Extended Path Expressions for XML

[Extended Abstract]

Makoto Murata
IBM Tokyo Research Lab/IUJ Research Institute
1623-14, Shimotsuruma, Yamato-shi,
Kanagawa-ken 242-8502, Japan
mmurata@trl.ibm.co.jp

ABSTRACT

Query languages for XML often use path expressions to locate elements in XML documents. Path expressions are regular expressions such that underlying alphabets represent conditions on nodes. Path expressions represent conditions on paths from the root, but do not represent conditions on siblings, siblings of ancestors, and descendants of such siblings. In order to capture such conditions, we propose to extend underlying alphabets. Each symbol in an extended alphabet is a triplet (e_1, a, e_2), where a is a condition on nodes, and e_1 (e_2) is a condition on elder (resp. younger) siblings and their descendants; e_1 and e_2 are represented by hedge regular expressions, which are as expressive as hedge automata (hedges are ordered sequences of trees). Nodes matching such an extended path expression can be located by traversing the XML document twice. Furthermore, given an input schema and a query operation controlled by an extended path expression, it is possible to construct an output schema. This is done by identifying where in the input schema the given extended path expression is satisfied.

1. INTRODUCTION

XML [5] has been widely recognized as one of the most important formats on the WWW. XML documents are ordered trees containing text, and thus have structures more flexible than relations of relational databases.

Query languages for XML have been actively studied [1, 14]. Typically, operations of such query languages can be controlled by path expressions. A path expression is a regular expression such that underlying alphabets represent conditions on nodes. For example, by specifying a path expression $(section^*, figure)$, we can extract figures in sections, figures in sections in sections, figures in sections in sections in sections, and so forth, where $section$ and $figure$ are conditions on nodes. Based on well-established theories of regular languages, a number of useful techniques (e.g., optimization [2, 8, 15, 19]) for path expressions have been developed.

PODS '01 Santa Barbara, California USA

However, when applied to XML, path expressions do not take advantage of orderedness of XML documents. For example, path expressions cannot locate all <figure> elements whose immediately following siblings are <table> elements.

On the other hand, industrial specifications such as XPath [10] have been developed. Such specifications address orderedness of XML documents. In fact, XPath can capture the above example. However, these specifications are not driven by any formal models, but rather designed in an ad-hoc manner[1]. Lack of formal models prevents generalization of useful techniques originally developed for path expressions.

As a formal framework for addressing orderedness, this paper shows a natural extension of path expressions. First, we introduce hedge regular expressions, which generate hedges (ordered sequences of ordered trees). Hedge regular expressions can be converted to hedge automata (variations of tree automata for hedges) and vice versa. Given a hedge and a hedge regular expression, we can determine which node in the hedge matches the given hedge regular expression by executing the hedge automaton. The computation time is linear to the number of nodes in hedges.

Second, we introduce pointed hedge representations. They are regular expressions such that each "symbol" is a triplet (e_1, a, e_2), where e_1, e_2 are hedge regular expressions and a is a condition on nodes. Intuitively, e_1 represent conditions on elder siblings and their descendants, while e_2 represent conditions on younger siblings and their descendants. As a special case, if every hedge regular expression in a pointed hedge representation generates all hedges, this pointed hedge representation is a path expression.

Given a hedge and a pointed hedge representation, we can determine which node in the hedge matches the given pointed hedge representation. For each node, (1) we determine which of the hedge regular expressions matches the elder siblings and younger siblings, respectively, (2) we then determine which of the triplets the node matches, and (3) we finally evaluate the pointed hedge representation. Again, the computation time is linear to the number of nodes in hedges.

Another goal of this work is schema transformation. Recall that query operations of relational databases construct not only relations but also schemas. For example, given input schemas (A, B) and (B, C), the join operation creates an output schema (A, B, C). Such output schemas allow

[1] However, a formal semantics for a subset of XPath has been later developed by Philip Wadler [35].

further processing of output relations.

It would be desirable for query languages for XML to provide such schema transformations. That is, we would like to construct output schemas from input schemas and query operations (e.g., select, delete), which utilize hedge regular expressions and pointed hedge representations. To facilitate such schema transformation, we construct match-identifying hedge automata from hedge regular expressions and pointed hedge representations. The computation of such automata assigns marked states to those nodes which match the hedge regular expressions and pointed hedge representations. Schema transformation is effected by first creating intersection hedge automata which simulate the match-identifying hedge automata and the input schemata, and then transforming the intersection hedge automata as appropriate to the query operation.

The rest of this paper is organized as follows. In Section 2, we consider related works. We introduce hedges and hedge automata in Section 3, and then introduce hedge regular expressions in Section 4. In Section 5, we introduce pointed hedges and pointed hedge representations. In Section 6, we define selection queries as pairs of hedge regular expressions and pointed hedge representations. In Section 7, we study how to locate nodes in hedges by evaluating pointed hedge representations. In Section 8, we construct match-identifying hedge automata, and then construct output schemas. In Section 9, we conclude and consider future works.

2. RELATED WORKS

Queries for structured documents have been studied by very many researchers (see surveys [3, 4]). Recently, formal approaches such as MSO have appeared.

Monadic second-order logic (MSO) [13, 33] allows the use of *set variables* ranging over *sets* of domain elements in addition to individual variables ranging over domain elements, where domain elements may be positions in strings or nodes in trees. Universal and existential quantifiers can be used for both types of variables. MSO formulas on strings can be translated to string automata, and vice versa.

MSO formulas can be used to represent selection queries. Selection queries definable by MSO can be translated to query automata, and vice versa [26], and can also be translated to boolean attribute grammars, and vice versa [25, 24]. Such query automata or attribute grammars can be evaluated efficiently, and the time complexity is linear to the number of nodes. However, translation of MSO formulas to automata or attribute grammars requires non-elementary space in the worst case [20].

Neven and Schwentick [27] defined F$OREG$ and F$OREG^*$, which are first-order logic extended with horizontal path expressions and vertical path expressions. After showing that these languages do not capture MSO, they introduced a logic called ETL, which is as expressive as MSO but can be evaluated efficiently. In fact, evaluation of an ETL formula takes time exponential to the size of the formula.

Our hedge regular expressions and pointed hedge regular representations also capture MSO queries. Furthermore, these expressions can be evaluated in time linear to the number of nodes and time exponential to the expression size. Exponential time is required for converting non-deterministic hedge automata to deterministic ones. However, we conjecture that such conversion is usually efficient, as does conversion of non-deterministic string automata to deterministic ones.

Papakonstantinou and Vianu [30] use regular expressions to navigate both vertically and horizontally (i.e., ancestors and siblings). Moreover, they provide schema transformation, which they call "DTD inference". Their input schemas, which they call regular loto (labeled ordered tree object) type definitions, represent hedge *local* languages, while their output schemas represent hedge *context-free* languages. The class of local languages is a proper subclass of the class of regular languages, which is in turn a proper subclass of context-free languages. However, in our framework, both input and output schemas represent hedge *regular* languages rather than hedge *local* languages. Since schema languages such as RELAX [18], TREX [9], and XML Schema [34] use hedge regular languages rather than hedge local languages, we would argue that our work is more applicable to such languages.

Although [27] and [30] allow variables in patterns, our framework does not allow variables. As a result, a pointed hedge representation cannot locate tuples of elements. Introduction of such variables are discussed in Section 9.

Nivat and Podelski [31, 28] introduced pointed binary tree representations, which inspired our work. We have extended them for hedges rather than binary trees. Furthermore, we have introduced an algorithm for evaluating pointed hedge representations and constructed match-identifying hedge automata for schema transformation.

Milo, Suciu, and Vianu [21] provides typechecking of XML transformers, which are modeled by k-pebble transducers. Given an input regular tree language, an output regular tree language, and a k-pebble transducer, their typechecking ensures whether the result of transforming any XML document in the input language is contained by the output language.

XDuce [17] is a programming language for handling XML documents. Types in XDuce are regular expressions of types. Operations in XDuce perform regular expression pattern matching. Furthermore, XDuce provides type inference by using tree automata.

Caterpillar expressions [6] capture conditions on ancestor nodes, sibling nodes, etc. Expressiveness of caterpillar expressions is compared with that of regular tree languages.

XPath [10] capture conditions on ancestors, siblings, descendants, and so forth. However, XPath is not as structurally expressive as combinations of hedge regular expressions and pointed hedge representations. In fact, some path expressions (e.g., a^* which implies "all ancestors are a") cannot be captured by XPath. On the other hand, XPath can capture conditions on attributes. For our framework to capture such conditions, we only have to allow terminal symbols to represent collections of tag names and conditions on attributes.

3. HEDGES AND HEDGE AUTOMATA

In this section, we introduce hedges (ordered sequences of ordered trees) [2] and hedge automata, which were originally introduced by Pair and Quere [29], and Takahashi [32].

Some researchers [26, 21] have used unranked tree automata for XML queries, probably because XML documents

[2]The term "hedge" was introduced by Courcelle [12]. Some authors use "forests", but forests are sets of trees rather than sequences of trees.

are unranked trees rather than hedges. However, we use hedge automata for two reasons. First, conditions on siblings can be naturally captured by hedge automata. Second, regular expressions for unranked trees cannot be introduced without introducing those for hedges in advance. In fact, Takahashi [32] studied both unranked tree languages and hedge languages, and pointed out that regular hedge languages provide a nicer generalization of regular string languages.

Hereafter, we assume that Σ is an alphabet and that X is a finite set of variables. They are disjoint and do not contain either $\langle$ or $\rangle$.

Definition 1. A *hedge over* Σ and X is (1) ϵ (the empty hedge), (2) x ($x \in X$), (3) $a\langle u \rangle$ ($a \in \Sigma$, u is a hedge), and (4) uv (u and v are hedges). The set of hedges is denoted $\mathcal{H}[\Sigma, X]$.

For example, $a\langle\epsilon\rangle$, $a\langle x\rangle$, and $a\langle\epsilon\rangle b\langle b\langle\epsilon\rangle\, x\rangle$ are hedges. Note that symbols in Σ are used as labels of non-leaf nodes, while variables in X are used as labels of leaf nodes[3]. We abbreviate $a\langle\epsilon\rangle$ as a. Thus, the third example is abbreviated as $ab\langle bx\rangle$.

Definition 2. The *ceil* of a hedge u, denoted $\lceil u \rceil$, is a string over $\Sigma \cup X$ recursively defined below:

$$\begin{aligned} \lceil\epsilon\rceil &= \epsilon \text{ (the empty string)}, \\ \lceil x\rceil &= x, \\ \lceil a\langle u\rangle\rceil &= a, \\ \lceil uv\rceil &= \lceil u\rceil\lceil v\rceil \end{aligned}$$

For example, the ceil of $a\langle x\rangle$ and that of $ab\langle b\, x\rangle$ are a and ab, respectively.

Definition 3. A *deterministic hedge automaton* M is a 6-tuple $(\Sigma, X, Q, \iota, \alpha, F)$ such that (1) Q is a finite set of states, (2) ι is a function from X to Q, (3) α is a mapping from $\Sigma \times Q^*$ to Q such that $\{q_1 q_2 \ldots q_k \mid \alpha(a, q_1 q_2 \ldots q_k) = q\}$, denoted $\alpha^{-1}(a, q)$, is regular for any $q \in Q$, $a \in \Sigma$, and (4) F is a regular set over Q and is called the final state sequence set.

As an example, we show a deterministic hedge automaton, which accepts any sequences of trees $d\langle p\langle x\rangle\rangle$, $d\langle p\langle x\rangle p\langle y\rangle\rangle$, $d\langle p\langle x\rangle p\langle y\rangle p\langle y\rangle\rangle, \ldots$. The deterministic hedge automaton is $M_0 = (\Sigma_0, X_0, Q_0, \iota_0, \alpha_0, F_0)$, where

$$\begin{aligned} \Sigma_0 &= \{d, p\}, \\ X_0 &= \{x, y\}, \\ Q_0 &= \{q_d, q_{p1}, q_{p2}, q_x, q_y, q_0\}, \\ \iota_0(x) &= q_x, \\ \iota_0(y) &= q_y, \\ \alpha_0(d, u) &= \begin{cases} q_d & (u \in L(q_{p1} q_{p2}^*)), \\ q_0 & (\text{otherwise}), \end{cases} \\ \alpha_0(p, u) &= \begin{cases} q_{p1} & (u = q_x), \\ q_{p2} & (u = q_y), \\ q_0 & (\text{otherwise}), \end{cases} \\ F_0 &= L(q_d^*) \end{aligned}$$

[3]We do not formally define nodes in hedges, but they are address-value pairs, where an address is a Dewey number and a value is a variable or symbol.

Given a hedge, a deterministic hedge automaton is executed in the bottom-up manner. First, we assign a state to every leaf node by computing $\iota(x)$, where x is the label of the leaf node. Then, we assign a state to each node by applying the transition function α to the label of this node and the states of the child nodes.

Definition 4. The *computation* of a hedge u by a deterministic hedge automaton M, denoted $M\|u$, is a hedge recursively defined below:

$$\begin{aligned} M\|\epsilon &= \epsilon, \\ M\|x &= \iota(x), \\ M\|a\langle u\rangle &= \alpha(a, \lceil M\|u\rceil)\langle M\|u\rangle, \\ M\|uv &= (M\|u)(M\|v) \end{aligned}$$

Definition 5. A hedge u is *accepted* by M if $\lceil M\|u\rceil$ is contained by F. The *language accepted* by M, denoted $L(M)$, is the set of hedges accepted by M.

Consider a hedge $d\langle p\langle x\rangle p\langle y\rangle\rangle d\langle p\langle x\rangle\rangle$. Its computation by M_0 (shown above) is $q_d\langle q_{p1}\langle q_x\rangle q_{p2}\langle q_y\rangle\rangle q_d\langle q_{p1}\langle q_x\rangle\rangle$. The ceil of this computation is $q_d q_d$, which is contained by F_0. Thus, this hedge is accepted by M_0.

Definition 6. A *non-deterministic hedge automaton* M is a 6-tuple $(\Sigma, X, Q, \iota, \alpha, F)$ such that (1) Q is a finite set of states, (2) ι is a mapping from X to 2^Q, (3) α is a mapping from $\Sigma \times Q^*$ to 2^Q such that $\{q_1 q_2 \ldots q_k \mid \alpha(a, q_1 q_2 \ldots q_k) \ni q\}$, denoted $\alpha^{-1}(a, q)$, is regular for any $q \in Q$, $a \in \Sigma$, and (4) F is a regular set over Q and is called the final state sequence set.

As an example, $M_1 = (\Sigma_1, X_1, Q_1, \iota_1, \alpha_1, F_1)$ is a non-deterministic hedge automaton where:

$$\begin{aligned} \Sigma_1 &= \{d, p\}, \\ X_1 &= \{x, y\}, \\ Q_1 &= \{q_d, q_{p1}, q_{p2}, q_x\}, \\ \iota_1(x) &= \{q_x\} \\ \iota_1(y) &= \emptyset, \\ \alpha_0(d, u) &= \begin{cases} \{q_d\} & (u \in L(q_{p1} q_{p2}^*)), \\ \emptyset & (\text{otherwise}), \end{cases} \\ \alpha_0(p, u) &= \begin{cases} \{q_{p1}, q_{p2}\} & (u = q_x q_x), \\ \{q_{p1}\} & (u = q_x), \\ \{q_{p2}\} & (u = q_x), \\ \emptyset & (\text{otherwise}), \end{cases} \\ F_0 &= L(q_d^*) \end{aligned}$$

Definition 7. The set of *computations* of a hedge u by a non-deterministic hedge automaton M, denoted $M\|u$, is the set of hedges recursively defined below:

$$\begin{aligned} M\|\epsilon &= \{\epsilon\}, \\ M\|x &= \iota(x), \\ M\|a\langle u\rangle &= \{a'\langle u'\rangle \mid u' \in M\|u, a' \in \alpha(a, \lceil u'\rceil)\} \\ M\|uv &= \{u'v' \mid u' \in M\|u,\ v' \in M\|v\} \end{aligned}$$

Definition 8. A hedge u is *accepted* by a non-deterministic hedge automaton M if for some $u' \in M\|u$, $\lceil u'\rceil$ is contained by F. The *language accepted* by M, denoted $L(M)$, is the set of hedges accepted by M.

As an example, we execute M_1 (shown above) for two hedges $d\langle p\langle x\rangle p\langle y\rangle\rangle$ and $d\langle p\langle xx\rangle p\langle xx\rangle\rangle$. The set of computations of the first hedge is empty. Thus, this hedge is not accepted. The set of computations of the second hedge is $\{q_d\langle q_{p1}\langle q_x q_x\rangle q_{p1}\langle q_x q_x\rangle\rangle,\ q_d\langle q_{p1}\langle q_x q_x\rangle q_{p2}\langle q_x q_x\rangle\rangle\}$. The ceils of these computations are q_d, which is contained by F_1. Thus, the second hedge is accepted.

The following theorem can be easily proved by subset construction.

THEOREM 1. *Deterministic hedge automata and nondeterministic hedge automata are equally expressive.*

4. HEDGE REGULAR EXPRESSIONS

In this section, we introduce hedge regular expressions, which are as expressive as hedge automata.

Although there are many works [16, 11] on binary tree regular expressions, hedge regular expressions have not been studied in the literature. To the best of our knowledge, the work closest to ours is Pair and Quere [29]. Their expressions capture the class of hedge local languages, which is a proper subclass of hedge regular languages. Since any hedge regular language can be obtained by applying some projection to some hedge local language, a pair of an expression and projection provides a hedge regular "expression". Our work differs in not using projections. In other words, our hedge regular expressions directly capture hedge regular languages.

Recall that regular expressions for strings have the concatenation and the closure (*) operator. Hedge regular expressions require two sets of these operators. The first set creates new hedges by aligning hedges in the horizontal direction. Meanwhile, the second set creates new hedges by embedding hedges in hedges.

Although it is easy to align hedges in the horizontal direction, it is not straightforward to embed hedges in hedges. Where in a hedge do we embed other hedges? As a target for such embedding, we introduce substitution symbols.

Let Z be a set of substitution symbols. We assume that Z and $\Sigma \cup X \cup \{\langle, \rangle\}$ are disjoint.

Definition 9. A hedge over Σ and X with substitution symbols in Z is recursively defined below:

- ϵ,
- $x\ (x \in X)$,
- $a\langle z\rangle\ (a \in \Sigma, z \in Z)$,
- $a\langle u\rangle\ (a \in \Sigma, u$ is a hedge with substitution symbols)
- $u_1 u_2\ (u_1, u_2$ are hedges with substitution symbols)

The set of hedges with substitution symbols is denoted by $\mathcal{H}[\Sigma, X, Z]$.

Definition 10. The *embedding* of $U(\subseteq \mathcal{H}[\Sigma, X, Z])$ in $v(\in \mathcal{H}[\Sigma, X, Z])$ at a substitution symbol z, denoted by $U \circ_z v$, is the set of hedges with substitution symbols obtained by replacing every occurrence of z in v by some element of U; different occurrences of z may be replaced by different elements of V. The *embedding* of $U(\subseteq \mathcal{H}[\Sigma, X, Z])$, in $V(\subseteq \mathcal{H}[\Sigma, X, Z])$ at z, denoted by $U \circ_z V$, is $\bigcup_{v \in V} U \circ_z v$.

For example, let $U = \{a, b\}$ and $v = c\langle z\rangle c\langle z\rangle$, where $a, b, c \in \Sigma$ and $z \in Z$. Then, $U \circ_z v$ is $\{c\langle a\rangle c\langle a\rangle,\ c\langle a\rangle c\langle b\rangle,\ c\langle b\rangle c\langle a\rangle,\ c\langle b\rangle c\langle b\rangle\}$. Note that $c\langle a\rangle c\langle b\rangle$ is constructed by replacing the first occurrence of z with a and the second with b. Let $V = \{c\langle z\rangle c\langle z\rangle, c\langle z\rangle\}$. Then, $U \circ_z V$ is $\{c\langle a\rangle c\langle a\rangle,\ c\langle a\rangle c\langle b\rangle,\ c\langle b\rangle c\langle a\rangle,\ c\langle b\rangle c\langle b\rangle,\ c\langle a\rangle, c\langle b\rangle\}$.

Now, we are ready to introduce hedge regular expressions.

Definition 11. A *hedge regular expression* over an alphabet Σ, a set X of variables, and a set Z of substitution symbols is recursively defined below:

- $\emptyset$,
- ϵ,
- $x\ (x \in X)$,
- $a\langle e\rangle\ (a \in \Sigma, e$ is a hedge regular expression),
- $e_1 e_2\ (e_1, e_2$ are hedge regular expressions),
- $e_1|e_2\ (e_1, e_2$ are hedge regular expressions),
- $e^*\ (e$ is a hedge regular expression),
- $a\langle z\rangle\ (a \in \Sigma, z \in Z)$,
- $e_1 \circ_z e_2\ (e_1, e_2$ are hedge regular expressions, and $z \in Z)$, and
- $e^z\ (e$ is a hedge regular expression, and $z \in Z)$,

Definition 12. A hedge regular expression e *represents* a set $L(e)$ of hedges with substitution symbols recursively defined below:

$$
\begin{aligned}
L(\emptyset) &= \emptyset,\\
L(\epsilon) &= \{\epsilon\},\\
L(x) &= \{x\},\\
L(a\langle e\rangle) &= \{a\langle u\rangle \mid u \in L(e)\},\\
L(e_1 e_2) &= \{u_1 u_2 \mid u_1 \in L(e_1), u_2 \in L(e_2)\},\\
L(e_1|e_2) &= L(e_1) \cup L(e_2),\\
L(e^*) &= \{\epsilon\} \cup L(e) \cup L(ee) \cup L(eee) \cup \ldots,\\
L(a\langle z\rangle) &= \{a\langle z\rangle\},\\
L(e_1 \circ_z e_2) &= L(e_1) \circ_z L(e_2)\\
L(e^z) &= L(e^{1,z}) \cup L(e^{2,z}) \cup L(e^{3,z}) \cup \ldots\\
&\quad L(e^{1,z}) = L(e),\\
&\quad L(e^{2,z}) = L(e^{1,z}) \circ_z L(e) \cup L(e^{1,z}),\\
&\quad L(e^{3,z}) = L(e^{2,z}) \circ_z L(e) \cup L(e^{2,z}),\\
&\quad \ldots
\end{aligned}
$$

For example, consider a hedge regular expression $a\langle z\rangle^{*z}$. Obviously, $L(a\langle z\rangle^*)$ is $\{\epsilon, a\langle z\rangle, a\langle z\rangle a\langle z\rangle, a\langle z\rangle a\langle z\rangle a\langle z\rangle, \ldots\}$. To compute $L(a\langle z\rangle^{*z})$, we have to compute $L(a\langle z\rangle^{*1,z})$, $L(a\langle z\rangle^{*2,z})$, $L(a\langle z\rangle^{*3,z})$, and so forth.

For every positive integer i, $L(a\langle z\rangle^{*i,z})$ contains all hedges such that (1) their height is equal to or less than i, (2) every symbol is a, and (3) every substitution symbol is z. Therefore, $L(a\langle z\rangle^{*z})$ contains all hedges such that (1) every symbol is a, and (2) every substitution symbol is z.

LEMMA 1. *Given a hedge regular expression e, we can construct a hedge automaton that accepts $L(e)$.*

PROOF. We do not provide a whole proof, but sketch how a hedge hedge automaton is constructed from a given hedge regular expression.

For each sub-expression e' of e, we construct a nondeterministic hedge automaton $M(e')$ that accepts $L(e')$. Since substitution symbols occur in hedges in $L(e)$, we allow substitution symbols as variables of hedge automata.

For each substitution symbol z in Z, we introduce a state $\bar{z}$ and always use this state for z, and denote set $\{\bar{z} \mid z \in Z\}$ by $\bar{Z}$. States in $\bar{Z}$ are used only for leaf nodes; they are never used for nodes labeled with symbols in Σ. Moreover, they never occur in final state sequences.

Case 1: $\emptyset$

$M(\emptyset) = (\emptyset, \emptyset, \emptyset, \alpha_1, \iota_1, \emptyset)$, where the domain of α_1 and that of ι_1 are empty.

Case 2: ϵ

$M(\epsilon) = (\emptyset, \emptyset, \emptyset, \alpha_1, \iota_1, \{\epsilon\})$, where the domain of α_1 and that of ι_1 are empty.

Case 3: x

$M(x) = (\emptyset, \{x\}, \{q\}, \alpha_1, \iota_1, \{q\})$, where the domain of α_1 is empty and $\iota_1(x) = \{q\}$.

Case 4: $a\langle e \rangle$

Let $M(e)$ be $(\Sigma_1, X_1, Q_1, \alpha_1, \iota_1, F_1)$. Then, $M(a\langle e \rangle) = (\Sigma_1 \cup \{a\}, X_1, Q_1 \cup \{q_2\}, \alpha_2, \iota_1, \{q_2\})$, where q_2 is a state not contained by Q_1 and

$$\alpha_2^{-1}(a, q_2) = F_1,$$

$$\alpha_2^{-1}(a, q_1) = \begin{cases} \alpha_1^{-1}(a, q_1) & (a \in \Sigma_1, q_1 \in Q_1), \\ \emptyset & (a \notin \Sigma_1, q_1 \in Q_1), \end{cases}$$

$$\alpha_2^{-1}(i, q_2) = \emptyset \quad (i \in \Sigma_1 \setminus \{a\}),$$

$$\alpha_2^{-1}(i, q_1) = \alpha_1^{-1}(i, q_1) \quad (i \in \Sigma_1 \setminus \{a\}, q_1 \in Q_1).$$

Case 5: $e_1 e_2$

Let $M(e_1)$ be $(\Sigma_1, X_1, Q_1, \alpha_1, \iota_1, F_1)$ and let $M(e_2)$ be $(\Sigma_2, X_2, Q_2, \alpha_2, \iota_2, F_2)$. By renaming states not contained by $\bar{Z}$, we assume $Q_1 \cap Q_2 \subseteq \bar{Z}$.

$M(e_1 e_2) = (\Sigma_1 \cup \Sigma_2, X_1 \cup X_2, Q_1 \cup Q_2, \alpha_3, \iota_3, F_1 F_2)$, where

$$\iota_3(x) = \begin{cases} \iota_1(x) \cup \iota_2(x) & (x \in X_1 \cap X_2), \\ \iota_1(x) & (x \in X_1 \setminus X_2), \\ \iota_2(x) & (x \in X_2 \setminus X_1), \end{cases}$$

$$\alpha_3^{-1}(i, q) = \begin{cases} \alpha_1^{-1}(i, q) & ((i, q) \in \Sigma_1 \times Q_1, \\ & \quad (i, q) \notin \Sigma_2 \times Q_2), \\ \alpha_2^{-1}(i, q) & ((i, q) \in \Sigma_2 \times Q_2, \\ & \quad (i, q) \notin \Sigma_1 \times Q_1), \\ \emptyset & (\text{otherwise}) \end{cases}$$

Case 6: $e_1 | e_2$

The construction of $M(e_1|e_2)$ is very similar to that of $M(e_1 e_2)$. The only difference is that the final state sequence set is $F_1 \cup F_2$ rather than $F_1 F_2$.

Case 7: e^*

Let $M(e)$ be $(\Sigma_1, X_1, Q_1, \alpha_1, \iota_1, F_1)$. Then, $M(e^*) = (\Sigma_1, X_1, Q_1, \alpha_1, \iota_1, F_1^*)$.

Case 8: $a\langle z \rangle$

$M(a\langle z \rangle) = (\{a\}, \{z\}, \{q, \bar{z}\}, \alpha_1, \iota_1, \{q\})$, where

$$\iota_1(z) = \bar{z},$$

$$\alpha_1^{-1}(a, q) = \{\bar{z}\},$$

$$\alpha_1^{-1}(a, \bar{z}) = \emptyset.$$

Case 9: $e_1 \circ_z e_2$

Let $M(e_1)$ be $(\Sigma_1, X_1, Q_1, \alpha_1, \iota_1, F_1)$ and let $M(e_2)$ be $(\Sigma_2, X_2, Q_2, \alpha_2, \iota_2, F_2)$. Again, we assume $Q_1 \cap Q_2 \subseteq \bar{Z}$.

Let $X_2' = X_2 \setminus \{z\}$ and $Q_2' = Q_2 \setminus \{\bar{z}\}$. Then, $M(e_1 \circ_z e_2) = (\Sigma_1 \cup \Sigma_2, X_1 \cup X_2', Q_1 \cup Q_2', \alpha_3, \iota_3, F_2)$, where

$$\iota_3(x) = \begin{cases} \iota_1(x) \cup \iota_2(x) & (x \in X_1 \cap X_2'), \\ \iota_1(x) & (x \in X_1 \setminus X_2'), \\ \iota_2(x) & (x \in X_2' \setminus X_1), \end{cases}$$

$$\alpha_3^{-1}(i, q) = \begin{cases} \alpha_1^{-1}(i, q) & ((i, q) \in \Sigma_1 \times Q_1, \\ & \quad (i, q) \notin \Sigma_2 \times Q_2'), \\ \alpha_2^{-1}(i, q) & ((i, q) \in \Sigma_2 \times (Q_2' \setminus \alpha_2(i, \bar{z})), \\ & \quad (i, q) \notin \Sigma_1 \times Q_1), \\ (\alpha_2^{-1}(i, q) \setminus \{\bar{z}\}) & ((i, q) \in \Sigma_2 \times \alpha_2(i, \bar{z}), \\ \quad \cup F_1 & \quad (i, q) \notin \Sigma_1 \times Q_1), \\ \emptyset & (\text{otherwise}) \end{cases}$$

Case 10: e^z

Let $M(e)$ be $(\Sigma_1, X_1, Q_1, \alpha_1, \iota_1, F_1)$. Then, $M(e^z) = (\Sigma_1, X_1, Q_1, \alpha_2, \iota_1, F_1)$, where

$$\alpha_2^{-1}(i, q) = \begin{cases} \alpha_1^{-1}(i, q) & (i \in \Sigma_1, q \in Q_1 \setminus \alpha_1(i, \bar{z})) \\ \alpha_1^{-1}(i, q) \cup F_1 & (i \in \Sigma_1, q \in \alpha_1(i, \bar{z})) \end{cases}$$

□

LEMMA 2. *Given a hedge automaton M, we can construct a hedge regular expression that represents $L(M)$.*

PROOF. We do not provide a whole proof, but sketch how a hedge regular expression is constructed from a given hedge automaton.

Informally, the key idea is to pick a state of a hedge automaton, decompose the hedge automaton to small hedge automata at every occurrence of that state, and combine these small hedge automata with operators of hedge regular expressions. By repeatedly applying this procedure, we eventually have a hedge regular expression.

Let M be $(\Sigma, X, Q, \alpha, \iota, F)$, and q be a state in Q. Consider a hedge accepted by M and those of its nodes to which M assigns q. At these nodes, we decompose this hedge into small hedges. Upper hedges contain nodes of the form $a\langle q \rangle$, where a is a symbol in Σ. Such nodes are said to be *connector nodes*. If we embed lower hedges within connector nodes of upper hedges, we can reconstruct the original hedge. Since connector nodes have states in Q as labels of leaf nodes, upper small hedges are contained by $\mathcal{H}[\Sigma, X \cup Q]$ rather than $\mathcal{H}[\Sigma, X]$.

After decomposing hedges at a state q, we would like to uniquely determine the label of the connector nodes. For this purpose, we assume that, for each state q in Q, there exists one and at most one symbol $a \in \Sigma$ such that $\alpha^{-1}(a, q) \neq \emptyset$ and denote this symbol by $\zeta(q)$. If this assumption does not hold, we only have to use $(Q \times \Sigma) \cup Q$ as a state set and reconstruct α so that it returns the input symbol as the first component.

Next, we create another deterministic hedge automaton M' by extending M for decomposed hedges. For each q in Q, we introduce a state $\hat{q}$ and denote $\{\hat{q} \mid q \in Q\}$ by $\hat{Q}$. We also introduce a dead-end state $q_\perp$, which is not contained by Q. New transition functions ι' and α' are defined below:

$$\iota'(q) = \hat{q} \quad (q \in Q)$$
$$\iota'(x) = \iota(x) \quad (x \in X)$$
$$\alpha'(a,u) = \begin{cases} q & (u = \hat{q}, a = \zeta(q)), \\ \alpha(a,u) & (u \in Q^*), \\ q_\perp & (\text{otherwise}), \end{cases}$$

Now, $M' = (\Sigma, X \cup Q, Q \cup \hat{Q} \cup \{q_\perp\}, \alpha', \iota', F)$ is a deterministic hedge automaton. Moreover, $L(M') \cap \mathcal{H}[\Sigma, X] = L(M)$.

Let Q_1, Q_2 be subsets of Q, and q be a state in Q. We define $R(q, Q_1, Q_2)$ as the set of hedges u ($\in \mathcal{H}[\Sigma, X \cup Q]$) such that

- if a node is neither a leaf nor a connector, the state assigned to this node by M' is contained by Q_1,
- if a node is a connector, the state assigned to this node by M' is contained by Q_2, and
- $\lceil M' \| u \rceil$ is contained by $\alpha^{-1}(\zeta(q), q)$.

We show that $R(q, Q_1, Q_2)$ can be represented by hedge regular expressions by induction on the cardinality of Q_1.

Base case: Since the cardinality of Q_1 is 0, this set is empty. Thus, any node in a hedge in $R(q, \emptyset, Q_2)$ is either a leaf node (i.e., labeled with a variable in X or state in Q) or a connector node. For each state $r \in Q$, the set of leaf nodes or connector nodes which reach r is finite and can thus be represented by a hedge regular expression, say e_r. By replacing each r in $\alpha^{-1}(\zeta(q), q)$ with e_r, we have a hedge regular expression representing $R(q, \emptyset, Q_2)$.

Inductive case: We consider $R(q, Q_1 \cup \{p\}, Q_2)$, where $p \notin Q_1$.

Obviously, $R(q, Q_1 \cup \{p\}, Q_2)$ includes $R(q, Q_1, Q_2)$. Consider a hedge in $R(q, Q_1 \cup \{p\}, Q_2) \setminus R(q, Q_1, Q_2)$. The computation of such a hedge has at least one occurrence of state p. We decompose this hedge at each of the highest occurrences of state p. The set of hedges below such an occurrence of p is captured by $R(p, Q_1 \cup \{p\}, Q_2)$, while the set of hedges above such an occurrence of p is captured by $R(q, Q_1, Q_2 \cup \{p\})$. Thus, we have the following equation.

$$R(q, Q_1 \cup \{p\}, Q_2) = R(p, Q_1 \cup \{p\}, Q_2) \circ_p R(q, Q_1, Q_2 \cup \{p\}) \cup R(q, Q_1, Q_2).$$

Again, $R(p, Q_1 \cup \{p\}, Q_2)$ includes $R(p, Q_1, Q_2)$. Consider a hedge in $R(p, Q_1 \cup \{p\}, Q_2) \setminus R(p, Q_1, Q_2)$. The computation of such a hedge has at least one occurrence of state p. We decompose this hedge at each of the lowest occurrences of state p. The set of hedges below such an occurrence of p is captured by $R(p, Q_1, Q_2)$, while the set of hedges above such an occurrence of p is captured by $R(p, Q_1 \cup \{p\}, Q_2 \cup \{p\})$. Thus, we have the second equation.

$$R(p, Q_1 \cup \{p\}, Q_2) = R(p, Q_1, Q_2) \circ_p R(p, Q_1 \cup \{p\}, Q_2 \cup \{p\}) \cup R(p, Q_1, Q_2).$$

Finally, we consider $R(p, Q_1 \cup \{p\}, Q_2 \cup \{p\})$. At each occurrence of state p, we decompose this hedge into hedges in $R(p, Q_1, Q_2 \cup \{p\})$. Thus, we have the third equation.

$$R(p, Q_1 \cup \{p\}, Q_2 \cup \{p\}) = R(p, Q_1, Q_2 \cup \{p\})^p.$$

It follows from these three equations that

$$R(q, Q_1 \cup \{p\}, Q_2) = (R(p, Q_1, Q_2) \circ_p R(p, Q_1, Q_2 \cup \{p\})^p \cup R(p, Q_1, Q_2)) \circ_p R(q, Q_1, Q_2 \cup \{p\}) \cup R(q, Q_1, Q_2).$$

The right-hand side of this equation does not use $Q_1 \cup \{p\}$ but rather uses Q_1 as the second argument of R. By the induction hypothesis, the right-hand side can be represented by a hedge regular expression. Thus, the left-hand side can also be represented.

By replacing each $r \in Q$ occurring in F with a hedge regular expression representing $R(r, Q, \emptyset)$, we have a hedge regular expression representing $L(M)$. □

The next theorem directly follows from the two lemmas.

THEOREM 2. *Hedge regular expressions and hedge automata are equally expressive.*

5. POINTED HEDGE REPRESENTATIONS

In this section, we introduce pointed hedge representations, which naturally extend path expressions. Pointed binary tree representations were originally introduced by [31, 28] and their applications to structured documents were studied [22]. We extend them for hedges rather than binary trees. Furthermore, pointed hedge representations use hedge regular expressions rather than hedge (or tree) automata.

Definition 13. A *pointed hedge* over Σ and X is a hedge with one substitution symbol η (in other words, an element of $\mathcal{H}[\Sigma, X, \{\eta\}]$) such that η occurs once and only once.

Definition 14. The *product* of pointed hedges u and v, denoted by $u \oplus v$, is the only element of $\{u\} \circ_\eta v$.

For example, $a\langle x\rangle b\langle\eta\rangle$ and $a\langle x\rangle b\langle c\langle\eta\rangle y\rangle$ are pointed hedges; the product of the former and latter is $a\langle x\rangle b\langle c\langle a\langle x\rangle b\langle\eta\rangle\rangle y\rangle$ (Figure 1).

The associative law holds; that is, $(u \oplus v) \oplus w = u \oplus (v \oplus w)$ for any pointed hedges u, v, w.

Definition 15. A *pointed base hedge* is a pointed hedge of the form $u_1 a\langle\eta\rangle u_2$, where u_1, u_2 are hedges and a is a symbol in Σ.

For example, $a\langle x\rangle b\langle\eta\rangle$ is a pointed base hedge. On the other hand, $a\langle x\rangle b\langle c\langle\eta\rangle y\rangle$ is not.

Any pointed hedge can be uniquely decomposed into a sequence of pointed base hedges. After such unique decomposition, η is introduced as the child of each node in the path from the top-level to η (Figure 2). For example, $a\langle x\rangle b\langle c\langle\eta\rangle y\rangle$ can be decomposed into $c\langle\eta\rangle y$ and $a\langle x\rangle b\langle\eta\rangle$.

Definition 16. A *pointed base hedge representation* is a triplet (e_1, a, e_2), where $a \in \Sigma$ and e_1, e_2 are hedge regular expressions.

Definition 17. A pointed base hedge *matches* (e_1, a, e_2) if it is of the form $u_1 a\langle\eta\rangle u_2$, where u_1 and u_2 are contained by $L(e_1)$ and $L(e_2)$, respectively.

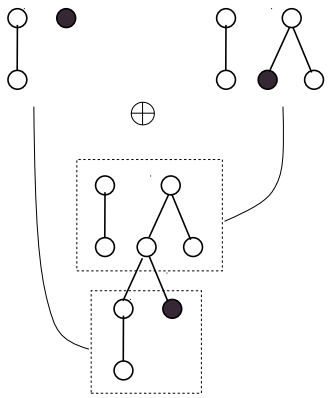

Figure 1: Pointed hedges and their product (The left-top example is $a\langle x\rangle b\langle\eta\rangle$ and the right-top example is $a\langle x\rangle b\langle c\langle\eta\rangle y\rangle$. Their product is $a\langle x\rangle b\langle c\langle a\langle x\rangle b\langle\eta\rangle\rangle y\rangle$)

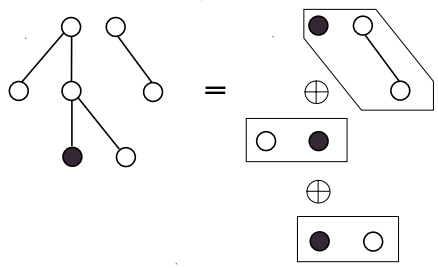

Figure 2: Decomposition of pointed hedges (the right-hand side begins at the bottom and ends at the top).

As an example, consider a pointed base hedge representation $(a\langle z\rangle^{*z}, b, a\langle z\rangle^{*z})$, where $a\langle z\rangle^{*z}$ is an example hedge regular expression shown in the previous section. Recall that this hedge regular expression generates all hedges such that every symbol is a and every substitution symbol is z. Therefore, a pointed hedge matches $(a\langle z\rangle^{*z}, b, a\langle z\rangle^{*z})$, when the parent of η is labeled with b, the other symbols are a, and the substitution symbols are z.

Definition 18. A *pointed hedge representation* is a regular expression over a finite set of pointed base hedge representations.

Definition 19. A pointed hedge u *matches* a pointed hedge representation e if

- u is decomposed into a sequence of pointed base hedges $u_1, u_2, \ldots, u_k$ (i.e., $u = u_1 \oplus u_2 \oplus \ldots \oplus u_k$),
- e generates $(e_{11}, a_1, e_{12}), (e_{21}, a_2, e_{22}), \ldots, (e_{k1}, a_k, e_{k2})$, and
- u_i matches (e_{i1}, a, e_{i2}) for every i $(1 \leq i \leq k)$.

As an example, consider a pointed hedge representation $(a\langle z\rangle^{*z}, b, a\langle z\rangle^{*z})^*$. A pointed hedge matches this pointed hedge representation if (1) the parent of η is labeled with b, (2) all its ancestor nodes are labeled with b, (3) all other nodes are labeled with a, and (4) the substitution symbols are z.

6. SELECTION QUERIES

Having introduced hedge regular expressions and pointed hedge representations, we can introduce selection queries. They form the backbone for queries on structured documents and have been studied by many researchers [22, 25, 26, 24, 23].

Definition 20. A *selection query* is $\mathbf{select}(e_1, e_2)$, where e_1 is a hedge regular expression and e_2 is a pointed hedge representation.

To define how selection queries work, we need two auxiliary definitions.

Definition 21. The *subhedge* of a node n in a hedge u is the hedge comprising all descendants of n. The *envelope* of a node n in a hedge u is the result of removing the subhedge of n and adding η as the child of n.

For example, consider $ba\langle a\langle bx\rangle b\rangle$. This hedge has two nodes at the top-level, and the second top-level node has two second-level nodes. The subhedge and envelope of the first second-level node is bx and $ba\langle a\langle\eta\rangle b\rangle$, respectively.

Definition 22. A node n in a hedge u is located by selection query $\mathbf{select}(e_1, e_2)$ if the subhedge of n is contained by $L(e_1)$ and the envelope of n is contained by $L(e_2)$.

As an example, let e_1 be $(b|x)^*$ and e_2 be $(\epsilon, a, b)(b, a, \epsilon)$. Then, the first second-level node of the second top-level node of $ba\langle a\langle bx\rangle b\rangle$ is located by $\mathbf{select}(e_1, e_2)$.

Next, we study evaluation of selection queries. Evaluation of pointed hedge regular representations is not straightforward and we will consider such evaluation in Section 7. Here we consider evaluation of hedge regular expressions.

The following theorem for evaluating hedge regular expressions appears trivial, but it is also useful for schema transformation. Furthermore, we will later introduce a similar theorem for pointed hedge representations.

THEOREM 3. *Given a hedge regular expression e, we can construct a deterministic hedge automaton $M \downarrow e$ and a set of marked states such that*

- *any hedge u is accepted by $M \downarrow e$, and*
- *a node occurring in u is located by e if and only if $(M \downarrow e)\|u$ assigns a marked state to this node.*

PROOF. Given a hedge regular expression e_1, we first construct a deterministic hedge automaton

$$M_1 = (\Sigma, X, Q, \iota, \alpha, F)$$

from e_1.

For a string $u = (q_1, i_1)(q_2, i_2)\ldots(q_n, i_n)$ over $Q \times \{0, 1\}$, we denote $q_1 q_2 \ldots q_n$ by $u[1]$, and denote $i_1 i_2 \ldots i_n$ by $u[2]$.

Now, we can construct $M \downarrow e_1$.

$$M \downarrow e_1 = (\Sigma, X, Q \times \{0,1\}, \iota', \alpha', (Q \times \{0,1\})^*)$$

where

$$\begin{aligned}
\iota'(x) &= (\iota(x), 0), \\
\alpha'^{-1}(a,(q,1)) &= \{u \in (Q \times \{0,1\})^* \mid u[1] \in \alpha^{-1}(a,q) \cap F\}, \\
\alpha'^{-1}(a,(q,0)) &= \{u \in (Q \times \{0,1\})^* \mid u[1] \in \alpha^{-1}(a,q) \setminus F\}, \\
F' &= \{u \in (Q \times \{0,1\})^* \mid u[1] \in F\}.
\end{aligned}$$

The rest of the proof is straightforward. □

For example, given $(b|x)^*$, we can construct a deterministic hedge automaton $M = (\Sigma, X, Q, \iota, \alpha, F)$ where

$$\begin{aligned}
\Sigma &= \{a, b\}, \\
X &= \{x\}, \\
Q &= \{q_0, q_1, q_2\}, \\
\iota(x) &= q_1, \\
\alpha(i,u) &= \begin{cases} q_0 & (i = b, u = \epsilon), \\ q_2 & (\text{otherwise}), \end{cases} \\
F &= L((q_0|q_1)^*)
\end{aligned}$$

We then construct

$$M \downarrow e = (\Sigma, X, Q \times \{0,1\}, \iota', \alpha', (Q \times \{0,1\})^*)$$

where

$$\begin{aligned}
\iota'(x) &= (q_1, 0), \\
\alpha'(i,u) &= \begin{cases} (q_0,0) & (i = b, u = \epsilon), \\ (q_2,0) & (u \in L(((q_0,0)|(q_0,1)| \\ & \qquad (q_1,0)|(q_1,1))^*), \\ (q_2,1) & (\text{otherwise}). \end{cases}
\end{aligned}$$

Then, the computation of $ba\langle a\langle bx\rangle b\rangle$ by $M \downarrow e$ is a hedge $(q_2,0)(q_2,0)\langle(q_2,1)\langle(q_0,0)(q_1,0)\rangle(q_2,0)\rangle$. The state assigned to the first second-level node of the second top-level node is $(q_2,1)$. This node is thus located, while the other nodes are not located.

Next, we consider complexities of hedge regular expression evaluation. Conversion from a hedge regular expression to a non-deterministic hedge automaton requires time linear to the size of the expression, but conversion from a non-deterministic hedge automaton to a deterministic one requires time exponential to the size of the non-deterministic hedge automaton. Note that these conversions are performed before we start to examine hedges. Once a deterministic hedge automaton is created, we can evaluate it against a given hedge by traversing the hedge in the depth-first manner. Such evaluation takes time linear to the number of nodes.

Finally, we consider expressiveness of selection queries. It is well known that a language of finite strings is accepted by a string automaton if and only if it is MSO-definable [7]. This observation has be generalized for unranked trees or hedges [26]. It readily follows that our selection queries are definable in MSO. Conversely, by applying the techniques by Neven and Schwentick [26, 27], one can show that our selection queries express exactly the MSO definable selection queries.

7. EVALUATION OF POINTED HEDGE REPRESENTATIONS

In Section 6, we have studied how we evaluate hedge regular expression e_1 of a selection query $\mathbf{select}(e_1, e_2)$. In this section, we show an algorithm for evaluating pointed hedge representation e_2. Given a hedge, this algorithm locates those nodes whose envelopes match the pointed hedge representation by traversing the hedge twice.

We informally present the key idea. Although the underlying alphabet of e_2 contains many hedge regular expressions, we would like to evaluate all of them by performing one traversal. This is done by creating a single deterministic hedge automaton which captures all of these hedge regular expressions. In other words, this automaton simluates execution of the deterministic hedge automata created from these hedge regular expressions.

For each node, this automaton computes a sequence of states for elder siblings, and another for younger siblings. By examining the label of this node and these two sequences, we can determine which of the pointed based hedge representations occurring in e_2 is satisfied. This idea is formally captured by the following theorem.

THEOREM 4. *Given a pointed hedge representation r, we can construct*

- *a deterministic hedge automaton $M = (\Sigma, X, Q, \alpha, \iota, \emptyset)$,*
- *a right-invariant equivalence relation $\equiv$ of finite index over Q^*, and*
- *a regular set L over $(Q^*/\equiv) \times \Sigma \times (Q^*/\equiv)$*

such that

for any pointed hedge u,

u matches r if and only if $\Gamma_1\Gamma_2\ldots\Gamma_n \in L$, where

- *$u_{11}a_i\langle\eta\rangle u_{12}, u_{21}a_i\langle\eta\rangle u_{22}, \ldots, u_{n1}a_n\langle\eta\rangle u_{n2}$ is the decomposition of u, and*
- *$\Gamma_i = ([\lceil M \| u_{i1} \rceil]_\equiv, a_i, [\lceil M \| u_{i2} \rceil]_\equiv)$ $(1 \le i \le n)$.*

PROOF. Recall that a pointed hedge representation is a regular expression over a finite set of pointed base hedge representations. Let this finite set for r be $\{(e_{11}, a_1, e_{12}), (e_{21}, a_2, e_{22}), \ldots, (e_{n1}, a_n, e_{n2})\}$.

For each e_{i1} and e_{i2} $(1 \le i \le n)$, we construct deterministic hedge automata M_{i1} and M_{i2}.

Without loss of generality, we can assume that M_{i1}, M_{i2} share the state set Q, the transition function ι, and the transition function α. That is,

$$\begin{aligned}
M_{i1} &= (\Sigma, X, Q, \iota, \alpha, F_{i1}) \ \ (1 \le i \le n), \\
M_{i2} &= (\Sigma, X, Q, \iota, \alpha, F_{i2}) \ \ (1 \le i \le n)
\end{aligned}$$

If they did not share Q, ι, α, we only have to use the cross product of all state sets as a new state set; that is, we use $Q_{11} \times Q_{12} \times Q_{21} \times Q_{22} \times \ldots Q_{n1} \times Q_{n2}$ as the state set, where Q_{ij} is the state set of M_{ij}. We then reconstruct transition functions and final state sequences for this new state set.

It is well known that any regular (string) set is the union of some of the equivalence classes of a right-invariant equivalence relationship of finite index; we say that the equivalence relationship *saturates* the regular set. Since F_{i1} $(1 \le i \le n)$

is regular over Q, it is saturated by a right-invariant equivalence relationship $\equiv_{i1}$ of finite index over Q^*. Likewise, F_{i2} $(1 \leq i \leq n)$ is saturated by another right-invariant equivalence relationship $\equiv_{i2}$ of finite index over Q^*. Let

$$\equiv \ = \ (\equiv_{11} \cap \equiv_{12} \cap \equiv_{21} \cap \equiv_{22} \cap \cdots \cap \equiv_{n1} \cap \equiv_{n2}).$$

In other words, $\equiv$ holds if and only if $\equiv_{11}$, $\equiv_{12}$, $\equiv_{21}$, $\equiv_{22}$, ..., $\equiv_{n1}$, $\equiv_{n2}$ hold. Then, $\equiv$ is a right-invariant equivalence relationship of finite index over Q^*. Furthermore, $\equiv$ saturates F_{i1} and F_{i2} $(1 \leq i \leq n)$.

Assume that

$$F_{i1} = C_{i1}^1 \cup C_{i1}^2 \cup \ldots \cup C_{i1}^{n_{i1}}$$

and

$$F_{i2} = C_{i2}^1 \cup C_{i2}^2 \cup \ldots \cup C_{i2}^{n_{i2}},$$

where $C_{i1}^{j_1}, C_{i2}^{j_2}$ $(1 \leq j_1 \leq n_{i1}, 1 \leq j_2 \leq n_{i2})$ are equivalence classes of $\equiv$ over Q^*. Then, a pointed base hedge $u_1 a \langle \eta \rangle u_2$ matches (e_{i1}, a_i, e_{i2}) if and only if

$$([\lceil M \| u_1 \rceil]_{\equiv}, a, [\lceil M \| u_2 \rceil]_{\equiv})$$

is contained by

$$\bigcup_{1 \leq j_1 \leq n_{i1}, 1 \leq j_2 \leq n_{i2}} C_{i1}^{j_1} \times \{a_i\} \times C_{i2}^{j_2}.$$

Let ξ be a mapping from a pointed base hedge representation to a subset of $(Q^*/\equiv) \times \Sigma \times (Q^*/\equiv)$ such that $\xi((e_{i1}, a_i, e_{i2}))$ is the set shown above. ξ can be naturally extended as a homomorphism from sequences of pointed base hedge representations. Let

$$L = \xi(L(r)).$$

Since the image of a regular language by a homomorphism is regular, L is regular. Obviously, a pointed hedge u matches r if and only if

$$\begin{aligned}&([\lceil M \| u_{11} \rceil]_{\equiv}, a_1, [\lceil M \| u_{12} \rceil]_{\equiv})\\ &\quad ([\lceil M \| u_{21} \rceil]_{\equiv}, a_2, [\lceil M \| u_{22} \rceil]_{\equiv}) \ldots\\ &\quad\quad ([\lceil M \| u_{n1} \rceil]_{\equiv}, a_n, [\lceil M \| u_{n2} \rceil]_{\equiv})\end{aligned}$$

is contained by L, where the decomposition of u is $u_{11} a_i \langle \eta \rangle u_{12}$, $u_{21} a_i \langle \eta \rangle u_{22}$, ..., $u_{n1} a_n \langle \eta \rangle u_{n2}$. □

In this construction, conversion from hedge regular expressions to deterministic hedge automata takes time exponential to the size of hedge regular expressions. Construction of right-invariant equivalence classes $\equiv_{i1}, \equiv_{i2}$ requires determinization of finite (string) automata, thus taking exponential time. The rest of the construction takes polynomial time. Thus, this construction takes exponential time.

Consider the mirror image of L, namely $\{w_k \ldots w_2 w_1 \mid w_1 w_2 \ldots w_k \in L\}$. Since the mirror image of a regular set is regular, we can construct a deterministic (string) automaton

$$N = (S, \mu, s_0, S_{\text{fin}})$$

that accepts this set, where S is a finite set of states, s_0 $(\in S)$ is a start state, and S_{fin} $(\subseteq S)$ is a set of final states.

Now, we are ready to sketch an algorithm for locating those nodes which satisfy a pointed hedge representation. This algorithm requires two depth-first traversals, and takes time linear to the number of nodes. Intuitively speaking, deterministic hedge automaton M and equivalence relation$\equiv$ are evaludated during the first traversal, and then deterministic (string) automaton N is evaluated during the second traversal.

Algorithm 1

First traversal Let e be a node. Let $e_1, e_2, \ldots, e_k$ be its elder siblings, $e'_1, e'_2, \ldots, e'_l$ be its younger siblings.

- When we visit e, we compute an element of $Q^*/\equiv$ for the sequence of states assigned to $e_1, e_2, \ldots, e_i$. The computed element, together with the number i, is recorded at the parent of e.
- When we leave e, we assign a state in Q to e by computing α or ι.

  We start computing an element of $Q^*/\equiv$ for the sequence of states assigned to $e'_1, e'_2, \ldots, e'_l$; this computation is completed when we later leave e'_l.

  If e does not have younger siblings (i.e., $l = 0$), we compute an element of $Q^*/\equiv$ for the sequence of states assigned to $e_1, e_2, \ldots, e_k, e$. The computed value, together with the number 1, is recorded at the parent of e. We also compute another element of $Q^*/\equiv$ for the sequence of states assigned to $e_2, e_3, \ldots, e_k, e$. Again, the computed value, together with the number 2, is recorded at the parent of e. And so forth.

Second traversal We only have to evaluate N so as to locate elements which match the given pointed hedge representation.

When we visit a node e, by examining its label and value recorded at its parent, we can determine an element of $(Q^*/\equiv) \times \Sigma \times (Q^*/\equiv)$.

By applying μ to this element and the state (in S) of its parent node, we assign a state to e. If and only if a final state in S_{fin} is assigned to e, it is located by the given pointed hedge representation.

We note that Neven and Van den Bussche [25] have shown that selection queries definable by MSO formulas can be evaluated by traversing a tree twice.

8. MATCH-IDENTIFYING HEDGE AUTOMATA

In this section, to facilitate schema transformation, we construct a match-identifying hedge automaton. A match-identifying hedge automaton accepts the same language as does the input schema, but further identifies matches to hedge regular expressions and pointed hedge representations at the schema level.

For each query operation, an output schema can be created by modifying the match-identifying automaton as appropriate to the query operation. In the case of selection, we only have to use marked states as final state sequences (To be precise, we have to use only those marked states from which final state sequences can be reached).

As for a hedge regular expression e, Theorem 3 ensures that we can construct a hedge automaton $M \downarrow e_1$ that captures e_1. As for a pointed hedge representation, the following theorem provides a similar solution.

THEOREM 5. *Given a pointed hedge representation e_2, we can construct a non-deterministic hedge automaton $M \uparrow e_2$ and a set of marked states such that*

- *for any hedge u in $\mathcal{H}[\Sigma, X]$, there exists one and at most one successful computation of u, and*
- *a node occurring in u is located by e_2 if and only if the only successful computation of u assigns a marked state to this node.*

If this theorem holds, construction of match-identifying hedge automata is straightforward. Given an input schema, $M \downarrow e_1$, and $M \uparrow e_2$, we can construct a match-identifying hedge automaton by creating an intersection of these hedge automata.

PROOF. In Theorem 4, given a hedge regular expression, we have constructed a deterministic hedge automaton $M = (\Sigma, X, Q, \alpha, \iota, \emptyset)$, a right-invariant equivalence relation $\equiv$ of finite index over Q^*, and a regular set L over $(Q^*/\equiv) \times \Sigma \times (Q^*/\equiv)$. We have also constructed a deterministic (string) automaton $N = (S, \mu, s_0, S_{\rm fin})$ that accepts the mirror image of L. On the basis of M, $\equiv$, L, and N, we construct a match-identifying non-deterministic hedge automaton.

The key idea is to make a non-deterministic (string) automaton N' which simulates N in reverse (Figure 3). That is,

- if N has a transition labeled δ from a state s_1 to another state s_2, then N' has a transition labeled δ from s_2 to s_1 via δ, where δ is a pointed base hedge representation,
- the start state of N is a final state of N', and
- any state of N is a start state of N'.

Formally, $N' = (S, \mu', S, \{s_0\})$ where $\mu'(\delta, s_2) \ni s_1$ if and only if $\mu(\delta, s_1) = s_2$.

Suppose that N' has successful computations (sequences of states) for a string. If we reverse these computations, we obtain computations of N for the mirror image of this string. Since N is a deterministic automaton, it has only one computation per string. Therefore, each string has one and at most one successful computation by N'. At each point in a string, N' may have multiple choices. But only one of them leads to a final state.

We construct a match-identifying non-deterministic hedge automaton

$$M' = (\Sigma, X, Q', \kappa, \beta, F').$$

This automaton simulates M (which is deterministic) and also simulates N' for every path from a leaf node to the top-level. Thus, any hedge has one and at most one successful computation by M'.

First, the set of states Q' is defined below:

$$Q' = (Q \times S \times \Sigma) \cup (Q \times \{s_\perp\} \times \{a_\perp\}).$$

Since we intend to use $(Q^*/\equiv) \times \Sigma \times (Q^*/\equiv)$ in the definition of β, a state in Q' comprises a symbol in Σ. Use of S and Q in Q' allows simulation of N and M_{i1}, M_{i2}, respectively. $s_\perp$ and $a_\perp$ are additional values for leaf nodes.

A set $Q'_{\rm mark}$ of marked states is defined below:

$$Q'_{\rm mark} = Q \times S_{\rm fin} \times \Sigma.$$

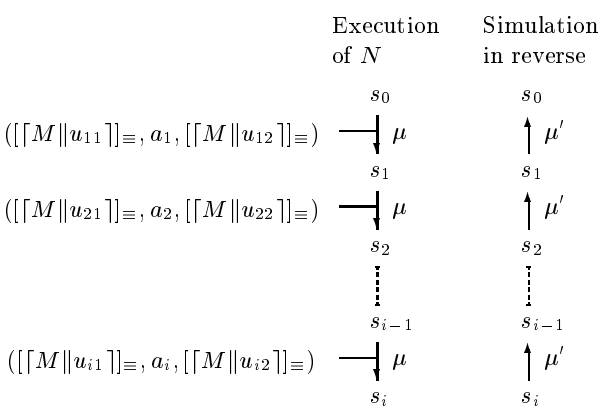

Figure 3: Simulation of N in reverse

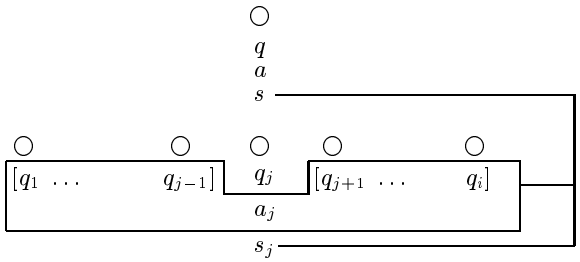

Figure 4: Given the equivalence class of $q_1 \ldots q_{j-1}$, the label a_j of the i-th child, the equivalence class of $q_{j+1} \ldots q_i$, and state s assigned to the parent element, function μ must return s_j.

Use of $S_{\rm fin}$ implies that N' begins with one of its start states or N reaches one of its final states.

Mapping κ from Σ to Q' is defined below:

$$\kappa(x) = (\iota(x), s_\perp, a_\perp).$$

Next, we define a function β from $\Sigma \times Q'^*$ to the power set of Q'. Given a symbol and a sequence of states in Q', this function returns a set of states in Q'. The first component ($\in Q$) simulates α. The second component ($\in S \cup \{s_\perp\}$) simulates N' (Figure 4). The third component ($\in \Sigma \cup \{a_\perp\}$) is the symbol given as an input.

$$\begin{aligned}&\beta(a, (q_1, s_1, a_1)(q_2, s_2, a_2) \ldots (q_i, s_i, a_i)) = \\ &\quad \{(\alpha(a, q_1 q_2 \ldots q_i), s, a) \mid \\ &\qquad \text{for every } j\ (1 \le j \le i), \text{ either } a_j = a_\perp \text{ or} \\ &\qquad s \in \mu'(([q_1 \ldots q_{j-1}]_\equiv, a_j, [q_{j+1} \ldots q_i]_\equiv), s_j)\}\end{aligned}$$

Next, we show that β satisfies the regularity condition. We can easily show that $\beta^{-1}(a, (q, s, b))$ $(a, b \in \Sigma, a \ne b, q \in Q, s \in S)$ and $\beta^{-1}(a, (q, s_\perp, a_\perp))$ $(q \in Q)$ are regular.

$$\begin{aligned}\beta^{-1}(a, (q, s, b)) &= \emptyset \\ \beta^{-1}(a, (q, s_\perp, a_\perp)) &= \emptyset\end{aligned}$$

Next, we consider $\beta^{-1}(a, (q, s, a))$ $(q \in Q, s \in S, a \in \Sigma)$. The key idea is to eliminate those child node sequences which cause failures to simulate N'.

$$\beta^{-1}(a, (q, s, a)) = h(\alpha^{-1}(a, q)) \setminus \bigcup_{C_1, C_2 \in Q^*/\equiv} h(C_1)\, \Omega\, h(C_2),$$

where

$$\Omega = \{(q', s', a') \mid s' \neq \mu((C_1, a', C_2), s), q' \in Q\}$$

and h is a homomorphism such that

$$h(q) = (\{q\} \times S \times \Sigma) \cup \{(q, s_\perp, a_\perp).\}$$

Intuitively speaking, the inequality in the definition of Y implies that N fails to reach s' or N' fails to reach s.

Since regular sets are closed under homomorphisms, concatenation, and boolean operations, the inverse image of β is a regular set.

Finally, we construct a final state sequence set F'. The first component ($\in Q$) simulates F. The second component ($\in S \cup \{s_\perp\}$) and third component ($\in \Sigma \cup \{a_\perp\}$) are defined so that N' reaches its final state s_0.

$$\begin{aligned} F' = \{&(q_1, s_1, a_1)(q_2, s_2, a_2) \ldots (q_i, s_i, a_i) \mid \\ &\text{for every } j\ (1 \leq j \leq i),\ \text{either } a_j = a_\perp \text{ or} \\ &\quad s_0 \in \mu(([q_1 \ldots q_{j-1}]_\equiv, a_j, [q_{j+1} \ldots q_i]_\equiv), s_j)\} \end{aligned}$$

It remains to show that F' is regular. As in the construction of the inverse image of β, we can rewrite F as below:

$$F' = h(Q^*) \setminus \bigcup_{C_1, C_2 \in Q^*/\equiv} h(C_1) Y h(C_2)$$

where Ω and h are the same as in the inverse image of β.

Again, since regular sets are closed under homomorphisms, concatenation, and boolean operations, F' is a regular set. □

Finally, we show how this construction can be simplified when pointed hedge representations do not impose conditions on siblings or their descendants. In other words, such pointed hedge representations are traditional path expressions. In this case, $\equiv$ is $Q^* \times Q^*$. Thus, we can use Σ rather than $(Q^*/\equiv) \times \Sigma \times (Q^*/\equiv)$ We can also assume that the transition function μ of N is a mapping from $\Sigma \times S$ to S.

The match-identifying non-deterministic hedge automaton for a traditional path expression is

$$M' = (\Sigma, X, Q', \beta, \kappa, F')$$

and Q'_{mark} is the set of marked states where

$$\begin{aligned} Q' &= (S \times \Sigma) \cup (\{s_\perp\} \times \{a_\perp\}), \\ Q'_{\text{mark}} &= S_{\text{fin}} \times \Sigma, \\ \kappa(x) &= (s_\perp, a_\perp), \\ \beta^{-1}(a, (s, b)) &= \emptyset\ (a \neq b), \\ \beta^{-1}(a, (s_\perp, a_\perp)) &= \emptyset, \\ \beta^{-1}(a, (s, a)) &= (\{(s', a') \mid s' \in S, a' \in \Sigma, \\ &\qquad \mu(a', s) = s'\} \cup \{(s_\perp, a_\perp)\})^*, \\ F' &= (\{(s', a') \mid s' \in S, a' \in \Sigma, \\ &\qquad \mu(a', s_0) = s'\} \cup \{(s_\perp, a_\perp)\})^* \end{aligned}$$

9. CONCLUSIONS AND FUTURE WORKS

We have assumed XML documents as hedges and have presented a formal framework for XML queries. Our selection queries are combinations of hedge regular expressions and pointed hedge representations. A hedge regular expression captures conditions on descendant nodes. To locate nodes, a hedge regular expression is first converted to a deterministic hedge automaton and then it is executed by a single depth-first traversal. Meanwhile, a pointed hedge representation captures conditions on non-descendant nodes (e.g., ancestors, siblings, siblings of ancestors, and descendants of such siblings). To locate nodes, a pointed hedge representation is first converted to triplets: (1) a deterministic hedge automaton, (2) a finite-index right-invariant equivalence of states, and (3) a string automaton over the equivalence classes. Then, this triplet is executed by two depth-first traversals. Schema transformation is effected by identifying where in an input schema the given hedge regular expression and pointed hedge representation is satisfied.

Interestingly enough, as it turns out our framework exactly captures the selection queries definable by MSO, as do boolean attribute grammars and query automata. On the other hand, our framework has two advantages over MSO-driven approaches. First, conversion of MSO formulas to query automata or boolean attribute grammars requires non-elementary space, thus discouraging implementations. On the other hand, our framework employs determinization of hedge automaton, which requires exponential time. However, we conjecture that such determinization usually works, as does determinization of string automata. Second, (string) regular expressions have been so widely and successfully used by many users because they are very easy to understand. We hope that hedge regular expressions and pointed hedge representations will become commodities for XML in the near future.

There are some interesting open issues. First, is it possible to generalize useful techniques (e.g., optimization) developed for path expressions to hedge regular expressions and pointed hedge representations? Second, we would like to introduce variables to hedge regular expressions so that query operations can use the values assigned to such variables. For this purpose, we have to study unambiguity of hedge regular expressions. An ambiguous expression may have more than one way to match a given hedge, while an unambiguous expression has at most only one such way. Variables can be safely introduced to unambiguous expressions.

10. ACKNOWLEDGMENTS

I appreciate Kilho Shin who pointed me to hedge regular languages. Comments from anonymous reviewers were extremely helpful. Dongwon Lee, Murali Mani, Takeshi Umezawa, Akihiko Tozawa, and Tom Bauer read earlier drafts and provided many useful comments.

11. REFERENCES

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.

[2] S. Abiteboul and V. Vianu. Regular path queries with constraints. In *PODS 97*, 1997.

[3] T. Arnold-Moore, M. Fuller, and R. Sacks-Davis. System architecture of a content management server for XML document applications. *Markup Languages*, 2(1), 2000.

[4] R. Baeza-Yates and G. Navarro. Integrating contents and structure in text retrieval. *SIGMOD Record*, 25(1):67–79, Mar. 1996.

[5] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. W3C

Recommendation. http://www.w3.org/TR/REC-xml, Feburary 1998.

[6] A. Brüggemann-Klein and D. Wood. Caterpillars: A context specification technique. *Markup Languages: Theory and Practice*, 2(1):81–106, Winter 2000.

[7] J. Buchi. Weak second-order arithmetic and finite automata, 1960.

[8] P. Buneman, W. Fan, and S. Weinsten. Path constraints on semistructured and structured data. In *PODS 98*, 1998.

[9] J. Clark. TREX - tree regular expressions for XML, 2001. http://www.thaiopensource.com/trex/.

[10] J. Clark and S. DeRose. XML Path Language (XPath) version 1.0. W3C Recommendation. http://www.w3.org/TR/xpath, November 1999.

[11] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. "Tree Automata Techniques and Applications", 1997. http://www.grappa.univ-lille3.fr/tata.

[12] B. Courcelle. On recognizable sets and tree automata. In M. Nivat and H. Ait-Kaci, editors, *Resolution of Equations in Algebraic Structures*. Academic Press, 1989.

[13] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, Berlin, 1995.

[14] M. Fernandez, J. Simeon, and P. Wadler. XML query languages: Experiences and exemplars. http://www-db.research.bell-labs.com/user/simeon/xquery.html.

[15] M. Fernandez and D. Suciu. Optimizing regular path expressions using graph schemas. In *ICDE 98*, 1998.

[16] F. Gécseg and M. Steinby. Tree languages. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages, Volume 3, Beyond Words*, volume 3, pages 1–68. Springer-Verlag, 1997.

[17] H. Hosoya and B. C. Pierce. Regular expression pattern matching for XML. In *POPL 01*, 2001.

[18] ISO/IEC. *Information Technology – Document Description and Processing Languages – Regular Language Description for XML (RELAX) – Part 1: RELAX Core*, 2000. DTR 22250-1.

[19] A. Mendelzon and P. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6), 1995.

[20] A. R. Meyer. Weak monadic second order theory of successor is Not elementary-recursive. In *LOGCOLLOQ: Logic Colloquium*, volume 453 of *LNM*. Springer-Verlag, 1975.

[21] T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. In *PODS 00*, 2000.

[22] M. Murata. Transformation of documents and schemas by patterns and contextual conditions. In *PODP 96*, volume 1293 of *LNCS*. Springer-Verlag, 1997.

[23] 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.

[24] F. Neven. Extension of attribute grammars for structured document queries. In *DBPL 99*, 1999.

[25] F. Neven and J. V. den Bussche. Expressiveness of structured document query languages based on attribute grammars. In *PODS 98*, pages 11–17, 1998.

[26] F. Neven and T. Schwentick. Query automata. In *PODS 99*, 1999.

[27] F. Neven and T. Schwentick. Expressive and efficient pattern languages for tree-structured data. In *PODS 00*, 2000.

[28] M. Nivat and A. Podelski. Another variation on the common subexpression problem. *Discrete Mathematics*, 114:379–401, 1993.

[29] C. Pair and A. Quere. Définition et etude des bilangages réguliers. *Information and Control*, 13(6):565–593, Dec. 1968.

[30] Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *PODS 00*, 2000.

[31] A. Podelski. A monoid approach to tree automata. In M. Nivat and A. Podelski, editors, *Tree Automata and Languages*, pages 41–56. North-Holland, 1992.

[32] M. Takahashi. Generalizations of regular sets and their application to a study of context-free languages. *Information and Control*, 27:1–36, 1975.

[33] W. Thomas. Languages, automata and logic. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages, Volume 3, Beyond Words*, volume 3, pages 389–449. Springer-Verlag, 1997.

[34] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. "XML Schema Part 1: Structures", 2001. http://www.w3.org/TR/xmlschema-1/.

[35] P. Wadler. A formal semantics of patterns in XSLT. In *Markup Technologies 99*, 1999.