# 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 [14,7,13]. 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 [10]. When it comes to processing XML streams, quite some algorithms [12,15,9,5] are based on nested word automata (NWAs), for which an efficient projection algorithm exists [19].

More recently, it was noticed that stepwise hedge automata (SHA) [18] have important advantages over NWAs when it comes to determinization and earliest query answering [9]. SHAs are a recent variant of standard hedge automata that go back to the sixties [4,20]. 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 [9]. 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 [5]. 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 [19].

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 [19]. 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 [16], except that they apply to unlabeled hedges instead of labeled forests. NWAs are known to operate similarly on nested words [8], 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 [19], 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. For instance, reconsider the XPath filter `[self::list][child::item]`. It 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). This compiler permits us to distinguish a projection state $\Pi$ in which subhedges can be ignored. We note however 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 the needed part of the SHA$^{\downarrow}$s dynamically on the fly when using it to evaluate some hedge with subhedge projection. A sketch of the soundness proof of our compiler is provided.

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 im-

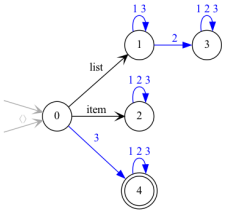



Fig. 1: A unique minimal deterministic SHA (with initial state equal tree initial state) for the XPath filter `[self::list][child::item]`.

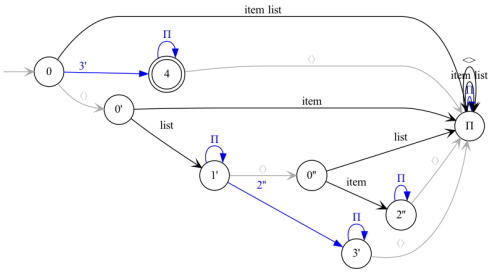



Fig. 2: The deterministic SHA$^{\downarrow}$ with subhedge projection state $\Pi$ obtained by our compiler.

proves 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 [18] for the forward regular XPath queries of the XPathMark benchmark [6] 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 [5] (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. Supplementary material including the soundness proof of our compiler and further discussion on related work can be found in [1].

## 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 $\Sigma$ be a set. The set of words over $\Sigma$ 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 $\Sigma^*$ 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* $\Sigma$ *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 $\Delta$ have three forms: If $(q, q') \in a^\Delta$ then we have a letter rule that we write as $q \xrightarrow{a} q'$ in $\Delta$. If $(q, p, q') \in @^\Delta$ then we have an apply rule that we write as: $q@p \to q'$ in $\Delta$. 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 $\Delta$. For any hedge $h \in \mathcal{H}_\Sigma$ we define the transition relation $\xrightarrow{h}$ wrt $\Delta$ such that for all $q, q' \in \mathcal{Q}$, $a \in \Sigma$, and $h, h' \in \mathcal{H}_\Sigma$:

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

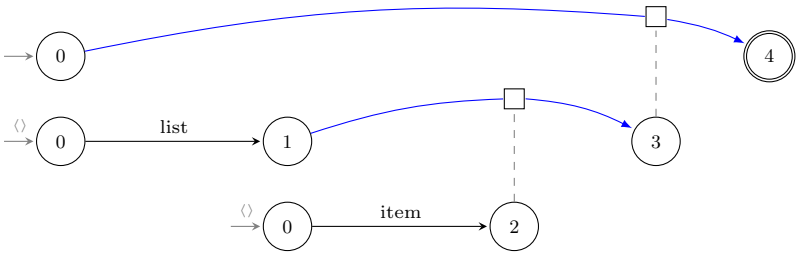



Fig. 3: A successful run of the SHA in Fig. 1 on $\langle list \cdot \langle item \rangle \rangle$.

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

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 $\Delta$. 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 $\Delta$. This run eventually ends up in state 3 justifying the transition $0 \xrightarrow{h} 3$ wrt $\Delta$. The run of the upper level is then resumed from state 0. Given that $0@3 \to 4$ in $\Delta$ 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 $\Delta$ works in analogy.

A hedge is accepted if its transition started in some initial states 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 $F$.

## 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

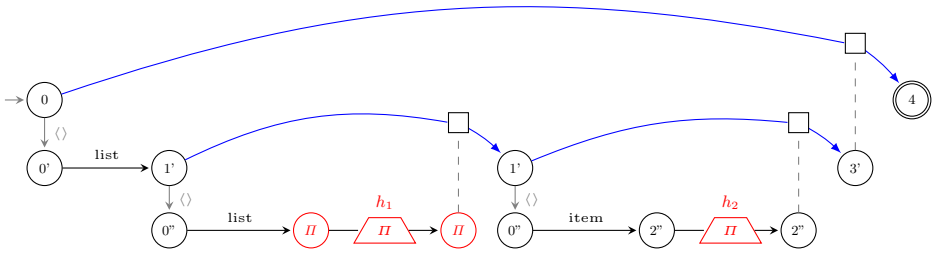



Fig. 4: A run of the SHA$^{\downarrow}$ in Fig. 2 on $\langle list \cdot \langle list \cdot h_1 \rangle \cdot \langle item \cdot h_2 \rangle \rangle$.

of Neumann and Seidl's pushdown forest automata [17] 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* $\Sigma$ *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* $\langle\rangle^{\Delta}$*,* $a^{\Delta}$*, 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 $\Delta$. 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:

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

This means that the evaluation of the subhedge $h$ starts in some state of $q'$ such that $q \xrightarrow{\langle\rangle} q'$ in $\Delta$. So the restart state $q'$ 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 $q'$ such that $q \xrightarrow{\langle\rangle} q'$ (rather than in some tree initial state that is independent of $q$). This 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 $\Pi$, which holds finite-state information on the current level that was passed top-down.

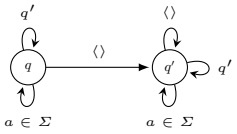
Any SHA can be identified with a SHA$^{\downarrow}$: we fix for this some state $q_0 \in \mathcal{Q}$ arbitrarily and replace $\langle\rangle^{\Delta}$ by $\{q_0\} \times \langle\rangle^{\Delta}$. As for other kinds of automata, making them multi-way does not add expressiveness. So we can convert any $d$SHA$^{\downarrow}$ $A$ 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 $d$SHA$^{\downarrow}$.

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 $\Delta$ complete if all its relations $(a^{\Delta})_{a\in\Sigma}$, $\langle\rangle^{\Delta}$ and $@^{\Delta}$ are total. We call $A$ complete if $\Delta$ is complete and $I \neq \emptyset$.

**Definition 3.** *We call a state $q \in \mathcal{Q}$ a* subhedge projection state of $\Delta$ *if there exists $q' \in \mathcal{Q}$ called the witness of $q$ such that the set of transition rules of $\Delta$ 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 $\Pi$ is a subhedge projection state with witness $\Pi$, but also the states $3'$, 4, and $2''$ are subhedge projection states with witness $\Pi$. 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 $\Delta$ 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 $\Delta$. We define the transition relation with projection $\xrightarrow{h}_{\mathcal{P}} \subseteq \mathcal{Q} \times \mathcal{Q}$ with respect to $\Delta$ such that for all hedges $h, h' \in \mathcal{H}_{\Sigma}$ and letters $a \in \Sigma$:

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

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

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

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 $\Delta$. Then for all hedges $h \in \mathcal{H}_\Sigma$ and states $q, q' \in \mathcal{Q}$: $q \xrightarrow{h} q'$ wrt $\Delta$ iff $q \xrightarrow{h}_{\mathcal{P}} q'$ wrt $\Delta$.*

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 $\Delta$. 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 $\Delta$. We consider pairs $(q, Q)$ consisting of a current state $q$ and a set of forbidden states $Q$ that must not be reached at the end of the hedge. We define:

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

The states in $sdown^{\Delta}(q, Q)$ can only access states $p \in \mathcal{Q}$ such that $q@^{\Delta}p$ is included in $Q$. They are safe for $\{p \in \mathcal{Q} \mid q@^{\Delta}p \subseteq Q\}$. The states in $no\text{-}change^{\Delta}(q, Q)$ safely either go to states $p$ whose application doesn't change $q$ or lead to $Q$. For instance in Fig. 1, $no\text{-}change^{\Delta}(1, \{2, 4\}) = sdown^{\Delta}(1, \{1, 2, 4\}) = \{1, 3, 4\}$. Note that, not only $1@^{\Delta}1 = 1 \subseteq \{1, 2, 4\}$, but also $1@^{\Delta}0 = 1@^{\Delta}4 = \emptyset \subseteq \{1, 2, 4\}$. Therefore, $sdown^{\Delta}(1, \{1, 2, 4\}) = safe^{\Delta}(\{0, 1, 3, 4\}) = \{1, 3, 4\}$.

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 $\Pi$ be a fresh symbol and consider the state set: $\mathcal{Q}^{\pi} = \{\Pi\} \uplus (\mathcal{Q} \times 2^{\mathcal{Q}})$. A pair $(q, Q)$ means that the evaluator is in state $q$ but must not reach any state in $Q$. We next define projection of such pairs with respect to $\Delta$:

$$\pi(q, Q) = \textit{if } q \in Q \textit{ then } \Pi \textit{ else } (q, Q)$$

The sets of initial and final states are defined as follows:

$$I^{\pi} = \{\pi(q, safe^{\Delta}(\mathcal{Q} \setminus F)) \mid q \in I\} \qquad F^{\pi} = \{(q, safe^{\Delta}(\mathcal{Q} \setminus F)) \mid q \in F\}$$

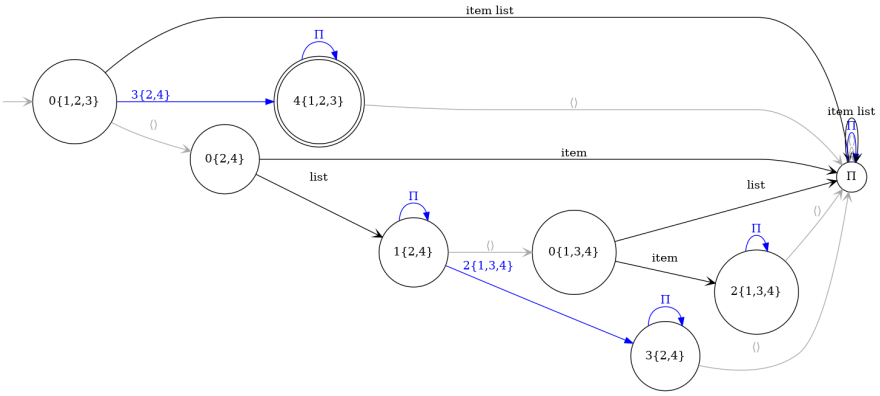



Fig. 5: The $d\text{SHA}^{\downarrow}$ $A^{\pi}$ constructed from the dSHA $A$ in Fig. 1 except for useless state transitions leading out of the schema of our application (see Fig. 6).

So at the beginning, the set of forbidden states are those in $safe^{\Delta}(\mathcal{Q} \setminus F)$. The transition rules in $\Delta^{\pi}$ are given by the following inference rules where $p, q \in \mathcal{Q}$, $P, Q \subseteq \mathcal{Q}$ and $a \in \Sigma$.

$$\frac{q \xrightarrow{a} q' \text{ in } \Delta}{\pi(q,Q) \xrightarrow{a} \pi(q',Q) \text{ in } \Delta^{\pi}} \qquad \frac{\xrightarrow{\langle\rangle} q' \text{ in } \Delta \qquad P = \mathit{no\text{-}change}^{\Delta}(q,Q)}{\pi(q,Q) \xrightarrow{\langle\rangle} \pi(q',P) \text{ in } \Delta^{\pi}}$$

$$\frac{q@p \to q' \text{ in } \Delta \qquad P = \mathit{no\text{-}change}^{\Delta}(q,Q)}{\pi(q,Q)@\pi(p,P) \to \pi(q',Q) \text{ in } \Delta^{\pi}}$$

When going down to a subhedge from state $q$ with forbidden states $Q$, the next set of forbidden states is $\mathit{no\text{-}change}^{\Delta}(q,Q)$. This is where finite state information is passed down. States in $\mathit{no\text{-}change}^{\Delta}(q,Q)$ cannot lead to any change of state $q$, so that subhedge projection can be applied.

When applied to the SHA in Fig. 1 for `[self::list][child::item]`, the construction yields the $\text{SHA}^{\downarrow}$ in Fig. 5 which is indeed equal to the $\text{SHA}^{\downarrow}$ 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 $\Pi$.

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

*Proof.* For the inclusion $\mathcal{L}(A) \subseteq \mathcal{L}(A^{\pi})$ we can show for all hedge $h \in \mathcal{H}_{\Sigma}$ and states $q, q' \in \mathcal{Q}$ that $q \xrightarrow{h} q'$ wrt $\Delta$ implies $q \xrightarrow{h} q'$ wrt $\Delta^{\pi}$. This is straightforward by induction on the structure of $h$.

The inverse inclusion $\mathcal{L}(A^{\pi}) \subseteq \mathcal{L}(A)$ is less obvious. We have to show that not inspecting projected subhedges does not change the language. Intuitively, this is since the projected subhedges are irrelevant for detecting acceptance. They either don't change the state or lead to rejection.

**Claim.** For all $h \in \mathcal{H}_\Sigma$, $q \in \mathcal{Q}$, $Q' \subseteq \mathcal{Q}$, $Q = \mathit{safe}^\Delta(Q')$, and $\mu \in \mathcal{Q}^\pi$ the hypothesis $(q, Q) \xrightarrow{h} \mu$ wrt $\Delta^\pi$ implies:

1. if exists $q' \in \mathcal{Q}$ such that $\mu = (q', Q)$ then $q \xrightarrow{h} q'$ wrt $\Delta$.
2. if $\mu = \Pi$ then there exists $q' \in Q$ such that $q \xrightarrow{h} q'$ wrt $\Delta$.

The lengthy proof is by induction on the structure of $h$. It remains to show that the Claim implies $\mathcal{L}(A^\pi) \subseteq \mathcal{L}(A)$. So let $h \in \mathcal{L}(A^\pi)$. Then there exist $q \in I$ and $q' \in F$ such that for $Q = \mathit{safe}^\Delta(\mathcal{Q} \setminus F)$: $(q, Q) \xrightarrow{h} (q', Q)$ wrt $\Delta^\pi$ Part 1. of the Claim implies that $q \xrightarrow{h} q'$ wrt $\Delta$, so that $h \in \mathcal{L}(A)$. (Part 2 intervenes only for proving Part 1 by induction.) □

The projecting in-memory evaluator of $A^\pi$ will be more efficient than that the nonprojecting evaluator of $A$. Note, however, that the size of $A^\pi$ 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 $A^\pi$ 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 $\Sigma^*$ 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 $\Sigma$ 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 $A_n^\pi$ must be exponential in the size of $A_n$.*

## 6 Streaming Evaluators for SHA$^\downarrow$s

Any SHA$^\downarrow$ yields a visibly pushdown machine [11] that evaluates nested words in a streaming manner. The same property was already noticed for Neumann and Seidl's pushdown forest automata [8].

Let $\mathcal{A} = (\Sigma, \mathcal{Q}, \Delta, I, F)$ be a 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 [8], we can show for any hedge $h$ that the streaming transition relation $\xrightarrow{nw(h)}^{\text{str}}$ wrt $\Delta$ is correct for its in-memory transition relation $\xrightarrow{h}$ wrt $\Delta$:

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

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 [19].

Let $\mathcal{P} \subseteq \mathcal{Q}$ be the subset of subhedge projection states of $\Delta$. We define a transition relation with subhedge projection $\xrightarrow{h}_{\mathcal{P}}^{\text{str}} \subseteq \mathcal{K} \times \mathcal{K}$ with respect to $\Delta$ 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 $\sigma, \sigma', \sigma'' \in \mathcal{Q}^*$:

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

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

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

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 $\hat{\Sigma}^*$, $\Delta$ 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 $\Delta$: $(q,\sigma) \overset{v}{\rightarrow}^{str} (q',\sigma')$ wrt $\Delta$ iff $(q,\sigma) \overset{v}{\rightarrow}^{str}_{\mathcal{P}} (q',\sigma')$ wrt $\Delta$.*

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

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

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.

Given an $\mathcal{A} = (\Sigma, \mathcal{Q}, \Delta, I, F)$, at the beginning all states in $S_0 = \mathit{safe}^{\Delta}(F)$ are safe for selection and all states in $R_0 = \mathit{safe}^{\Delta}(\mathcal{Q} \setminus F)$ are safe for rejection. We will have to update these sets when moving down a tree. The states of our SHA$^{\downarrow}$ 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. We define:

$$\mathit{no\text{-}change}^{\Delta}_{e}(q,Q,R,S) = \mathit{no\text{-}change}^{\Delta}(q,Q) \setminus \ \mathit{sdown}^{\Delta}(q,R) \setminus \mathit{sdown}^{\Delta}(q,S)$$

That is, state changes are now relevant only if they don't move to states that are safe for rejection or selection. Let *sel* and *rej* be two fresh symbols beside of $\Pi$. We adapt tuple projection as follows:

$$\pi_e(q,Q,R,S) = \begin{cases} \mathit{if}\ q \in S\ \mathit{then}\ \mathit{sel}\ \mathit{else\ if}\ q \in R\ \mathit{then}\ \mathit{rej} \\ \mathit{else\ if}\ q \in S\ \mathit{then}\ \Pi\ \mathit{else}\ (p,Q,R,S) \end{cases}$$

We next compile the given SHA $A$ to a SHA$^{\downarrow}$ $A^{\pi}_{e} = (\Sigma, \mathcal{Q}^{\pi}_{e}, \Delta^{\pi}_{e}, I^{\pi}_{e}, F^{\pi}_{e})$. The state sets of $A^{\pi}_{e}$ are:

$$\mathcal{Q}^{\pi}_{e} = \pi_{e}(\mathcal{Q} \times 2^{\mathcal{Q}} \times 2^{\mathcal{Q}} \times 2^{\mathcal{Q}}) \quad I^{\pi}_{e} = \{\pi_{e}(q, R_0, R_0, S_0) \mid q \in I\}$$
$$F^{\pi}_{e} = \{\pi_{e}(q, R_0, R_0, S_0) \mid q \in F\}$$

The transition rules in $\Delta^{\pi}_{e}$ are given by the following inference rules where $p, q \in \mathcal{Q}$, $P, Q, R, S \subseteq \mathcal{Q}$ and $a \in \Sigma$.

$$\frac{q \xrightarrow{a} q' \text{ in } \Delta}{\pi_{e}(q, Q, R, S) \xrightarrow{a} \pi_{e}(q', Q, R, S) \text{ in } \Delta^{\pi}_{e}}$$

$$\frac{\xrightarrow{\langle\rangle} q' \quad P = \mathit{no\text{-}change}^{\Delta}_{e}(q, Q, R, S)}{\pi_{e}(q, Q, R, S) \xrightarrow{\langle\rangle} \pi_{e}(q', P, \mathit{sdown}^{\Delta}(q, R), \mathit{sdown}^{\Delta}(q, S)) \text{ in } \Delta^{\pi}_{e}}$$

$$\frac{q@p \to q' \text{ in } \Delta \qquad P = \mathit{no\text{-}change}^{\Delta}_{e}(q, Q, R, S)}{\pi_{e}(q, Q, R, S)@\pi_{e}(p, P, \mathit{sdown}^{\Delta}(q, R), \mathit{sdown}^{\Delta}(q, S)) \to \pi_{e}(q', Q, R, S) \text{ in } \Delta^{\pi}_{e}}$$

The dSHA from Fig. 1 is not sufficiently complete to obtain the expected results. The problem is that $\mathit{sdown}^{\Delta}(0, \{4\}) = \mathcal{Q}$ there, but only state 3 is really safe for selection. We therefore add a sink state 5 to it, yielding to the dSHA in Fig. 8. For this $A$ we get $\mathit{sdown}^{\Delta}(0, \{4\}) = \{3\}$, so indeed, 3 is the only state that is safe for selection. Applying the earliest construction to this dSHA $A$ yields the $d$SHA$^{\downarrow}$ $A^{\pi}_{e}$ in Fig. 9.

Running the SHA$^{\downarrow}$s $A^{\pi}_{e}$ in streaming mode with subtree projection yields an earliest membership tester for $d$SHA*s* with subtree projection. Two adaptations are in order. Whenever the safe rejection state *rej* is reached, the computation can stop and the hedge on the input stream is rejected. And whenever the safe selection state *sel* is reached, the evaluation can be stopped and the input hedge on the stream is accepted.

**Theorem 1.** *For any $d$SHA $\mathcal{A} = (\Sigma, \mathcal{Q}, \Delta, I, F)$ and hedge $h \in \mathcal{H}_{\Sigma}$ with $[\![h]\!](I) \neq \emptyset$ wrt $\Delta$ the streaming evaluator $[\![h]\!]^{str}_{\{\Pi, sel, rej\}}(I^{\pi}_{e})$ with respect to $\Delta^{\pi}_{e}$ can check 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* and rejected once the evaluator reaches state *rej*. If neither happens the truth value of $q \in F$ is returned where $q$ is the state in the final tuple.

*Proof (sketch).* The streaming membership tester with safe selection and rejection by computing $[\![h]\!]^{str}_{\{sel, rej\}}(I^{\pi}_{e})$ wrt. $\Delta^{\pi}_{e}$ can be shown to be similar to the earliest membership tester from Proposition 6 of [3] enhanced with safe rejection. So we can rely on the definitions of earliest membership testing and the result given there. When adding subtree projection by computing $[\![h]\!]^{str}_{\{\Pi, sel, rej\}}(I^{\pi}_{e})$ wrt. $\Delta^{\pi}_{e}$, the only difference is that the evaluator ignores some subtrees in which the state does not change. Clearly, this does not affect earliest selection and rejection, so we still have an earliest membership tester, but now with subtree projection.

## 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 [6] $A1 - A8$, see Table 1. We used the deterministic SHAs for all these XPath queries constructed by the compiler from [18]. 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 [5], 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 [19]. 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.

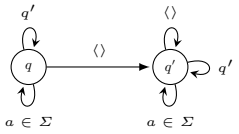
## References


1. A. Al Serhali and J. Niehren. Subhedge Projection for Stepwise Hedge Automata. `https://inria.hal.science/hal-04165835`, 2023.
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 *International Conference on Implementation and Application of Automata* 2023.
4. 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.
5. 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.
6. M. Franceschet. XPathMark performance test. `https://users.dimi.uniud.it/~massimo.franceschet/xpathmark/PTbench.html`. Accessed: 2023-05-30.
7. A. Frisch. Regular tree language recognition with static information. In *Exploring New Frontiers of Theoretical Informatics, IFIP 3rd International Conference on Theoretical Computer Science*, pages 661–674, 2004.
8. O. Gauwin, J. Niehren, and Y. Roos. Streaming tree automata. *Information Processing Letters*, 109(1):13–17, 2008.
9. O. Gauwin, J. Niehren, and S. Tison. Earliest query answering for deterministic nested word automata. In *17th International Symposium on Fundamentals of Computer Theory*, volume 5699 of *LNCS*, pages 121–132. 2009.
10. M. Kay. The Saxon XSLT and XQuery processor, 2004. https://www.saxonica.com.
11. 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.
12. P. Madhusudan and M. Viswanathan. Query automata for nested words. In *34th International Symposium on Mathematical Foundations of Computer Science*, vol. 5734 of *LNCS*, p. 561–573. 2009.
13. S. Maneth and K. Nguyen. XPath whole query optimization. *VLPB Journal*, 3(1):882–893, 2010.
14. A. Marian and J. Siméon. Projecting XML documents. In *VLDB*, 213–224, 2003.
15. B. Mozafari, K. Zeng, and C. Zaniolo. High-performance complex event processing over XML streams. In *SIGMOD Conference*, pages 253–264. ACM, 2012.
16. A. Neumann and H. Seidl. Locating matches of tree patterns in forests. In *Foundations of Software Technology and TCS*, vol. 1530 of *LNCS*, p. 134–145. 1998.
17. A. Neumann and H. Seidl. Locating matches of tree patterns in forests. In *18-th International Conference on Foundations of Software Technology and TCS*, 1998.
18. J. Niehren and M. Sakho. Determinization and minimization of automata for nested words revisited. *Algorithms*, 14(3):68, 2021.
19. 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)*, 2016.
20. 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 $\Delta$. Then for all hedges $h \in \mathcal{H}_\Sigma$ and states $q, q' \in \mathcal{Q}$: $q \xrightarrow{h} q'$ wrt $\Delta$ iff $q \xrightarrow{h}_{\mathcal{P}} q'$ wrt $\Delta$.*

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




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

**Related Work**. We can convert any $d$SHA$^{\downarrow}$ $A$ 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 [18] or to hedge automata [8]. 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 [20,4,27, 26] have the same expressiveness as Neumann and Seidl's pushdown forest automata [16] and also as Bojanczyk's forest automata (see Section 3.3 of [23]). 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.

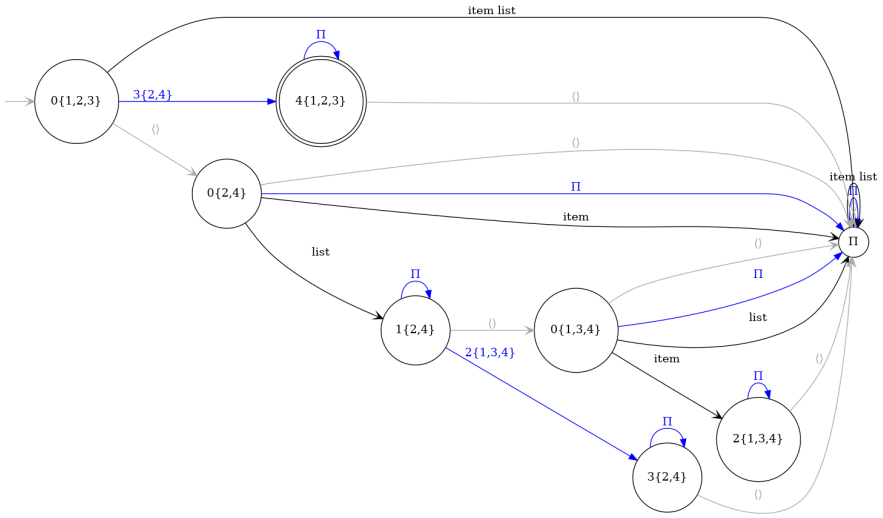



Fig. 6: The full $d\text{SHA}^{\downarrow}$ $A^{\pi}$ constructed from the dSHA $A$ in Fig. 1.

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

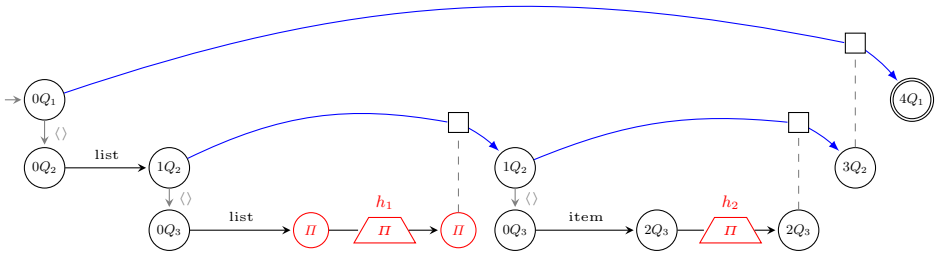



Fig. 7: A successful run of the $\text{SHA}^{\downarrow}$ in Fig. 5 on $\langle list \cdot \langle list \cdot h_1 \rangle \cdot \langle item \cdot h_2 \rangle \rangle$ with $Q_1 = \{1, 2, 3\}$, $Q_2 = \{2, 4\}$ and $Q_3 = \{1, 3, 4\}$

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

– Case $h = \varepsilon$. The following inference rule proves $(q, Q) \xrightarrow{h} \mu$ wrt $\Delta^{\pi}$:

$$\frac{true}{(q, Q) \xrightarrow{\varepsilon} (q, Q) \text{ wrt } \Delta^{\pi}}$$

1. If $\mu = (q', Q)$ for some $q'$ then $q' = q$ and $q \xrightarrow{h} q'$ wrt $\Delta$
2. The case $\mu = \Pi$ is not possible.

– Case $h = a$. The following inference rule got applied for some $q' \in \mathcal{Q}$:

$$\frac{q \xrightarrow{a} q' \text{ in } \Delta}{\pi(q, Q) \xrightarrow{a} \pi(q', Q) \text{ in } \Delta^\pi}$$

where $\mu = \pi(q', Q)$.

1. If $\mu = (q', Q)$ then $q \xrightarrow{a} q'$ wrt $\Delta$.
2. if $\mu = \Pi$ then $q' \in Q$ so that $q \xrightarrow{a} q'$ wrt $\Delta$ for some $q' \in Q$.

– Case $h = h_1 \cdot h_2$. The following inference rule got applied for some $\mu_1 \in \mathcal{Q}^\pi$:

$$\frac{(q, Q) \xrightarrow{h_1} \mu_1 \text{ wrt } \Delta^\pi \quad \mu_1 \xrightarrow{h_2} \mu \text{ wrt } \Delta^\pi}{(q, Q) \xrightarrow{h_1 \cdot h_2} \mu \text{ wrt } \Delta^\pi}$$

1. Case $\mu = (q', Q)$. Then $\mu_1 = (q_1, Q)$ for some $q_1$. The induction hypothesis applied to $h_1$ yields
$$q \xrightarrow{h_1} q_1 \text{ wrt } \Delta$$
and the induction hypothesis applied to $h_2$ yields
$$q_1 \xrightarrow{h_2} q' \text{ wrt } \Delta$$
Hence $q \xrightarrow{h_1 \cdot h_2} q'$ wrt $\Delta$ can derived by the following inference rule:
$$\frac{q \xrightarrow{h_1} q_1 \text{ wrt } \Delta \quad q_1 \xrightarrow{h_2} q' \text{ wrt } \Delta}{q \xrightarrow{h_1 \cdot h_2} q' \text{ wrt } \Delta}$$
2. Case $\mu = \Pi$. We distinguish two cases.
**Case** $\mu_1 = \Pi$ By induction hypothesis applied to $h_1$ there exists $q_1 \in \{q\} \cup Q$ such that $q \xrightarrow{h_1} q_1$ wrt $\Delta$. Since $A$ is complete there exists $q' \in Q$ such that $q_1 \xrightarrow{h_2} q'$ wrt $\Delta$. Since $Q = \mathit{safe}^\Delta(Q')$ and $q_1 \xrightarrow{h_2} q'$ wrt $\Delta$. it follows that $q' \in Q$ too.
**Case** $\mu_1 = (q_1, Q)$ **for some** $q_1$**.** By induction hypothesis applied to $h_1$ it follows that $q \xrightarrow{h_1} q_1$ wrt $\Delta$.
   (a) If $\mu = (q', Q)$ for some $q'$ then the induction hypothesis applied to $h_2$ yields $q_1 \xrightarrow{h_2} q'$ wrt $\Delta$. Hence $q \xrightarrow{h} q'$ wrt $\Delta$.
   (b) If $\mu = \pi$ then the induction hypothesis applied to $h_2$ yields $q_1 \xrightarrow{h_2} q'$ wrt $\Delta$ for some $q' \in Q$. Hence $q \xrightarrow{h} q'$ wrt $\Delta$ for some $q' \in Q$.

– Case $h = \langle h_1 \rangle$. The following inference rule got applied to infer $(q, Q) \xrightarrow{h} \mu$ for some $\mu_1, \mu_2 \in \mathcal{Q}^\pi$:

$$\frac{(q, Q) \xrightarrow{\langle\rangle} \mu_1 \text{ in } \Delta \quad \mu_1 \xrightarrow{h_1} \mu_2 \text{ wrt } \Delta^\pi \quad (q, Q)@\mu_2 \to \mu \text{ in } \Delta^\pi}{(q, Q) \xrightarrow{\langle h_1 \rangle} \mu \text{ wrt } \Delta^\pi}$$

Furthermore, the hypothesis $(q, Q) \xrightarrow{\langle\rangle} \mu_1$ in $\Delta$ must be derived by the inference rule:

$$\frac{\xrightarrow{\langle\rangle} q_1 \text{ in } \Delta \qquad P = \mathit{no\text{-}change}^{\Delta}(q, Q)}{\pi(q, Q) \xrightarrow{\langle\rangle} \pi(q_1, P) \text{ in } \Delta^{\pi}}$$

with $q \notin Q$ so that $\pi(q, Q) = (q, Q)$

**Case** $\mu_1 = \Pi$**.** Hence $q_1 \in P$ and $\pi(q_1, P) = \mu_1$. The hypothesis, $\mu_1 \xrightarrow{h_1} \mu_2$ wrt $\Delta^{\pi}$ implies that $\mu_2 = \Pi$. Furthermore, $(q, Q)@\mu_2 \to \mu$ in $\Delta^{\pi}$ must be infered by the rule:

$$\frac{q@p \to q' \text{ in } \Delta \qquad P = \mathit{no\text{-}change}^{\Delta}(q, Q)}{\pi(q, Q)@\pi(p, P) \to \pi(q', Q) \text{ in } \Delta^{\pi}}$$

By completeness there exists $q_2$ such that $q_1 \xrightarrow{h_1} q_2$ wrt $\Delta$. Since $q_1 \in P = d^{\Delta}(\{q\} \cup Q)$ it follows that $q@^{\Delta}q_2 \subseteq \{q\} \cup Q$. Since it follows that $q' \in q@^{\Delta}q_2$, so that $q' = q$ or $q \in Q$. In the first case, we have $\mu = (q, Q)$ and in the second $\mu = \Pi$. n any case $q \xrightarrow{\langle h_1\rangle} q'$ wrt $\Delta$. can be inferred as follows:

$$\frac{\xrightarrow{\langle\rangle} q_1 \text{ in } \Delta \quad q_1 \xrightarrow{h_1} q_2 \text{ wrt } \Delta \quad q@q_2 \to q' \text{ in } \Delta}{q \xrightarrow{\langle h_1\rangle} q' \text{ wrt } \Delta}$$

1. Case $\mu = \Pi$. Then $q \in Q$, so there exists $q \in Q$ such that $q \xrightarrow{h} q$ wrt $\Delta$.
2. Case $\mu = (q, Q)$. Then $q' = q$ so $q \xrightarrow{h} q$ wrt $\Delta$.

**Case** $\mu_1 = (q_1, P)$**.** So $q_1 \notin P$. The induction hypothesis applied to the assumption $(q_1, P) \xrightarrow{h_1} \mu_2$ leaves 2 cases:

1. If $\mu_2 = \Pi$ then there exists $q_2 \in P$ such that $q_1 \xrightarrow{h_1} q_2$ wrt $\Delta$. Since $P = \mathit{sdown}^{\Delta}(\{q\} \cup Q)$ we have $q@^{\Delta}q_2 \in \{q\} \cup Q$. So $q \xrightarrow{h} q'$ wrt $\Delta$ for $q' = q$ or $q' \in Q$.
   (a) Case $\mu = \Pi$. Then $q \xrightarrow{h} q'$ wrt $\Delta$ for some $q' \in Q$.
   (b) Case $\mu = (q, Q)$. Then $q' = q$ so that $q \xrightarrow{h} q$ wrt $\Delta$.

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

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

*Proof.* Straightforward.

**Proposition 8.** *Let $v$ be a word in $\hat{\Sigma}^*$, $\Delta$ 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 $\Delta$: $(q, \sigma) \xrightarrow{v}{}^{str} (q', \sigma')$ wrt $\Delta$ iff $(q, \sigma) \xrightarrow{v}{}^{str}_{\mathcal{P}} (q', \sigma')$ wrt $\Delta$.*

*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) [21], whose streaming evaluator is given by a visibly pushdown machine [22], previously known as *input driven automata* [25, 28, 24].

**Definition 10.** *A nested word automata (*NWA*) is a tuple* $(\Sigma, \mathcal{Q}, \Gamma, \Delta, I, F)$*, where* $\Sigma$*,* $\Gamma$ *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 $\Gamma$ are called stack symbols. The transition rules in $\Delta$ 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 $S \in \Gamma^*$:

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

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

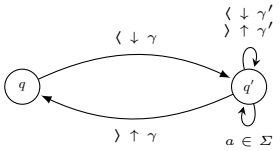
For any $d$SHA$^{\downarrow}$ $\mathcal{A} = (\Sigma, \mathcal{Q}, \Delta, I, F)$ we can define the dNWA $A^{nwa} = (\Sigma, \mathcal{Q}, \Gamma, \Delta^{nwa}, I^{nwa}, F^{nwa})$ such that $\Gamma = \mathcal{Q}$, while $\Delta^{nwa}$ contains for all $a \in \Sigma$ and $q, p \in \mathcal{Q}$ the transition 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 [19]. They are based on the following notion of states of irrelevant subtrees for NWAs.

**Definition 12 (Variant of Definition 3 of [19]).** *We call a state $q$ of an* NWA *a* state of irrelevant subtrees *if there exist two different stack symbols $\gamma, \gamma'$ and a state $q'$ such that the transitions shown on the right exist, but no further opening transitions with $\gamma$, no further transitions with $\gamma'$, and no further closing transition in $q$ popping $\gamma$. 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 [19] 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)

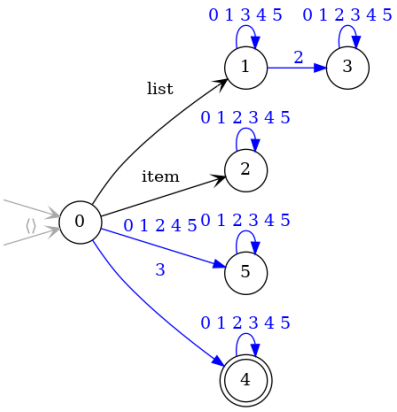



Fig. 8: A unique minimal complete dSHA for the XPath filter `[self::list][child::item]`.

**Lemma 14.** *Let $\mathcal{P}$ be a subset of projection states of $\Delta_e^\pi$. 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 $\sigma \in \mathcal{Q}^*$:*

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

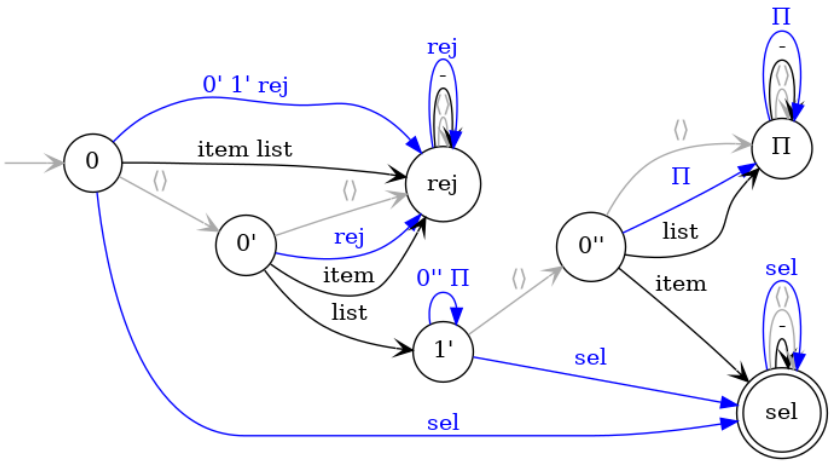



Fig. 9: The earliest $d\text{SHA}^{\downarrow}$ with subtree projection $A^{\pi}_{e}$ for the SHA in Fig. 8. The states are rename with the following mapping to tuples:
$0 = (0, \{1,2,3,5\}, \{1,2,3,5\}, \{4\})$, $0' = (0, \emptyset, \{2,4,5\}, \{3\})$, $1' = (1, \emptyset, \{2,4,5\}, \{3\})$ and $0'' = (0, \{1,3,4,5\}, \emptyset, \{2\})$

## 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 |

## References for the Appendix


21. R. Alur. Marrying words and trees. In *26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 233–242. ACM-Press, 2007.
22. R. Alur and P. Madhusudan. Visibly pushdown languages. In L. Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM, 2004.
23. M. Bojanczyk and I. Walukiewicz. Forest algebras. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 107–132. Amsterdam University Press, 2008.
24. A. Okhotin and K. Salomaa. Complexity of input-driven pushdown automata. *SIGACT News*, 45(2):47–67, 2014.
25. K. Mehlhorn. Pebbling moutain ranges and its application of dcfl-recognition. In J. W. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 422–435. Springer, 1980.
26. M. Murata. Hedge automata: a formal model for XML schemata. Web page, 2000.
27. C. Pair and A. Quéré. Définition et etude des bilangages réguliers. *Information and Control*, 13(6):565–593, 1968.
28. B. von Braunmühl and R. Verbeek. Input driven languages are recognized in log n space. In M. Karplnski and J. van Leeuwen, editors, *Topics in the Theory of Computation*, volume 102 of *North-Holland Mathematics Studies*, pages 1 – 19. North-Holland, 1985.