Subhedge Projection for Stepwise Hedge Automata

Antonio Al Serhali, Joachim Niehren

Subhedge Projection for Stepwise Hedge Automata

Antonio Al Serhali and Joachim Niehren

Inria and University of Lille, France

Abstract. We show how to evaluate stepwise hedge automata (SHAs) with subhedge projection. Since this requires passing finite state information top-down, we introduce the notion of downward stepwise hedge automata. We use them to define an in-memory and a streaming evaluator with subhedge projection for SHAs. We then tune the streaming evaluator so that it 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 subhedge projection of SHAs can indeed speed up earliest query answering on XML streams.

1 Introduction

Projection is necessary for running automata on words, trees, hedges or nested words efficiently without having to evaluate irrelevant parts of the input structure. Projection is most relevant for XML processing as already noticed by [15,8,14]. 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 [11]. When it comes to processing XML streams, quite some algorithms [13,16,10,6] are based on nested word automata (NWAs), for which an efficient projection algorithm exists [20].

More recently, it was noticed that stepwise hedge automata (SHA) [19] have important advantages over NWAs when it comes to determinization and earliest query answering [10]. SHAs are a recent variant of standard hedge automata that go back to the sixties [5,21]. They mix up bottom-up processing of standard tree automata with the left-to-right processing of finite word automata (NFAs), but do neither support top-down processing nor have an explicit stack in contrast to NWAs. In particular, it could be shown that earliest query answering for regular queries defined by deterministic SHAs [3] has a lower worst case complexity than for deterministic NWAs [10]. SHAs have the advantage that the set of states that are accessible over some hedge from a given set of start states can be computed in linear time, while for NWAs this requires cubic time.

Based on deterministic SHAs, earliest query answering for regular queries became feasible in practice [3], as shown for a collection of deterministic SHAs for real word regular XPath queries on XML documents [2]. On the other hand side, it is still experimentally slower than the best non-earliest approaches [6]. We believe that this is due to the fact that projection algorithms for SHA evaluation

are missing. Projecting in-memory evaluation assumes that the full graph of the input hedge is constructed at beforehand. Nevertheless, projection may still save time, if one has to run several queries on the same input hedge, or, if the graph got constructed for different reasons anyway. In the streaming case with subhedge projection, the situation is similar: the whole input hedge on the stream needs to be parsed. But only for the nodes that are not projected away, the automaton transitions need to be computed. Given that pure parsing is by two or three orders of magnitude faster, one can save considerable time as noticed in [20].

Consider the example of the XPath filter `[self::list][child::item]` that is satisfied by an XML document if its root is an `list` element that has some `item` child. When evaluating this filter on an XML document, it is sufficient to inspect its roots for having label `list` and then all its children until some `item` is found. The subhedges of these children can be projected away. 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. It would not be sufficient to map SHAs to NWAs and use their projecting evaluators [20]. The NWAs obtained by compilation from SHAs do not push any information top-down, so no projection is enabled. Thus, the objective of the present paper is to develop evaluators with subhedge projection for SHAs.

As more general machines we propose *downward stepwise hedge automata* (SHA$^{\downarrow}$s), a variant of SHAs that support top-down processing in addition. They are basically Neumann and Seidl's pushdown forest automata [17], except that they apply to unlabeled hedges instead of labeled forests. NWAs are known to operate similarly on nested words [9], while allowing for 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. Alternatively, subtree projecting evaluators for SHA$^{\downarrow}$s could be obtained by compiling them to NWAs, distinguishing irrelevant subtrees there [20], and using them for subtree projecting evaluation via projecting NWAs.

As a first and main contribution, we show how to compile SHAs to SHA$^{\downarrow}$s so that one can distinguish appropriate subhedge projection states. The idea is to distinguish contexts in which 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. 1, which our compiler maps to the SHA$^{\downarrow}$ in Fig. 2 (up to renaming of states). The context information made explicit is about the levels of the states. This permits us to distinguish a projection state Π taken from level 2 on in which subhedges can be ignored.

We prove the soundness of our compiler given in the appendix. The proof is based on an nontrivial invariant, that we establish for a slight adaptation of the original projection algorithm published at FCT [4]. We also note that our compiler may in the worst case increase the size of the automata exponentially. Therefore, we avoid constructing the SHA$^{\downarrow}$s statically but rather construct only

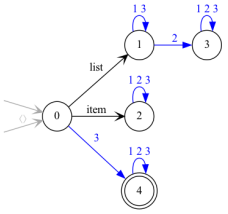


Fig. 1: A unique minimal deterministic SHA (with initial state equal tree initial state) for the XPath filter `[self::list][child::item]`.

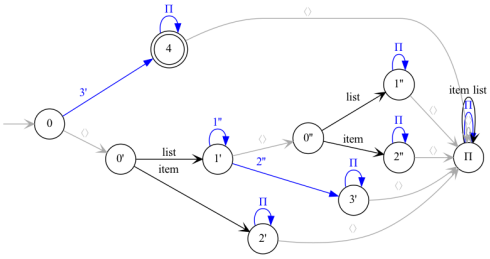


Fig. 2: The deterministic SHA$^{\downarrow}$ with subhedge projection state Π obtained by our compiler.

the needed part of the SHA$^{\downarrow}$s dynamically on the fly when using it to evaluate some hedge with subhedge projection.

Our second contribution is a refinement of the compiler from SHAs to SHA$^{\downarrow}$s for distinguishing safe states for rejection and selection. In this way, we obtain an earliest membership tester for deterministic SHAs in streaming mode which improves the recent earliest membership tester of [3] with subhedge projection. The property of being earliest carries over from there. We lifted this earliest membership tester to an earliest query answering algorithm with subhedge projection for monadic queries defined by deterministic SHAs but omit the details.

Our third contribution is an implementation and experimental evaluation of an earliest query answering algorithm for dSHAs with subhedge projection (not only of earliest membership testing), by introducing subhedge projection into the AStream tool [3]. For the evaluation, we consider the deterministic SHAs constructed with the compiler from [19] for the forward regular XPath queries of the XPathMark benchmark [7] and real-world XPath queries [2]. It turns out that we can reduce the running time for all regular XPath queries that contain only child axes considerably since large parts of the input hedges can be projected away. For such XPath queries, the earliest query answering algorithm of AStream with projection becomes competitive in efficiency with the best existing streaming algorithm from QuiXPath [6] (which is non-earliest on some queries though). The win is smaller for XPath queries with descendant axis, where only few subhedge projection is possible.

Outline. After some preliminaries in Section 2 and 3. In Section 4 we introduce SHA$^{\downarrow}$s and show that they enable in-memory evaluation with subhedge projection. In Section 5 we show how to compile SHAs to SHA$^{\downarrow}$s with subhedge projection states. Streaming evaluators for SHA$^{\downarrow}$s with subhedge projection follow in Section 6. Section 7 improves the compiler from SHAs to SHA$^{\downarrow}$s for obtaining an earliest membership tester. Section 8 discusses our practical experiments.

Publication Comments. This orignial version of this paper paper was published at FCT [4] without appendix. The appendix of the present longer version [1] contains supplementary material including in particular the soundness proof of the safe-no-change subhedge projection algorithm. It also contains further discussion on related work.

2 Preliminaries

Let A and B be sets and $r \subseteq A \times B$ a binary relation. The domain of r is $dom(r) = \{a \in A \mid \exists b \in B.\ (a,b) \in r\}$. We call r total if $dom(r) = A$. A partial function $f : A \hookrightarrow B$ is a relation $f \subseteq A \times B$ that is functional. A total function $f : A \to B$ is a partial function $f : A \hookrightarrow B$ that is total.
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 $\Sigma^* = \cup_{n\in\mathbb{N}}\Sigma^n$. A word $(a_1,\ldots,a_n) \in \Sigma^n$ where $n \in \mathbb{N}$ is written as $a_1 \ldots a_n$. 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 $v_1, v_2 \in \Sigma^*$.
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:

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

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 $\mathcal{H}_\Sigma$, i.e., $\Sigma^* \ni aab = a \cdot a \cdot b \in \mathcal{H}_\Sigma$.
Nested Words. Hedges can be identified with nested words, i.e., words over the alphabet $\hat{\Sigma} = \Sigma \cup \{\langle, \rangle\}$ in which all parentheses are well-nested. This is done by the function $nw(h) : \mathcal{H}_\Sigma \to (\Sigma \cup \{\langle, \rangle\})^*$ such that: $nw(\varepsilon) = \varepsilon$, $nw(\langle h \rangle) = \langle \cdot\, nw(h) \cdot \rangle$, $nw(a) = a$, and $nw(h \cdot h') = nw(h) \cdot nw(h')$.

3 Stepwise Hedge Automata (SHAs)

Stepwise hedge automata (SHAs) are automata for hedges mixing up bottom-up tree automata and left-to-right word automata.

Definition 1. *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 $@^\Delta : (\mathcal{Q} \times \mathcal{Q}) \times \mathcal{Q}$. A* SHA *is deterministic or equivalently a d*SHA *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 state $q \in \mathcal{Q}$, 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: $q@p \to q'$ in Δ. And if $q \in \langle\rangle^\Delta \in \mathcal{Q}$ then we have a tree initial rule that we denote as $\xrightarrow{\langle\rangle} q$ in Δ.

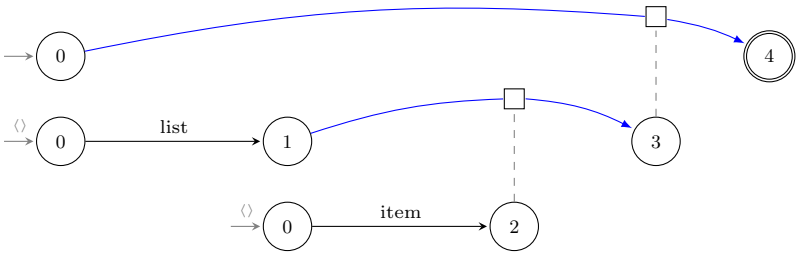


Fig. 3: A successful run of the SHA in Fig. 1 on $\langle list \cdot \langle item \rangle \rangle$.

For any hedge $h \in \mathcal{H}_\Sigma$ we define the transition relation $\xrightarrow{h}$ wrt Δ such that for all $q, q', p, p' \in \mathcal{Q}$, $a \in \Sigma$, and $h, h' \in \mathcal{H}_\Sigma$:

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

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

A run of the dSHA in Fig. 1 on the tree $\langle h \rangle$ with subhedge $h = list \cdot \langle item \rangle$ is illustrated graphically in Fig. 2. It justifies the transition $0 \xrightarrow{\langle h \rangle} 4$ wrt Δ. The run starts on the top-most level of $\langle h \rangle$ in the initial state 0 of the automaton. The run on the topmost level is suspended immediately. Instead, a run on the tree's subhedge h on the level below is started in the tree initial state, which is 0 since $\xrightarrow{\langle\rangle} 0$ in Δ. This run eventually ends up in state 3 justifying the transition $0 \xrightarrow{h} 3$ wrt Δ. The run of the upper level is then resumed from state 0. Given that $0@3 \to 4$ in Δ it continues in state 4. In the graph, this instance of the suspension/resumption mechanism is illustrated by the box in the edge $0 \longrightarrow\!\!\square\!\!\longrightarrow 4$. The box stands for a future value. Eventually, the box is filled by state 3, as illustrated by $3 \; \text{- -}\square$ so that the computation can continue. But in state 4, the upper hedge ends. Since state 4 is final the run ends successfully. The run on the subhedge justifying $0 \xrightarrow{h} 3$ wrt Δ works in analogy.

A hedge is accepted if its transition started in some initial state reaches some final state. The language $\mathcal{L}(A)$ is the set of all accepted hedges:

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

For any subset $Q \subseteq \mathcal{Q}$ and hedge $h \in \mathcal{H}_\Sigma$ we define the in-memory evaluation: $[\![h]\!](Q) = \{q' \mid q \xrightarrow{h} q' \text{ wrt } \Delta,\ q \in Q\}$. An in-memory membership tester for $h \in \mathcal{L}(A)$ can be obtained by computing $[\![h]\!](I)$ by applying the transition relation to all elements recursively and testing whether it contains some final state in F.

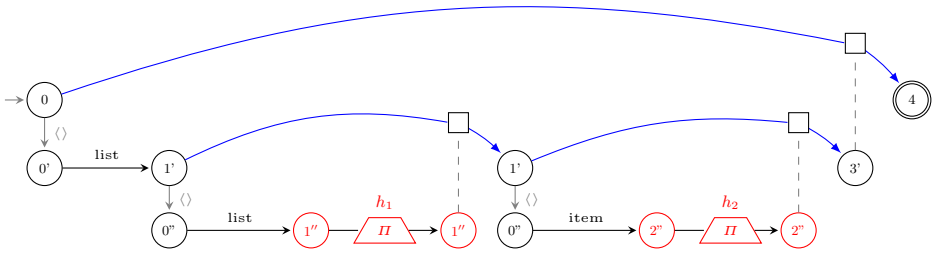


Fig. 4: A run of the SHA$^{\downarrow}$ in Fig. 2 on $\langle list \cdot \langle list \cdot h_1 \rangle \cdot \langle item \cdot h_2 \rangle \rangle$.

4 Downward Stepwise Hedge Automata (SHA$^{\downarrow}$s)

SHAs process information bottom-up and left-to-right exclusively. We next propose an extension to downward stepwise hedge automata with the ability to pass finite state information top-down. These can also be seen as an extension of Neumann and Seidl's pushdown forest automata [18] from (labeled) forests to (unlabeled) hedges.

Definition 2. *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* $@^{\Delta} : (\mathcal{Q} \times \mathcal{Q}) \times \mathcal{Q}$*. A* SHA$^{\downarrow}$ *is deterministic or equivalently a d*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: $q \xrightarrow{\langle\rangle} q'$ in Δ. So here the state q' where the evaluation of a subhedge starts depends on the state q of the parent. The definition of the transition relation and thus the evaluator of a SHA$^{\downarrow}$ differs from that of a SHA by the following equation:

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

This means that the evaluation of the subhedge h starts in some state p such that $q \xrightarrow{\langle\rangle} p$ in Δ. So the restart state p now depends 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.

An example of an in-memory evaluation on the *d*SHA$^{\downarrow}$ in Fig. 2 for the filter `[self::list][child::item]` is shown in Fig. 4. The run of SHA$^{\downarrow}$s works quite similarly to the runs of SHAs, just that when restarting a computation in the subhedge of some tree in state q, then it will start in some state p such that $q \xrightarrow{\langle\rangle} p$ (rather than in some tree initial state that is independent of q). This

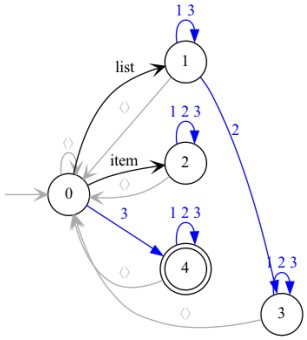


Fig. 5: The SHA$^{\downarrow}$ A^{down} for the dSHA A in Fig. 1.

can be noticed for example when opening the first subtree labeled with `item` where a transition rule $1' \xrightarrow{\langle\rangle} 0''$ is applied. One can see that all nodes of the subtrees h_1 and h_2 are evaluated to the projection state Π, which holds finite-state information on the current level that was passed top-down.

Any SHA $\mathcal{A} = (\Sigma, \mathcal{Q}, \Delta, I, F)$ can be mapped to a SHA$^{\downarrow}$ $A^{down} = (\Sigma, \mathcal{Q}, \Delta^{down}, I, F)$ with the same runs and language. The only change is described by the following rule.

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

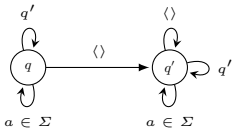
Independently of the current state $q \in \mathcal{Q}$, the SHA$^{\downarrow}$ A^{down} can start the evaluation of the subhedge of a subtree in any open tree state $p \in \Delta$. Note that the conversion preserves determinism. For illustration, the dSHA$^{\downarrow}$ for dSHA from Fig. 1 of the introduction is given in Fig. 5. As for other kinds of automata, making them multi-way does not add expressiveness. So we can convert any dSHA$^{\downarrow}$ A into an equivalent SHA by introducing nondeterminism . Since SHAs can be determinized in at most exponential time, the same holds for SHA$^{\downarrow}$s. It is sufficient to convert it to a SHA, determinize it, and identify the resulting dSHA with a dSHA$^{\downarrow}$.

We next show how to get subhedge projection for SHA$^{\downarrow}$s. Two notions will be relevant here, automata completeness and subhedge projection states.

So let $\mathcal{A} = (\Sigma, \mathcal{Q}, \Delta, I, F)$ be a SHA$^{\downarrow}$. We call Δ complete if all its relations $(a^{\Delta})_{a\in\Sigma}$, $\langle\rangle^{\Delta}$ and $@^{\Delta}$ are total. We call A complete if Δ is complete and $I \neq \emptyset$.

Definition 3. *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 with q on the leftmost position is included in:*

$$\{q \xrightarrow{\langle\rangle} q',\ q@q' \to q,\ q' \xrightarrow{\langle\rangle} q',\ q'@q' \to q'\}$$
$$\cup\{q' \xrightarrow{a} q',\ q \xrightarrow{a} q \mid a \in \Sigma\}$$



In the example SHA$^{\downarrow}$ in Fig. 2 Π is a subhedge projection state with witness Π, but also the states $3'$, 4, and $2''$ are subhedge projection states with witness Π. Note that only inclusion holds for the latter but not equality since this automaton is not complete.

For complete SHA$^{\downarrow}$s A, the above set must be equal to the set of transition rules of Δ with q or q' on the leftmost position. In the soundness expressed in Proposition 4, completeness will be assumed and the proof relies on it. In the examples, however, we will consider automata that are not complete. Still they are "sufficiently complete" to illustrate the constructions.

Note that a subhedge projection state q may be equal to its witness q'. Therefore the witness q' of any subhedge projection state is itself a subhedge projection state with witness q'.

Let $\mathcal{P} \subseteq \mathcal{Q}$ be a subset of subhedge projection states of Δ. We define the transition relation with projection $\xrightarrow{h}_{\mathcal{P}} \subseteq \mathcal{Q} \times \mathcal{Q}$ with respect to Δ such that for all hedges $h, h' \in \mathcal{H}_\Sigma$ and letters $a \in \Sigma$:

$$\frac{q \in \mathcal{P}}{q \xrightarrow{h}_{\mathcal{P}} q \text{ wrt } \Delta} \qquad \frac{q \notin \mathcal{P} \quad q \xrightarrow{a} q' \text{ in } \Delta}{q \xrightarrow{a}_{\mathcal{P}} q' \text{ wrt } \Delta} \qquad \frac{q \notin \mathcal{P}}{q \xrightarrow{\varepsilon}_{\mathcal{P}} q \text{ wrt } \Delta}$$

$$\frac{q \notin \mathcal{P} \quad q \xrightarrow{h}_{\mathcal{P}} q' \text{ wrt } \Delta \quad q' \xrightarrow{h'}_{\mathcal{P}} q'' \text{ wrt } \Delta}{q \xrightarrow{h \cdot h'}_{\mathcal{P}} q'' \text{ wrt } \Delta}$$

$$\frac{q \notin \mathcal{P} \quad q \xrightarrow{\langle\rangle} p \text{ in } \Delta \quad p \xrightarrow{h}_{\mathcal{P}} p' \quad q@p' \to q' \text{ in } \Delta}{q \xrightarrow{\langle h \rangle}_{\mathcal{P}} q'' \text{ wrt } \Delta}$$

Transitions with respect to $\mathcal{P}$ stay in states $q \in \mathcal{P}$ until the end of the current subhedge is reached. This is correct if p is a subhedge projection state since transitions without subhedge projection don't change state p nor if the run is not blocking.

Proposition 4. *Let $\mathcal{A} = (\Sigma, \mathcal{Q}, \Delta, I, F)$ be a complete* SHA$^{\downarrow}$ *and $\mathcal{P}$ a subset of subhedge projection states for Δ. Then for all hedges $h \in \mathcal{H}_\Sigma$ and states $q, q' \in \mathcal{Q}$: $q \xrightarrow{h} q'$ wrt Δ iff $q \xrightarrow{h}_{\mathcal{P}} q'$ wrt Δ.*

For any subset $Q \subseteq \mathcal{Q}$ and hedge $h \in \mathcal{H}_\Sigma$, we define the in-memory evaluation with subhedge projection: $[\![h]\!]_{\mathcal{P}}(Q) = \{q' \in \mathcal{Q} \mid q \xrightarrow{h}_{\mathcal{P}} q' \text{ wrt } \Delta,\ q \in Q\}$. An in-memory membership tester for $h \in \mathcal{L}(A)$ with subtree projection can be obtained by computing $[\![h]\!]_{\mathcal{P}}(I)$ and testing whether it contains some state in F.

5 Compiling SHAs to SHA$^{\downarrow}$s with Projection States

We show how to compile any SHA to some SHA$^{\downarrow}$ with subhedge projection states, yielding an evaluator with appropriate subhedge projection for the SHA via the SHA$^{\downarrow}$. This compiler is the most original contribution of the paper.

Let $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ be a SHA. For any set $Q \subseteq \mathcal{Q}$ we define the set $acc^{\Delta}(Q) = \{q' \in \mathcal{Q} \mid \exists\ q \in Q,\ h \in \mathcal{H}_{\Sigma}.\ q \xrightarrow{h} q' \text{ wrt } \Delta\}$. We note that $acc^{\Delta}(Q)$ can be computed in linear time in the size of Δ. We define:

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

If A is complete and deterministic then safety can be used to characterize universal states, since for all $q \in \mathcal{Q}$: $L(A[I/\{q\}]) = \mathcal{H}_{\Sigma}$ if and only if $q \in safe^{\Delta}(F)$. See Lemma 5 of [3]. Note that $safe^{\Delta}(Q)$ can be computed in linear time in the size of Δ. We define the set of states that may no more change by:

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

Note that $q \in no\text{-}change^{\Delta}$ if and only if $acc^{\Delta}(\{q\}) \subseteq \{q\}$. In the example automaton from Fig. 1 we have $no\text{-}change^{\Delta} = \{2, 3, 4\}$. For any state $q \in \mathcal{Q}$ and subset of states $Q \subseteq \mathcal{Q}$ we define:

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

A state belongs to $s\text{-}down^{\Delta}(q, Q)$ if it can either no more change or if all accessible states p satisfy that $q@^{\Delta}p \subseteq Q$.

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

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

A pair (q, P) means that the evaluator in state q may project subhedges if $q \in P$ since these will no more lead to any relevant change. The sets of initial and final states are defined as follows:

$$I^{\pi} = \{(q, \emptyset) \mid q \in I\} \qquad F^{\pi} = \{(q, \emptyset) \mid q \in F\}$$

How to generated the transition rules of A^{π} from those of A is described in Fig. 6.

When applied to the SHA in Fig. 1 for `[self::list][child::item]`, the construction yields the SHA$^{\downarrow}$ in Fig. 7 which is indeed equal to the SHA$^{\downarrow}$ from Fig. 2 up to state renaming. When run on the hedge $\langle list \cdot \langle list \cdot h_1 \rangle \cdot \langle item \cdot h_2 \rangle \rangle$ as shown in Fig. 4, 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.

Proposition 5 (Soundness). *$\mathcal{L}(A^{\pi}) = \mathcal{L}(A)$ for any complete* SHA *A.*

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

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

$$\frac{\xrightarrow{\langle\rangle} q' \text{ in } \Delta \quad q \notin P \cup \mathit{no\text{-}change}^{\Delta}}{(q,P) \xrightarrow{\langle\rangle} (q', \mathit{s\text{-}no\text{-}change}^{\Delta}(q)) \text{ in } \Delta^{\pi}} \qquad \frac{q \in P \cup \mathit{no\text{-}change}^{\Delta}}{(q,P) \xrightarrow{\langle\rangle} \Pi \text{ in } \Delta^{\pi}}$$

$$\frac{q@p \to q' \text{ in } \Delta \quad q \notin P \cup \mathit{no\text{-}change}^{\Delta}}{(q,P)@(p, \mathit{s\text{-}no\text{-}change}^{\Delta}(q)) \to (q',P) \text{ in } \Delta^{\pi}} \qquad \frac{q \in P \cup \mathit{no\text{-}change}^{\Delta}}{(q,P)@\Pi \to (q,P) \text{ in } \Delta^{\pi}}$$

$$\frac{a \in \Sigma}{\Pi \xrightarrow{a} \Pi \text{ in } \Delta^{\pi}} \qquad \frac{\mathit{true}}{\Pi@\Pi \to \Pi \text{ in } \Delta^{\pi}} \qquad \frac{\mathit{true}}{\Pi \xrightarrow{\langle\rangle} \Pi \text{ in } \Delta^{\pi}}$$

Fig. 6: The transition rules of the SHA$^{\downarrow}$ A^{π} inferred from those of the SHA A.

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

Example 6. *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 family of hedge languages $H_n \in \mathcal{H}_{\Sigma}$ *such that each node of* $h \in H_n$ *is labeled by one symbol in* Σ *and so that the sequence of labels of some root-to-leave path of* h_n *belongs to* L_n*. Note that* H_n *can be recognized in a bottom-up manner by the d*SHA 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 a* 2^{n+1} *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* 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^{π} *must be exponential in the size of* A_n*.*

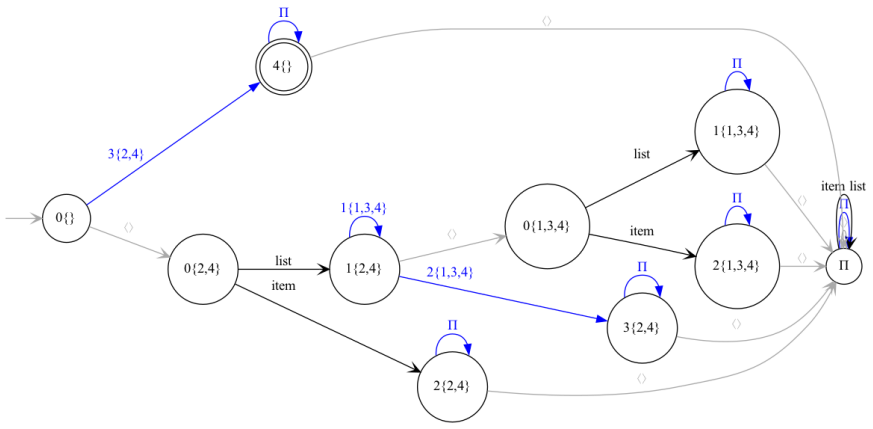


Fig. 7: The $d\text{SHA}^{\downarrow}$ A^{π} constructed from the dSHA A in Fig. 1 except for useless state transitions leading out of the schema of our application. Note that $\mathit{no\text{-}change}^{\Delta} = \{2, 3, 4\}$. It is equal to the $\text{SHA}^{\downarrow}$ in Fig. 2 up to the state renaming $0 = (0, \{\})$, $0' = (0, \{2, 4\})$, $0'' = (0, \{1, 3, 4\})$, $1' = (1, \{2, 4\})$, $1'' = (1, \{1, 3, 4\})$, $2' = (2, \{2, 4\})$, $2'' = (2, \{1, 3, 4\})$, $3' = (3, \{2, 4\})$, $4 = (4, \{\})$.

6 Streaming Evaluators for SHA$^{\downarrow}$s

Any $\text{SHA}^{\downarrow}$ yields a visibly pushdown machine [12] that evaluates nested words in a streaming manner. The same property was already noticed for Neumann and Seidl's pushdown forest automata [9].

Let $\mathcal{A} = (\Sigma, \mathcal{Q}, \Delta, q_{init}, F)$ be a $\text{SHA}^{\downarrow}$. A configuration of the corresponding visibly pushdown machine is a pair in $\mathcal{K} = \mathcal{Q} \times \mathcal{Q}^*$ containing a state and a stack of states. For any word $v \in \hat{\Sigma}^*$ we define the transition relation of the visibly pushdown machine $\xrightarrow{v}{}^{\text{str}} \subseteq \mathcal{K} \times \mathcal{K}$ such that for all $q, q' \in \mathcal{Q}$ and $\sigma \in \mathcal{Q}^*$:

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

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

The same visibly pushdown machine can be obtained by compiling the SHA to an NWA. In analogy to Theorem 4 of [9], we can show for any hedge h that the streaming transition relation $\xrightarrow{nw(h)}{}^{\text{str}}$ wrt Δ is correct for its in-memory transition relation $\xrightarrow{h}$ wrt Δ:

Proposition 7. $L(A) = \{h \in \mathcal{H}_\Sigma \mid (q, \varepsilon) \xrightarrow{nw(h)}{}^{str} (q', \varepsilon) \ \mathit{wrt}\ \Delta,\ q \in I,\ q' \in F\}$.

Any nested word $v \in \hat{\Sigma}^*$ can be evaluated in streaming mode on any subset of configurations $K \subseteq \mathcal{K}$: $[\![v]\!]_{str}(K) = \{(q', \sigma') \mid (q, \sigma) \xrightarrow{v}^{\text{str}} (q', \sigma') \text{ wrt } \Delta,\ (q, \sigma) \in K\}$. So any hedge can be evaluated in streaming mode by computing $[\![nw(h)]\!]_{str}(I \times \{\varepsilon\})$. The hedge is accepted if it can reach some final configuration in $F \times \{\varepsilon\}$.

Going one step further, we show how to enhance the streaming evaluator of an SHA$^{\downarrow}$ with subhedge projection, in analogy to the in-memory evaluator. This approach yields a similar result in a more direct manner, as obtained by mapping SHA$^{\downarrow}$s to NWAs, identifying subtree projection states there, and mapping NWAs with subtree projection states to projecting NWAs [20].

Let $\mathcal{P} \subseteq \mathcal{Q}$ be the subset of subhedge projection states of Δ. We define a transition relation with subhedge projection $\xrightarrow{h}{}^{\text{str}}_{\mathcal{P}} \subseteq \mathcal{K} \times \mathcal{K}$ with respect to Δ such that for all nested words $v, v' \in \mathcal{N}_\Sigma$, letters $a \in \Sigma$, states $p, q, q', q'' \in \mathcal{Q}$ and stacks $\sigma, \sigma', \sigma'' \in \mathcal{Q}^*$:

$$\frac{q \in \mathcal{P}}{(q,\sigma) \xrightarrow{v}{}^{\text{str}}_{\mathcal{P}} (q,\sigma) \text{ wrt } \Delta} \qquad \frac{q \notin \mathcal{P} \qquad q \xrightarrow{a} q' \text{ in } \Delta}{(q,\sigma) \xrightarrow{a}{}^{\text{str}}_{\mathcal{P}} (q',\sigma) \text{ wrt } \Delta} \qquad \frac{q \notin \mathcal{P}}{q \xrightarrow{\varepsilon}{}^{\text{str}}_{\mathcal{P}} q \text{ wrt } \Delta}$$

$$\frac{q \notin \mathcal{P} \qquad (q,\sigma) \xrightarrow{v}{}^{\text{str}}_{\mathcal{P}} (q',\sigma') \text{ wrt } \Delta \qquad (q',\sigma') \xrightarrow{v'}{}^{\text{str}}_{\mathcal{P}} (q'',\sigma'') \text{ wrt } \Delta}{(q,\sigma) \xrightarrow{v \cdot v'}{}^{\text{str}}_{\mathcal{P}} (q'',\sigma'') \text{ wrt } \Delta}$$

$$\frac{q \notin \mathcal{P} \qquad q \xrightarrow{\langle\rangle} q' \text{ in } \Delta}{(q,\sigma) \xrightarrow{\langle}{}^{\text{str}}_{\mathcal{P}} (q', \sigma \cdot q) \text{ wrt } \Delta} \qquad \frac{p \notin \mathcal{P} \qquad q@p \to q' \text{ in } \Delta}{(p, \sigma \cdot q) \xrightarrow{\rangle}{}^{\text{str}}_{\mathcal{P}} (q', \sigma) \text{ wrt } \Delta}$$

The projecting transition relation stays in a configuration with a projection state until the end of the current subhedge is reached. This is correct since the state of the non-projecting transition relation would not change the state either, while the visible stack comes back to its original value after the evaluation of a nested word (that by definition is well-nested).

Proposition 8. *Let v be a word in $\hat{\Sigma}^*$, Δ a set of transition rules of a complete* SHA$^{\downarrow}$ *with state set $\mathcal{Q}$, $q \in \mathcal{Q}$ a state and $\sigma \in \mathcal{Q}^*$ a stack. For any subset $\mathcal{P} \subseteq \mathcal{Q}$ of subhedge projection states of Δ: $(q,\sigma) \xrightarrow{v}{}^{str} (q',\sigma')$ wrt Δ iff $(q,\sigma) \xrightarrow{v}{}^{str}_{\mathcal{P}} (q',\sigma')$ wrt Δ.*

For any subset $K \subseteq \mathcal{K}$ and nested word $v \in \mathcal{N}_\Sigma$ we define the streaming evaluation with subhedge projection:

$$[\![v]\!]^{str}_{\mathcal{P}}(K) = \{(q',\sigma') \mid (q,\sigma) \xrightarrow{v}{}^{\text{str}}_{\mathcal{P}} (q',\sigma') \text{ wrt } \Delta,\ (q,\sigma) \in K\}$$

A streaming membership tester for $h \in \mathcal{L}(A)$ with subtree projection can be obtained by computing $[\![nw(h)]\!]^{str}_{\mathcal{P}}(I \times \{\varepsilon\})$ and testing whether it contains some state in $F \times \{\varepsilon\}$.

7 Earliest Membership with Subhedge Projection

We next enhance our compiler from SHAs to SHA$^{\downarrow}$s for introducing subtree projection such that it can take safe rejection and safe selection into account. The streaming version for deterministic SHAs leads us to an earliest membership tester, which enhances the previous earliest membership tester for dSHAs from [3] with subtree projection.

The idea is as follows: A state is called safe for rejection if whenever the evaluator reaches this state on some subhedge then it can safely reject the hedge independently of the parts that were not yet evaluated. In analogy, a state is safe for selection if whenever the evaluator reaches this state for some subhedge, the full hedge will be accepted.

Consider a dSHA $\mathcal{A} = (\Sigma, \mathcal{Q}, \Delta, I, F)$. The states of our dSHA$^{\downarrow}$ will contain tuples (q, Q, R, S) stating that the evaluator is in state q, that the states in Q are safe no-changes, the states in R are safe for rejection, and the states in S safe for selection. State changes are relevant only if they are not safe for no-change, rejection or selection. Let *sel* be a fresh symbols beside of Π.

We next compile the given SHA A to a SHA$^{\downarrow}$ $A^{\pi}_{e} = (\Sigma, \mathcal{Q}^{\pi}_{e}, \Delta^{\pi}_{e}, I^{\pi}_{e}, F^{\pi}_{e})$. We start with set set of states that are safe for selection $S_0 = \mathit{safe}^{\Delta}(F)$ and respectively safe for rejection $R_0 = \mathit{safe}^{\Delta}(\mathcal{Q} \setminus F)$. The state sets of A^{π}_{e} are then:

$$\mathcal{Q}^{\pi}_{e} = (\mathcal{Q} \times 2^{\mathcal{Q}} \times 2^{\mathcal{Q}} \times 2^{\mathcal{Q}}) \cup \{\Pi, \mathit{sel}\}$$
$$I^{\pi}_{e} = \{(q, \emptyset, R_0, S_0) \mid q \in I,\ q \notin R_0 \cup S_0\} \cup \{\mathit{sel} \mid I \cap S_0 \neq \emptyset\}$$
$$F^{\pi}_{e} = \{(q, \emptyset, R_0, S_0) \mid q \in F\} \cup \{\mathit{sel}\}$$

The transition rules in Δ^{π}_{e} are given by the in Fig. 8. For illustration, reconsider the dSHA from Fig. 1. However, it is not sufficiently complete to obtain the expected results. The problem is that $\mathit{s\text{-}down}^{\Delta}(0, \{4\}) = \{0, 1, 2, 3, 4\}$ there, but only the states 3 is really safe for selection one level down. We therefore add a sink state to it – that we call 5 – yielding the dSHA$^{\downarrow}$ in Fig. **??**. For this dSHA, which is still not complete but complete for the intended schema $\mu X.\langle(\mathit{item}+\mathit{list})\cdot X^*\rangle^*$. We then get $\mathit{s\text{-}down}^{\Delta}(0, \{4\}) = \{3, 4\}$. I may seem counter intuitive that note 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. **??** yields the dSHA$^{\downarrow}$ in Fig. 9. This automaton is the best dSHA$^{\downarrow}$ that we could hope for the XPath filter `[self::list][child::item]`, enabling earliest query answering with earliest rejection and perfect subhedge projection.

Running the SHA$^{\downarrow}$s A^{π}_{e} in streaming mode with subtree projection yields an earliest membership tester for dSHA*s* with subtree projection. A single adaptations are in order. Whenever the safe selection state *sel* is reached, the evaluation can be stopped and the input hedge on the stream is accepted.

Theorem 1. *Let $\mathcal{A} = (\Sigma, \mathcal{Q}, \Delta, I, F)$ and P the set of its projection states of Δ^{π}_{e} with witness either sel or Π. For any hedge $h \in \mathcal{H}_{\Sigma}$ with $[\![h]\!](I) \neq \emptyset$ wrt Δ*

$$\frac{(q,P) \xrightarrow{a} (q',P) \text{ in } \Delta^\pi \quad q' \notin S \cup R}{(q,P,S,R) \xrightarrow{a} (q',P,S,R) \text{ in } \Delta^\pi_e}$$

$$\frac{(q,P) \xrightarrow{\langle\rangle} (q',P') \text{ in } \Delta^\pi \quad q' \notin S \cup R}{(q,P,S,R) \xrightarrow{\langle\rangle} (q',P',s\text{-}down^\Delta(q,S),s\text{-}down^\Delta(q,R)) \text{ in } \Delta^\pi_e}$$

$$\frac{(q,P)@(p,P') \to (q',P) \text{ in } \Delta^\pi \quad q' \notin S \cup R}{(q,P,S,R)@(p,P',s\text{-}down^\Delta(q,S),s\text{-}down^\Delta(q,R)) \to (q',P,S,R) \text{ in } \Delta^\pi_e}$$

$$\frac{(q,P)@\Pi \to (q,P) \text{ in } \Delta^\pi \quad q' \notin S \cup R}{(q,P,S,R)@\Pi \to (q,P,S,R) \text{ in } \Delta^\pi_e}$$

$$\frac{(q,P) \xrightarrow{a} (q',P) \text{ in } \Delta^\pi \quad q' \in S}{(q,P,S,R) \xrightarrow{a} sel \text{ in } \Delta^\pi_e} \qquad \frac{(q,P) \xrightarrow{\langle\rangle} (q',P) \text{ in } \Delta^\pi \quad q' \in S}{(q,P,S,R) \xrightarrow{\langle\rangle} sel \text{ in } \Delta^\pi_e}$$

$$\frac{(q,P)@(p,P') \to (q',P) \text{ in } \Delta^\pi \quad q' \in S}{(q,P,S,R)@(p,P',s\text{-}down^\Delta(q,S),s\text{-}down^\Delta(q,R)) \to sel \text{ in } \Delta^\pi_e}$$

$$\frac{(q,P)@\Pi \to (q,P) \text{ in } \Delta^\pi \quad q' \in S}{(q,P,S,R)@\Pi \to sel \text{ in } \Delta^\pi_e}$$

$$\frac{a \in \Sigma}{\Pi \xrightarrow{a} \Pi \text{ in } \Delta^\pi_e} \qquad \frac{true}{\Pi@\Pi \to \Pi \text{ in } \Delta^\pi_e} \qquad \frac{true}{\Pi \xrightarrow{\langle\rangle} \Pi \text{ in } \Delta^\pi_e}$$

$$\frac{a \in \Sigma}{sel \xrightarrow{a} sel \text{ in } \Delta^\pi_e} \qquad \frac{\mu \in \mathcal{Q}^\pi_e}{sel@\mu \to sel \text{ in } \Delta^\pi_e} \qquad \frac{\mu \in \mathcal{Q}^\pi_e}{\mu@sel \to sel \text{ in } \Delta^\pi_e} \qquad \frac{true}{sel \xrightarrow{\langle\rangle} sel \text{ in } \Delta^\pi_e}$$

Fig. 8: The transition rules of the SHA$^\downarrow$ A^π_e inferred from those of the SHA A.

the streaming evaluation $[\![nw(h)]\!]^{str}_P(I^\pi_e)$ *with respect to* Δ^π_e *checks membership* $h \in \mathcal{L}(A)$ *at the earliest event when streaming* $nw(h)$.

The hedge h is accepted once the evaluator reaches state *sel*. It this doesn't happend online, the truth value of $q \in F$ is returned where q is the state in the final tuple.

Proof (sketch). Let P' be the set of projection states of A that use state *sel* as witness (but not state Π). We obtain an streaming membership tester with earliest selection and earliest rejection (but no safe-no-change subhedge projection) by computing: $[\![h]\!]^{str}_{P'}(I^\pi_e)$ wrt. Δ^π_e Earliest selection is detected by reaching the state *sel*, while earliest rejection happens when the run fails. We notice that this algorithm is basically the same as the earliest membership tester from Proposition 6 of [3], except that it also check for safe rejection. The fact that nested word automata are used there instead of SHA$^\downarrow$s here is not essential. So we can rely on the definitions of earliest membership testing and the result given there.

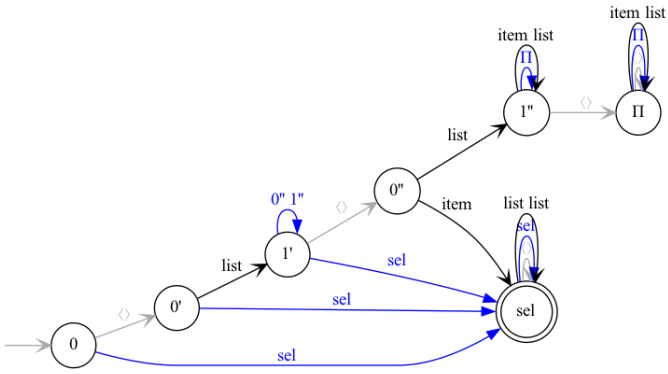


Fig. 9: The earliest $d\text{SHA}^{\downarrow}$ A_e^{π} with safe-no-change subhedge projection for the the SHA A in Fig. 10 adding sink 5 to the dSHA in Fig. 1. The states of A_e^{π} correspond to the following tuples: $0 = (0, \emptyset, \{1,2,3,5\}, \{4\})$, $0' = (0, \emptyset, \{2,4,5\}, \{3,4\})$, $1' = (1, \emptyset, \{2,4,5\}, \{3,4\})$, $0'' = (0, \{1,3,4,5\}, \emptyset, \{2,4,5\})$, and $1'' = (1, \{1,3,4,5\}, \emptyset, \{2,4,5\})$.

In order to add safe-no-change subtree projection, we consider the set of all projection state P of Δ_e^{π} whose witnesses are either Π or *sel*. We then compute: $[\![h]\!]_P^{str}(I_e^{\pi})$ wrt. Δ_e^{π}. The difference is that the evaluator based on P does also ignore subtrees that go to safe-no-change projection states. Clearly, this does not affect earliest selection nor earliest rejection, so we still have an earliest membership tester, but now safe-no-change subtree projection is added.

8 Experimental Evaluation

We integrated subhedge projection into the earliest query answering tool AStream [3]. It is implemented in Scala while computing safety with ABC Datalog.

In order to benchmark AStream 2.01 with subhedge projection for efficiency, and to compare it to AStream 1.01 without projection, we considered the regular XPath queries from the XPathMark [7] $A1 - A8$. We used the deterministic SHAs for all these XPath queries constructed by the compiler from [19]. These were evaluated on XML documents of variable size created by the XPathMark generator. We did further experiments on a sub-corpus of 79 regular XPath queries extracted by Lick and Schmitz from real-world XSLT and XQUERY programs, for which dSHAs are available [2]. These experiments confirm the results presented here, so we don't describe them in detail.

The XPath queries of the XPathMark without descendant axis are A1,A4 and A6-A8. The evaluation time on these queries a 1.2 GB document are reduced

between $88 - 97\%$. In average, it is 92.5%, so the overall time is divided by 12. While the parsing time remains unchanged the gain on the automaton evaluation time is proportional to the percentage of subhedge projection for the respective query. This remains true for the other queries with the descendant axis, just that the projection percentage is much lower.

Finally, we compared AStream with for QuiXPath [6], the best previous streaming tool that can answer A1-A8 in an earliest manner. QuiXPath compiles regular XPath queries to possibly nondeterministic early NWAs, and evaluates them with subtree and descendant projection [20]. QuiXPath is not generally earliest though. On the queries without descendant axis, AStream 1.01 without projection is by a factor of 60 slower than QuiXPath [3]. With subhedge projection in version 2.01, the overhead goes down to a factor of $5 = 60/12$. So our current implementation is close to becoming competitive with the best existing streaming tool while guaranteeing earliest query answering in addition.

9 Conclusion and Future Work

We developed evaluators with subhedge projection for SHAs in in-memory mode and in streaming mode. One 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 a compiler from SHAs to downward SHAs. This compiler propagates safety information about non-changing states, similar to the propagation of safety information proposed for earliest query answering for dSHA queries on nested word streams. We confirmed the usefulness of our novel subhedge projection algorithm for SHAs experimentally. We showed that it can indeed speed up the best previously existing earliest query answering algorithm for dSHA queries on nested word streams, as needed for answering regular XPath queries on XML streams. In future work, we plan to improve on subhedge projection for SHAs with descendant projection for SHAs and to use it for efficient stream processing. Another question is whether and how to obtain completeness results for subhedge projection.

References

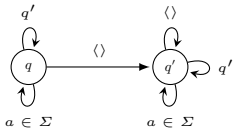
1. A. Al Serhali and J. Niehren. Subhedge projection for stepwise hedge automata. Longer version with appendix of the present paper published at FCT [4].
2. A. Al Serhali and J. Niehren. A Benchmark Collection of Deterministic Automata for XPath Queries. In *XML Prague 2022*, Prague, Czech Republic, June 2022.
3. A. Al Serhali and J. Niehren. Earliest query answering for deterministic stepwise hedge automata. In B. Nagy, editor, *Implementation and Application of Automata - 27th International Conference, CIAA 2023, Famagusta, North Cyprus, September 19-22, 2023, Proceedings*, volume 14151 of *Lecture Notes in Computer Science*, pages 53–65. Springer, 2023.
4. A. Al Serhali and J. Niehren. Subhedge projection for stepwise hedge automata. In H. Fernau and K. Jansen, editors, *Fundamentals of Computation Theory - 24th International Symposium, FCT 2023, Trier, Germany, September 18-21, 2023,*

Proceedings, volume 14292 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2023.
5. H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available online since 1997: `http://tata.gforge.inria.fr`, 2007.
6. D. Debarbieux, O. Gauwin, J. Niehren, T. Sebastian, and M. Zergaoui. Early nested word automata for XPath query answering on XML streams. *Theor. Comput. Sci.*, 578:100–125, 2015.
7. M. Franceschet. Xpathmark performance test. `https://users.dimi.uniud.it/~massimo.franceschet/xpathmark/PTbench.html`. Accessed: 2020-10-25.
8. A. Frisch. Regular tree language recognition with static information. In *Exploring New Frontiers of Theoretical Informatics, IFIP 18th World Computer Congress, TCS 3rd International Conference on Theoretical Computer Science*, pages 661–674, 2004.
9. O. Gauwin, J. Niehren, and Y. Roos. Streaming tree automata. *Information Processing Letters*, 109(1):13–17, 2008.
10. O. Gauwin, J. Niehren, and S. Tison. Earliest query answering for deterministic nested word automata. In *17th International Symposium on Fundamentals of Computer Theory*, volume 5699 of *Lecture Notes in Computer Science*, pages 121–132. Springer Verlag, 2009.
11. M. Kay. The saxon xslt and xquery processor, 2004. https://www.saxonica.com.
12. V. Kumar, P. Madhusudan, and M. Viswanathan. Visibly pushdown automata for streaming XML. In *16th international conference on World Wide Web*, pages 1053–1062. ACM-Press, 2007.
13. P. Madhusudan and M. Viswanathan. Query automata for nested words. In *34th International Symposium on Mathematical Foundations of Computer Science*, volume 5734 of *Lecture Notes in Computer Science*, pages 561–573. Springer Verlag, 2009.
14. S. Maneth and K. Nguyen. XPath whole query optimization. *VLPB Journal*, 3(1):882–893, 2010.
15. A. Marian and J. Siméon. Projecting XML documents. In *VLDB*, pages 213–224, 2003.
16. B. Mozafari, K. Zeng, and C. Zaniolo. High-performance complex event processing over XML streams. In K. S. Candan, Y. Chen, R. T. Snodgrass, L. Gravano, A. Fuxman, K. S. Candan, Y. Chen, R. T. Snodgrass, L. Gravano, and A. Fuxman, editors, *SIGMOD Conference*, pages 253–264. ACM, 2012.
17. A. Neumann and H. Seidl. Locating matches of tree patterns in forests. In *Foundations of Software Technology and Theoretical Computer Science*, volume 1530 of *Lecture Notes in Computer Science*, pages 134–145. Springer Verlag, 1998.
18. A. Neumann and H. Seidl. Locating matches of tree patterns in forests. In *18-th Conference on Foundations of Software Technology and Theoretical Computer Science*, 1998.
19. J. Niehren and M. Sakho. Determinization and minimization of automata for nested words revisited. *Algorithms*, 14(3):68, 2021.
20. T. Sebastian and J. Niehren. Projection for Nested Word Automata Speeds up XPath Evaluation on XML Streams. In *International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, Harrachov, Czech Republic, 2016.
21. J. W. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of automata theory. *Journal of Computer and System Science*, 1:317–322, 1967.

A Proofs for Section 4 (Downward Stepwise Hedge Automata (SHA$^{\downarrow}$s))

Proposition 4. *Let $\mathcal{A} = (\Sigma, \mathcal{Q}, \Delta, I, F)$ be a complete* SHA$^{\downarrow}$ *and $\mathcal{P}$ a subset of subhedge projection states for Δ. Then for all hedges $h \in \mathcal{H}_\Sigma$ and states $q, q' \in \mathcal{Q}$: $q \xrightarrow{h} q'$ wrt Δ iff $q \xrightarrow{h}_{\mathcal{P}} q'$ wrt Δ.*

Proof. If $q \notin \mathcal{P}$, then for any $p \in \mathcal{Q}$, $q \xrightarrow{h} p$ wrt Δ iff $q \xrightarrow{h}_{\mathcal{P}} p$ wrt Δ by definition of the projecting transition relation. If $q \in \mathcal{P}$ then p is a subhedge projection state of Δ, so the set of transition rules of Δ containing q' or with q on the leftmost position are included in the following:



Since Δ is complete, equality holds. Hence for any hedge $h \in \mathcal{H}_\Sigma$ it holds that $q \xrightarrow{h} q$ wrt Δ while $q \not\xrightarrow{h} p$ wrt Δ for all $p \neq q$. The projecting transition relation does the same: $q \xrightarrow{h}_{\mathcal{P}} q$ wrt Δ while $q \not\xrightarrow{h}_{\mathcal{P}} p$ wrt Δ for all $p \neq q$. □

Related Work. We can convert any dSHA$^{\downarrow}$ A into an equivalent SHA $elim^{\downarrow}(A)$ as follows.

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

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

Proposition 9. $\mathcal{L}(A) = \mathcal{L}(elim^{\downarrow}(A))$.

Proof. The construction is analogous to the conversion of NWAs to SHAs [19] or to hedge automata [9]. The correctness proofs for these compilers are standard.

It should be noticed that unique minimization fails for SHA$^{\downarrow}$, as usual for deterministic multiway automata. This even happens for deterministic two-way finite state automata on words. In contrast, the class of dSHAs with the same initial and tree initial state enjoys unique minimization.

Standard hedge automata [21,5,**?**,**?**] have the same expressiveness as Neumann and Seidl's pushdown forest automata [17] and also as Bojanczyk's forest automata (see Section 3.3 of [**?**]). Note, however, that there is an exponential difference in succinctness between SHAs and Bojanczyk's forest automata. Therefore, these forest automata are of quite different nature from those of Neumann and Seidl.

B Proofs for Section 5 (Compiling SHAs to SHA$^{\downarrow}$s with Projection States)

Proposition 5 (Soundness). $\mathcal{L}(A^\pi) = \mathcal{L}(A)$ *for any complete* SHA A.

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

We prove the first inclusion $\mathcal{L}(A) \subseteq \mathcal{L}(A^\pi)$ based on the following three Claims 5.1a, 5.2a, and 5.3a:

Claim 5.1a. $\Pi \xrightarrow{h} \Pi$ wrt Δ^π 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^π in Fig. 6 permitting to always stay in Π for whatever hedge follows.

Claim 5.2a. For all $h \in \mathcal{H}_\Sigma$, $q \in \mathcal{Q}$, and $P \subseteq \mathcal{Q}$ such that $q \in P \cup \mathit{no\text{-}change}^{\Delta}$:

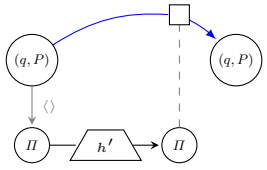
$$(q, P) \xrightarrow{h} (q, P) \text{ wrt } \Delta^\pi$$

We prove Claim 5.2a by induction on the structure of h. Note that the completeness of Δ will be needed.

Case $h = \langle h' \rangle$**.** In this case, we can use Claim 5.1a to show $\Pi \xrightarrow{h'} \Pi$ wrt Δ^π and the inference rules

$$\frac{q \in P \cup \mathit{no\text{-}change}^{\Delta}}{(q, P) \xrightarrow{\langle\rangle} \Pi \text{ in } \Delta^\pi} \qquad \frac{q \in P \cup \mathit{no\text{-}change}^{\Delta}}{(q, P)@\Pi \to (q, P) \text{ in } \Delta^\pi}$$

in order to close the following diagram with respect to Δ^π:



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

Case $h = a$**.** By completeness of Δ there exists some state q' such that $q \xrightarrow{a} q'$ in Δ. Since $q \in P \cup \mathit{no\text{-}change}^{\Delta}$ we can apply the inference rule:

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

This proves this case of the claim.

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

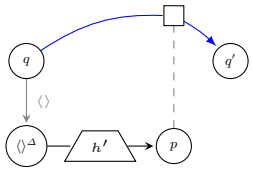
Case $h = h' \cdot h''$**.** By induction hypothesis applied to h' and h'' we have: $(q, P) \xrightarrow{h'} (q, P)$ and $(q, P) \xrightarrow{h''} (q, P)$. Hence $(q, P) \xrightarrow{h' \cdot h''} (q, P)$.

This ends the proof of Claim 5.2. The next claim, in which the induction step is a little more tedious to prove, is the key of the soundness proof.

Claim 5.3a. 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 $acc^\Delta(P) \subseteq P$ and $q \notin P \cup no\text{-}change^\Delta$. If $q \xrightarrow{h} q'$ wrt Δ then there exists q'' such that $(q, P) \xrightarrow{h} (q'', P)$ wrt Δ^π and $(q' = q''$ or $q', q'' \in P)$.

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 some state $p \in \mathcal{Q}$ closing the following diagram:

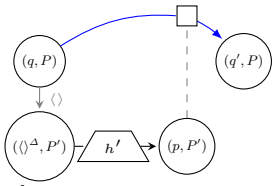


Let $P' = s\text{-}no\text{-}change^\Delta(q)$ and note that $acc^\Delta(P') \subseteq P'$. Since $q \notin P \cup no\text{-}change^\Delta$ we can infer:

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

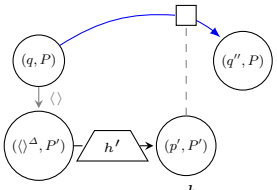
Subcase $\langle\rangle^\Delta \notin P' \cup no\text{-}change^\Delta$**.** The induction hypothesis applies to h' yielding that shows that there exists p' such that $(\langle\rangle^\Delta, P') \xrightarrow{h'} (p, P')$ wrt Δ^π and $p = p' \vee p, p' \in P$. We distinguish the latter two cases:

Subsubcase $(\langle\rangle^\Delta, P') \xrightarrow{h'} (p, P')$ **wrt** Δ^π**.** We can close the diagram as follows:



This shows that $(q, P) \xrightarrow{h} (q', P)$ wrt Δ^π, so the first disjunct of the claim holds for h.

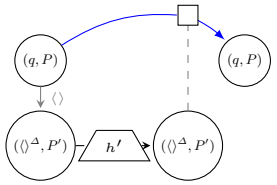
Subsubcase $p \in P' \wedge \exists p' \in P'$. $(\langle\rangle^\Delta, P') \xrightarrow{h'} (p', P')$ **wrt** Δ^π**.** Since Δ is complete, there exists a state q'' such that $q'' \in q@^\Delta p'$. Since both $p, p' \in P'$ we have $q' = q = q''$ by definition of $P' = s\text{-}no\text{-}change^\Delta(q)$. Hence we can close the diagram as follows:



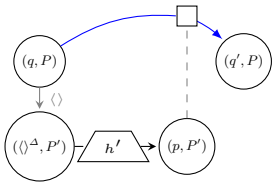
Since $q'' = q'$ this shows that $(q, P) \xrightarrow{h} (q', P)$ wrt Δ^π, so the first disjunct of the claim holds again.

Subcase $\langle\rangle^{\Delta} \in P' \cup \mathit{no\text{-}change}^{\Delta}$**.** Claim 5.2a then shows that $(\langle\rangle^{\Delta}, P') \xrightarrow{h'} (\langle\rangle^{\Delta}, P')$ wrt Δ^π.

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



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



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

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

This shows that $(q, P) \xrightarrow{h} (q', P)$, so claim is valid with the first disjunct.

Case $h = \varepsilon$**.** In this case we have $q = q'$ and $(q, P) \xrightarrow{\varepsilon} (q, P)$ so the first property required by 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 $q \notin P \cup \mathit{no\text{-}change}^{\Delta}$, we apply the induction hypothesis on h_1. This implies that there exists q_1' such that:

$$(q, P) \xrightarrow{h_1} (q_1', P) \text{ wrt } \Delta^\pi \ \wedge (q_1 = q_1' \vee q_1, q_1' \in P)$$

We distinguish the two cases of the disjunction:

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

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

$$\exists q''.(q_1, P) \xrightarrow{h_2} (q'', P) \text{ wrt } \Delta^\pi \wedge (q' = q'' \vee q', q'' \in P)$$

Hence

$$\exists q''.\ (q, P) \xrightarrow{h} (q'', P) \text{ wrt } \Delta^\pi \wedge (q' = q'' \vee q', q'' \in P)$$

Subsubcase $q_1 \in P$**.** By Claim 5.2a, we have $(q_1, P) \xrightarrow{h_2} (q_1, P)$. We also have $q' \in acc(\{q_1\})$ and since we assume $acc(P) \subseteq P$, this implies $q' \in P$. Hence $(q, P) \xrightarrow{h} (q_1, P)$ and $q', q_1 \in P$ implying the claim with the second disjunct valid.

Subcase $q_1, q_1' \in P$**.** Since $q_1' \in P$, Claim 5.2a, implies $(q_1', P) \xrightarrow{h_2} (q_1', P)$ wrt Δ^π. Thus $(q, P) \xrightarrow{h} (q_1', P)$ wrt Δ^π. Since $q' \in acc^\Delta(\{q_1\})$ and $q_1 \in P$ it follows that $q' \in acc^\Delta(P) \subseteq P$. Here we used as in the previous subsubcase that $acc(P) \subseteq P$ is assumed by the claim. Let $q'' = q_1'$. Then we have $(q, P) \xrightarrow{h} (q'', P)$ wrt Δ^π and $q', q'' \in P$ showing the claim.

This ends the proof of Claim 5.3a.

Proof of inclusion $\mathcal{L}(A) \subseteq \mathcal{L}(A^\pi)$**.** 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 $q_0 \in$ *no-change*$^\Delta$**.** By definition of *no-change*$^\Delta$ and since $q \in acc^\Delta(q_0)$ we have $q_0 = q$. Claim 5.2a shows that $(q_0, \emptyset) \xrightarrow{h} (q_0, \emptyset)$ wrt Δ^π and thus $(q_0, \emptyset) \xrightarrow{h} (q, \emptyset)$ so that $h \in \mathcal{L}(A^\pi)$.

Case $q_0 \notin$ *no-change*$^\Delta$**.** Claim 5.3a with $P = \emptyset$ shows that $(q_0, \emptyset) \xrightarrow{h} (q, \emptyset)$ wrt Δ^π and hence $h \in \mathcal{L}(A^\pi)$.

This ends the proof of the first inclusion. We next want to show the inverse inclusion $\mathcal{L}(A^\pi) \subseteq \mathcal{L}(A)$. It will eventually follow from the following three Claims 5.1b., 5.2b, and 5.3b.

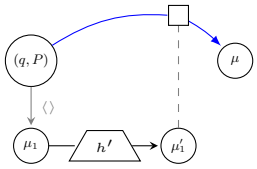
Claim 5.1b. For any hedge h and state $\mu \in \mathcal{Q}^\pi$, if $\Pi \xrightarrow{h} \mu$ wrt Δ^π then $\mu = \Pi$.

The proof is straightforward by induction on the structure of h: the only transitions rules of Δ^π with Π on the left hand side are inferred by the last three rules in Fig. 6. These require to stay in Π whatever hedge follows.

Claim 5.2b. For any hedge h, set $P \subseteq \mathcal{Q}$, state $q \in P \cup$ *no-change*$^\Delta$, and state $\mu \in \mathcal{Q}^\pi$: if $(q, P) \xrightarrow{h} \mu$ wrt Δ^π then $\mu = (q, P)$.

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

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



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

$$\frac{q \in P \cup \mathit{no\text{-}change}^{\Delta}}{(q,P) \xrightarrow{\langle\rangle} \Pi \text{ in } \Delta^\pi}$$

Therefore $\mu_1 = \Pi$. Claim 5.1b shows that $\mu_1' = \Pi$ too. So μ must have been infered by applying the rule:

$$\frac{q \in P \cup \mathit{no\text{-}change}^{\Delta}}{(q,P)@\Pi \to (q,P) \text{ in } \Delta^\pi}$$

So $\mu = (q,P)$ as required.

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

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

Hence, $\mu = (q,P)$.

Case $h = \varepsilon$**.** Obvious.

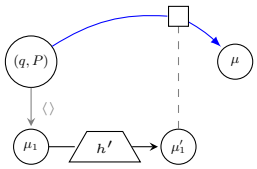
Case $h = h_1 \cdot h_2$**.** There must exist μ_1 such that $(q,P) \xrightarrow{h_1} \mu_1 \xrightarrow{h_2} \mu$ wrt Δ^π. By induction hypothesis applied to h_1, we have $\mu_1 = (q,P)$. We can thus apply the induction hypothesis to h_2 to obtain $\mu_2 = (q,P)$.

This ends the proof of Claim 5.2b. We next need an inverse of Claim 5.3a.

Claim 5.3b. Let $h \in \mathcal{H}_\Sigma$ a hedge, $q \in \mathcal{Q}$ states and $P \subseteq \mathcal{Q}$ a subset of states such that $\mathit{acc}^{\Delta}(P) \subseteq P$ and $q \notin P \cup \mathit{no\text{-}change}^{\Delta}$. If $(q,P) \xrightarrow{h} \mu$ wrt Δ^π then there exists q', q'' such that $\mu = (q',P)$, $q \xrightarrow{h} q''$ wrt Δ and ($q' = q''$ or $q', q'' \in P$).

Proof. Let $P \subseteq \mathcal{Q}$ and $q \in \mathcal{Q}$ such that $\mathit{acc}^{\Delta}(P) \subseteq P$ and $q \notin P \cup \mathit{no\text{-}change}^{\Delta}$. Assume that $(q,P) \xrightarrow{h} \mu$ wrt Δ^π. We prove by induction on h that there exists q' such that $\mu = (q',P)$ and $\exists q''.\ (q' = q'' \vee q', q'' \in P) \wedge q \xrightarrow{h} q''$ wrt Δ.

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

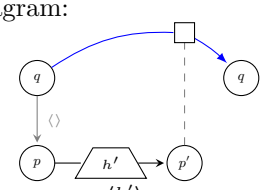


Since $q \notin P \cup \mathit{no\text{-}change}^{\Delta}$, the following inference rule got applied:

$$\frac{\xrightarrow{\langle\rangle} p \text{ in } \Delta \quad q \notin P \cup \mathit{no\text{-}change}^{\Delta}}{(q,P) \xrightarrow{\langle\rangle} (p, \mathit{s\text{-}no\text{-}change}^{\Delta}(q)) \text{ in } \Delta^{\pi}}$$

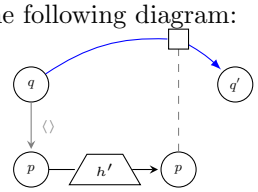
Let $P' = \mathit{s\text{-}no\text{-}change}^{\Delta}(q)$. Hence there exists $p \in \langle\rangle^{\Delta}$ such that $\mu_1 = (p, P')$.
Subcase $p \in P' \cup \mathit{no\text{-}change}^{\Delta}$**.**

Subsubcase $p \in P'$**.** Since $p \in \mathit{acc}^{\Delta}(\{p\})$ and $P' = \mathit{s\text{-}no\text{-}change}^{\Delta}(q)$ we have $q@p = \{q\}$. So $q = q'$. Furthermore, by completeness of Δ there exists p' such that $p \xrightarrow{h'} p'$ wrt Δ. In particular $p' \in \mathit{acc}^{\Delta}(\{p\})$ so by definition of P', it follows that $q@p' \to q$ wrt Δ. Hence, we can close the following diagram:



Since $q = q'$ this shows that $q \xrightarrow{\langle h'\rangle} q'$ wrt Δ.

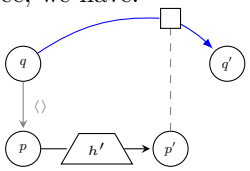
Subsubcase $p \in \mathit{no\text{-}change}^{\Delta}$**.** By completeness of Δ there exists p' such that $p \xrightarrow{h'} p'$ wrt Δ. Since $p \in \mathit{no\text{-}change}^{\Delta}$ it follows that $p' = p$. Hence, we can close the following diagram:



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

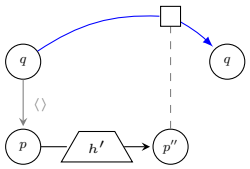
Subcase $p \notin P' \cup \mathit{no\text{-}change}^{\Delta}$**.** By induction hypothesis, there exists p', p'' such that $\mu_1' = (p', P')$, $p \xrightarrow{h'} p''$ wrt Δ, and $(p' = p''$ or $p', p'' \in P'$. By completeness of Δ there exist $q'' \in q@^{\Delta}p''$ and $q' \in q@^{\Delta}p'$.

Subsubcase $p' = p''$**.** Hence, we 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 $q \notin P \cup \mathit{no\text{-}change}^{\Delta}$, the following inference rule must be used:

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

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^{π} shows that there exists μ_1 such that $(q, P) \xrightarrow{h_1} \mu_1 \xrightarrow{h_2} \mu$ wrt Δ^{π}. Since $q \notin P \cup \mathit{no\text{-}change}^{\Delta}$, we can apply the induction hypothesis to h_1. It shows that there exists q_1' and q_1'' such that $\mu_1 = (q_1', P)$ and $q \xrightarrow{h_1} q_1''$ wrt $\Delta \wedge (q_1' = q_1'' \vee q_1', q_1'' \in P)$.

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

Subsubcase $q_1' \notin P \cup \mathit{no\text{-}change}^{\Delta}$**.** In this case, we can apply the induction hypothesis to $(q_1', P) \xrightarrow{h_2} \mu$ wrt Δ^{π} showing the existence of q' such that $\mu = (q', P)$ and $\exists q''.\ q_1' \xrightarrow{h_2} q'' \wedge (q' = q'' \vee q', q'' \in P)$. Hence $\exists q''.\ q \xrightarrow{h} q''$ wrt $\Delta \wedge (q' = q'' \vee q', q'' \in P)$, so the claim holds.

Subsubcase $q_1' \in P \cup \mathit{no\text{-}change}^{\Delta}$**.** Claim 5.2b applied to $(q_1', P) \xrightarrow{h_2} \mu$ wrt Δ^{π} shows that $\mu = (q_1', P)$.

Subcase $q_1', q_1'' \in P$**.** Recall that $(q_1'', P) \xrightarrow{h_2} \mu$ wrt Δ^{π} and $q_1'' \in P$. Claim 5.2b shows that $\mu = (q_1'', P)$ wrt Δ^{π}. We also have $q \xrightarrow{h_1} q_1''$ wrt Δ. By completeness of Δ, there exist a state q'' such that $q_1'' \xrightarrow{h} q''$ wrt Δ. Since $q_1' \in P$ and P is closed by accessibility, we have $q'' \in \mathit{acc}(\{q_1''\}) \subseteq \mathit{acc}(P) \subseteq P$. From $q \xrightarrow{h_1} q_1''$ wrt Δ, we get $q \xrightarrow{h} q''$ wrt Δ. The claim follows since $q_1'', q'' \in P$.

This ends the proof of Claim 5.3b.

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

Case $q_0 \in \mathit{no\text{-}change}^{\Delta}$**.** Claim 5.2b shows that $q = q_0$. Claim 5.2a proves that $q_0 \xrightarrow{h} q_0$ wrt Δ. Hence $q_0 \xrightarrow{h} q$ wrt Δ and thus $h \in \mathcal{L}(A)$.

Case $q_0 \notin \mathit{no\text{-}change}^{\Delta}$**.** Claim 5.3b with $P = \emptyset$ shows that $q_0 \xrightarrow{h} q$ wrt Δ^{π} and hence $h \in \mathcal{L}(A^{\pi})$.

This end the proof of the inverse inclusion, and thus of $L(A) = L(A^{\pi})$. □

C Proofs for Section 6 (Streaming Evaluators for SHA$^{\downarrow}$s)

Proposition 7. $L(A) = \{h \in \mathcal{H}_{\Sigma} \mid (q, \varepsilon) \xrightarrow{nw(h)}{}^{str} (q', \varepsilon) \text{ wrt } \Delta,\ q \in I,\ q' \in F\}$.

Proof. Straightforward.

Proposition 8. *Let v be a word in $\hat{\Sigma}^*$, Δ a set of transition rules of a complete* SHA$^{\downarrow}$ *with state set $\mathcal{Q}$, $q \in \mathcal{Q}$ a state and $\sigma \in \mathcal{Q}^*$ a stack. For any subset $\mathcal{P} \subseteq \mathcal{Q}$ of subhedge projection states of Δ: $(q,\sigma) \xrightarrow{v}{}^{str} (q',\sigma')$ wrt Δ iff $(q,\sigma) \xrightarrow{v}{}^{str}_{\mathcal{P}} (q',\sigma')$ wrt Δ.*

Proof. Analogous to the proof of Proposition 4, i.e., the soundness of the in-memory evaluator with projection for complete SHA$^{\downarrow}$s.

Related Work. A 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) [**?**], whose streaming evaluator is given by a visibly pushdown machine [**?**], previously known as *input driven automata* [**?**,**?**,**?**].

Definition 10. *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 $\rangle^{\Delta} : \mathcal{Q} \times \Gamma \times \mathcal{Q}$. A* NWA *is deterministic or equivalently a d*NWA *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 $q \xrightarrow{a} q'$ as for SHAs, opening rules $q \xrightarrow{\langle\downarrow\gamma} q'$ if $\langle^{\Delta}(q) = (q',\gamma)$ and closing rules $q \xrightarrow{\rangle\uparrow\gamma} q'$ if $\rangle^{\Delta}(q,\gamma) = q'$.

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 $\xrightarrow{v}{}^{\text{str}} \subseteq \mathcal{K} \times \mathcal{K}$ such that for all $q, q' \in \mathcal{Q}$ and $S \in \Gamma^*$:

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

$$\frac{q \xrightarrow{a} q' \text{ in } \Delta}{q \xrightarrow{a}{}^{\text{str}} q' \text{ wrt } \Delta} \qquad \frac{q \xrightarrow{\langle\downarrow\gamma} q' \text{ in } \Delta}{(q,S) \xrightarrow{\langle}{}^{\text{str}} (q',S) \text{ wrt } \Delta} \qquad \frac{q \xrightarrow{\rangle\uparrow\gamma} q' \text{ in } \Delta}{(q,S\cdot\gamma) \xrightarrow{\rangle}{}^{\text{str}} (q',S) \text{ wrt } \Delta}$$

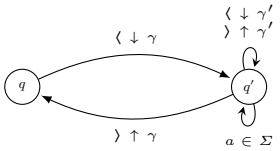
For any dSHA$^{\downarrow}$ $\mathcal{A} = (\Sigma, \mathcal{Q}, \Delta, I, F)$ we can define the dNWA $A^{nwa} = (\Sigma, \mathcal{Q}, \Gamma, \Delta^{nwa}, I^{nwa}, F^{nwa})$ such that $\Gamma = \mathcal{Q}$, while Δ^{nwa} contains for all $a \in \Sigma$ and $q, p \in \mathcal{Q}$ the transition rules:

$$\frac{q \xrightarrow{a} q' \text{ in } \Delta}{q \xrightarrow{a} q' \text{ in } \Delta^{nwa}} \qquad \frac{q \xrightarrow{\langle\rangle} q' \text{ in } \Delta}{q \xrightarrow{\langle\downarrow q} q \text{ in } \Delta^{nwa}} \qquad \frac{p@q \to q' \text{ in } \Delta}{q \xrightarrow{\rangle\uparrow p} q' \text{ in } \Delta^{nwa}}$$

Lemma 11. $\mathcal{L}(A^{nwa}) = \mathcal{L}(A)$.

The runs of SHA$^{\downarrow}$s A and NWAs A^{nwa} can be identified. Projecting evaluators for NWAs were proposed in the context of projecting NWAs [20]. They are based on the following notion of states of irrelevant subtrees for NWAs.

Definition 12 (Variant of Definition 3 of [20]). *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 13. *Any subhedge projection state of a complete* SHA$^{\downarrow}$ *A is a state of irrelevant subtrees of A^{nwa}.*

It was then shown in [20] how to map an NWA with a subset $\mathcal{P}$ 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 notice that projecting NWAs support descendant projection beside of subhedge projection. For SHAs this is left to future research.

D Proofs for Section 7 (Earliest Membership with Subhedge Projection)

Lemma 14. *Let $\mathcal{P}$ be a subset of projection states of Δ_e^π. For any hedge h and prefix v of $nw(h)$, state $q \in \mathcal{Q}$, state sets $Q, R, S \subseteq \mathcal{Q}$, and stack $\sigma \in \mathcal{Q}^*$:*

$$\text{if } (I_e^\pi, \varepsilon)_e^\pi \xrightarrow{v}{}^{str}_{\mathcal{P}} ((q, Q, R, S), \sigma) \text{ wrt. } \Delta_e^\pi \text{ and } q \in S \text{ then } h \in \mathcal{L}(A).$$

E Proofs for Section 8 (Experimental Evaluation)

Table 1: XPathMark list of queries.

Id	XPath Query
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`

Table 2: Timings in seconds for XPathMark queries that have child axis exclusively with QuiXPath and the two versions of our tool.

	QuiXPath	Astream 1.01 nonproj.	Astream 2.01 proj.	Gain: Astream2.01 vs Astream1.01	Factor: QuiXPath/ Astream2.01
A1	11	644.67	72.83	88.7%	*6.62
A4	11.6	723.03	78.5	89.14%	*6.77
A6	10.7	780.78	65.26	91.64%	*6.1
A7	8.6	601.26	24.64	95.9 %	*2.87
A8	8.8	801.96	24.85	96.9%	*2.82
average	10.14	710.34	53.2	92.45%	*5.036

Table 3: Timings in seconds for XPathMark queries that have descendant axis with QuiXPath and the two versions of our tool.

	QuiXPath	Astream 1.01 nonproj.	Astream 2.01 proj.	Gain: Astream2.01 vs Astream1.01	Factor: QuiXPath/ Astream2.01
A2	11.4	569.0112	664.6	-16.8%	*58.3
A3	11.5	673.716	666.124	1.13%	*57.92
A5	12	593.4648	77.785	86.89%	*6.48

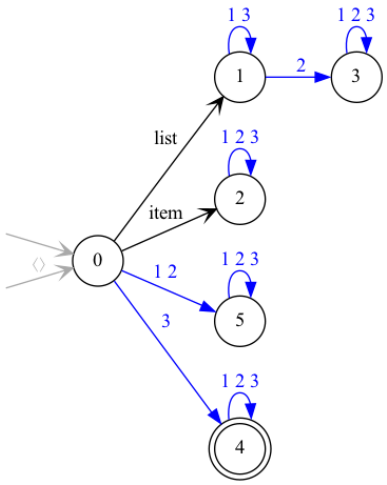


Fig. 10: Adding a sink 5 to the dSHA from Fig. 1 for the XPath filter `[self::list][child::item]`.