Transformation of Documents and Schemas by Patterns and Contextual Conditions

Makoto Murata[1]

Fuji Xerox Information Systems, Co., Ltd.,
KSP 9A7, 2-1 Sakado 3-chome, Takatsu-ku, Kawasaki-shi, Kanagawa 213, Japan

Abstract. On the basis of the tree-regular language theory, we study document transformation and schema transformation. A document is represented by a tree t, and a schema is represented by a tree-regular language $\mathcal{L}$. Document transformation is defined as a composition of a marking function $m_{\mathcal{C}}^{\mathcal{P}}$ and a linear tree homomorphism h, where $\mathcal{P}$ is a pattern and $\mathcal{C}$ is a contextual condition. Pattern $\mathcal{P}$ is a tree-regular language, and contextual condition $\mathcal{C}$ is a pointed tree representation. Marking function $m_{\mathcal{C}}^{\mathcal{P}}$ marks a node if the subtree rooted by this node matches $\mathcal{P}$ and the envelope (the rest of the tree) satisfies $\mathcal{C}$. Linear tree homomorphism h then rewrites the tree, for example, by deleting or renaming marked nodes. Schema transformation is defined by naturally extending document transformation; that is, the result of transforming a schema $\mathcal{L}$, denoted $h(m_{\mathcal{C}}^{\mathcal{P}}(\mathcal{L}))$, is $\{h(m_{\mathcal{C}}^{\mathcal{P}}(t)) \mid t \in \mathcal{L}\}$. Given a tree automaton that accepts $\mathcal{L}$, we can effectively construct a tree automaton that accepts $h(m_{\mathcal{C}}^{\mathcal{P}}(\mathcal{L}))$. This observation provides a theoretical basis for document transformation engines and document database systems.

1 Introduction

In this paper we study tree transformations for document information. Such tree transformations require more powerful patterns and contextual conditions than do tree transformations for programs (see Wilhelm [13]). Furthermore, tree transformations for document information must accompany transformations of tree schemas.

In preparation, we give general background on documents. Many document models, most notably SGML [9], introduce tree structures to documents. Document processing, such as the generation of new documents by assembling components of existing documents, can thus be computerized by writing tree transformation programs. Furthermore, some document models, including SGML, introduce *schemas* of documents. Schemas describe which types of nodes may appear in documents and which hierarchical relationships such nodes may have. Such described information helps programmers to write tree transformation programs. A schema is typically represented by an *extended context-free grammar*. A tree is *permitted* by this schema if it is a parse tree of that grammar. It is important that schemas be extended context-free grammars rather than simple context-free grammars; that is, we can specify that a section may have an arbitrary number of figures and paragraphs.

Let us consider three challenges in tree transformations for document processing (Figure 1). First, we need powerful patterns so as to specify relevant nodes in trees. As an example, assume that, given PODP'96 papers, we want to retrieve sections containing the word "preliminaries" in their titles. Among the subordinates of a section node, only the section title node is significant, and the type and number of the other subordinates are irrelevant. What is needed here is a powerful pattern that allows the section to have an arbitrary number of other subordinates of any type. As a result, the pattern cannot be a single tree with don't-care-characters [8]. Rather, it must be a specification of an infinite number of trees with any number of subordinates.

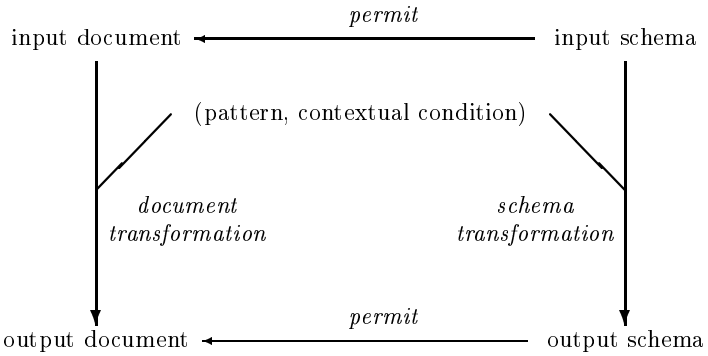

Fig. 1. Document transformation and schema transformation

Second, we need to introduce conditions on contexts, where the *context* of a node is its superior nodes, sibling nodes, and subordinates of these sibling nodes. For instance, assume that in the previous example we are interested only in those sections of *automaton* papers. Then, we want to introduce a contextual condition: *the section node must be directly or indirectly subordinate to a paper root node such that its title node or its summary node contains the word "automaton"*. This contextual condition concerns a node (the section node), a directly or indirectly superior node (the paper root node), and subordinate nodes of this superior node (the paper title node and the summary node).

Third, we need to transform schemas as well as trees. That is, given schemas for input documents and a tree transformation program, we must be able to create a schema for output documents. A document is permitted by this output schema if and only if that document can be generated by transforming some documents that are permitted by the input schemas. The creation of the output schema is crucial, as the output schema again helps programmers to write programs for further transformations. For example, Boeing wants to create documents for Boeing 747 maintenance by assembling those documents written by Boeing with those written by GE (an engine maker), and so on. These assem-

bled documents must be accompanied by schemas, since a purchasing airline such as Northwest wants to further update those documents by incorporating information specific to Northwest.

Researchers have made a number of attempts to overcome these challenges (see a survey [2]). Among such attempts are programming languages for documents [1] and data models for documents [3,4,6,7]. Even commercial products (OmniMark, etc.) and a standard (DSSSL [10]) have been developed. However, to the best of our knowledge, no attempt has fulfilled all three of the requirements (powerful patterns, powerful contextual conditions, and transformation of schemas). Some [1,6] completely ignore the schema transformation. Others [4,7] provide the schema transformation, but allow very weak patterns and no contextual conditions.

Why is it difficult to fulfill the three requirements simultaneously? One reason is that some contextual conditions lead to schemas that cannot be expressed by extended context-free grammars. For example, suppose that we want to delete all footnote nodes below appendix nodes, where "below appdendix nodes" is a contextual condition. After such deletion, paragraph nodes below appendix nodes do not have subordinate footnotes, but those below other sections do. However, no extended context-free grammar can capture such context dependencies. Another reason is that the class of local sets is mathematically intractable. (A set of trees is *local* if it is the set of parse trees of some extended context-free grammar.) If boolean operators are applied to local sets, the result is not always local. Moreover, some finite sets are not local.

To overcome these problems, we formalize a schema as a tree-regular language [5] rather than an extended context-free grammar. We believe that this approach is more appropriate for the following reasons. First, any schema in existing document models can be represented, since every local set is tree-regular. Second, any tree-regular language can be *localized*; that is, for every tree-regular language $\mathcal{L}$, we can construct a unique minimum local set that includes $\mathcal{L}$. Third, a pattern can also be formalized as a tree-regular language. Fourth, since the class of tree-regular languages forms a boolean algebra, we can apply boolean operators to both schemas and patterns. In particular, we can construct the intersection automaton of a schema automaton and a deterministic tree automaton genereated from a pattern, thus identifying where pattern matches occur.

Recent research on pointed trees [11,12] provides a very good basis for the study of contextual conditions. We represent a contextual condition with a pointed tree representation. We then construct an unambiguous non-deterministic tree automaton from that representation. Again, by automaton intersection, we can identify where the contextual condition is satisfied.

Key contributions of this paper are as follows:

- a powerful class of patterns,
- a powerful class of contextual conditions and an algorithm for testing them in a time linear to the size of a tree, and
- the construction of a minimally sufficient output schema.

The third contribution is probably the most significant. It provides a theoretical basis for document transformation engines and document database systems.

The remainder of this paper is organized as follows. In Section 2, we limit our concerns to strings rather than trees. After introducing preliminaries, we first formalize patterns, contextual conditions, and transformation rules. We then introduce algorithms for pattern matching and contextual condition checking. Finally, we show the construction of output schemas. In Section 3, we extend our observations from Section 2 for binary trees. Extension for general trees does not require any new ideas and is left to the reader.

2 Transformations of Strings

2.1 Preliminaries

A *string* over a finite alphabet Σ is an element of the free monoid Σ^*. The *addresses* of a string s are $1, 2, \ldots, n$, where n is the length of s. The character at an address i is denoted by $s[i]$. The *prefix* of s at i, denoted $s{\downarrow}i$, is a string $s[1]s[2]\ldots s[i \Leftrightarrow 1]s[i]$. The *suffix* of s at i, denoted $s{\uparrow}i$, is $s[i+1]\ldots s[n \Leftrightarrow 1]s[n]$. The mirror image of s, namely $s[n]s[n \Leftrightarrow 1]\ldots s[2]s[1]$, is denoted by s^{r}.

A *deterministic string automaton* (DSA) is a 5-tuple $<Q, \Sigma, \delta, q_0, Q_{\mathrm{f}}>$, where Q is a finite set of states, δ is a function from $Q \times \Sigma$ to Q, q_0 (initial state) is an element of Q, and Q_{f} (final states) is a subset of Q.

For a string s of length n and a DSA $M = <Q, \Sigma, \delta, q_0, Q_{\mathrm{f}}>$, the *computation* of s by M, denoted $M\|s$, is a string over Q such that the length of $M\|s$ is $n+1$, $(M\|s)[1] = q_0$, and $(M\|s)[i+1] = \delta((M\|s)[i], s[i])$ $(1 \leq i \leq n)$. If $(M\|s)[n+1] \in Q_{\mathrm{f}}$, this computation is *successful* and s is *accepted* by M. The set of strings accepted by M is denoted by $\mathcal{L}(M)$. If a language $\mathcal{L}$ (a set of strings) is accepted by some DSA M, $\mathcal{L}$ is *string-regular*.

A *non-deterministic string automaton* (NSA) is a 5-tuple $<Q, \Sigma, \delta, Q_0, Q_{\mathrm{f}}>$, where Q and Q_{f} are as above, δ is a relation from $Q \times \Sigma$ to Q, and Q_0 (initial states) is a subset of Q.

For a string s of length n and a NSA $M = <Q, \Sigma, \delta, Q_0, Q_{\mathrm{f}}>$, a *computation* of s by M is a string $\hat{s}$ over Q such that the length of $\hat{s}$ is n, $\hat{s}[1] \in Q_0$, and $\delta(\hat{s}[i], s[i], \hat{s}[i+1])$ $(1 \leq i \leq n)$. If $\hat{s}[n+1] \in Q_{\mathrm{f}}$, this computation is *successful*. If there is at least one successful computation, s is *accepted* by M. It is well known that a language is string-regular if and only if it is accepted by some NSA. If every string has at most one successful computation, M is *unambiguous*. If an unambiguous NSA M accepts s, we denote the successful computation of s by $M\|s$.

2.2 Transformation Rules

We first define marking functions and linear string homomorphisms, and then define transformation rules.

Marking Functions. A *pattern* $\mathcal{P}$ is a string-regular language. Given a string s, the prefix of s at an address i *matches* $\mathcal{P}$ if $s{\downarrow}i \in \mathcal{P}$. A *contextual condition* $\mathcal{C}$ is also a string-regular language. The suffix of s at i *satisfies* $\mathcal{C}$ if $s{\uparrow}i \in \mathcal{C}$.

For each symbol $x \in \Sigma$, we introduce a marked symbol $\overline{x}$. A marked alphabet $\overline{\Sigma}$ is defined as $\{\overline{x} \mid x \in \Sigma\}$.

A *marking function* $m_{\mathcal{C}}^{\mathcal{P}}$ is a mapping from Σ^* to $(\Sigma \cup \overline{\Sigma})^*$. Intuitively, $m_{\mathcal{C}}^{\mathcal{P}}$ marks the symbol at i for every address i such that the prefix of s at i matches $\mathcal{P}$, and the suffix of s at i satisfies $\mathcal{C}$. Formally, $m_{\mathcal{C}}^{\mathcal{P}}$ is defined as below:

$$m_{\mathcal{C}}^{\mathcal{P}}(s[1]s[2]\ldots s[k \Leftrightarrow 1]s[k]) = (s[1])'(s[2])'\ldots(s[k \Leftrightarrow 1])'(s[k])', \tag{1}$$

where

$$(s[i])' = \begin{cases} \overline{s[i]} & (s{\downarrow}i \in \mathcal{P}, s{\uparrow}i \in \mathcal{C}) \\ s[i] & (otherwise)\ . \end{cases} \tag{2}$$

Linear String Homomorphisms. A *replacement string* s' over Σ is a string over $\Sigma \cup \{z\}$ $(z \notin \Sigma)$ such that z either does not occur in s' or occurs as the first symbol. The result of replacing z with a string s is denoted $s'(z \leftarrow s)$.

Let $h_{\overline{\Sigma}}$ be a function from $\overline{\Sigma}$ to the set of replacement strings over Σ. The *linear string homomorphism* h determined by $h_{\overline{\Sigma}}$ is the function from $(\Sigma \cup \overline{\Sigma})^*$ to Σ^* defined as below:

$$h(s) = \begin{cases} h(s{\downarrow}(n \Leftrightarrow 1))\, s[n] & (s[n] \in \Sigma, n \text{ is the length of } s) \\ h_{\overline{\Sigma}}(s[n])(z \leftarrow h(s{\downarrow}(n \Leftrightarrow 1))) & (s[n] \in \overline{\Sigma}, n \text{ is the length of } s)\ . \end{cases} \tag{3}$$

Transformation Rules. A *transformation rule* is a triplet $<\mathcal{P}, \mathcal{C}, h>$, where $\mathcal{P}$ is a pattern, $\mathcal{C}$ is a contextual condition, and h is a linear string homomorphism. The result of applying this rule to a string s is defined as $h(m_{\mathcal{C}}^{\mathcal{P}}(s))$.

2.3 Applying Transformation Rules to Strings

To implement transformation rules, we need algorithms for pattern matching and contextual condition testing. Given a pattern $\mathcal{P}$, a contextual condition $\mathcal{C}$, and a string s, how do we find all i's such that $s{\downarrow}i \in \mathcal{P}$ and $s{\uparrow}i \in \mathcal{C}$?

It is simple to find all i's such that $s{\downarrow}i \in \mathcal{P}$. Let

$$\mathcal{P}_{\mathrm{M}} = <P, \Sigma, \alpha, p_0, P_{\mathrm{f}}> \tag{4}$$

be a DSA that accepts $\mathcal{P}$. Then, by executing $\mathcal{P}_{\mathrm{M}}$ for s, we obtain a computation $\mathcal{P}_{\mathrm{M}} \| s$. $s{\downarrow}i \in p$ if and only if the $(i+1)$-th state of this computation is a final state, namely $(\mathcal{P}_{\mathrm{M}} \| s)[i+1] \in P_{\mathrm{f}}$.

We can also use a DSA to find all i's such that $s{\uparrow}i \in \mathcal{C}$. Let

$$\mathcal{C}^{\mathrm{M}} = <C, \Sigma, \beta, c_0, C_{\mathrm{f}}> \tag{5}$$

be a DSA that accepts the mirror image of $\mathcal{C}$; that is, $\mathcal{C}^{\mathrm{M}}$ accepts a string s if and only if $s^{\mathrm{r}} \in \mathcal{C}$. Then, by executing $\mathcal{C}^{\mathrm{M}}$ for s^{r} (in other words, for s from its tail to its head), we obtain a computation $\mathcal{C}^{\mathrm{M}} \| s^{\mathrm{r}}$. $s \uparrow i \in \mathcal{C}$ if and only if the $(n \Leftrightarrow i+1)$-th character of this computation is a finite state, namely $(\mathcal{C}^{\mathrm{M}} \| s^{\mathrm{r}})[n \Leftrightarrow i+1] \in C_{\mathrm{f}}$.

Now that we have algorithms for pattern matching and contextual condition testing, it is a simple matter to write a computer program that applies a transformation rule to a string. The marking function $m_{\mathcal{P}}^{\mathcal{C}}$ can easily be derived from our algorithms, and the linear string homomorphism h is a simple recursive program.

2.4 Schema Transformation

Now, we can formally state the schema-transformation problem. We want to prove the following theorem.

Theorem 1. *The image of a string-regular language $\mathcal{L}$ over Σ by a transformation rule $<\mathcal{P}, \mathcal{C}, h>$ is string-regular over Σ.*

This theorem directly follows from Lemmas 2 and 3.

Lemma 2. *The image of $\mathcal{L}$ by $m_{\mathcal{P}}^{\mathcal{C}}$ is string-regular over $\Sigma \cup \overline{\Sigma}$.*

This lemma implies that after constructing the image of $\mathcal{L}$ by $m_{\mathcal{P}}^{\mathcal{C}}$, we no longer need the original language $\mathcal{L}$, the pattern $\mathcal{P}$, and the contextual condition $\mathcal{C}$. We only have to consider the constructed image.

Lemma 3. *The image of a string-regular language $\mathcal{L}'$ over $\Sigma \cup \overline{\Sigma}$ by h is string-regular over Σ.*

Lemma 3 is a special case of Theorem 4.16 (linear tree homomorphism) in Gécseg and Steinby [5]. Thus, we will not prove this lemma in this paper.

Proof (Lemma 2). We effectively construct an NSA that accepts the image as depicted by Figure 2. The key idea is the construction of a *match-identifying NSA* that identifies matches at the schema-level while accepting $\mathcal{L}$.

Recall that DSA $\mathcal{P}_{\mathrm{M}} = <P, \Sigma, \alpha, p_0, P_{\mathrm{f}}>$ accepts $\mathcal{P}$. By allowing any state as a final state, we obtain a DSA

$$\mathcal{P}_{\mathrm{I}} = <P, \Sigma, \alpha, p_0, P> \ . \tag{6}$$

Obviously, $\mathcal{P}_{\mathrm{I}}$ accepts any string. Furthermore, for any string s,

$$(\mathcal{P}_{\mathrm{I}} \| s)[i+1] \in P_{\mathrm{f}} \Leftrightarrow s \downarrow i \in \mathcal{P} \ . \tag{7}$$

Likewise, recall that DSA $\mathcal{C}^{\mathrm{M}} = <C, \Sigma, \beta, c_0, C_{\mathrm{f}}>$ accepts $\{s^{\mathrm{r}} \mid s \in \mathcal{C}\}$. The DSA obtained by allowing any state as a final state, namely $<C, \Sigma, \beta, c_0, C>$,

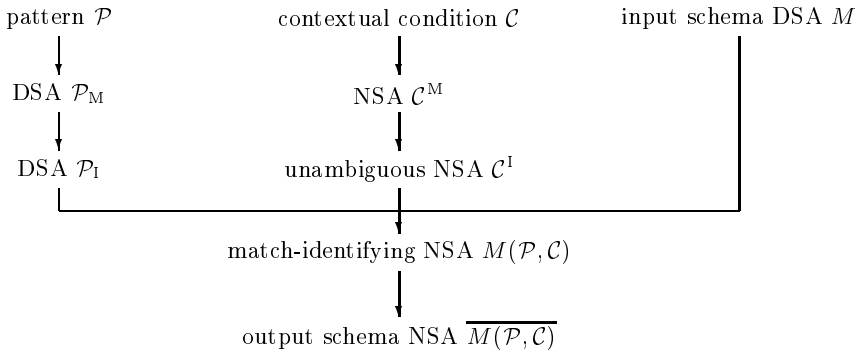

Fig. 2. Constructing the image of $\mathcal{L}$ by $m^{\mathcal{C}}_{\mathcal{P}}$ (string case)

accepts any string. We introduce an NSA $\mathcal{C}^{\mathrm{I}}$ that simulates this DSA in the reverse order (from the tail to the head). Formally,

$$\mathcal{C}^{\mathrm{I}} = <C, \Sigma, \beta', C, \{c_0\}>, \tag{8}$$

where β' is defined as

$$\beta'(c_1, x, c_2) \Leftrightarrow \beta(c_2, x) = c_1 \ . \tag{9}$$

It can be easily seen that every string s has one and only one successful computation by $\mathcal{C}^{\mathrm{I}}$, namely the mirror image of the computation of s^{r} by $\mathcal{C}^{\mathrm{M}}$. That is,

$$\mathcal{C}^{\mathrm{I}} \| s = (\mathcal{C}^{\mathrm{M}} \| s^{\mathrm{r}})^{\mathrm{r}} \ . \tag{10}$$

Furthermore,

$$s{\uparrow}i \in \mathcal{C} \Leftrightarrow (\mathcal{C}^{\mathrm{I}} \| s)[i+1] \in C_{\mathrm{f}}, \tag{11}$$

since

$$s{\uparrow}i \in \mathcal{C} \Leftrightarrow (\mathcal{C}^{\mathrm{M}} \| s^{\mathrm{r}})[n \Leftrightarrow i+1] \in C_{\mathrm{f}} \ . \tag{12}$$

Suppose that $\mathcal{L}$ is accepted by a DSA

$$M = <Q, \Sigma, \delta, q_0, Q_{\mathrm{f}}> \ . \tag{13}$$

Let us define a match-identifying NSA $M(\mathcal{P},\mathcal{C})$ by augmenting M with $\mathcal{P}_{\mathrm{I}}$ and $\mathcal{C}^{\mathrm{I}}$. First, we define a state set R, initial state set R_0, and final state set R_{f}.

$$R = Q \times P \times C \qquad R_0 = \{q_0\} \times \{p_0\} \times C \qquad R_{\mathrm{f}} = Q_{\mathrm{f}} \times P \times \{c_0\} \ . \tag{14}$$

Second, we define a transition relation η that simulates δ, α, and β'.

$$\eta((q_1, p_1, c_1), x, (q_2, p_2, c_2)) \Leftrightarrow \delta(q_1, x) = q_2,\ \alpha(p_1, x) = p_2,\ \beta'(c_1, x, c_2)\ . \quad (15)$$

Now, we can define a match-identifying NSA $M(\mathcal{P}, \mathcal{C})$ and a *marked state set* R_{m} as follows:

$$M(\mathcal{P}, \mathcal{C}) = <R, \Sigma, \eta, R_0, R_{\mathrm{f}}>, \quad (16)$$

$$R_{\mathrm{m}} = Q \times P_{\mathrm{f}} \times C_{\mathrm{f}}\ . \quad (17)$$

Obviously, $M(\mathcal{P}, \mathcal{C})$ is unambiguous and accepts $\mathcal{L}$. Furthermore, R_{m} *identifies* matches; that is,

$$(M(\mathcal{P}, \mathcal{C}) \| s)[i+1] \in R_{\mathrm{m}} \Leftrightarrow s{\downarrow}i \in \mathcal{P}, s{\uparrow}i \in \mathcal{C}\ . \quad (18)$$

Now, we are ready to construct an NSA $\overline{M(\mathcal{P}, \mathcal{C})}$ that accepts the image. We first extend the alphabet from Σ to $\Sigma \cup \overline{\Sigma}$. Second, we define a transition relation η' from $R \times (\Sigma \cup \overline{\Sigma})$ to R; intuitively speaking, we mark the labels of those transitions in η which lead to marked states. Formally, $\overline{M(\mathcal{P}, \mathcal{C})}$ is defined as follows:

$$\overline{M(\mathcal{P}, \mathcal{C})} = <R, \Sigma \cup \overline{\Sigma}, \eta', R_0, R_{\mathrm{f}}>, \quad (19)$$

where

$$\eta'(r_1, x, r_2) \Leftrightarrow \begin{cases} \eta(r_1, x, r_2), r_2 \notin R_{\mathrm{m}} & (x \in \Sigma) \\ \eta(r_1, y, r_2), r_2 \in R_{\mathrm{m}}, x = \overline{y} & (x \in \overline{\Sigma})\ . \end{cases} \quad (20)$$

It can be easily seen that

$$M(\mathcal{P}, \mathcal{C}) \,\|\, s = \overline{M(\mathcal{P}, \mathcal{C})} \,\|\, m_{\mathcal{C}}^{\mathcal{P}}(s)\ . \quad (21)$$

Therefore, $\overline{M(\mathcal{P}, \mathcal{C})}$ accepts $\{m_{\mathcal{P}}^{\mathcal{C}}(s) \mid s \in \mathcal{L}\}$. □

3 Transformations of Binary Trees

In this section we extend our observations for the binary tree case. Hereafter, we do not say "binary trees" but rather simply say "trees".

3.1 Preliminaries

A *tree* over a finite alphabet Σ is ϵ (the null tree) or $a\langle u\, v\rangle$, where a is a symbol in Σ, and u and v are trees. We assume that $a(\in \Sigma)$ and $a\langle \epsilon\, \epsilon\rangle$ are identical. The set of trees is denoted by $\Sigma^{\#}$. We assign to each $t(\in \Sigma^{\#})$, a set of *addresses* $D(t)(\subset \{1,2\}^*)$ such that

$$D(t) = \begin{cases} \emptyset & (t = \epsilon), \\ \{\epsilon\} \cup \{1d \mid d \in D(u)\} \cup \{2d \mid d \in D(v)\} & (t = a\langle u\, v\rangle)\ . \end{cases} \quad (22)$$

For example, $D(a\langle b\langle c\,\epsilon\rangle d\rangle) = \{\epsilon, 1, 11, 2\}$. An address d in $D(t)$ is a *leaf address* if $d1 \notin D(t)$ and $d2 \notin D(t)$. For example, 2 is a leaf address of $a\langle b\langle c\,\epsilon\rangle d\rangle$, but 1 is not.

The symbol at an address $d \in D(t)$ is denoted by $t[d]$. That is, if $t = a\langle u\,v\rangle$, then $t[1] = a, t[1d] = u[d]$, and $t[2d'] = v[d']$ $(d \in D(u),\ d' \in D(v))$. For example, $a\langle b\langle c\,\epsilon\rangle d\rangle[1] = b$. A *subtree* of t at an address d, denoted $t{\downarrow}d$, is

$$t[d]\langle t[d1]\langle t[d11]\langle\ldots\rangle t[d12]\langle\ldots\rangle\rangle t[d2]\langle t[d21]\langle\ldots\rangle t[d22]\langle\ldots\rangle\rangle\rangle \ . \tag{23}$$

For example, $a\langle b\langle c\,\epsilon\rangle d\rangle{\downarrow}1 = b\langle c\,\epsilon\rangle$.

A *deterministic tree automaton* (DTA) is a 5-tuple $<Q, \Sigma, \delta, q_0, Q_{\rm f}>$, where Q is a finite set of states, δ is a function from $Q \times Q \times \Sigma$ to Q, q_0 (initial state) is an element of Q, and $Q_{\rm f}$ (final state set) is a subset of Q.

For a tree t and a DTA $M = <Q, \Sigma, \delta, q_0, Q_{\rm f}>$, the *computation* of t by M, denoted $M\|t$, is a tree over Q such that

$$M\|t = \begin{cases} q_0 & (t = \epsilon), \\ \delta((M\|u)[\epsilon], (M\|v)[\epsilon], a)\langle M\|u\ M\|v\rangle & (t = a\langle u\,v\rangle)\ . \end{cases} \tag{24}$$

If $(M\|t)[\epsilon] \in Q_{\rm f}$, this computation is *successful* and t is *accepted* by M. The set of trees accepted by M is denoted by $\mathcal{L}(M)$. If a language $\mathcal{L}$ (a set of trees) is accepted by some DSA M, $\mathcal{L}$ is *tree-regular*.

A *non-deterministic tree automaton* (NTA) is a 5-tuple $<Q, \Sigma, \delta, Q_0, Q_{\rm f}>$, where Q and $Q_{\rm f}$ are as above, δ is a relation from $Q \times Q \times \Sigma$ to Q, and Q_0 (initial state set) is a subset of Q.

For a tree t and an NTA $M = <Q, \Sigma, \delta, Q_0, Q_{\rm f}>$, a *computation* of t by M is a tree $\hat{t}$ over Q such that

$$\hat{t} \in Q_0 \qquad (t = \epsilon), \tag{25}$$

$$\hat{t} = \delta(\hat{u}[\epsilon], \hat{v}[\epsilon], a)\langle\hat{u}\,\hat{v}\rangle \qquad (t = a\langle u\,v\rangle)\ . \tag{26}$$

where $\hat{u}$ and $\hat{v}$ are computations of u and v, respectively. If $\hat{t}[\epsilon] \in Q_{\rm f}$, this computation is *successful*. If there is at least one successful computation, t is *accepted* by M. It is well known that a language is tree-regular if and only if it is accepted by some NTA. If every tree has at most one successful computation, M is *unambiguous*. Furthermore, if an unambiguous NTA M accepts tree t, we denote the successful computation of t by $M\|t$.

The rest is borrowed from Nivat and Podelski [11,12]. A *pointed tree* over a finite alphabet Σ is a tree t over $\Sigma \cup \{\varsigma\}$ $(\varsigma \notin \Sigma)$ such that ς occurs in t once and only once and the only occurrence is as a leaf. The set of pointed trees over Σ is denoted by $\Sigma^{(\#)}$. The result of replacing ς with another pointed tree t' is denoted by $t' \circ t$. For example, $b\langle\varsigma\,\epsilon\rangle \circ a\langle\varsigma\,\epsilon\rangle = a\langle b\langle\varsigma\,\epsilon\rangle\epsilon\rangle$. Obviously, $(\Sigma^{(\#)}; \circ, \varsigma)$ is a monoid.

An *envelope* of a tree t at an address d, denoted $t{\uparrow}d$, is a pointed tree obtained from t by replacing $t{\downarrow}d$ with ς. For example, $a\langle b\langle c\,\epsilon\rangle d\rangle{\uparrow}1 = a\langle\varsigma\,d\rangle$.

A *pointed-base tree* is a pointed tree of the form $a\langle\varsigma\,t\rangle$ or $a\langle t\,\varsigma\rangle$. Any pointed tree t uniquely decomposes into a sequence of pointed-base trees $t_1, t_2, \ldots, t_k$

such that $t = t_1 \circ t_2 \circ \cdots \circ t_k$ $(k \geq 0)$. For example, $a\langle b\langle \varsigma\epsilon\rangle\epsilon\rangle$ uniquely decomposes into $b\langle\varsigma\epsilon\rangle, a\langle\varsigma\epsilon\rangle$.

A *pointed-base tree representation* is either a triplet $<a, \varsigma, \mathcal{S}>$ or a triplet $<a, \mathcal{S}, \varsigma>$, where $a \in \Sigma$, and $\mathcal{S}$ is a tree-regular language over Σ. The represented language is defined as below:

$$\mathcal{L}(<a, \varsigma, \mathcal{S}>) = \{a\langle\varsigma\, t\rangle \mid t \in \mathcal{S}\}, \tag{27}$$

$$\mathcal{L}(<a, \mathcal{S}, \varsigma>) = \{a\langle t\, \varsigma\rangle \mid t \in \mathcal{S}\}\ . \tag{28}$$

For example, $\mathcal{L}(<a, \varsigma, \{b\langle c\,\epsilon\rangle\}>) = \{a\langle\varsigma b\langle c\,\epsilon\rangle\rangle\}$.

A *pointed tree representation* is a pair $<\psi, \mathcal{E}>$, where ψ is a bijection from a finite alphabet to a finite set of pointed-base tree representations, and $\mathcal{E}$ is a string-regular language over the domain of ψ. The represented language is defined as below:

$$\mathcal{L}(<\psi, \mathcal{E}>) = \{t_1 \circ t_2 \circ \cdots \circ t_k \mid t_i \in \mathcal{L}(\psi(e_i)) \text{ for some } e_1 e_2 \ldots e_k \in \mathcal{E}\}\ . \tag{29}$$

For example, if $dom(\psi) = \{\omega\}$, $\psi(\omega) = <a, \varsigma, \{b\langle c\,\epsilon\rangle\}>$, and $\mathcal{E} = \{\omega\omega\}$, then $\mathcal{L}(<\psi, \mathcal{E}>) = \{a\langle\varsigma b\langle c\,\epsilon\rangle\rangle \circ a\langle\varsigma b\langle c\,\epsilon\rangle\rangle\} = \{a\langle a\langle\varsigma b\langle c\,\epsilon\rangle\rangle b\langle c\,\epsilon\rangle\rangle\}$.

3.2 Transformation Rules

We first define marking functions and linear tree homomorphisms, and then define transformation rules.

Marking Functions. A *pattern* $\mathcal{P}$ is a tree-regular language. Given a tree t, the subtree of t at an address d *matches* $\mathcal{P}$ if $t{\downarrow}d \in \mathcal{P}$. A *contextual condition* $\mathcal{C}$ is a language represented by a pointed tree representation $<\psi, \mathcal{E}>$. The envelope of t at d *satisfies* $\mathcal{C}$ if $t{\uparrow}d \in \mathcal{C}$.

For each symbol $x \in \Sigma$, we introduce a marked symbol $\overline{x}$. A marked alphabet $\overline{\Sigma}$ is defined as $\{\overline{x} \mid x \in \Sigma\}$.

A *marking function* $m^{\mathcal{P}}_{\mathcal{C}}$ is a mapping from $\Sigma^{\#}$ to $(\Sigma \cup \overline{\Sigma})^{\#}$. Intuitively, $m^{\mathcal{P}}_{\mathcal{C}}$ replaces $t[d]$ with $\overline{t[d]}$ for every address d such that $t{\downarrow}d \in \mathcal{P}$ and $t{\uparrow}d \in \mathcal{C}$. Formally, $m^{\mathcal{P}}_{\mathcal{C}}$ is defined as follows:

$$\begin{aligned} m^{\mathcal{P}}_{\mathcal{C}}(t[\epsilon]\langle t[1]\langle t[11]\langle\ldots\rangle\, t[12]\langle\ldots\rangle\rangle\, t[2]\langle t[21]\langle\ldots\rangle\, t[22]\langle\ldots\rangle\rangle\rangle) \\ = (t[\epsilon])'\langle(t[1])'\langle(t[11])'\langle\ldots\rangle\,(t[12])'\langle\ldots\rangle\rangle \\ (t[2])'\langle(t[21])'\langle\ldots\rangle\,(t[22])'\langle\ldots\rangle\rangle\rangle, \end{aligned} \tag{30}$$

where

$$(t[d])' = \begin{cases} \overline{t[d]} & (t{\downarrow}d \in \mathcal{P}, t{\uparrow}d \in \mathcal{C}) \\ t[d] & (otherwise)\ . \end{cases} \tag{31}$$

Linear Tree Homomorphisms. A *replacement tree* t' over Σ is a tree over $\Sigma \cup \{z_1, z_2\}$ ($z_1, z_2 \notin \Sigma$) such that 1) z_1 and z_2 occur in t' only as leaf nodes, 2) z_1 occurs at most once, and 3) z_2 occurs at most once. For example, $a\langle b\langle z_1\, \epsilon\rangle z_2\rangle$ and $a\langle b\langle z_1\, \epsilon\rangle \epsilon\rangle$ are replacement trees over $\{a, b\}$, but $a\langle z_1\langle b\, \epsilon\rangle z_2\rangle$ is not. The result of replacing z_1 and z_2 with trees t_1 and t_2 respectively, is denoted $t'(z_1 \leftarrow t_1, z_2 \leftarrow t_2)$.

Let $h_{\overline{\Sigma}}$ be a function from $\overline{\Sigma}$ to the set of replacement trees over Σ. The *linear tree homomorphism* h determined by $h_{\overline{\Sigma}}$ is the function from $(\Sigma \cup \overline{\Sigma})^{\#}$ to $\Sigma^{\#}$ defined as below:

$$h(t) = \begin{cases} \epsilon & (t = \epsilon), \\ t[\epsilon]\langle h(t{\downarrow}1)\, h(t{\downarrow}2)\rangle & (t[\epsilon] \in \Sigma), \\ h_{\overline{\Sigma}}(t[\epsilon])(z_1 \leftarrow h(t{\downarrow}1), z_2 \leftarrow h(t{\downarrow}2)) & (t[\epsilon] \in \overline{\Sigma})\ . \end{cases} \tag{32}$$

For example, if $\Sigma = \{a, b\}$, $h_{\overline{\Sigma}}(\overline{a}) = a\langle b\langle z_1\, \epsilon\rangle z_2\rangle$, and $h_{\overline{\Sigma}}(\overline{b}) = b\langle z_1\, z_2\rangle$, then $h(\overline{a}\langle \overline{b}\, a\rangle) = a\langle b\langle b\, \epsilon\rangle a\rangle$.

Transformation Rules. A *transformation rule* is a triplet $<\mathcal{P}, \mathcal{C}, h>$, where $\mathcal{P}$ is a pattern, $\mathcal{C}$ is a contextual condition, and h is a linear tree homomorphism. The result of applying this rule to a tree t is defined as $h(m^{\mathcal{P}}_{\mathcal{C}}(t))$.

3.3 Applying Transformation Rules to Trees

To implement transformation rules, we need an algorithm for pattern matching and contextual condition testing. As in the string case, the rest is straightforward.

It is simple to find all d's such that $t{\downarrow}d \in \mathcal{P}$. Let

$$\mathcal{P}_{\mathrm{M}} = <P, \Sigma, \alpha, p_0, P_{\mathrm{f}}> \tag{33}$$

be a DTA that accepts $\mathcal{P}$. Then, by executing $\mathcal{P}_{\mathrm{M}}$ for t, we obtain a computation $\mathcal{P}_{\mathrm{M}}\|s$. $s{\downarrow}d \in \mathcal{P}$ if and only if $(\mathcal{P}_{\mathrm{M}}\|s)[d] \in Q_{\mathrm{f}}$.

Unlike the string case, it is more complicated (but still efficient) to find all d's such that $t{\uparrow}d \in \mathcal{P}$. In preparation we first introduce some definitions and then introduce a lemma that provides an algorithm for contextual condition testing.

A *pseudo-DTA* is a 4-tuple $<S, \Sigma, \lambda, s_0>$, where S is a finite set of states, λ is a function from $S \times S \times \Sigma$ to S, and s_0 (initial state) is an element of S. The only difference from DTA's is that a pseudo-DTA does not have a final set state. The computation of a pseudo-DTA is defined similarly to that of a DTA.

Given a pseudo-DTA $N = <S, \Sigma, \lambda, s_0>$ and a function θ from $\Sigma \times S \times \{1, 2\}$ to some finite alphabet Δ, we define a function $\theta \oplus N$ from $\Sigma^{(\#)}$ to Δ as below:

$$(\theta \oplus N)(t) = t_1{}'t_2{}' \ldots t_k{}', \tag{34}$$

where $t_1, t_2, \ldots, t_k$ is the decomposition of t and

$$t_i{}' = \begin{cases} \theta(a, (N\|u)[\epsilon], 1) & (t_i = a\langle \varsigma\, u\rangle), \\ \theta(a, (N\|u)[\epsilon], 2) & (t_i = a\langle u\, \varsigma\rangle)\ . \end{cases} \tag{35}$$

Lemma 4. *There exist a pseudo-DTA* $N = <S, \Sigma, \lambda, s_0>$*, a function* θ *from* $\Sigma \times S \times \{1, 2\}$ *to some finite alphabet* Δ*, and a string-regular language* $\mathcal{F}$ *over* Δ *such that* $t \in \mathcal{C}$ *if and only if* $(\theta \oplus N)(t) \in \mathcal{F}$*.*

Proof. We effectively construct $N, \theta, \mathcal{F}$, and leave the rest of the proof to the reader. The key idea is to make a pseudo-DTA that "accepts" every tree-regular language that appears as a constituent of some pointed-base tree representation in $range(\psi)$.

By enumerating those constituent tree-regular languages, we obtain a sequence $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n$; that is,

$$\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\} = \{S \mid <a, \varsigma, S> \in range(\psi) \text{ or } <a, S, \varsigma> \in range(\psi)\}. \quad (36)$$

For a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ in $\{\Leftrightarrow\!\!1, 1\}^n$, we introduce $\mathcal{B}(\mathbf{x})$ (a subset of $\Sigma^{\#}$) as below:

$$\mathcal{B}(\mathbf{x}) = \mathcal{Y}_1 \cap \mathcal{Y}_2 \cap \cdots \cap \mathcal{Y}_n, \quad (37)$$

where

$$\mathcal{Y}_i = \begin{cases} \mathcal{S}_i & (x_i = 1), \\ \Sigma^{\#} \Leftrightarrow \mathcal{S}_i & (x_i = \Leftrightarrow\!\!1) \ . \end{cases} \quad (38)$$

Obviously, if $\mathbf{x} \neq \mathbf{y}$, then $\mathcal{B}(\mathbf{x})$ and $\mathcal{B}(\mathbf{y})$ are disjoint. Furthermore,

$$\bigcup_{x \in \{-1,1\}^m} \mathcal{B}(\mathbf{x}) = \Sigma^{\#} \ . \quad (39)$$

Given a DTA

$$M_i = <S_i, \Sigma, \lambda_i, s_i^0, S_i^{\mathrm{f}}> \quad (40)$$

that accepts $\mathcal{S}_i$ $(1 \leq i \leq n)$, let us define a pseudo-DTA N. First, we define a state set S and an initial state s_0.

$$S = S_1 \times S_2 \times \cdots \times S_n \qquad s_0 = (s_1^0, s_2^0, \ldots, s_n^0) \quad (41)$$

Second, we define a transition function λ.

$$\lambda((s_1, s_2, \ldots, s_n), (s_1', s_2', \ldots, s_n'), x) = (\lambda_1(s_1, s_1', x), \lambda_2(s_2, s_2', x), \ldots, \lambda_n(s_n, s_n', x)) \ . \quad (42)$$

Now, we can define N as below:

$$N = <S, \Sigma, \lambda, s_0> \ . \quad (43)$$

For each vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ in $\{\Leftrightarrow\!\!1, 1\}^n$, we introduce $S(\mathbf{x})$ (a subset of S^n) as below:

$$S(\mathbf{x}) = Z_1 \cap Z_2 \cap \cdots \cap Z_n, \quad (44)$$

where

$$Z_i = \begin{cases} S_1 \times \cdots \times S_{i-1} \times S_i^{\mathrm{f}} \times S_{i+1} \times \cdots \times S_n & (x_i = 1), \\ S_1 \times \cdots \times S_{i-1} \times (S_i \Leftrightarrow S_i^{\mathrm{f}}) \times S_{i+1} \times \cdots \times S_n & (x_i = \Leftrightarrow 1) \ . \end{cases} \tag{45}$$

Obviously, if $\mathbf{x} \neq \mathbf{y}$, then $S(\mathbf{x})$ and $S(\mathbf{y})$ are disjoint. Furthermore, the DTA created by adding $S(\mathbf{x})$ to N as a final state set accepts $\mathcal{B}(\mathbf{x})$; that is,

$$\mathcal{L}(<S, \Sigma, \lambda, s_0, S(\mathbf{x})>) = \mathcal{B}(\mathbf{x}) \ . \tag{46}$$

Next, we construct a function θ from $\Sigma \times S \times \{1, 2\}$ to Δ, where Δ is a finite set as below:

$$\Delta = \{<a, \varsigma, \mathcal{B}(\mathbf{x})>, <a, \mathcal{B}(\mathbf{x}), \varsigma> \mid a \in \Sigma, \mathbf{x} \in \{\Leftrightarrow 1, 1\}^m\} \ . \tag{47}$$

Observe that for any $s \in S$, there exists one and only one $\mathbf{x}$ such that $s \in S(\mathbf{x})$. So, the following is a sound definition.

$$\theta(a, s, 1) = <a, \varsigma, \mathcal{B}(\mathbf{x})>, \text{ where } \mathbf{x} \in \{\Leftrightarrow 1, 1\}^m \text{ such that } s \in S(\mathbf{x}), \tag{48}$$

$$\theta(a, s, 2) = <a, \mathcal{B}(\mathbf{x}), \varsigma>, \text{ where } \mathbf{x} \in \{\Leftrightarrow 1, 1\}^m \text{ such that } s \in S(\mathbf{x}) \ . \tag{49}$$

Finally, we define a string-regular language $\mathcal{F}$ over Δ. Let ξ be a substitution function from the domain of ψ to the powerset of Δ as below:

$$\xi(\omega) = \begin{cases} \{<a, \varsigma, \mathcal{B}(\mathbf{x})> \mid \mathbf{x} \in \{\Leftrightarrow 1, 1\}^m, x_i = 1\} & (\psi(\omega) = <a, \varsigma, \mathcal{S}_i>) \\ \{<a, \mathcal{B}(\mathbf{x}), \varsigma> \mid \mathbf{x} \in \{\Leftrightarrow 1, 1\}^m, x_i = 1\} & (\psi(\omega) = <a, \mathcal{S}_i, \varsigma>) \ . \end{cases} \tag{50}$$

Language $\mathcal{F}$ is defined as the image of $\mathcal{E}$ by ξ; that is

$$\mathcal{F} = \{f_1 f_2 \ldots f_i \mid e_1 e_2 \ldots e_i \in \mathcal{E}, f_j \in \xi(e_j), 1 \leq j \leq i\} \ . \tag{51}$$

□

Lemma 4 yields the following algorithm for contextual condition testing.

Initialization: We first construct $N, \theta, \mathcal{F}$ of Lemma 4 as shown in the proof above, and then construct a DSA $\mathcal{F}^{\mathrm{M}} = <F, \Delta, \mu, f_0, F_{\mathrm{f}}>$ that accepts the mirror image of $\mathcal{F}$.

Evaluation of N: By evaluating N for a given tree t, we construct a computation $N \| t$.

Evaluation of θ: By evaluating θ at each address d, we construct a tree u over Δ ($D(u) = D(t)$). If $d = d'1$, then $u[d]$ is $\theta(t[d], (\mathcal{P}_{\mathrm{M}} \| t)[d2], 1)$; if $d = d'2$, then $u[d]$ is $\theta(t[d], (N \| t)[d1], 2)$. The value at ϵ can be anything, as it is not important.

Evaluation of $\mathcal{F}^{\mathrm{M}}$: By evaluating $\mathcal{F}^{\mathrm{M}}$ from the root to the leaf nodes, we construct another tree v over F ($D(v) = D(t)$) such that (1) if $d = \epsilon$, then $v[d]$ is f_0, and (2) if $d = d'1$ or $d'2$, then $v[d]$ is $\mu(v[d'], u[d])$. Then, $t{\uparrow}d \in \mathcal{C}$ if and only if $v[d] \in F_{\mathrm{f}}$.

The initialization does not depend on t. The other steps only require time linear to the size of t, as we only have to evaluate λ, θ, and μ for each node. Thus, this algorithm is linear-time.

3.4 Schema Transformation

Theorem 5. *The image of a tree-regular language $\mathcal{L}$ over Σ by a transformation rule $<\mathcal{P},\mathcal{C},h>$ is tree-regular over Σ.*

This theorem directly follows from Lemmas 6 and 7.

Lemma 6. *The image of $\mathcal{L}$ by $m^{\mathcal{C}}_{\mathcal{P}}$ is tree-regular over $\Sigma \cup \overline{\Sigma}$.*

Lemma 7. *The image of a tree-regular language $\mathcal{L}'$ over $\Sigma \cup \overline{\Sigma}$ by h is tree-regular over Σ.*

Again, we will not prove Lemma 7 as it is a special case of Theorem 4.16 (linear tree homomorphism) in Gécseg and Steinby [5].

Proof (Lemma 6). We effectively construct an NSA that accepts the image as depicted by Figure 3. As in the proof of Theorem 1, the key idea is the construction of a *match-identifying NTA*.

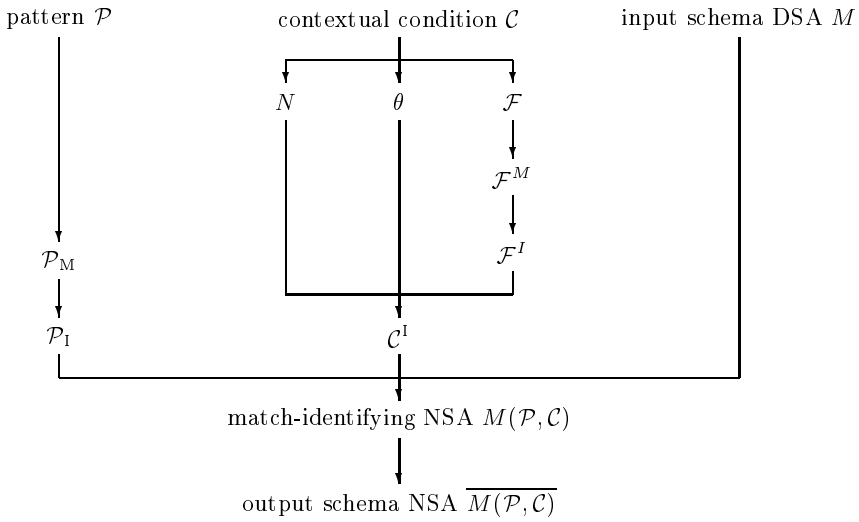

Fig. 3. Constructing the image of $\mathcal{L}$ by $m^{\mathcal{C}}_{\mathcal{P}}$ (tree case)

Recall that DTA $\mathcal{P}_{\mathrm{M}} = <P, \Sigma, \alpha, p_0, P_{\mathrm{f}}>$ accepts $\mathcal{P}$. By allowing any state as a final state, we obtain a DTA

$$\mathcal{P}_{\mathrm{I}} = <P, \Sigma, \alpha, p_0, P> \ . \tag{52}$$

Obviously, $\mathcal{P}_{\mathrm{I}}$ accepts any tree. Furthermore, for any tree t,

$$(\mathcal{P}_{\mathrm{I}} \| t)[d] \in Q_{\mathrm{f}} \Leftrightarrow s{\downarrow}d \in \mathcal{P} \ . \tag{53}$$

We suppose an unambiguous NTA

$$\mathcal{C}^{\mathrm{I}} = <C, \Sigma, \beta, C_0, C_{\mathrm{f}}> \tag{54}$$

and a subset C_{m} of C such that (1) $\mathcal{C}^{\mathrm{I}}$ accepts any tree t, and (2) $(\mathcal{C}^{\mathrm{I}} \| t)[d] \in C_{\mathrm{m}}$ if and only if $t \uparrow d \in \mathcal{C}$. We later construct $\mathcal{C}^{\mathrm{I}}$ and C_{m} from $N = <S, \Sigma, \lambda, s_0>, \theta$, and $\mathcal{F}$ of Lemma 4.

Suppose that $\mathcal{L}$ is accepted by a DTA

$$M = <Q, \Sigma, \delta, q_0, Q_{\mathrm{f}}> \ . \tag{55}$$

Let us define a match-identifying NTA $M(\mathcal{P}, \mathcal{C})$ by augmenting M with $\mathcal{P}_{\mathrm{I}}$ and $\mathcal{C}^{\mathrm{I}}$. First, we define a state set R, initial state set R_0, and final state set R_{f}.

$$R = Q \times P \times C \qquad R_0 = \{q_0\} \times \{p_0\} \times C_0 \qquad R_{\mathrm{f}} = Q_{\mathrm{f}} \times P \times C_{\mathrm{f}} \ . \tag{56}$$

Second, we define a transition relation η that simulates δ, α, and β'.

$$\eta((q_1, p_1, c_1), (q_2, p_2, c_2), x, (q_3, p_3, c_3)) \ \Leftrightarrow \ \delta(q_1, q_2, x) = q_3, \alpha(p_1, p_2, x) = p_3, \beta(c_1, c_2, x, c_3) \ . \tag{57}$$

Now, we can define a match-identifying NTA $M(\mathcal{P}, \mathcal{C})$ and a *marked state set* R_{m} as follows:

$$M(\mathcal{P}, \mathcal{C}) = <R, \Sigma, \eta, R_0, R_{\mathrm{f}}>, \tag{58}$$

$$R_{\mathrm{m}} = Q \times P_{\mathrm{f}} \times C_{\mathrm{m}} \ . \tag{59}$$

Obviously, $M(\mathcal{P}, \mathcal{C})$ is unambiguous and accepts $\mathcal{L}$. Furthermore, R_{m} *identifies* matches; that is,

$$(M(\mathcal{P}, \mathcal{C}) \| t)[d] \in R_{\mathrm{m}} \Leftrightarrow t \downarrow d \in \mathcal{P}, t \uparrow d \in \mathcal{C} \ . \tag{60}$$

Now, we are ready to construct an NTA $\overline{M(\mathcal{P}, \mathcal{C})}$ that accepts the image. We first extend the alphabet from Σ to $\Sigma \cup \overline{\Sigma}$. Second, we define a transition relation η' from $R \times R \times (\Sigma \cup \overline{\Sigma})$ to R; intuitively speaking, we mark the labels of those transitions in η which lead to marked states. Formally, $\overline{M(\mathcal{P}, \mathcal{C})}$ is defined as follows:

$$\overline{M(\mathcal{P}, \mathcal{C})} = <R, \Sigma \cup \overline{\Sigma}, \eta', R_0, R_{\mathrm{f}}>, \tag{61}$$

where

$$\eta'(r_1, r_2, x, r_3) \Leftrightarrow \begin{cases} \eta(r_1, r_2, x, r_3), r_3 \notin R_{\mathrm{m}} & (x \in \Sigma), \\ \eta(r_1, r_2, y, r_3), r_3 \in R_{\mathrm{m}}, x = \overline{y} & (x \in \overline{\Sigma}) \ . \end{cases} \tag{62}$$

It can be easily seen that

$$M(\mathcal{P}, \mathcal{C}) \| t = \overline{M(\mathcal{P}, \mathcal{C})} \| m_{\mathcal{C}}^{\mathcal{P}}(t) \ . \tag{63}$$

Therefore, $\overline{M(\mathcal{P},\mathcal{C})}$ accepts $\{m^{\mathcal{C}}_{\mathcal{P}}(t) \mid t \in \mathcal{L}\}$.

It remains to show the construction of an NTA

$$\mathcal{C}^{\mathrm{I}} = <C, \Sigma, \beta, C_0, C_{\mathrm{f}}> \tag{64}$$

and a subset C_{m} of C from $N = <S, \Sigma, \lambda, s_0>, \theta$, and $\mathcal{F}$ of Lemma 4. The first idea is, as in Section 2, to construct an unambiguous NSA $\mathcal{F}^{\mathrm{I}}$ from $\mathcal{F}^{\mathrm{M}} = <F, \Delta, \mu, f_0, F_{\mathrm{f}}>$. Formally,

$$\mathcal{F}^{\mathrm{I}} = <F, \Delta, \mu', F, \{f_0\}>, \tag{65}$$

where μ' is defined as

$$\mu'(f_1, x, f_2) \Leftrightarrow \mu(f_2, x) = f_1 \ . \tag{66}$$

The second idea is to simulate the execution of N and $\mathcal{F}^{\mathrm{I}}$ from every leaf d to the root. A state of $\mathcal{C}^{\mathrm{I}}$ is thus a pair of $s \in S$ and $f \in F$; that is,

$$C = S \times F \ . \tag{67}$$

The first constituent s simulates N. The second constituent f simulates $\mathcal{F}^{\mathrm{I}}$ for every path. Notice that we do not need more than one state of $\mathcal{F}^{\mathrm{I}}$ since all paths should merge. If not, we make this NTA fail.

An initial state is a pair of the initial state of N and an initial state of $\mathcal{F}^{\mathrm{I}}$; that is,

$$C_0 = \{s_0\} \times F \ . \tag{68}$$

A final state is a pair of any state of N and the final state of $\mathcal{F}^{\mathrm{I}}$; that is,

$$C_{\mathrm{f}} = S \times \{f_0\} \ . \tag{69}$$

A marked state is a pair of any state of N and a final state f of $\mathcal{F}^{\mathrm{M}}$; that is,

$$C_{\mathrm{m}} = S \times F_{\mathrm{f}} \ . \tag{70}$$

Finally, we define transition relation β as

$$\beta((s_1, f_1), (s_2, f_2), x, (s_3, f_3)) \Leftrightarrow s_3 = \lambda(s_1, s_2, x), \mu'(\theta(x, s_2, 1), f_1, f_3), \mu'(\theta(x, s_1, 2), f_2, f_3) \ . \tag{71}$$

The proof that $\mathcal{C}^{\mathrm{I}}$ satisfies our assumptions is left to the reader. □

References

1. Arnon, D.: Scrimshaw: A language for document queries and transformations. Electronic Publishing – Origination, Dissemination, and Design **6** (1993) 385–396
2. Baeza-Yates, R., Navarro, G.: Integrating contents and structure in text retrieval. SIGMOD Record, 25(1):(1996) 67–79
3. Christophidese, V., Abiteboul, S., Cluet, S., Scoll, M.: From structured documents to novel query facilities. In SIGMOD 1994, (1994) 313–324
4. Colby, L., Saxton, L., Van Gucht, D.: Concepts for modeling and querying list-structured data. Information Processing and Management. 30(5): (1994) 687–709
5. Gécseg, F., and Steinby, M.: Tree automata. Akadémiai Kiaddá, Budapest, Hungary, 1984.
6. Gonnet, G., and Tompa, F.: Mind your grammar – a new approach to modelling text. In Proceedings of VLDB'87, (1987) 339–346
7. Gyssens, M., Paredaens, J., and Van Gucht, D.: A grammar-based approach towards unifying hierarchical data models. SIAM Journal on Computing, 23, (1994) 1093–1097
8. Hoffmann, C., and O'Donnell, M.: Pattern matching in trees. Journal of the ACM. 29(1):(1982) 68–95
9. International Organization for Standardization. Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML), 1986.
10. International Organization for Standardization. Information Technology – Text and Office Systems – Document Style Semantics and Specification Language (DSSSL), 1994.
11. Nivat, M. and Podelski, A.: Another variation on the common subexpression problem. Theoretical Computer Science, **114**, (1993) 11-11
12. Podelski, A.: A monoid approach to tree automata. In Nivat and Podelski, editors, Tree Automata and Languages, Studies in Computer Science and Artificial Intelligence **10**. North-Holland, (1992) 11-11
13. Wilhelm, R.: Tree transformations, functional languages, and attribute grammars. In Pierre Deransart and Martin Jourdan, editors, Attribute grammars and their applications, Springer-Verlag **461**, (1990) 116–129