ -th letter from the end is equal to "a". The minimal left-to-right DFA for L_n has 2^{n+1} many states, since needs to memoize a window of $n + 1$ -letters. In contrast, its minimal right-to-left DFA has only $n + 1$ states; in this direction, it is sufficient to memoize the distance from the end modulo $n + 1$.*

We next consider the schema S defined by $\mu z. \langle (a + b) \cdot z \rangle^$. It contains all hedges having in all its subtrees a label from $\Sigma = \{a, b\}$ as the first letter, and no further occurrences of letters of Σ elsewhere. We then consider the family of hedge languages $H_n \in \mathcal{H}_\Sigma$ with schema S such that the sequence of labels of some root-to-leaf path of h_n belongs to L_n . Note that H_n can be recognized in a bottom-up manner by the dSHA A_n with $O(n + 1)$ states, which simulates the minimal deterministic DFA of L_n on all paths of the input hedge. For an evaluator with subhedge projection the situation is different. When moving top-down, it needs to memoize the sequence of labels of the $n + 1$ -last ancestors, possibly filled with b 's, and there are 2^{n+1} of such sequences. If for some leaf, its sequence starts with an "a" then the following subhedges with the following leaves can be projected away. As a consequence, there cannot be any $d\text{SHA}^\downarrow$ recognizing H_n that projects away all irrelevant subhedges with less than 2^{n+1} states. In particular, the size of $A_n^{\text{gr}(S)}$ must be exponential in the size of A_n .*

8. Earliest Membership with Subhedge Projection

We next enhance our compilers for introducing subhedge projection states such that they can also detect certain membership or nonmembership in an earliest manner. For this we show how to combine the previous earliest membership tester for dSHAs from [21] with any subtree projection algorithm, so that it applies to both, to safe-no-change projection and to congruence projection. By combining both aspects orthogonally, we obtain an earliest dSHA with subhedge projection, that can be evaluated either in inmemory or in streaming mode.

8.1. Earliest Membership

We start with a semantic characterization of earliest membership, i.e., of prefixes of nested words of hedges whose suffixes are irrelevant for membership, when assuming top-down and left-to-right processing. Note that the same characterization up to various presentation choices was given earlier in the context of stream processing [21,22].

Definition 52. *Let $S \subseteq \mathcal{H}_\Sigma$ be a hedge schema and $L \subseteq S$ a hedge language satisfying this schema. A nested word prefix v is certain for membership in L with schema S if for all suffixes w of nested words such that $v \cdot w \in \text{nw}(S)$ it holds that $v \cdot w \in \text{nw}(L)$. It is called certain for nonmembership in L with schema S if for all suffixes w of nested words such that $v \cdot w \in \text{nw}(S)$ it holds that $v \cdot w \notin \text{nw}(L)$.*

We are now interested in SHA^\downarrow that are able to detect certain membership and nonmembership syntactically at the earliest possible prefix or node of the run.

Definition 53. *A $d\text{SHA}^\downarrow A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ with schema $S \subseteq \mathcal{H}_\Sigma$ is called earliest for schema S if there exists a state $\text{sel} \in \mathcal{Q}$ such that for all hedges $h \in S$ the unique run R of $\text{compl}(A)$ on h satisfies:*

- *it goes to the state sel for exactly all those prefixes of $\text{nw}(h)$ that are certain for membership in $\mathcal{L}(A)$ wrt S , and*
- *it goes to the sink for all prefixes of $\text{nw}(h)$ that are certain for nonmembership in $\mathcal{L}(A)$ with S .*

For dSHA A that is schema-complete for schema $S = \mathcal{L}(A[F/F_S])$, we can construct a $d\text{SHA}^\downarrow A_{e(S)}$ that is earliest for S , as shown in Appendix N. The idea is to adapt the earliest

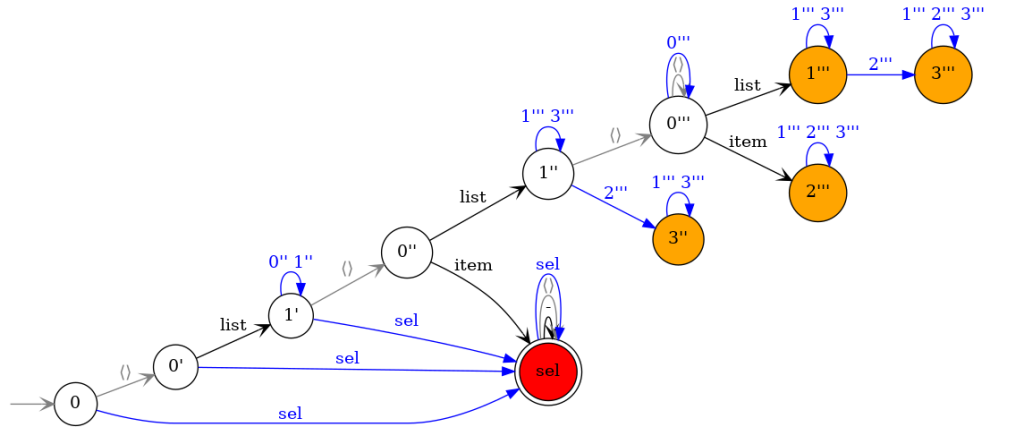


Figure 19. The earliest $\text{dSHA}^\downarrow A_{e(\mathbf{S})}$ for the dSHA A in Fig. 4 with schema $\mathbf{S} = \llbracket \text{doc} \rrbracket$ for the XPath filter $[\text{self}::\text{list}/\text{child}::\text{item}]$.

membership tester for deterministic SHAs from [21] so that it takes schema restrictions into account. Furthermore, we need to compile the input dSHAs to dSHA^\downarrow s rather than to dNWA's as used there. This change is straightforward once the relationship between both automata models becomes clear.

Given that the construction of $A_{e(\mathbf{S})}$ is not part of the core of the present paper, we only illustrate its outcome at our running example: For the dSHA A with schema $\mathbf{S} = \llbracket \text{doc} \rrbracket$ in Fig. 4 we obtain the $\text{SHA}^\downarrow A_{e(\mathbf{S})}$ in Fig. 19. Note that on the prefix $\langle \text{list} \langle \text{item}$ it goes to state sel showing that it is certain for membership. On prefix $\langle \text{item}$ it blocks showing that it is certain for nonmembership. However, on the prefix $\langle \text{list} \langle \text{list}$ it does not go into a subhedge projection state even though this prefix is subhedge irrelevant.

8.2. Adding Subhedge Projection

We now enhance any earliest membership for dSHAs with subhedge projection. The idea is to combine subhedge projection with earliest membership. So suppose that we are given a schema \mathbf{S} and two automata A^π and A_e with schema \mathbf{S} that recognize the same language up to the schema, so $\mathcal{L}(A^\pi) \cap \mathcal{L}(\mathbf{S}) = \mathcal{L}(A_e) \cap \mathcal{L}(\mathbf{S})$. Furthermore assume that A_e has a selection state sel indicating certain membership at the earliest possible prefix.

In order to obtain earliest membership with subhedge projection, we combine the two SHA^\downarrow s into a single $\text{SHA}^\downarrow A_e^\pi = (\Sigma, Q_e^\pi, \Delta_e^\pi, I_e^\pi, F_e^\pi)$ basically running both automata in parallel, but under a shared control. The state set of A_e^π are as follows:

$$\begin{aligned} Q_e^\pi &= (Q^\pi \times (Q_e \setminus \{sel\})) \cup \{sel\} \\ I_e^\pi &= (I^\pi \times (I_e \setminus \{sel\})) \cup \{sel \mid sel \in I_e\} \\ F_e^\pi &= (F^\pi \times (F_e \setminus \{sel\})) \cup \{sel\} \end{aligned}$$

The transition rules in Δ_e^π are given by the in Fig. 20. Any run of A_e^π synchronizes parallel runs of A^π and A_e as follows: whenever A_e goes to sel , then A_e^π does so too, and whenever A^π goes into a subhedge projection state then it makes A_e jump over the subsequent subhedge.

For illustration, we reconsider the dSHA from Fig. 4. The earliest dSHA^\downarrow with subhedge projection $A_{e(\mathbf{S})}^{snc}$ is given in Fig. 21. Here we combine the safe-no-change projection A_e^{snc} from Fig. 11 with the earliest membership tester $A_{e(\mathbf{S})}$ from Fig. 19. Their combination $A_{e(\mathbf{S})}^{snc}$ represents the pattern $[\text{self}::\text{list}/\text{child}::\text{item}]$ with schema $\llbracket \text{doc} \rrbracket$ in a concise manner. After prefix $\langle \text{list} \langle \text{item}$ it detects certain membership, for the prefix $\langle \text{list} \langle \text{list}$ it detects that the subhedge is irrelevant, and after prefix $\langle \text{item}$ it detects certain nonmembership.

$$\begin{array}{c}
\frac{q \xrightarrow{a} q' \text{ in } \Delta^\pi \quad r \xrightarrow{a} r' \text{ in } \Delta_e \quad q \notin P \quad r' \neq \text{sel}}{(q, r) \xrightarrow{a} (q', r') \text{ in } \Delta_e^\pi} \quad \frac{q \xrightarrow{a} q' \text{ in } \Delta^\pi \quad r \xrightarrow{a} \text{sel} \text{ in } \Delta_e}{(q, r) \xrightarrow{a} \text{sel} \text{ in } \Delta_e^\pi} \\
\\
\frac{q \xrightarrow{a} q' \text{ in } \Delta^\pi \quad r \xrightarrow{a} r' \text{ in } \Delta_e \quad q \in P \quad r' \neq \text{sel}}{(q, r) \xrightarrow{a} (q', r) \text{ in } \Delta_e^\pi} \\
\\
\frac{q@p \rightarrow q' \text{ in } \Delta^\pi \quad r@s \rightarrow r' \text{ in } \Delta_e \quad r' \neq \text{sel} \quad q \notin P}{(q, r)@(p, s) \rightarrow (q', r') \text{ in } \Delta_e^\pi} \quad \frac{q@p \rightarrow q' \text{ in } \Delta^\pi \quad r@s \rightarrow \text{sel} \text{ in } \Delta_e}{(q, r)@(p, s) \rightarrow \text{sel} \text{ in } \Delta_e^\pi} \\
\\
\frac{q@p \rightarrow q' \text{ in } \Delta^\pi \quad r@s \rightarrow r' \text{ in } \Delta_e \quad q \in P \quad r' \neq \text{sel}}{(q, r)@(p, s) \rightarrow (q', r) \text{ in } \Delta_e^\pi} \\
\\
\frac{q \xrightarrow{\diamond} q' \text{ in } \Delta^\pi \quad r \xrightarrow{\diamond} r' \text{ in } \Delta_e \quad q \notin P \quad r' \neq \text{sel}}{(q, r) \xrightarrow{\diamond} (q', r') \text{ in } \Delta_e^\pi} \quad \frac{q \xrightarrow{\diamond} q' \text{ in } \Delta^\pi \quad r \xrightarrow{\diamond} \text{sel} \text{ in } \Delta_e}{(q, r) \xrightarrow{\diamond} \text{sel} \text{ in } \Delta_e^\pi} \\
\\
\frac{q \xrightarrow{\diamond} q' \text{ in } \Delta^\pi \quad r \xrightarrow{\diamond} r' \text{ in } \Delta_e \quad q \in P \quad r' \neq \text{sel}}{(q, r) \xrightarrow{\diamond} (q', r) \text{ in } \Delta_e^\pi} \\
\\
\frac{a \in \Sigma}{\text{sel} \xrightarrow{a} \text{sel} \text{ in } \Delta_e^\pi} \quad \frac{\mu \in Q_e^\pi}{\text{sel}@\mu \rightarrow \text{sel} \text{ in } \Delta_e^\pi} \quad \frac{\mu \in Q_e^\pi}{\mu@\text{sel} \rightarrow \text{sel} \text{ in } \Delta_e^\pi} \quad \frac{\text{true}}{\text{sel} \xrightarrow{\diamond} \text{sel} \text{ in } \Delta_e^\pi}
\end{array}$$

Figure 20. The transition rules Δ_e^π inferred from those of the SHA^\downarrow s A^π with projection states P and A_e with selection state sel .

8.3. Soundness and Completeness

We next show that the soundness and completeness of a projection algorithm are preserved when combined with some earliest membership tester when assuming determinism.

Proposition 54. *Let A^π and A_e be $d\text{SHA}^\downarrow$ s with the same schema \mathbf{S} and the same language up to the schema, i.e. $\mathcal{L}(A^\pi) \cap \mathcal{L}(\mathbf{S}) = \mathcal{L}(A_e) \cap \mathcal{L}(\mathbf{S})$. The combination $d\text{SHA}^\downarrow A_e^\pi$ then has the same language up to schema \mathbf{S} too. Furthermore, if A_e is earliest for \mathbf{S} then A_e^π is earliest for \mathbf{S} too and goes into some subhedge projection state whenever A^π does.*

Proof. If q is a subhedge projection state of A^π and r a state of A_e then (q, r) is a subhedge projection state of A_e^π . The evaluator for automaton A_e^π runs A^π and A_e in parallel on the input hedge, while skipping subhedges starting in subhedge projection states of A^π . These include the subhedges starting in a projection state when evaluating A_e^π on h . By Proposition 20 applied to A^π such subhedges are irrelevant for $\mathcal{L}(A^\pi)$ with respect to \mathbf{S} since A^π is assumed to be deterministic. Since $\mathcal{L}(A^\pi) \cap \mathbf{S} = \mathcal{L}(A_e) \cap \mathbf{S}$, skipping such subhedge does not affect acceptance by A_e for hedges inside schema \mathbf{S} . Therefore the evaluator of A_e^π is earliest with respect to \mathbf{S} too. \square

The above proposition shows that if A^π is complete for subhedge projection then A_e^π is also complete for subhedge projection while being earliest in addition.

Theorem 5. *For any complete $d\text{SHAs}$ A with schema \mathbf{S} , the $d\text{SHA} A^{\text{cgr}}(\mathbf{S})_{e(\mathbf{S})}$ is earliest and sound complete for subhedge projection for schema \mathbf{S} .*

Proof. The congruence projection $A^{\text{cgr}}(\mathbf{S})$ is sound by Proposition 2 and complete for subhedge projection wrt \mathbf{S} by Theorem 3. The automaton $A_{e(\mathbf{S})}$ discussed in Section 8.1 is

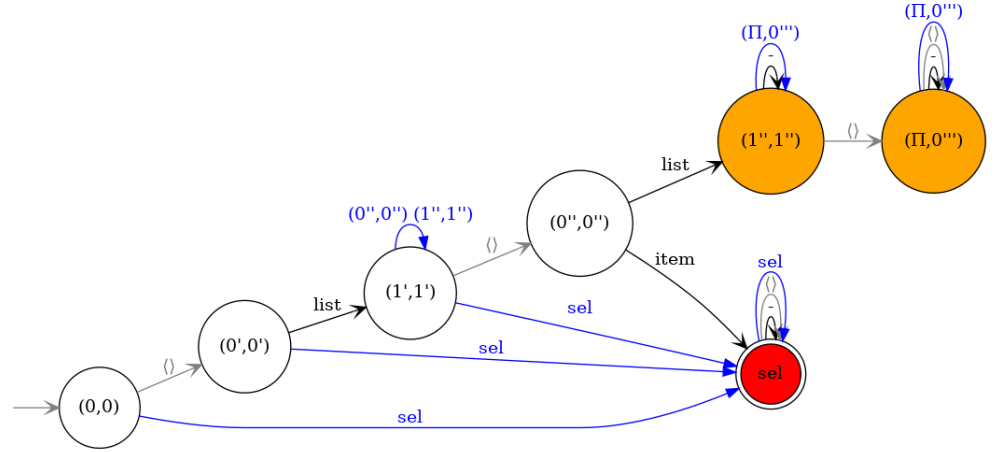


Figure 21. The earliest $\text{dSHA}^\downarrow A_{e(\mathbf{S})}^{\text{SNC}}$ with safe-no-change subhedge projection for the dSHA A in Fig. 4 with schema $\mathbf{S} = \llbracket \text{doc} \rrbracket$ for the XPath filter $[\text{self}::\text{list}/\text{child}::\text{item}]$.

earliest for \mathbf{S} . So their combination $A^{\text{cgr}}(\mathbf{S})_{e(\mathbf{S})}$ yields an earliest automaton with complete subhedge projection wrt \mathbf{S} by Proposition 54. \square

8.4. In-Memory Complexity

We next discuss the complexity of earliest membership testing with complete subhedge projection by an in-memory evaluator for $A^{\text{cgr}}(\mathbf{S})_{e(\mathbf{S})}$.

Lemma 55. *The number of states in $\mathcal{Q}^{\text{cgr}}(\mathbf{S})_{e(\mathbf{S})}$ is in $O(2^{n^2+2n+\log(n)})$ where $n = |\mathcal{Q}|$.*

Proof. By Lemma 50 the number of states of $A^{\text{cgr}}(\mathbf{S})$ is in $O(2^{n^2+n})$ where $n = |\mathcal{Q}|$. The number of states of $A_{e(\mathbf{S})}$ is in $O(n2^n)$ and thus in $O(2^{n+\log(n)})$. The number of states of $\mathcal{Q}^{\text{cgr}}(\mathbf{S})_{e(\mathbf{S})}$ is thus in $O(2^{n^2+n} 2^{n+\log(n)})$ which is in $O(2^{n^2+2n+\log(n)})$ \square

Corollary 6. *Let the signature Σ be fixed. For any complete dSHA A with schema $\mathbf{S} \subseteq \mathcal{H}_\Sigma$, earliest membership on a input hedge $h \in \mathbf{S}$ can be tested in-memory in time $O(|1|)$ per nonprojected node of h after a preprocessing time of $O(|A^{\text{cgr}}(\mathbf{S})_{e(\mathbf{S})}| + n^3d + ns)$ where d is the number of difference relations and s the number of subset of safe states in $\mathcal{Q}^{\text{cgr}}(\mathbf{S})_{e(\mathbf{S})}$. The preprocessing time is also in $O(2^{2n^2+4n+2\log(n)})$ where n is the number of states of A .*

Proof. Since A is complete, Theorem 5 shows that $A^{\text{cgr}}(\mathbf{S})_{e(\mathbf{S})}$ is earliest and sound and complete for subhedge projection for schema \mathbf{S} . The in-memory evaluator of $A^{\text{cgr}}(\mathbf{S})_{e(\mathbf{S})}$ can be run on any input hedge $h \in \mathbf{S}$ in time $O(|1|)$ per nonprojected node of h after a preprocessing time of $O(|A^{\text{cgr}}(\mathbf{S})_{e(\mathbf{S})}| + n^3d + ns)$. Since Σ is fixed the size $|A^{\text{cgr}}(\mathbf{S})_{e(\mathbf{S})}|$ bounded by $O(|\mathcal{Q}^{\text{cgr}}(\mathbf{S})_{e(\mathbf{S})}|^2)$. Lemma 55 shows that $|\mathcal{Q}^{\text{cgr}}(\mathbf{S})_{e(\mathbf{S})}|$ is in $O(2^{n^2+2n+\log(n)})$, so $|A^{\text{cgr}}(\mathbf{S})_{e(\mathbf{S})}|$ is in $O(2^{2n^2+4n+2\log(n)})$. \square

As before, instead of precomputing $A^{\text{cgr}}(\mathbf{S})_{e(\mathbf{S})}$ from the input dSHA A and $F_{\mathbf{S}}$ we can compute only the part needed for evaluating the input hedge h on-the-fly. If the needed part of $A^{\text{cgr}}(\mathbf{S})_{e(\mathbf{S})}$ for evaluating h is small, the overall time and space go down considerably.

9. Streaming Algorithms

We show that dSHA^\downarrow s can be evaluated on the nested word of a hedge in a streaming manner, so that they produce the same results and projection behavior as by evaluating them on hedges in-memory in a top-down and left-to-right manner.

All aspects related to subhedge projection remain unchanged. Earliest query answering in streaming mode means that nested word suffixes are ignored if irrelevant – and not only nested words as with subhedge projection. What changes between both evaluation modes is memory consumption. For in-memory evaluation, the whole graph of the input hedge must be stored, while in streaming mode, only a stack and a state are maintained at any event. For testing regular language membership, the state space is finite, while the stack space become infinite, since the depth of the stack depends on the depth of the input hedge.

The property of having equivalent streaming and in-memory evaluators was already noticed in [26] for Neumann and Seidl's pushdown forest automata [25] and for nested word automata [11,15]. We here show how to transpose this result to dSHA^\downarrow s. This permits us to show the correctness of our evaluation algorithms based on properties of the constructed automata. It also permits obtaining correctness properties of streaming algorithms by reasoning about the in-memory evaluator of SHA^\downarrow s.

9.1. Streaming Evaluators for Downward Hedge Automata

We define the streaming evaluator of a $\text{SHA}^\downarrow A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ by a visibly pushdown machine. The set of configurations of this machine is $\mathcal{K} = \mathcal{Q} \times \mathcal{Q}^*$ containing pairs of states and stacks of states. The visibly pushdown machine provides for any word $v \in \hat{\Sigma}^*$ a streaming transition relation $\xrightarrow{v}^{str} \subseteq \mathcal{K} \times \mathcal{K}$ such that for all states $q, q' \in \mathcal{Q}$ and stacks $\sigma \in \mathcal{Q}^*$ and words $v, v' \in \hat{\Sigma}^*$:

$$\frac{\text{true}}{(q, \sigma) \xrightarrow{\varepsilon}^{str} (q, \sigma) \text{ wrt } \Delta} \quad \frac{(q, \sigma) \xrightarrow{v}^{str} (q', \sigma) \quad (q', \sigma) \xrightarrow{v'}^{str} (q'', \sigma) \text{ wrt } \Delta}{(q, \sigma) \xrightarrow{v \cdot v'}^{str} (q'', \sigma) \text{ wrt } \Delta}$$

$$\frac{q \xrightarrow{a} q' \text{ in } \Delta}{q \xrightarrow{a}^{str} q' \text{ wrt } \Delta} \quad \frac{q \xrightarrow{\langle \rangle} q' \text{ in } \Delta}{(q, \sigma) \xrightarrow{\langle \rangle}^{str} (q', \sigma \cdot q) \text{ wrt } \Delta} \quad \frac{q @ p \rightarrow q' \text{ in } \Delta}{(p, \sigma \cdot q) \xrightarrow{\rangle}^{str} (q', \sigma) \text{ wrt } \Delta}$$

We note that the same visibly pushdown machine can be obtained by compiling the SHA^\downarrow to an NWA, whose streaming evaluator is defined by a visibly pushdown machine too (see Section 12.3 on related work).

The notion of partial runs of the in-memory evaluator is closely related to streaming runs. To see this, we first note that each partial run v wrt Δ naturally defines a stack as follows. Since v is a nested word prefix, there exist unique nested words $v_0, \dots, v_n \in \text{nw}(\mathcal{H}_{\Sigma \cup \mathcal{Q}})$ such that:

$$v = v_0 \cdot \langle \cdot v_1 \cdot \langle \dots \langle \cdot v_n$$

These nested words must be partial runs of Δ too, so there exist states $p_0, \dots, p_n, q_0, \dots, q_n \in \mathcal{Q}$ such that v_i goes from p_i to q_i for all $0 \leq i \leq n$. We define the stack σ of v by $\sigma = q_0 \cdot \dots \cdot q_{n-1}$ and say that v goes from p_0 to q_n and has stack σ . Note that the stack is empty if v is a nested word, since in this case $n = 0$ so that $v = v_0$ and $\sigma = \varepsilon$. For instance, the partial run $v_0 = 0 \cdot \langle \cdot 0 \cdot \text{list} \cdot 1 \cdot \langle \cdot 0 \cdot \text{item} \cdot 2$ has the stack $\sigma_0 = 0 \cdot 1$ while the partial run $v_1 = v_0 \cdot \rangle \cdot 3 \cdot \rangle \cdot 4$ has stack $\sigma_1 = \varepsilon$.

Lemma 56. *Consider a $\text{SHA} A = (\Sigma, \mathcal{Q}, \Delta, I, F)$, states $q, q' \in \mathcal{Q}$, and a stack $\sigma \in \mathcal{Q}^*$. Let v be a prefix of some nested word in $\text{nw}(\mathcal{H}_\Sigma)$. Then there exists a partial run r wrt Δ from q to q' with stack σ such that $\text{proj}_\Sigma(r) = v$ iff $(q, \varepsilon) \xrightarrow{v}^{str} (q', \sigma) \text{ wrt } \Delta$.*

Proof. By induction on the length of prefix v . \square

This lemma implies that any in-memory transition of A on a hedge h from q to q' can be identified with some streaming transition of A on $nw(h)$ from (q, ε) to (q', ε) .

Proposition 57. $q \xrightarrow{h} q' \text{ wrt } \Delta \Leftrightarrow (q, \varepsilon) \xrightarrow{nw(h) \text{ str}} (q', \varepsilon) \text{ wrt } \Delta$.

Proof. If $q \xrightarrow{h} q' \text{ wrt } \Delta$ there exists a run R of h with Δ that starts with q and ends with q' . Consider the partial run $v = nw(R)$ on h . Since v is a nested word, its stack is empty. By Lemma 56, we have $(q, \varepsilon) \xrightarrow{proj_{\Sigma}(v) \text{ str}} (q', \varepsilon) \text{ wrt } \Delta$ and $proj_{\Sigma}(v) = nw(h)$. For the inverse implication, assume that $(q, \varepsilon) \xrightarrow{nw(h) \text{ str}} (q', \varepsilon) \text{ wrt } \Delta$. Let $u = nw(h)$. By Lemma 56 there exists a partial run v of Δ from q to q' with stack ε . Then there exists a run R of Δ on h such that $v = nw(R)$. Hence, $q \xrightarrow{h} q' \text{ wrt } \Delta$. \square

For any deterministic $d\text{SHA}^{\downarrow} A$, hedge h , and state q , the streaming transition on $nw(h)$ with respect to Δ starting with (q, ε) can be computed in time $O(1)$ per letter after a precomputation in time $O(|A|)$. So the overall computation time is in $O(|A| + |h|)$. The streaming memory needed to store a configuration is of size $O(\text{depth}(h) + |A|)$ since the size of the visible stack is bounded by the depth of the input hedge.

9.2. Adding Subhedge Projection

We next extend the streaming transition relation with subhedge projection, in analogy to what we did for the in-memory transition relation. We define a transition relation with subhedge projection $\xrightarrow[h \text{ shp}]{h \text{ str}} \subseteq \mathcal{K} \times \mathcal{K}$ with respect to Δ such that for all words $v, v' \in \hat{\Sigma}^*$, letters $a \in \Sigma$, states $p, q, q', q'' \in \mathcal{Q}$ and stacks $\sigma, \sigma', \sigma'' \in \mathcal{Q}^*$:

$$\frac{q \in \mathcal{Q}_{shp}^{\Delta} \quad v \in nw(\mathcal{H}_{\Sigma})}{(q, \sigma) \xrightarrow[h \text{ shp}]{v \text{ str}} (q, \sigma) \text{ wrt } \Delta}$$

$$\frac{q \notin \mathcal{Q}_{shp}^{\Delta} \quad q \xrightarrow{a} q' \text{ in } \Delta}{(q, \sigma) \xrightarrow[h \text{ shp}]{a \text{ str}} (q', \sigma) \text{ wrt } \Delta} \quad \frac{q \notin \mathcal{Q}_{shp}^{\Delta}}{(q, \sigma) \xrightarrow[h \text{ shp}]{\varepsilon \text{ str}} (q, \sigma) \text{ wrt } \Delta}$$

$$\frac{q \notin \mathcal{Q}_{shp}^{\Delta} \quad (q, \sigma) \xrightarrow[h \text{ shp}]{v \text{ str}} (q', \sigma') \text{ wrt } \Delta \quad (q', \sigma') \xrightarrow[h \text{ shp}]{v' \text{ str}} (q'', \sigma'') \text{ wrt } \Delta}{(q, \sigma) \xrightarrow[h \text{ shp}]{v \cdot v' \text{ str}} (q'', \sigma'') \text{ wrt } \Delta}$$

$$\frac{q \notin \mathcal{Q}_{shp}^{\Delta} \quad q \xrightarrow{\langle \rangle} q' \text{ in } \Delta}{(q, \sigma) \xrightarrow[h \text{ shp}]{\langle \rangle \text{ str}} (q', \sigma \cdot q) \text{ wrt } \Delta} \quad \frac{p \notin \mathcal{Q}_{shp}^{\Delta} \quad q @ p \rightarrow q' \text{ in } \Delta}{(p, \sigma \cdot q) \xrightarrow[h \text{ shp}]{\rangle \text{ str}} (q', \sigma) \text{ wrt } \Delta}$$

The projecting transition relation stays in a configuration with a subhedge projection state until the end of the current subhedge is reached. This is correct since a subhedge projection state cannot be changed anyway.

Proposition 58. For any $\text{SHA}^{\downarrow} A = (\Sigma, \mathcal{Q}, \Delta, I, F)$, states $q, q' \in \mathcal{Q}$, nested word v on which Δ has no blocking partial run starting from q :

$$(q, \varepsilon) \xrightarrow{v \text{ str}} (q', \varepsilon) \text{ wrt } \Delta \Leftrightarrow (q, \varepsilon) \xrightarrow[h \text{ shp}]{v \text{ str}} (q', \varepsilon) \text{ wrt } \Delta$$

Proof. Let $h \in \mathcal{H}_\Sigma$ be the hedge such that $v = nw(h)$. The proof is by induction on the size of v . For the implication from the left to the right we assume that $(q, \varepsilon) \xrightarrow{v \text{ str}} (q', \varepsilon)$ wrt Δ . Since $v = nw(h)$, Proposition 57 proves $q \xrightarrow{h \text{ str}} q'$ wrt Δ .

Case $q \in \mathcal{Q}_{shp}^\Delta$. By Lemma 19, it follows from $q \xrightarrow{h \text{ str}} q'$ wrt Δ that $q = q'$. We can then apply the following rule since v is a nested word:

$$\frac{q \in \mathcal{Q}_{shp}^\Delta \quad v = nw(\mathcal{H}_\Sigma)}{(q, \varepsilon) \xrightarrow[shp]{v \text{ str}} (q, \varepsilon) \text{ wrt } \Delta}$$

Hence $(q, \varepsilon) \xrightarrow[shp]{v \text{ str}} (q, \varepsilon)$ wrt Δ .

Case $q \notin \mathcal{Q}_{shp}^\Delta$. We distinguish all possible forms of hedge h .

Subcase $h = h' \cdot h''$. Let $v' = nw(h')$ and $v'' = nw(h'')$ so $v = v' \cdot v'' = nw(h)$. The following rule must have been applied:

$$\frac{(q, \varepsilon) \xrightarrow{v' \text{ str}} (p, \sigma) \text{ wrt } \Delta \quad (p, \sigma) \xrightarrow{v'' \text{ str}} (q', \varepsilon) \text{ wrt } \Delta}{(q, \varepsilon) \xrightarrow{v' \cdot v'' \text{ str}} (q', \varepsilon) \text{ wrt } \Delta}$$

Since v' is a nested word, it follows that $\sigma = \varepsilon$. By induction hypothesis applied to the nested words v' and v'' we get $(q, \varepsilon) \xrightarrow[shp]{v' \text{ str}} (p, \varepsilon)$ wrt Δ and $(p, \varepsilon) \xrightarrow[shp]{v'' \text{ str}} (q', \varepsilon)$ wrt Δ .

Hence, we can apply the following rule:

$$\frac{q \notin \mathcal{Q}_{shp}^\Delta \quad (q, \varepsilon) \xrightarrow[shp]{v' \text{ str}} (p, \varepsilon) \text{ wrt } \Delta \quad (p, \varepsilon) \xrightarrow[shp]{v'' \text{ str}} (q', \varepsilon) \text{ wrt } \Delta}{(q, \varepsilon) \xrightarrow[shp]{v' \cdot v'' \text{ str}} (q', \varepsilon) \text{ wrt } \Delta}$$

This shows $(q, \varepsilon) \xrightarrow[shp]{v \text{ str}} (q', \varepsilon)$ as required.

Subcases $h = \langle h' \rangle$ or $h = a$ or $h = \varepsilon$. Straightforward.

For the implication from the right to the left we assume that $(q, \varepsilon) \xrightarrow[shp]{v \text{ str}} (q', \varepsilon)$ wrt Δ .

Case $q \in \mathcal{Q}_{shp}^\Delta$. Then the following rule must have been applied with $q = q'$:

$$\frac{q \in \mathcal{Q}_{shp}^\Delta \quad v = nw(\mathcal{H}_\Sigma)}{(q, \varepsilon) \xrightarrow[shp]{v \text{ str}} (q, \varepsilon) \text{ wrt } \Delta}$$

Since we assume that v does not have any blocking partial runs with Δ , there exists a partial run on v with Δ that starts with q . Since v is a nested word, any partial run on v is the nested word of some run, so there exists a run on the hedge h that starts with q . This shows the existence of some state p such that $q \xrightarrow{h} p$ wrt Δ . Since $q \in \mathcal{Q}_{shp}^\Delta$, Lemma 19 shows $q = p$ and thus $q \xrightarrow{h} q$ wrt Δ . Proposition 57 then proves that $(q, \varepsilon) \xrightarrow[nw(h) \text{ str}]{v \text{ str}} (q, \varepsilon)$ wrt Δ , which is equal to $(q, \varepsilon) \xrightarrow[shp]{v \text{ str}} (q', \varepsilon)$ wrt Δ as required.

Case $q \notin \mathcal{Q}_{shp}^\Delta$. We distinguish all possible forms of the hedge h .

Subcase $h = h' \cdot h''$. Let $v' = nw(h')$ and $v'' = nw(h'')$ so $v = v' \cdot v'' = nw(h)$. The following rule must have been applied:

$$\frac{q \notin \mathcal{Q}_{shp}^\Delta \quad (q, \varepsilon) \xrightarrow[shp]{v'} (p, \sigma) \text{ wrt } \Delta \quad (p, \sigma) \xrightarrow[shp]{v''} (q', \varepsilon) \text{ wrt } \Delta}{(q, \varepsilon) \xrightarrow[shp]{v' \cdot v''} (q', \varepsilon) \text{ wrt } \Delta}$$

Since v' is a nested word, it follows that $\sigma = \varepsilon$ too. By induction hypothesis applied to the nested words v' and v'' we get $(q, \varepsilon) \xrightarrow{v'} (p, \varepsilon) \text{ wrt } \Delta$ and $(p, \varepsilon) \xrightarrow{v''} (q', \varepsilon) \text{ wrt } \Delta$. Hence, we can apply the following rule:

$$\frac{(q, \varepsilon) \xrightarrow{v'} (p, \varepsilon) \text{ wrt } \Delta \quad (p, \varepsilon) \xrightarrow{v''} (q', \varepsilon) \text{ wrt } \Delta}{(q, \varepsilon) \xrightarrow{v' \cdot v''} (q', \varepsilon) \text{ wrt } \Delta}$$

This shows $(q, \varepsilon) \xrightarrow{v} (q', \varepsilon)$ as required.

Subcases $h = \langle h' \rangle$ or $h = a$ or $h = \varepsilon$. Straightforward.

This ends the proofs of both directions. \square

A streaming evaluator with subhedge projection for deterministic SHA^\downarrow s A on hedges h can thus be obtained by computing the streaming transition relation with subhedge projection for A of $nw(h)$ starting with the initial configuration. This costs time at most $O(1)$ per letter of $nw(h)$, i.e. constant time per event of the stream.

9.3. Streaming Complexity of Earliest Membership with Subhedge Projection

By using streaming evaluators for SHA^\downarrow s we can test membership earliest and with subhedge projection in streaming mode by running the $\text{SHA}^\downarrow A^{cgr}(\mathbf{S})_{e(\mathbf{S})}$.

Corollary 7. *For any complete dSHAs $(\Sigma, \mathcal{Q}, \Delta, I, F)$, $F \subseteq F_S \subseteq \mathcal{Q}$, language $L = \mathcal{L}(A)$, and schema $\mathbf{S} = \mathcal{L}(A[F/F_S])$, earliest membership with complete subhedge projection for L wrt \mathbf{S} can be tested in streaming mode for any hedge $h \in \mathbf{S}$ in time $O(|1|)$ per nonprojected letter of $nw(h)$ after a preprocessing time of $O(|A^{cgr}(\mathbf{S})_{e(\mathbf{S})}| + n^3 d + n s)$, where d is the number of difference relations and s the number of safe subsets in states of $\mathcal{Q}_{e(\mathbf{S})}^{cgr}(\mathbf{S})$. The space required is in $O(\text{depth}(h) + |A^{cgr}(\mathbf{S})_{e(\mathbf{S})}|)$.*

Proof. By Theorem 5 the $\text{SHA}^\downarrow A^{cgr}(\mathbf{S})_{e(\mathbf{S})}$ is earliest and sound and complete for subhedge projection. By Propositions 57 and 58 we can thus obtain a streaming membership tester with subhedge projection for a hedge $h \in \mathbf{S}$ by running the streaming evaluator with subhedge projection of $A^{cgr}(\mathbf{S})_{e(\mathbf{S})}$ on the nested word $nw(h)$. Since this SHA^\downarrow is deterministic, the streaming evaluation can be done in time $O(|1|)$ per letter of $nw(h)$ after a preprocessing time of $O(|A^{cgr}(\mathbf{S})_{e(\mathbf{S})}| + n^3 d + n s)$. \square

As for safe-no-change projection, the exponential preprocessing time can again be avoided by creating the part of the $\text{SHA}^\downarrow A^{cgr}(\mathbf{S})_{e(\mathbf{S})}$ needed for the evaluation of the hedge on the fly. The size of the needed part is in $O(|h|)$. Hence, the space requirements can also be reduced to $O(|h|)$ which may be too much for large streams $nw(h)$. The needed part of $A^{cgr}(\mathbf{S})_{e(\mathbf{S})}$, however, may be much smaller than $|h|$. Furthermore, a stack of size $\text{depth}(h)$ has to be maintained, but this dependence on h may be acceptable even for large streams $nw(h)$.

10. Monadic Regular Queries and XPath

We next consider the problem of how to answer monadic regular queries on hedges in an earliest manner and with subhedge projection. Each monadic query defines a set of positions for each input hedge. The XPath queries `self::list[child::item]`, for instance can be applied to the hedge $\langle \text{list}.\langle \text{list}.\langle \text{item} \rangle.\langle \text{list}.\langle \text{item} \rangle \rangle \rangle$ where it selects the position of the first top-most tree labeled by `list`, since it contains a child tree labeled by `item`. We identify positions of hedges by integers. For instance, the selected `list` element in the hedge above is identified by the integer 2. Therefore, the answer set of the above query on the above hedge is $\{2\}$. Note that monadic queries do not return any other information about the selected positions, in contrast to the subtree semantics of XPath in the official W3C standard.

Answering this query is more complicated than verifying whether the XPath filter `[self::list/child::item]` matches at the root of the first subtree of the hedge, since the answer set needs to be constructed too. Concerning memory consumption, alive answer candidates need to be buffered for monadic query answering. Therefore the memory consumption of streaming evaluators of monadic queries may grow linearly in the size of the stream, if the number of the alive candidates does.

10.1. Monadic Regular Queries

Monadic queries on hedges in \mathcal{H}_Σ can be identified with languages of annotated hedges in $\mathcal{H}_{\Sigma \cup \{x\}}$ where $x \notin \Sigma$ in an arbitrarily fixed selection variable. A monadic query on hedges in \mathcal{H}_Σ can then be identified with the language of all annotated hedges in $\mathcal{H}_{\Sigma \cup \{x\}}$, in which a single position got x -annotated and this position is selected by query.

Therefore, regular monadic queries on hedges in \mathcal{H}_Σ can be defined by nested regular expressions in $nRegExp_{\Sigma \cup \{x\}}$ or by a dSHA with the same signature. The regular XPath query `self::list[child::item]`, for instance, can be defined by following nested regular expression:

$$\langle \text{list} \cdot x \cdot \top \cdot \langle \cdot \text{item} \cdot \top \cdot \rangle \cdot \top \rangle \cdot \top$$

where \top is like a wildcard accepting any hedge in \mathcal{H}_Σ without x , i.e., for some recursion variable $z \in V$:

$$\top = \mu z. (+_{a \in \Sigma} a + \langle z \rangle)^*$$

Note that the nested regular expression for the XPath filter `[self::list/child::item]` in $nRegExp_\Sigma$ can be obtained from that of the XPath query `self::list[child::item]` by removing the x . In addition, for any monadic query on hedges, we have to restrict the annotated hedges to be recognized by the language of the query such that exactly one position in each hedge is annotated by x . This can be done by intersection with the following nested regular expression one_x :

$$\mu z. (T.x.T + T.\langle z \rangle.T)$$

The previous schema $\llbracket \text{doc} \rrbracket$ for pattern matching has now to be changed from to $\mathbf{S} = \llbracket \text{doc}_x \rrbracket \cap \llbracket one_x \rrbracket$ where $\text{doc}_x \in nRegExp_{\Sigma \cup \{x\}}$ is like `doc` but over the signature $\Sigma \cup \{x\}$. For the regular XPath query `self::list[child::item]` we obtain the dSHA in Fig. 22. The transition rules with the special letter x indicate, where positions may be bound to the selection variable: x can be bound only in state 2, that is in subtrees starting with a `list` element. The same automaton can be used to define the schema $\mathbf{S} = \llbracket \text{doc}_x \rrbracket \cap \llbracket one_x \rrbracket$ by using the schema-final states $F_S = \{14, 20\}$ instead of the final states $F = \{20\}$. Indeed, we constructed the automaton by intersecting the automaton obtained from the XPath expression with the automata computed from the regular expressions `docx` and `onex` for the schema. Note that 20 is a final state for the query (and the schema), while 14 is a sink state for the query (but not for the schema).

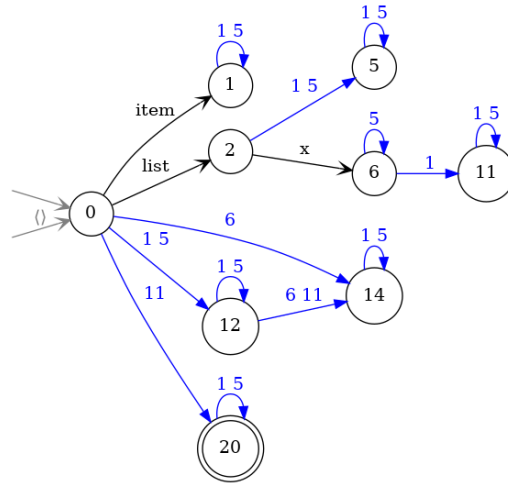


Figure 22. The dSHA for the XPath query `self::list[child::item]`. The selection position is indicated by x .

10.2. Earliest Query Answering with Subhedge Projection

A naive query evaluation algorithm for a hedge automaton with signature $\Sigma \cup \{x\}$ receives an input hedge $h \in \mathcal{H}_\Sigma$, guesses all possible positions of where to insert x into h , and tests for all of them whether the annotated hedge obtained is accepted by A . In a less naive manner, one inserts x only at those position where the current state has an outgoing x -transition. All this can be improved if the hedge automaton under consideration supports selection, rejection, or projection states. In all cases, the algorithm has to buffer all binding candidates, that are not yet in a selection or rejection state. In the worst case it has to buffer the bindings to all positions. Therefore, the state space of the evaluator of monadic queries is no more finite.

Selection and rejection states can be identified by adapting the earliest query answering algorithm from [21]. This can be made such that one obtains an $\text{SHA}^\downarrow A_{e(\mathbf{S})}$ with signature $\Sigma \cup \{x\}$ depending on the schema. It should be noticed that automata used in [21] are dNWA's instead of SHA^\downarrow s, and that schemas are ignored there. The needed adaptations are therefore discussed in Appendix N. How to obtain an efficient query answering algorithm in streaming mode from $A_{e(\mathbf{S})}$ is the main topic of [21]. This is the base algorithm that we will extend with subhedge projection for our experiments.

In order to add congruence projection to the earliest query answering algorithm, we have to run the automaton $A_{e(\mathbf{S})}^{\text{cgr}}$. The earliest congruence projection for the XPath query `self::list[child::item]` is shown in Fig. 23. It binds x in state 2D1 after having read a top-level `list`-element and then checks for the existence of a child element `item`, going into selection state 1D3 once it is found. All other children of the top-level `list` element are projected in state 2D3. For adding earliest query answering to self-no-change projection, we have to run the automaton $A_{e(\mathbf{S})}^{\pi}$ instead of $A_{e(\mathbf{S})}^{\text{cgr}}$. We don't give the details, but claim for the present query that the projection power of both automaton is the same.

Fortunately, the soundness and completeness theorems for earliest membership testing with subhedge projection do only affect the constructions of these SHA^\downarrow s, but not how they are to be evaluated. However, there is one additional issue: Nested word prefixes cannot be considered as irrelevant for the subsequent subhedge, if the selection variable x can be bound there, even if the acceptance doesn't depend on where it will be bound. This is since the position of the binding is to be returned by any query answering algorithm.

Definition 59. We call a prefix v binding irrelevant for L and \mathbf{S} if there does not exist any hedge h containing x and suffix w such that $v \cdot w \in \text{nw}(\mathbf{S})$ and $v \cdot \text{nw}(h) \cdot w \in \text{nw}(L)$.

Our subhedge projecting evaluator can project only at subhedge irrelevant prefixes that are also binding irrelevant. Given an dSHA A and a set of schema final-states $F_{\mathbf{S}}$, one can decide whether a prefix is subhedge irrelevant if the current state q does accept

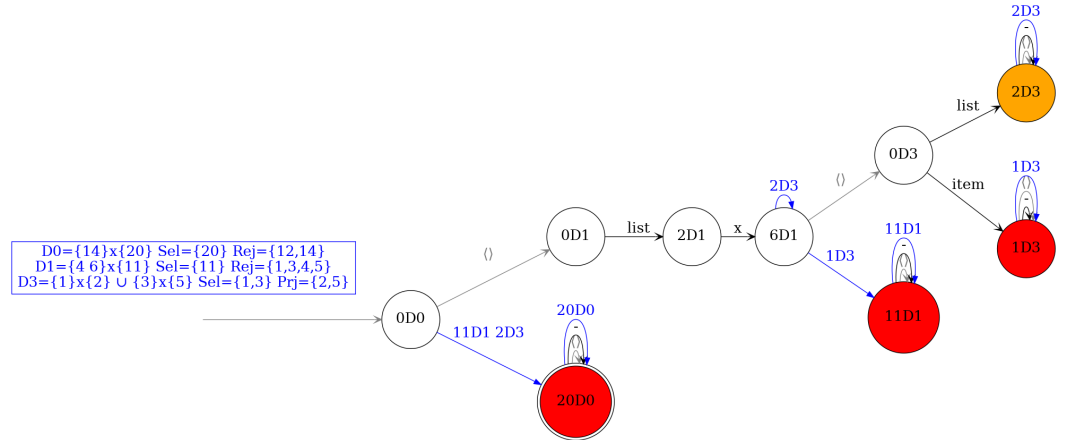


Figure 23. The earliest congruence projection $A^{cgr}(\llbracket \text{doc} \rrbracket \cap \llbracket \text{one}_x \rrbracket)_{e(\llbracket \text{doc} \rrbracket \cap \llbracket \text{one}_x \rrbracket)}$ for the dSHA A in Fig. 22 with $F_S = \{14, 20\}$. It defines the query of the XPath `self::list[child::item]`.

hedges containing x but can access some other state that is not safe for rejection and permits x -bindings. In ground Datalog, we can distinguish binding irrelevant states q by imposing the following rules for all $q, q' \in Q$:

`binding_irrelev(q) :- state(q), not binding_relev(q).`

`binding_relev(q) :- not bind_x(q), acc(q, q'), bind_x(q'), not rej(q').`

Note in our example of the XPath query `self::list[child::item]` that state 2D3 of the $\text{dSHA} \downarrow A^{cgr}(\llbracket \text{doc} \rrbracket \cap \llbracket \text{one}_x \rrbracket)_{e(\llbracket \text{doc} \rrbracket \cap \llbracket \text{one}_x \rrbracket)}$ in Fig. 23 is binding irrelevant, since x must be bound on the top-level `list` element in state 2D1, so it cannot be bound on any `list` element below.

11. Experiments with Streaming XPath Evaluation

We compare experimentally three streaming evaluators for dSHA, for earliest monadic query answering, without subhedge projection, with safe-no-change projection, and with congruence projection.

11.1. Benchmark dSHA for XPath Queries

We start from dSHA defining monadic queries for the regular XPath queries A1-A8 from the XPathMark benchmark [7] that are given in Table 24. These XPath queries show most of the features of the regular fragment of XPath. In Table 25, we added 14 further XPath path queries that we found useful for testing too.

Deterministic SHAs for A1-A8 were provided earlier in [21]. For the other XPath queries we compiled them to dSHAs via nested regular expression.

In order to produce the input dSHAs for our evaluators, we intersect these automata with a dSHA for the schema $\llbracket \text{doc}'_x \rrbracket \cap \llbracket \text{one}_x \rrbracket$ where doc'_x is the schema we use for hedge encodings of real-world XML documents with x annotations. Thereby we could identify the schema final states F_S . We minimized and completed the result. Fig. 26 reports the size of the input dSHAs for our evaluators obtained by the above procedure. For each dSHA A , the size is given by two numbers $\text{size}(n)$, the first for the overall size and the second for the number of states n .

For the input dSHA A of each query, we statically computed the whole $\text{dSHA} \downarrow_s A^{cgr}(\mathbf{S})_{e(\mathbf{S})}$ while using the necessary parts of the determinization algorithm from [16]. The size of the $\text{dSHA} \downarrow_s$ obtained and the number d of difference relations are reported in Fig. 26 too. The biggest size is 2091(504) for the input dSHA for A8. The largest number of difference relations $d=24$ is also obtained for this query. So, indeed the size of these automata is much smaller than one might expect from a construction that is

Query ID	XPath expression
A1	/site/closed_auctions/closed_auction/annotation/description/text/keyword
A2	//closed_auction//keyword
A3	/site/closed_auctions/closed_auction//keyword
A4	/site/closed_auctions/closed_auction[annotation/description/text/keyword]/date
A5	/site/closed_auctions/closed_auction[descendant::keyword]/date
A6	/site/people/person[profile/gender and profile/age]/name
A7	/site/people/person[phone or homepage]/name
A8	/site/people/person[address and (phone or homepage) and (creditcard or profile)]/name

Figure 24. Regular XPath queries A1-A8 from XPathMark collection.

A0	/site
A1_0a	/site/*
A1_0b	/site/@*
A1_0c	/site//@*
A1_1a	//bidder/personref[starts-with(@person, 'person0')]
A1_1d	//bidder/personref[@person='person0']
A1_2	//person
A1_3	/site/regions/africa/@*
A1_4	/site/regions/africa/*
A1_5	/site/regions/*
A1_6	//closed_auction/annotation//keyword
A2_1	//closed_auction[descendant::keyword]
A4_0	/site/closed_auctions/closed_auction[annotation]/date
A4_1	/site[open_auctions]/closed_auctions

Figure 25. Additional regular XPath queries for XPathMark documents.

highly exponential in the worst case. However, the only query that we couldn't obtain its correspondent deterministic earliest congruence projection $A^{cgr}(\mathbf{S})_{e(\mathbf{S})}$ is A5.

The time for computing the earliest congruence projection took between 2.3 to 26 seconds.

We note that we have not yet computed the complete dSHA^\downarrow s statically for safe-no-change projection A^{snc} , pure congruence projection $A^{cgr}(\mathbf{S})$ and earliest query answering $A_{e(\mathbf{S})}$. We believe that these automata may become bigger than $A^{cgr}(\mathbf{S})_{e(\mathbf{S})}$ that we constructed in a single shot.

11.2. Streaming Evaluation Tool: AStream

We are using and developing the AStream tool for answering monadic dSHA queries on XML streams with schema restrictions. Version 1.01 of AStream was presented in [21] supports earliest query answering without projection. Given a dSHA A defining a monadic query, a set of schema final states F_S and an XML stream w , it constructs on-the-fly the needed part of the $\text{SHA}^\downarrow A_{e(\mathbf{S})}$ while parsing w and evaluating A on the hedge encoding w .

For the FCT conference version of the present paper [24], we enhanced AStream with safe-no-change projection. This leads to version AStream 2.01. It constructs the earliest safe-no-change projection $\text{dSHA}^\downarrow A_{e(\mathbf{S})}^{snc}$ on the fly while evaluating the monadic query defined by A on the hedge encoding the XML stream w . Subhedge projection can be switched on or off, in order to compare both versions without further implementation differences.

For this present journal version, we added earliest congruence projection and integrated it into AStream leading to AStream 3.0.

AStream 3.0 is different from the two previous versions in that the $\text{dSHA}^\downarrow A_{e(\mathbf{S})}^{cgr}$ is constructed statically and entirely, independently of the input hedge. We then use a generic earliest streaming evaluator for SHA^\downarrow s that rejects in rejection states, selects in selection

Query ID	dSHA A size(n)	dSHA [↓] size(#state)	#diff-rel. d
A1	482(68)	1296(324)	16
A2	224(42)	316(82)	7
A3	320(53)	662(156)	10
A4	629(74)	1651(404)	18
A5	438(63)	1226(269)	13
A6	675(76)	2090(500)	22
A7	394(59)	728(184)	12
A8	648(79)	2091(504)	24
A0	203(40)	158(44)	5
A1_0a	224(42)	145(44)	6
A1_0b	203(39)	68(23)	5

Query ID	dSHA A size(n)	dSHA [↓] size(#state)	#diff-rel d
A1_0c	230(43)	238(62)	6
A1_1a	305(54)	384(101)	8
A1_1d	305(54)	382(101)	8
A1_2	194(39)	152(42)	5
A1_3	318(53)	672(159)	10
A1_4	312(52)	479(132)	10
A1_5	266(47)	293(84)	8
A1_6	265(47)	588(142)	9
A2_1	232(42)	295(78)	7
A4_0	392(59)	719(184)	12
A4_1	292(49)	287(78)	8

Figure 26. Overall size and number n of states of the input dSHA A for the query with schema-final states F_S , the overall size and number of states of the $\text{SHA}^{\downarrow} A_e^{cgr}(\mathbf{S})_{e(\mathbf{S})}$ obtained by earliest congruence projection, and the number d its difference relations.

states, and projects in all subhedge projection states. This evaluator could also be run with $A_e^{snc}(\mathbf{S})$ or $A_{e(\mathbf{S})}$, as long as these do not grow too big.

It should be noticed that $A_e^{cgr}(\mathbf{S})$ turned out to be nicely small for our whole benchmark, while the other $\text{dSHA}^{\downarrow}s$ $A_e^{snc}(\mathbf{S})$ and $A_{e(\mathbf{S})}$ risk becoming bigger. As stated earlier, we did not construct these $\text{dSHA}^{\downarrow}s$ so far for our benchmark queries. This is why we continued to run the earliest streaming with safe-no-change projection with AStream 2.01, while we used AStream 3.0 for earliest streaming with congruence projection.

All AStream versions rely on Java’s abc-Datalog for computing the least fixed points of Datalog programs in a bottom-up manner. Datalog programs are needed for all logical reasoning: for computing subsets of states that are safe for rejection, selection, or subhedge projection on all levels. Also, the difference relations are computed based on Datalog. Note that earliest on-the-fly query evaluation requires running Datalog during query evaluation, while with a static approach for constructing $\text{dSHA}^{\downarrow}s$ entirely, Datalog is only needed at preprocessing time during the automaton construction.

11.3. Evaluation Measures

We want to measure the time for running the three streaming query evaluators for all dSHAs obtained from the XPath expressions in the collection in Tables 24 and 25.

One advantage of the XPathMark benchmark is that it comes with a generator of XML documents to which the queries can be applied and that can be scaled in size. We created an XML document of size 1.1 GB, which is sufficiently large to make streaming mandatory. Our experiments show that the efficiency of query evaluation grows linearly with the size of the non-projected part of the XML document. This holds for each of our three evaluators. Therefore, measuring the time of the evaluator on XML documents of other sizes would not show any new insights, except that memory consumption remains moderate too.

The time gain of projection for a query Q is the percentage of time saved by projection during query evaluation when ignoring the pure document parsing time t_{parse} seconds:

$$\text{time-gain}_Q = 100 * (1 - (t_{\text{noproject}_Q} - t_{\text{parse}}) / (t_{\text{project}_Q} - t_{\text{parse}}))$$

We can measure the time gain for earliest safe-no-change projection by AStream 2.01 and for earliest congruence projection by AStream 3.0, since projection can be switched off in both of them. The disadvantage of the time gain is that it depends on the details of the implementation.

The event gain is a better measure for the projection power. The parser sends a stream of needed events to the SHA^{\downarrow} , while ignoring parts of the input document that can be projected away. For this, it must always be informed by the automaton about what kind of

Query ID	#ans- wer	time in sec	time gain	event gain	Query ID	#ans- wer	time in sec	time gain	event gain
A1	38267	29.3	98.1%	98.9%	A1_0c	3600816	226.6	72.9%	75.7%
A2	117389	171.8	77.9%	81.1%	A1_1a	2	187.8	78.1%	80.3%
A3	117389	33.4	97.4%	97.8%	A1_1d	2	217.3	74.8%	80.3%
A4	24959	33.2	97.8%	98.9%	A1_2	1062965	238.2	68.2%	76.0%
A5	50186				A1_3	25074	16.2	99.1%	99.8%
A6	30307	45.9	96.9%	98.2%	A1_4	5170	17.5	99.7%	100.0%
A7	179889	30.5	98.0%	98.7%	A1_5	6	14.3	100.0%	100.0%
A8	67688	37.8	97.3%	98.7%	A1_6	117389	188.7	75.4%	81.1%
A0	1	21.3	99.1%	100.0%	A2_1	50186	199.7	76.5%	81.1%
A1_0a	6	15.2	99.9%	100.0%	A4_0	91650	21.8	99.0%	99.3%
A1_0b	0	0.0	100.0%	100.0%	A4_1	1	12.0	100.0%	100.0%

Figure 27. Time gain and event gain by earliest congruence projection with AStream3.01 on a 1.1GB XPathMark stream.

events can be projected in the current state. The event gain of projection $event-gain_Q$ is then the percentage of events that are recognized to be irrelevant and thus no more created by a projecting evaluator for query Q . One might expect that the time gain is always smaller than the event gain.

$$time-gain_Q \leq event-gain_Q$$

Indeed, this will be confirmed by our experiments. We believe that these discrepancies between these two measure indicate how much room remains for optimizing the implementation of an evaluator. In this way, we can observe that some room for further optimizations AStream 3.0 still remains.

11.4. Experiments

We use Java's *XMLStreamReader* Interface v1.0 from *javax.xml.stream* package to parse and stream XML files in Scala. Parsing a 1.1 GB XML documents requires $t_{\text{parse}} = 15$ seconds, while querying it without projection took us in average $t_{\text{noproject}_{\text{avg}}} = 752$ seconds, while varying from 600 to 900 seconds. The average processing time for 1 MB is thus 0.72 seconds. Once knowing this, one can predict the expected evaluation time from the size of the non-projected part of the XML document. It is:

$$event-gain_Q * 1100 * 0.72 \text{ seconds} + t_{\text{parse}} \text{ seconds}$$

Without projection, the parser generates 1,012,018,728 events for the 1.1GB XML document. This is the baseline for computing $event-gain_Q$ for all queries Q .

We ran AStream with Scala v2.13.3, on a machine with the operating system Ubuntu 20.04.06 LTS (64-bit). The machine is Dell Precision-7750 machine, equipped with an Intel® Core™ i7-10875H CPU @ 2.30GHz × 16 and 32 GB of RAM.

11.4.1. Earliest Congruence Projection

We present the measures of our evaluator with the earliest congruence projection in Fig. 27. The event gain is high, varying between 75.7% for A1_0c to 100% for A0, A1_0a, A1_0b, A1_4, A1_5, and A4_1. The event gain for queries without descendant axis is above 98.2%. For these queries, all subtrees that are not on the main path of the query can be projected away. In particular, only subtrees until a fixed depth have to be inspected. These are the queries A1, A4, A6-A8 and A4_0 and the queries with event gain 100% listed above. The time gain for all these queries is a little smaller than the event gain but still above 98.2%.

We next consider the second type of queries having the descendant axis, specifically A2, A1_0c, A1_1a, A1_1d, A1_2, A1_6, and A2_1. Intuitively, a lower event gain is expected in these cases because subhedged projection with the descendant axis cannot directly exclude

Query ID	time (sec) snc	time (sec) congr	time gain snc	time gain congr	event gain congr
A1	72.8	29.3	92.3%	98.1%	98.9%
A2	664.6	171.8	8.5%	77.9%	81.1%
A3	666.1	33.4	10.9%	97.4%	97.8%
A4	78.5	33.2	92.4%	97.8%	98.9%
A5	77.7		92.3%		
A6	65.2	45.9	95.1%	96.9%	98.2%
A7	24.6	30.5	98.8%	98.0%	98.7%
A8	24.8	37.5	98.9%	97.3%	98.7%

Figure 28. Time gain for earliest safe-no-change and earliest congruence projection, and the event gain of earliest congruence projection.

an entire subtree under a given element; all descendant elements must be inspected. For instance, the query `A1_0c` equal to `/site//@*` must select all attributes of all elements under the root element `site`, requiring the inspection of every XML element in the document, for attribute presence, except for the root element `site`.

The lower event gain reported for the aforementioned queries align with the above expectation, showing percentages ranging from 75.7% to 81.1%. Also, these queries exhibit a larger gap between time gain and event gain, which ranges from 2.2% to 5.7% and will be subject to future optimization. It is worth noting that some of these queries combine multiple features at once, like queries `A1_1a` and `A1_1d` which have descendant axis, filter with attributes, and string comparison.

The only exception to this trend for the queries having descendant axis are `A3` and `A1_3`, with respective event gains of 97.8% and 99.8%. The reason behind this high gain is that these queries starts from the root, with a child axis path of depth 3 before querying the descendant axis. This initial path allows for the exclusion and projection of most elements under `site` that do not fit the specified path, with their entire subtrees.

11.4.2. Earliest Safe-No-Change Projection

We compare earliest safe-no-change projection with the earliest congruence projection in see Table 28. For this we restrict ourselves to the queries `A1` – `A8` from the XPathMark in Table 24.

The XPath queries `A1`, `A4`, `A6`, `A7`, `A8` don't have descendant axes. The time gain for safe-no-change projection on these queries is between 92.3 – 98.9%. For earliest congruence projection, the time gain was better for `A1`, `A4`, `A6` with at least 96.6%. For `A7` and `A7`, the time gain of safe-no-change projection is close to the event gain of congruence projection, but the time gain of congruence projection is 1.5% lower. So we hope that an optimized version of congruence projection could outperform safe-no-change projection in all cases.

The XPath queries with descendant axis are `A2`, `A3`, `A5`. For `A2` and `A3` in particular, the time gain of safe-no-change projection is very low with even not 10.9% while congruence projection yields at least 77.9%. We believe that this is bug in our current implementation of safe-no-change projection, which may be prohibiting the projection of attributes and text nodes for these queries (but not for the others). Congruence projection on `A2` gains 77.9% of time and 81.1% of events. This is much better, but still far from the projection power for queries without descendant axis. For `A3`, congruence projection gains 97.4% of time. This is much better than for `A2` since the descendant axis in `A3` is at the end, while for `A2` it is at the beginning of the path. But even for `A2`, congruence projection is very helpful.

11.4.3. Comparison to External Tools

QuiXPath was shown to be the most efficient large coverage external tool for regular XPath evaluation on XML streams [4]. We therefore compare the efficiency of congruence projection to QuiXPath in Table 29, and thereby by transitivity to the many alternative tools. We note that QuiXPath performs early query answering by compilation to possibly

Query ID	parse-free time (sec) QuiXPath	parse-free time (sec) congr. prj.	fract.
A1	11.0	14.3	1.3
A4	11.6	18.2	1.6
A6	10.7	30.9	2.9
A7	8.6	15.5	1.8
A8	8.8	22.5	2.6

Query ID	parse-free time (sec) QuiXPath	parse-free time (sec) congr. prj.	fract.
A2	11.4	156.8	13.8
A3	11.5	18.4	1.6
A5	12.0		

Figure 29. Comparison of of parse-free timings in seconds for XPathMark queries with QuiXPath and earliest congruence projection.

nondeterministic NWA's with selection states, without always being earliest. Apart from this, it bases its efficiency on both subtree projection and descendant projection.

The experiments with QuiXPath in [4] use the parse-free time. We therefore do the same for congruence projection by reducing the measured overall time by 15 seconds of parsing time.

Compared to the parsing-free times for the XPath queries without descendant axis, the earliest congruence projection demonstrates significant improvements.. On average, our projection is only slower by a factor of 1.3 - 2.9 than QuiXPath. This shows that our implementation is already close to be competitive with the existing XML streaming tools, while being the only one guaranteeing earliest query answering.

For query A2 with descendant axis, congruence projection is by a factor of 13.8 slower than QuiXPath. The reason is that QuiXPath supports descendant projection, while congruence projection concerns only subhedge. For query A3 with a descendant axis at the end, congruence projection is only by a factor of 1.6 behind, so descendant projection seems less relevant here.

12. Related Automata Models

A large number of alternative automata notions for trees, forest and hedges were proposed in the literature. We compare them to SHAS and SHA^{\downarrow} s in what follows, and discuss the implications.

12.1. SHAs versus Tree, Forest, and Hedge Automata

Standard Hedge Automata. Standard hedge automata [9,10,34,35] operate on labeled hedges in a bottom-up manner similarly to SHAs. Also they have the same expressiveness, but horizontal languages are specified differently, leading to a problematic notion of determinism [36]. For this reason unique minimization fails for deterministic standard hedge automata. Still, they have the same expressiveness as SHAs when restricted to labeled hedges.

Standard Tree Automata and SHA Minimization. Syntactically, any SHA A is a standard tree automaton over the following ranked signature:

- a unary symbol for any letter $a \in \Sigma$,
- a binary symbol $@$, and
- a constant $\langle \rangle$.

Semantically, however, there is no perfect general correspondence. Only for subclasses a perfect correspondence can be made up via binary encoding. This was shown in [8] for the subclass of dSHAs with $I = \langle \rangle^\Delta$. In [37], it could also be shown for the subclass of multi-module dSHAs. This leads to unique minimization results for both subclasses of deterministic SHAs.

Bojanczyk's Forest Automata. Standard hedge automata also have the same expressiveness as Bojanczyk's forest automata (see Section 3.3 of [38]). These are based on the idea of transition monoids, rather than on the idea of transition rules such as SHAs or finite state automata or Neuman's and Seidl's forest automata. As a consequence, however, there is an exponential difference in succinctness between SHAs and Bojanczyk's forest automata [37].

12.2. SHA^\downarrow s versus SHAs

We start explaining why SHA^\downarrow 's have the same expressiveness as SHAs. Basically it is the same reason for why NWA's have the same expressiveness as SHAs (see e.g. [8]). For the easier direction, we argued already that any SHA $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ can be converted to a $\text{SHA}^\downarrow \text{down}(A)$.

Downward Elimination. Conversely, we can convert any $\text{dSHA}^\downarrow A$ into an equivalent SHA $\text{elim}^\downarrow(A)$ while possibly introducing nondeterminism as follows:

$$\begin{array}{c}
 I^{\text{elim}^\downarrow} = I \times \mathcal{Q} \\
 F^{\text{elim}^\downarrow} = F \times \mathcal{Q}
 \end{array}
 \quad
 \frac{q \xrightarrow{\langle \rangle} q' \text{ in } \Delta}{\xrightarrow{\langle \rangle} (q, q') \text{ in } \Delta^{\text{elim}^\downarrow}}$$

$$\frac{q \xrightarrow{a} q' \text{ in } \Delta \quad r \in \mathcal{Q}}{(r, q) \xrightarrow{a} (r, q') \text{ in } \Delta^{\text{elim}^\downarrow}}
 \quad
 \frac{q@p \rightarrow q' \text{ in } \Delta \quad r \in \mathcal{Q}}{(r, q)@(q, p) \rightarrow (r, q') \text{ in } \Delta^{\text{elim}^\downarrow}}$$

Proposition 60. $\mathcal{L}(A) = \mathcal{L}(\text{elim}^\downarrow(A))$.

The construction is analogous to the conversion of NWA's to SHAs [8] or to hedge automata [26]. The correctness proofs for these compilers are standard.

Unique Minimization. Unique minimization fails for dSHA^\downarrow 's as usual for deterministic multiway automata. This even happens for deterministic two-way finite state automata on words. In contrast, the subclass of dSHAs with the restriction that the initial state and the tree initial state are the same enjoys unique minimization [8].

Neuman and Seidl's Pushdown Forest Automata. Standard hedge automata are also known to have the same expressiveness than Neumann and Seidl's pushdown forest automata [25]. The latter can be identified with SHA^\downarrow 's for labeled hedges. What is needed

to map pushdown forest automata to standard hedge automata is downward elimination, as for mapping SHA^\downarrow s to SHAs.

12.3. SHA^\downarrow s versus NWAs

The streaming evaluator for SHA^\downarrow s via a visibly pushdown machine can also be obtained by compiling a SHA^\downarrow to a nested word automaton (NWA) [11], whose streaming evaluator is given by a visibly pushdown machine [12], previously known as *input driven automata* [13–15].

Definition 61. A nested word automata (NWA) is a tuple $(\Sigma, \mathcal{Q}, \Gamma, \Delta, I, F)$, where Σ, Γ and \mathcal{Q} are sets, $I, F \subseteq \mathcal{Q}$, and $\Delta = ((a^\Delta)_{a \in \Sigma}, \langle^\Delta, \rangle^\Delta)$ contains relations: $a^\Delta \subseteq \mathcal{Q} \times \mathcal{Q}$, $\langle^\Delta \subseteq \mathcal{Q} \times (\Gamma \times \mathcal{Q})$ and $\rangle^\Delta \subseteq \mathcal{Q} \times \Gamma \times \mathcal{Q}$. A NWA is deterministic or equivalently a dNWA if I contains at most one element and all above relations are partial functions.

The elements of Γ are called stack symbols. The transition rules in Δ again have three forms: Internal rules $q \xrightarrow{a} q'$ in Δ as for SHAs, opening rules $q \xrightarrow{\langle^\Delta \gamma} q'$ in Δ if $\langle^\Delta(q) = (q', \gamma)$ and closing rules $q \xrightarrow{\rangle^\Delta \gamma} q'$ in Δ if $\rangle^\Delta(q, \gamma) = q'$.

Streaming evaluators for NWAs. The streaming evaluator for NWAs can be seen as pushdown machines for evaluating the nested words of hedges in streaming manner. A configuration of the pushdown machine is a pair in $\mathcal{K} = \mathcal{Q} \times \Gamma^*$ containing a state and a stack. For any word $v \in \hat{\Sigma}^*$ we define a streaming transition relation $\xrightarrow{v}^{str} \subseteq \mathcal{K} \times \mathcal{K}$ such that for all $q, q' \in \mathcal{Q}$ and $\sigma \in \Gamma^*$:

$$\begin{array}{c} \frac{\text{true}}{(q, \sigma) \xrightarrow{\varepsilon}^{str} (q, \sigma) \text{ wrt } \Delta} \quad \frac{(q, \sigma) \xrightarrow{v}^{str} (q', \sigma) \quad (q', \sigma) \xrightarrow{v'}^{str} (q'', \sigma) \text{ wrt } \Delta}{(q, \sigma) \xrightarrow{v \cdot v'}^{str} (q'', \sigma) \text{ wrt } \Delta} \\ \\ \frac{q \xrightarrow{a} q' \text{ in } \Delta}{q \xrightarrow{a}^{str} q' \text{ wrt } \Delta} \quad \frac{q \xrightarrow{\langle^\Delta \gamma} q' \text{ in } \Delta}{(q, \sigma) \xrightarrow{\langle^\Delta \gamma}^{str} (q', \sigma) \text{ wrt } \Delta} \quad \frac{q \xrightarrow{\rangle^\Delta \gamma} q' \text{ in } \Delta}{(q, \sigma \cdot \gamma) \xrightarrow{\rangle^\Delta \gamma}^{str} (q', \sigma) \text{ wrt } \Delta} \end{array}$$

From SHA^\downarrow s to NWAs. For any $\text{SHA}^\downarrow A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ we can define the NWA $A^{nwa} = (\Sigma, \mathcal{Q}, \Gamma, \Delta^{nwa}, I^{nwa}, F^{nwa})$ while preserving determinism, such that $\Gamma = \mathcal{Q}$, while Δ^{nwa} contains for all $a \in \Sigma$ and $q, p \in \mathcal{Q}$ the transition rules:

$$\frac{q \xrightarrow{a} q' \text{ in } \Delta}{q \xrightarrow{a} q' \text{ in } \Delta^{nwa}} \quad \frac{q \xrightarrow{\langle^\Delta} q' \text{ in } \Delta}{q \xrightarrow{\langle^\Delta q} q' \text{ in } \Delta^{nwa}} \quad \frac{q @ p \rightarrow q' \text{ in } \Delta}{p \xrightarrow{\rangle^\Delta q} q' \text{ in } \Delta^{nwa}}$$

For instance, the dNWA A^{nwa} of the $\text{dSHA}^\downarrow A$ in Fig. 9 obtained from the dSHA for the `[self::list/child::item]` from the introduction is given in Fig. 30. And the dNWA A^{nwa} of the $\text{dSHA}^\downarrow A$ in Fig. 21 is given in Fig. 31.

Lemma 62. $\mathcal{L}(A^{nwa}) = \mathcal{L}(A)$.

The runs of SHA^\downarrow s A and NWAs A^{nwa} can be identified.

Projection for NWAs. Projecting evaluators for NWAs were proposed in the context of projecting NWAs [4]. They are based on the following notion of states of irrelevant subtrees for NWAs.

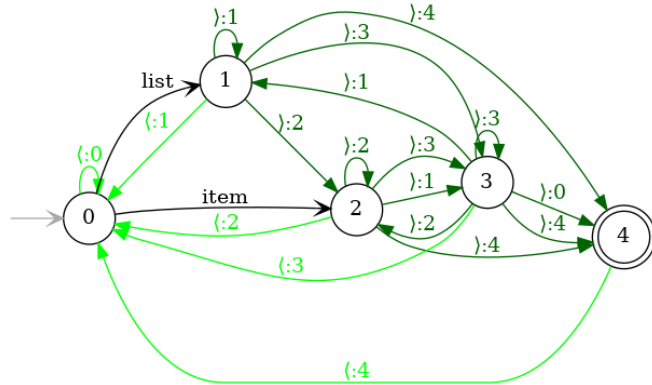


Figure 30. The dNWA A^{nwa} for the dSHA \downarrow A in Fig. 9 obtained by applying the *down* operator to the dSHA in Fig. 4 for the filter `[self::list/child::item]`.

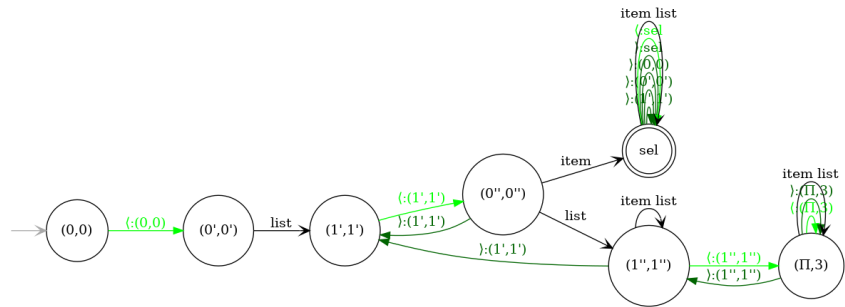
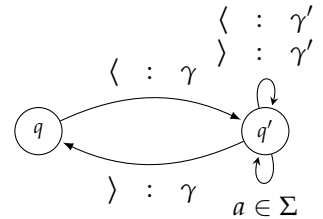


Figure 31. The dNWA $(A_e^{snc}(\llbracket doc \rrbracket))^{nwa}$ for the earliest dSHA \downarrow with safe-no-change projection in Fig. 21.

Definition 63 (Variant of Definition 3 of [4]). *We call a state q of an NWA a state of irrelevant subtrees if there exist two different stack symbols γ, γ' and a state q' such that the transitions shown on the right exist, but no further opening transitions with γ , no further transitions with γ' , and no further closing transition in q popping γ . In this case, we write $q \in i\text{-tree}_{\Sigma \setminus \emptyset}$.*



Lemma 64. *Any subhedge projection state of a complete SHA \downarrow A is a state of irrelevant subtrees of A^{nwa} .*

It was then shown in [4] how to map an NWA with a subset \mathcal{Q}_{shp}^Δ of states of irrelevant subtrees to a projecting NWA, whose streaming semantics yields a streaming evaluator with subhedge projection. The presentation of subhedge projection for SHA \downarrow s without passing via NWAs yields the same result in a more direct manner.

It should also be noticed that projecting NWAs support descendant projection beside of subhedge projection. For SHAs, this is left to future research.

sectionConclusion and Future Work We introduced the notion of irrelevant subhedges for membership to hedge language under schema restrictions. We developed algorithms with subhedge projection that jump over irrelevant subhedges for testing membership to the regular hedge languages defined by dSHAs. All our algorithms can be run in in-memory mode and in streaming mode. The difficulty was how to push the needed finite state information for subtree projection top-down given that SHAs operate bottom-up. We solved it based on compilers from SHAs to downward SHAs.

This first compiler propagates safety information about non-changing states. The second compiler, which we prove to be complete for subhedge projection, propagates

difference relations. We then combined subhedge projection with earliest membership testing and combine it earliest monadic query answering. We integrated both safe-no-change and complete congruence in our AStream tool and confirmed their usefulness experimentally: we showed that it can indeed speed up the the AStream tool, for answering algorithm regular XPath queries on XML streams. Moreover, congruence projection showed much higher event projection gain rates than safe-no-change, ranging between 75.7% and 100%, for the class of queries that does not have descendant axis and thus, becoming competitive in time with the best existing streaming tools for regular XPath queries without recursive axis.

A remaining open theoretical and practical question is how to obtain descendant projection for SHAs, as needed to speed up the evaluation of XPath queries with descendant axis. On the implementation and experimental side, it would be nice to investigate the in-memory version of our subhedge algorithms too. And finally, it would be nice to optimize streaming earliest-query answering tools further until they outperform the best streaming tools in all cases, while being better for queries where the other tools are not earliest.

References

1. Marian, A.; Siméon, J. Projecting XML Documents. In Proceedings of the VLDB, 2003, pp. 213–224.
2. Frisch, A. Regular Tree Language Recognition with Static Information. In Proceedings of the Exploring New Frontiers of Theoretical Informatics, IFIP 18th World Computer Congress, TCS 3rd International Conference on Theoretical Computer Science, 2004, pp. 661–674.
3. Maneth, S.; Nguyen, K. XPath Whole Query Optimization. *VLPB Journal* **2010**, *3*, 882–893.
4. Sebastian, T.; Niehren, J. Projection for Nested Word Automata Speeds up XPath Evaluation on XML Streams. In Proceedings of the International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM), Harrachov, Czech Republic, 2016.
5. Kay, M. The saxon XSLT and XQuery processor, 2004. <https://www.saxonica.com>.
6. Gienieczko, M.; Murlak, F.; Paperman, C. Supporting Descendants in SIMD-Accelerated JSONPath. In Proceedings of the Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1, ASPLOS 2024. ACM, 2024, pp. 15–29.
7. Al Serhali, A.; Niehren, J. A Benchmark Collection of Deterministic Automata for XPath Queries. In Proceedings of the XML Prague 2022, Prague, Czech Republic, 2022.
8. Niehren, J.; Sakho, M. Determinization and Minimization of Automata for Nested Words Revisited. *Algorithms* **2021**, *14*, 68. <https://doi.org/10.3390/a14030068>.
9. Comon, H.; Dauchet, M.; Gilleron, R.; Löding, C.; Jacquemard, F.; Lugiez, D.; Tison, S.; Tommasi, M. Tree Automata Techniques and Applications. Available online since 1997: <http://tata.gforge.inria.fr>, 2007.
10. Thatcher, J.W. Characterizing derivation trees of context-free grammars through a generalization of automata theory. *Journal of Computer and System Science* **1967**, *1*, 317–322. [https://doi.org/10.1016/S0022-0000\(67\)80022-9](https://doi.org/10.1016/S0022-0000(67)80022-9).
11. Alur, R. Marrying Words and Trees. In Proceedings of the 26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. ACM-Press, 2007, pp. 233–242.
12. Alur, R.; Madhusudan, P. Visibly pushdown languages. In Proceedings of the Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13–16, 2004; Babai, L., Ed. ACM, 2004, pp. 202–211. <https://doi.org/10.1145/1007352.1007390>.
13. Mehlhorn, K. Pebbling Mountain Ranges and its Application of DCFL-Recognition. In Proceedings of the Automata, Languages and Programming, 7th Colloquium, Noordwijkerhout, The Netherlands, July 14–18, 1980, Proceedings; de Bakker, J.W.; van Leeuwen, J., Eds. Springer, 1980, Vol. 85, *Lecture Notes in Computer Science*, pp. 422–435. https://doi.org/10.1007/3-540-1003-2_89.
14. von Braunmühl, B.; Verbeek, R. Input Driven Languages are Recognized in log n Space. In *Topics in the Theory of Computation*; Karplinski, M.; van Leeuwen, J., Eds.; North-Holland, 1985; Vol. 102, *North-Holland Mathematics Studies*, pp. 1 – 19. [https://doi.org/https://doi.org/10.1016/S0304-0208\(08\)73072-X](https://doi.org/https://doi.org/10.1016/S0304-0208(08)73072-X).
15. Okhotin, A.; Salomaa, K. Complexity of input-driven pushdown automata. *SIGACT News* **2014**, *45*, 47–67. <https://doi.org/10.1145/2636805.2636821>.

16. Niehren, J.; Sakho, M.; Al Serhali, A. Schema-Based Automata Determinization. In Proceedings of the Proceedings of the 13th International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2022, Madrid, Spain, September 21-23, 2022; Ganty, P.; Monica, D.D., Eds., 2022, Vol. 370, *EPTCS*, pp. 49–65. <https://doi.org/10.4204/EPTCS.370.4>.
17. Lick, A.; Schmitz, S. XPath Benchmark, Last visited April 13th 2022.
18. Mozafari, B.; Zeng, K.; Zaniolo, C. High-performance complex event processing over XML streams. In Proceedings of the SIGMOD Conference; Candan, K.S.; Chen, Y.; Snodgrass, R.T.; Gravano, L.; Fuxman, A.; Candan, K.S.; Chen, Y.; Snodgrass, R.T.; Gravano, L.; Fuxman, A., Eds. ACM, 2012, pp. 253–264. <https://doi.org/10.1145/2213836.2213866>.
19. Grez, A.; Riveros, C.; Ugarte, M. A Formal Framework for Complex Event Processing. 2019, pp. 5:1–5:18. <https://doi.org/10.4230/LIPIcs.ICDT.2019.5>.
20. Muñoz, M.; Riveros, C. Streaming Enumeration on Nested Documents. In Proceedings of the 25th International Conference on Database Theory, ICDT 2022, March 29 to April 1, 2022, Edinburgh, UK (Virtual Conference); Olteanu, D.; Vortmeier, N., Eds. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, Vol. 220, *LIPICs*, pp. 19:1–19:18. <https://doi.org/10.4230/LIPIcs.ICDT.2022.19>.
21. Al Serhali, A.; Niehren, J. Earliest Query Answering for Deterministic Stepwise Hedge Automata. In Proceedings of the Implementation and Application of Automata - 27th International Conference, CIAA 2023, Famagusta, North Cyprus, September 19-22, 2023, Proceedings; Nagy, B., Ed. Springer, 2023, Vol. 14151, *Lecture Notes in Computer Science*, pp. 53–65. https://doi.org/10.1007/978-3-031-40247-0_3.
22. Gauwin, O.; Niehren, J.; Tison, S. Earliest Query Answering for Deterministic Nested Word Automata. In Proceedings of the 17th International Symposium on Fundamentals of Computer Theory. Springer Verlag, 2009, Vol. 5699, *Lecture Notes in Computer Science*, pp. 121–132. https://doi.org/10.1007/978-3-642-03409-1_12.
23. Debarbieux, D.; Gauwin, O.; Niehren, J.; Sebastian, T.; Zergaoui, M. Early nested word automata for XPath query answering on XML streams. *Theor. Comput. Sci.* **2015**, 578, 100–125. <https://doi.org/10.1016/j.tcs.2015.01.017>.
24. Al Serhali, A.; Niehren, J. Subhedge Projection for Stepwise Hedge Automata. In Proceedings of the Fundamentals of Computation Theory - 24th International Symposium, FCT 2023, Trier, Germany, September 18-21, 2023, Proceedings; Fernau, H.; Jansen, K., Eds. Springer, 2023, Vol. 14292, *Lecture Notes in Computer Science*, pp. 16–31. https://doi.org/10.1007/978-3-031-43587-4_2.
25. Neumann, A.; Seidl, H. Locating Matches of Tree Patterns in Forests. In Proceedings of the Foundations of Software Technology and Theoretical Computer Science. Springer Verlag, 1998, Vol. 1530, *Lecture Notes in Computer Science*, pp. 134–145. https://doi.org/10.1007/978-3-642-03409-1_12.
26. Gauwin, O.; Niehren, J.; Roos, Y. Streaming Tree Automata. *Information Processing Letters* **2008**, 109, 13–17. <https://doi.org/10.1016/j.ipl.2008.08.002>.
27. Franceschet, M. XPathMark Performance Test. <https://users.dimi.uniud.it/~massimo.franceschet/xpathmark/PTbench.html>. Accessed: 2020-10-25.
28. Hosoya, H.; Pierce, B.C. XDuce: A Statically Typed XML Processing Language. *ACM Trans. Internet Technol.* **2003**, 3, 117–148. <https://doi.org/10.1145/767193.767195>.
29. Gécseg, F.; Steinby, M. *Tree Automata*; Akadémiai Kiadó, Budapest, 1984.
30. Gold, E.M. Language Identification in the Limit. *Inf. Control.* **1967**, 10, 447–474. [https://doi.org/10.1016/S0019-9958\(67\)91165-5](https://doi.org/10.1016/S0019-9958(67)91165-5).
31. Carme, J.; Niehren, J.; Tommasi, M. Querying Unranked Trees with Stepwise Tree Automata. In Proceedings of the Rewriting Techniques and Applications, 15th International Conference, RTA 2004, Aachen, Germany, June 3-5, 2004, Proceedings; van Oostrom, V., Ed. Springer, 2004, Vol. 3091, *Lecture Notes in Computer Science*, pp. 105–118. https://doi.org/10.1007/978-3-540-25979-4_8.
32. Neumann, A.; Seidl, H. Locating Matches of Tree Patterns in Forests. In Proceedings of the 18-th Conference on Foundations of Software Technology and Theoretical Computer Science, 1998. <https://doi.org/10.1007/b71635>.
33. Debarbieux, D.; Gauwin, O.; Niehren, J.; Sebastian, T.; Zergaoui, M. Early nested word automata for XPath query answering on XML streams. *Theor. Comput. Sci.* **2015**, 578, 100–125. <https://doi.org/10.1016/j.TCS.2015.01.017>.
34. Pair, C.; Quéré, A. Définition et étude des bilangages réguliers. *Information and Control* **1968**, 13, 565–593.

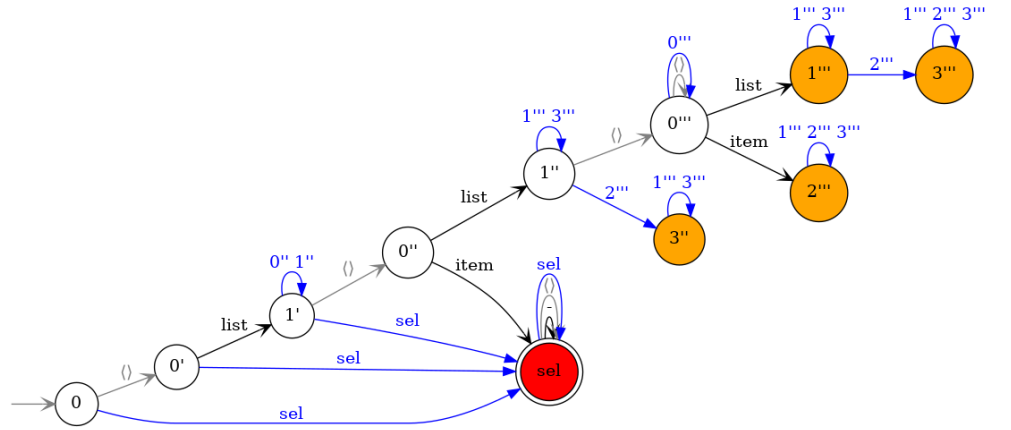


Figure A32. The earliest $dSHA^\downarrow A_e$ for the $dSHA A$ in Fig. 19 with schema $[[doc]]$ for the XPath filter $[self::list/child::item]$. The states of A_e correspond to the following tuples: $0 = (0, \{1, 2, 3, 5\}, \{4\})$, $0' = (0, \{2, 4, 5\}, \{3, 4\})$, $1' = (1, \{2, 4, 5\}, \{3, 4\})$, $0'' = (0, \emptyset, \{2, 4, 5\})$, $1'' = (1, \emptyset, \{2, 4, 5\})$, $0''' = (0, \emptyset, \emptyset)$, $1''' = (1, \emptyset, \emptyset)$, $2''' = (1, \emptyset, \emptyset)$, and $3''' = (1, \emptyset, \emptyset)$.

35. Murata, M. Hedge Automata: a Formal Model for XML Schemata. Web page, 2000.
36. Martens, W.; Niehren, J. On the Minimization of XML Schemas and Tree Automata for Unranked Trees. *Journal of Computer and System Science* **2007**, *73*, 550–583.
37. Niehren, J. Stepwise Hedge Automata are exponentially more succinct than Forest Automata, **in preparation**.
38. Bojanczyk, M.; Walukiewicz, I. Forest algebras. In Proceedings of the Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]; Flum, J.; Grädel, E.; Wilke, T., Eds. Amsterdam University Press, 2008, Vol. 2, *Texts in Logic and Games*, pp. 107–132.

Appendix N Earliest Membership

We adapt the earliest membership tester for dSHAs from [21] so that it compiles to SHA^\downarrow s instead of NWA's and such that it accounts for schemas.

Appendix N.1 Schema Safety is an Inclusion Problem

The idea for testing certain membership and certain nonmembership is to consider states are safe to reach final states except if going out of the schema. Let $A = (\Sigma, Q, \Delta, I, F)$ be an SHA with schema S such that there exists $F \subseteq F_S \subseteq Q$ with $S = \mathcal{L}(A[F/F_S])$. Note that $\mathcal{L}(A) \subseteq S \subseteq \mathcal{H}_\Sigma$ in particular. The set of states that are safe to reach a subset $P \subseteq Q$ for all hedges that do not go out of the schema S then is:

$$safe_S^\Delta(P) = safe^\Delta(P \cup (Q \setminus F_S))$$

If A is deterministic and schema-complete for S then the safety of a state of A with respect to schema S is an inclusion problem.

Lemma A65. Let $S = \mathcal{L}(A[F/F_S])$. If A is deterministic and schema-complete for S then for all states $q \in Q$:

$$\mathcal{L}(A[I/\{q\}, F/F_S]) \subseteq \mathcal{L}(A[I/\{q\}]) \Leftrightarrow q \in safe_S^\Delta(F)$$

Proof. “ \Leftarrow ”. Suppose that $q \notin safe_S^\Delta(F)$. The definition of schema-based safety yields $acc^\Delta(\{q\}) \cap (F_S \setminus F) \neq \emptyset$. So there exists some hedge $h \in \mathcal{H}_\Sigma$ such that such that $q \xrightarrow{h} F_S \setminus F$ wrt Δ . In particular, $q \in \mathcal{L}(A[I/\{q\}, F/F_S])$. By determinism there exists no other transition on h from q and thus $q \notin \mathcal{L}(A[I/\{q\}])$. Hence $\mathcal{L}(A[I/\{q\}, F/F_S]) \not\subseteq \mathcal{L}(A[I/\{q\}])$.

$$\begin{array}{c}
\frac{q \xrightarrow{a} q' \text{ in } \Delta \quad q' \notin S \cup R}{(q, S, R) \xrightarrow{a} (q', S, R) \text{ in } \Delta_e} \\
\frac{\Downarrow q' \text{ in } \Delta \quad (q, S, R) \in \text{in } \mathcal{Q}_e \quad q' \notin s\text{-down}^\Delta(q, S) \cup s\text{-down}^\Delta(q, R)}{(q, S, R) \Downarrow (q', s\text{-down}^\Delta(q, S), s\text{-down}^\Delta(q, R)) \text{ in } \Delta_e} \\
\frac{q @ p \rightarrow q' \text{ in } \Delta \quad q' \notin S \cup R}{(q, S, R) @ (p, s\text{-down}^\Delta(q, S), s\text{-down}^\Delta(q, R)) \rightarrow (q', S, R) \text{ in } \Delta_e} \\
\frac{q \xrightarrow{a} q' \text{ in } \Delta \quad q' \in S}{(q, S, R) \xrightarrow{a} sel \text{ in } \Delta_e} \quad \frac{\Downarrow q' \text{ in } \Delta \quad (q, S, R) \in \text{in } \mathcal{Q}_e \quad q' \in s\text{-down}^\Delta(q, S)}{(q, S, R) \Downarrow sel \text{ in } \Delta_e} \\
\frac{q @ p \rightarrow q' \text{ in } \Delta \quad q' \in S}{(q, S, R) @ (p, s\text{-down}^\Delta(q, S), s\text{-down}^\Delta(q, R)) \rightarrow sel \text{ in } \Delta_e} \\
\frac{a \in \Sigma}{sel \xrightarrow{a} sel \text{ in } \Delta_e} \quad \frac{\mu \in \mathcal{Q}_e}{sel @ \mu \rightarrow sel \text{ in } \Delta_e} \quad \frac{\mu \in \mathcal{Q}_e}{\mu @ sel \rightarrow sel \text{ in } \Delta_e} \quad \frac{true}{sel \Downarrow sel \text{ in } \Delta_e}
\end{array}$$

Figure A33. The earliest transition rules Δ_e inferred from transition rules Δ of some SHA.

" \Rightarrow ". Suppose that $q \in \text{safe}_S^\Delta(F)$. Let $h \in \mathcal{L}(A[I/\{q\}, F/F_S])$. Then there exists $q' \in F_S$ such that $q \xrightarrow{h} q'$ wrt Δ . Hence $q' \in \text{acc}^\Delta(\{q\})$ so that $\text{acc}^\Delta(\{q\}) \subseteq F \cup (\mathcal{Q} \setminus F_S)$ implies $q' \in F$. Thus, $h \in \mathcal{L}(A[I/\{q\}])$.
 \square

Lemma A65 with $F_S = \mathcal{Q}$ shows Lemma 5 of [21], which states for any complete dSHAs A that safety is a universality problem:

$$\mathcal{L}(A[I/\{q\}]) = \mathcal{H}_\Sigma \Leftrightarrow q \in \text{safe}^\Delta(F)$$

This is since the schema-completeness of A for $\mathbf{S} = \mathcal{H}_\Sigma$ implies the completeness of A and thus $\mathcal{L}(A[I/\{q\}, F/\mathcal{Q}]) = \mathcal{H}_\Sigma$.

Appendix N.2 Algorithm

We now present our earliest membership tester for dSHAs with schema restrictions. Given a SHA A we compute a $\text{SHA}^\downarrow A_{e(\mathbf{S})} = (\Sigma, \mathcal{Q}_e, \Delta_e, I_{e(\mathbf{S})}, F_{e(\mathbf{S})})$ as follows. Let sel be a fresh symbol. We start with set of states that are safe for selection $S_0 = \text{safe}_S^\Delta(F)$ and respectively safe for rejection $R_0 = \text{safe}_S^\Delta(\mathcal{Q} \setminus F)$. The state sets of $A_{e(\mathbf{S})}$ are then:

$$\begin{aligned}
\mathcal{Q}_e &= (\mathcal{Q} \times 2^{\mathcal{Q}} \times 2^{\mathcal{Q}}) \cup \{sel\} \\
I_{e(\mathbf{S})} &= \{(q, R_0, S_0) \mid q \in I, q \notin R_0 \cup S_0\} \cup \{sel \mid I \cap S_0 \neq \emptyset\} \\
F_{e(\mathbf{S})} &= \{(q, R_0, S_0) \mid q \in F\} \cup \{sel\}
\end{aligned}$$

The transition rules in Δ_e are given by the in Fig. A33. For illustration, reconsider the dSHA from Fig. 4. This dSHA is not complete but schema-complete for the schema $\llbracket \text{doc} \rrbracket$. We get $s\text{-down}^\Delta(0, \{4\}) = \{3, 4\}$. It may seem counter intuitive that not only state 3 but also state 4 down remains safe for selection. This reflects the fact that no proper subhedge of any hedge satisfying the intended schema may ever get into state 4. Applying the earliest construction with safe-no-change projection to dSHA in Fig. 4 yields the dSHA^\downarrow in Fig. A32.

Appendix N.3 Soundness and Completeness

We next provide soundness and completeness results for our earliest in-memory membership tester with projection.

Proposition A66 (Soundness). *For any SHA A with schema $\mathbf{S} = \mathcal{L}(A[F/F_S])$, hedge $h \in \mathbf{S}$, and partial run v of $A_{e(\mathbf{S})}$ on h that ends in state sel then $h \in \mathcal{L}(A)$.*

So if the partial run reaches state sel then $proj_{\Sigma}(v)$ is certain for membership to $\mathcal{L}(A)$ with respect of \mathbf{S} . Evaluating a $\text{SHA}^{\downarrow s} A_{e(\mathbf{S})}$ yields an earliest in-memory membership tester for dSHA A . A single adaptation is in order: whenever sel is reached, the evaluation can stop all over and accept the input hedge.

Theorem A8 (Completeness). *Let $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ be a dSHA that is schema-complete for schema $\mathbf{S} = \mathcal{L}(A[F/F_S])$. For any hedge $h \in \mathbf{S}$ and prefix v of $nw(h)$:*

- *if v is certain for membership in $\mathcal{L}(A)$ with \mathbf{S} then there exists a partial run w for $A_{e(\mathbf{S})}$ on h such that $v = proj_{\Sigma}(w)$ and w ends with state sel .*
- *if v is certain for nonmembership in $\mathcal{L}(A)$ with \mathbf{S} then there exists blocking partial run w for $A_{e(\mathbf{S})}$ on h such that $proj_{\Sigma}(w)$ is a prefix of v .*

This means that certainty for membership and nonmembership are detected at the earliest prefix when running the evaluator for $A_{e(\mathbf{S})}$.

Proof sketch. We notice that this algorithm is basically the same as the earliest membership tester from Proposition 6 of [21], except that it also check for safe rejection and that it accounts for schemas. The fact that NWA's are used there instead of $\text{SHA}^{\downarrow s}$ here is not essential. So we can lift the soundness and completeness proof from there without problem. \square