Approach to Document Structure Transformations

E. Kuikka, P. Leinonen, M. Penttonen

Report A/1999/7

ISBN 951-781-615-4
ISSN 0787-6416

UNIVERSITY OF KUOPIO

Department of Computer Science
and Applied Mathematics

P.O.Box 1627, FIN-70211 Kuopio, FINLAND

An Approach to Document Structure Transformations

Eila Kuikka
Department of Computer Science and Applied Mathematics
University of Kuopio
Eila.Kuikka@cs.uku.fi

Paula Leinonen
Department of Computer Science
University of Joensuu
pleinone@cs.joensuu.fi

Martti Penttonen
Department of Computer Science and Applied Mathematics
University of Kuopio
Martti.Penttonen@cs.uku.fi

Abstract

We characterize a class of structure transformations, called dense, hierarchic and local transformations, that can be efficiently implemented by a two-phase, semi-automatic procedure. In the first phase of our method, corresponding substructures are searched by an interactive procedure. In the second phase, the replacement of substructures is automatized by generating a tree transducer implementing it.

1 Introduction

Nowadays documents are produced in electronic form and often published in several formats using several applications. For example, in addition to a paper version, an HTML version can be published on the Internet. Because of several storage and formatting schemes in applications, there is a need to transform the format of the documents. If the logical structure of the document is marked, it may be possible to define an automatic transformation of the format or the structure of the document.

There are various transformation systems implemented in an ad hoc manner for transforming the format between two known presentational markups. For example, `latex2html` [Dra98] converts LaTeX markup [Lam86] to HTML. In many commercial systems there are tools for converting documents to HTML or SGML. However, often the user has to make the whole conversion or complete some automatic processing manually. If the structure of the document is defined using a grammar, tree transformation methods can be used for the structure transformations. The extended Backus-Naur Form (BNF) of the context-free grammar is suitable and widely used for defining the structure of the documents. The Standard Generalized Markup Language (SGML) is an ISO standard [ISO86] that provides the notation for defining the textual markup systems. This kind of markup system can be specified by an extended context-free grammar [KW97].

Some study about general structure transformations of documents has been done, but not many applications exist. Furuta and Stotts specify the problem in [FS88]. Akpotsui and Quint use a comparator tool to compare two grammars for creating conversion rules, which are used by a converter. No algorithm for the comparator tool has been provided. For a structure transformation between two closely related grammars, there are some systems based on a syntax-directed translation, like SYNDOC [KPV94, KP95, Kui96], HST [KLMN90], Integrated Chameleon Architecture (ICA) [MOB94] or the system of Chiba and Kyojima [CK95]. However, the syntax-directed translation is quite restricted. Attribute grammars [FW93] and tree transformation grammars [KPPM84] are used in structure transformations for increasing a power. A tree transformation grammar is an extension of a syntax-directed translation schema (SDTS). It describes an association between the source and the target grammars. In addition to defining an association between the nonterminals, the user defines an association between groups of productions. The tree transformation grammar is not restricted to a single level in a parse tree like the SDTS is.

SYNDOC is a research prototype for a syntax-directed document processing system, in which the document is considered as a tree structure in all phases of processing. The user defines the structure of a document with a skeleton grammar and the document instance is a parse tree of the grammar. Formatted documents are created from a parse tree using syntax-directed translation [AU72]. SYNDOC provides a simple facility for transforming documents — documents can be transformed to another format if the structure does not change or changes only in the limits of the SDTS. Omissions and additions and changes of order are allowed, but deeper changes are not possible.

Our aim in this work is to develop a semi-automatic procedure for defining an automatic transformation from a structure defined by one grammar to the structure defined by another grammar. This is an incremental procedure, where the transformation is defined by example documents in dialogue with the system. The completeness of the procedure is guaranteed by characterizing a sufficient set of examples, called characteristic trees. In Section 2, we define some concepts related with the syntax-directed approach to document processing. In Section 3, we develop a procedure for the transformation of a parse tree to another. Section 4 describes how transformation rules are expressed as rules of a tree transducer. Finally, Section 5 summarizes the work and discusses future developments.

2 Structured documents

In documents there is a content but virtually always also some other information about their formatting or structure. The formatting instructions for a document are called a *procedural markup*. A *declarative markup* describes the logical, hierarchical structure of the document. Documents with a declarative markup can be used directly in textual databases, or their format or structure may be possible to transform automatically. The structure definition can also help the writer through the writing process, if the text-processing system supports structures. SGML is a widely used international standard for describing the structure of a document. DSSSL (Document Style Semantics and Specification Language) [Cla96] is a standard for describing the formatting of the structured text. This chapter summarizes the principles of grammar-based document processing by considering a syntax-directed approach. First we will provide some definitions.

A *context-free grammar* [AU72] is a quadruple $G = (N, \Sigma, P, S)$, where N is a finite set of nonterminals, Σ is a finite set of terminals, P is a set of productions of the form $A \rightarrow \alpha$, where $A \in N$ and $\alpha \in (N \cup \Sigma)^*$ and $S \in N$ is the start symbol.

The rules of the grammar are used to derive documents. Such a derivation is expressed with a *derivation tree* as follows. The initial symbol S of the grammar is represented by a node labelled with S, that will be the *root* of the tree. In the beginning, when no rules have been applied, S is also a *leaf* of the derivation tree. Later, when a leaf of the tree is labelled with a nonterminal A, and there is a rule $A \rightarrow x_1 x_2 \dots x_k$, where $x_i \in N \cup \Sigma$ $(i = 1 \dots k)$, the derivation tree is extended with new leaves labelled with $x_1, x_2, \dots, x_k$ that a connected to A, which becomes an internal node. Sometimes the concept of a *parse tree* is used to emphasize the opposite of the derivation, a parsing according to a grammar.

With the text-processing system the user should be able to create, update, save and retrieve the documents and format them for delivery. Most present-day text-processing systems are based on the procedural approach. The user writes the contents of a document mixed with formatting instructions. Usually the elements of the documents, such as chapters, sections and figures are explicitly numbered, spaces between elements are fixed, and character types used for emphasizing are chosen. After adding or deleting some content element of the document, the user has to check whether the change has affected the numbering, references or other layout information and update it, if needed. However, for the writer of the document the idea of the text content should be most important, not the layout of the document.

In the syntax-directed approach to text processing a grammar describes the structure of the document, and a document instance is a parse tree of the grammar. Document processing based on a declarative definition of the structure can be considered to be an analogy of the compiling of programs. The input text is a "program" written in one language and the output text a "program" written in another language. The translation from the input text to the output text is divided into a syntactic and a semantic translation. The syntactic translation isolates the structured content from the input according to a source grammar. Its result is a parse tree which represents the logical elements and the syntactic relationships between them. The semantic translation adds formatting commands between the leaves of the parse tree according to the target grammar and generates the output text from the parse tree. If the logical structures of the source and the target grammars differ, text processing needs an extra phase. In addition to the two phases described above, the intermediate representation, the parse tree is transformed to follow a new structure.

The syntax-directed translation is widely used in structured document processing systems. The SYNDOC system [KPV94] defines the internal structure of the document with a *skeleton grammar*. Next example is a skeleton grammar of an article type of document.

```
article -> authors [date] title content.
authors -> (author)+.
author -> text.
date -> text.
title -> text.
content -> abstract (section)+.
abstract -> text.
section -> heading (paragraph)+.
heading -> text.
paragraph -> textpara.
paragraph -> itemizelist.
itemizelist -> (itemize)+.
textpara -> text.
itemize -> text.
```

Often it is useful to have special kinds of nonterminals called *content nonterminals*. These nonterminals do not have productions in the grammar, but they refer to the contents of the document. In the grammar above, nonterminal `text` can contain any sequence of characters. Usually the content nonterminal is the only nonterminal of the right-hand side of the production, and hence does not have siblings in the parse tree. The other nonterminals, having applicable rules, are called *structure nonterminals*.

In the current version of the SYNDOC system, the source grammar is identical to the skeleton grammar. It could, for example, use terminals for providing information to help the user in the writing process. The target grammar is created from the skeleton grammar by adding terminals to it. In the input phase of text processing there is an editor that uses the source grammar to prompt the structure elements to the user. The internal representation of the document is the parse tree of the skeleton grammar. The output of the document is generated from the parse tree for the skeleton grammar to an external representation for the target grammar using simple syntax-directed translation.

3 Associations in structures

Tree transformation methods are suitable for structure transformations of documents that have an extended context-free grammar as a structure definition. The tree transformation should be determined using the grammars for the source and for the target documents and the method should be automatic, or at least semi-automatic.

In this section we present a semiautomatic procedure for transforming a structure of a document from one grammar to another. A source grammar G_s and the document instance which is generated from this grammar, as well as a target grammar G_t exist. The user defines the correspondence between the old and the new structure. This is done in two phases. In the first phase, the correspondence between structure elements, called label association, is determined. The second phase is more elaborate. There substructures of the source document are mapped to the substructures of the target document. This phase is an interactive dialogue with the user and the system, and the outcome is what will be called tree association.

3.1 Characteristic trees

We use a concept of characteristic trees for defining an application of the label association to infinite amount of parse trees of the grammar in a finite manner.

We call *elementary trees* the set of derivation trees that have the start symbol S of the grammar as the root, terminals, empty string ϵ or content nonterminals as leaves, and no nonterminal occurring twice in any of the paths from the root to a leaf. We call *pumping factors* the derivation trees that have a nonterminal as the root and the same nonterminal as a leaf (or many leaves) and this is the only case when a nonterminal occurs twice on a path from the root to a leaf. *Characteristic* tree is the common name for elementary trees and pumping factors.

It is easy to see that

Lemma 3.1

1. *For any context-free grammar G_s the set of elementary trees and the set of pumping factors are finite.*
2. *Each parse tree can be decomposed to a elementary tree and a set of interleaved pumping factors.*
3. *All trees that are formed by interleaving pumping factors to elementary trees are parse trees.*

Because parse trees consist of characteristic trees and tree transformation can be made in parts, an automatic transformation for all parse trees can be derived from the transformation definition between characteristic trees of the source grammar and corresponding structures of the target grammar. The characteristic trees derived from the source grammar of Figure 2 are described in Figure 1.

3.2 Label association

A *label association* is a relation h from the set N_s of the nonterminals of the source grammar G_s to the set N_t of the nonterminals of the target grammar G_t. Relation h is not necessarily a mapping, because the member n of the domain N_s may have several images in the range N_t. The concept of the label association is used to show semantic similarity between the corresponding structure elements of two documents by associating the corresponding nonterminals of the source and the target grammars. For example, the nonterminal `name` of the source grammar is associated with

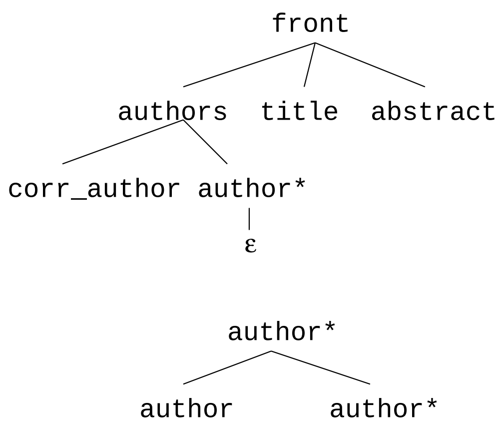

Figure 1: Characteristic trees of the source grammar

the nonterminal `author` of the target grammar, if the meaning of the element `name` in the source document corresponds to the meaning of the element `author` in the target document. If the associated nonterminals in two grammars have identical labels, the label association can be generated automatically, otherwise the user has to give the association between the nonterminals of the source and the target grammars. The label association alone does not determine the transformation but it helps to find matching substructures.

Figure 2 describes structures of the front parts of two articles. The label association between these grammars is described in Figure 3.

```
Source grammar:
  front        ->  authors title abstract
  authors      ->  corr_author (author)*
  title        ->  text
  abstract     ->  text
  corr_author  ->  text
  author       ->  text

Target grammar:
  front        ->  title shorttitle authors intro
  authors      ->  corr_author (author)*
  intro        ->  keywords abstract
  title        ->  text
  shorttitle   ->  text
  corr_author  ->  text
  author       ->  text
  keywords     ->  text
  abstract     ->  text
```

Figure 2: The source and the target grammars

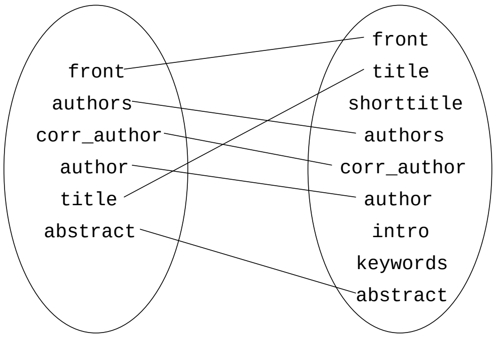

Figure 3: Label association of grammars

3.3 Tree association

Corresponding structures of the target grammar for the characteristic trees of the source grammar are derived by starting from the nonterminal of the target grammar which is associated with the start symbol of the source grammar. If there is only one rule for all the nonterminals of the target grammar, at most one corresponding structure tree for every elementary tree or pumping factor of the source grammar will be generated. This can be done automatically by starting from the root and continuing until the leaves are content nonterminals or empty strings ϵ. If there are several rules for the nonterminal n_t of the target grammar, the selection of the rule may be possible to do automatically if only one rule $n_t \to n_{t_1}, n_{t_2}, ..., n_{t_k}$ contains associated nonterminals with the rule $n_s \to n_{s_1}, n_{s_2}, ..., n_{s_l}$ of the source grammar which is used in the characteristic tree and in which n_s is associated with the nonterminal n_t. Otherwise, the user selects a suitable rule.

Consider two grammars G_s and G_t and a label association between these grammars. A *tree association* is a relation between the nodes of a derivation tree by grammar G_s and a derivation tree by grammar G_t such that the labels of related nodes are related in label association. The tree association is only meaningful when comparing the derivation trees of the same document by two grammars. If the grammars are structurally identical, the tree association is a bijective mapping. If the grammars are very different, the tree association is very partial and there is no obvious transformation of the document structure. Note that our definition allows even an empty relation being called a tree association. Such a tree association is, of course, useless from the point of view of transformation.

We say that a tree association h is *hierarchic* if for any $x \in h(u)$, $y \in h(v)$, if v is successor of u, y is a successor of x. A tree association is dense, if there is a constant c such that any non-root node with a nonempty label association has an ancestor with a nonempty label association within distance c. A tree association h is local, if there are constants d and e such that for all $x \in h(u)$, $y \in h(v)$ such that the distance of u from v is not greater than d, the distance of x from y is not greater than e.

Intuitively speaking, having a hierarchic and local tree association means that the transformation can be built top-down with a finite look-ahead in the tree.

Figure 4 describes a tree association between the nodes of the characteristic trees of the source grammar and the nodes of corresponding structures of the target grammar of Figure 2.

Now we will describe a procedure to build a derivation tree by the target grammar to an existing source grammar derivation tree so that a label association is respected as much as possible. A

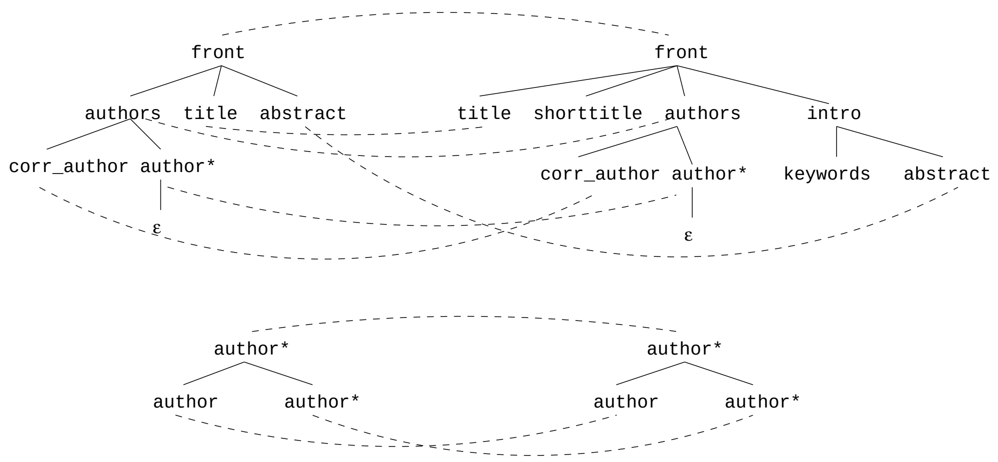

Figure 4: Tree association

sketch of the algorithm already appeared in [Kui96]. Given grammars $G_s = (N_s, \Sigma_s, P_s, S_s)$ and $G_t = (N_t, \Sigma_t, P_t, S_t)$ and a structured document T_s derived by G_s. If there is structured document T_t derived by G_t and a hierarchic, dense and local tree association between T_s and T_t, they can be found by the following procedure:

Tree association algorithm.
Given grammars G_s and G_t, a derivation tree T_s in G_s and a label association h_l such that the initial symbol of G_s is mapped to the initial symbol of G_t.

1 Associate the roots of T_s and T_t in the tree association h_t and mark them as *open* nodes.

2 If there are no more open nodes in T_s, stop. Otherwise choose an open node n and its associated node $h_t(n)$.

3 Expand the subtree whose root is n until in each branch a node m with a nonempty association $h_l(m)$ or a node without associated descendants in its subtree is found. (Due to denseness assumption, these nodes are found in bounded depth.) The nodes in frontier having a nonempty association are marked open, all other generated nodes and n are marked closed.

4 Expand the node $h_t(n)$ in T_t so that the open nodes generated in 3 can be associated in h_t. (Such nodes can be generated in finite time, because there is a hierarchic and local association.)

5 Go to 2.

Step 4 is nondeterministic, as there may be several ways to expand T_t, some of which may not lead to the required tree association. However, assuming that there is a hierarchic, dense and local tree association, it can be found in a finite time by backtracking. In the real implementation the search is guided by a suitable heuristic function providing “goodness” values for subtree candidates.

4 Automatizing the transformation

In the previous section, an interactive transformation procedure for a document tree was presented, when it is known that there is a hierarchic, dense and local transformation. The procedure can be applied to any document tree. However, the system should be as automatic as possible. We assume that the transformation is deterministic in the sense that substructures are transformed in the same way in all cases. Then, once the transformation of a structure is fixed, it can be automatically applied later on. This automatic transformation is formalized by tree transducers.

4.1 Tree transducer

A tree transducer is a tree automaton with an output. It works like a finite state automaton, but operates on trees, not on strings. It takes a finite labelled tree as an input and produces a finite labelled tree as an output.

A *descending tree transducer* [GS84] is a system $D = (\Sigma, X, Q, \Lambda, Y, P, Q')$, in which

1. Σ and Λ are ranked alphabets.
2. X and Y are the leaf alphabets.
3. Q is a ranked alphabet consisting of unary operators, the state set of D.
4. P is a finite set of productions of the following two types:
   (a) $q(x) \to t$,
   $(q \in Q, x \in X, t \in T_\Lambda(Y))$,
   (b) $q(\sigma(\xi_1, \ldots, \xi_m)) \to t$,
   $(q \in Q,\ \sigma \in \Sigma_m, m \geq 0,\ t \in T_\Lambda(Y \cup Q\Xi_m),\ (\xi_1, \ldots, \xi_m) \in \Xi_m)$,
   $\xi_1, \ldots, \xi_m$ are auxiliary variables for indicating occurrences of subtrees in a tree.

In the definition, $T_A(S)$ means the set of trees, whose internal nodes are taken from the (ranked) alphabet A and leaves from the set S. The (descending) tree transducer starts processing the source tree from the root towards the leaves and constructs the target tree structure by structure. For the node with associated state, the proper transformation rule is selected and the structure is transformed according to the rule. The states are inserted into the constructed structure for defining the unprocessed parts of the source tree.

For allowing more complex structures on the left-hand sides of the rules, we use an extension which is defined in [GS84]. In the extension a transformation step is applied to a tree and its hanging subtrees instead of applying it to a node and its hanging subtrees. Furthermore, we want to allow the addition of atomic nonterminal nodes to a target tree, which according to the previous definition is not possible. Because of the extensions, the rules are defined and presented as a term-rewriting system as follows:

P is a finite set of productions of the following two types:

1. $q(x) \to t, (q \in Q, x \in X, t \in T_\Lambda(Y))$,
2. $q(t') \to t,\ (q \in Q, t' \in (T_\Sigma \cup \Xi_m), m \geq 0, t \in T_\Lambda(Y \cup Q\Xi_m \cup \Lambda))$.

4.2 Constructing the tree automaton from the tree association

After the tree association is defined, transformation rules of the tree transducer can be generated automatically from the characteristic trees of the source grammar and associated structures of the target grammar. The nonterminals of the target grammar $G_t = (\Lambda, Y, P', S')$, which are roots in corresponding structures of characteristic trees of the source grammar, and the content nonterminals are associated with the states q_λ. The left hand side of the rule is constructed from the characteristic tree of the source grammar and from the state. Thus, the left hand side will be in the form

$$q_\lambda(\sigma(X_1, ..., X_n)) \quad \text{or} \quad q_\lambda(\sigma(X_1, ..., \sigma_i(X_j, ..., X_k), ..., X_n)).$$

The right hand side of the rule is constructed from the associated structure of the target grammar. It will be in the form

$$\lambda(q_{\lambda_1}(X_1), ..., q_{\lambda_m}(X_k)) \quad \text{or} \quad \lambda(q_{\lambda_1}(X_1), ..., \lambda_i(q_{\lambda_{i_1}}(X_1, ..., X_l), ...), q_{\lambda_m}(X_k)).$$

The leaf nodes of the trees are denoted using variables $X_i, i \geq 1$ and the interior nodes are expressed using their labels.

Figure 5 expresses the rules of the transducer, which are constructed from the trees of Figure 4. We assign states `p1` and `p2` to the roots `front` and `author` of the characteristic trees respectively and state `p3` to the content nonterminal `text`. The leaves of the trees are replaced by variables X_i, $i \geq 1$ which correspond to occurrences of content nonterminals.

```
[p1,front,
 p2,author,
 p3,text].

 1. p1(front(authors(corr_author(X1),(X2)),
          title(X3),abstract(X4)))
    -> front(title(p3(X3)),shorttitle,
          authors(corr_author(p3(X1)),p2(X2)),
          intro(keywords,abstract(p3(X4)))).
 2. p2(author*(author(X1),author*(X2))) ->
       author*(author(p3(X1)),author*(p2(X2))).
 3. p2(author*(e)) -> author*(e).
 4. p3(text(X1)) -> text(X1).
```

Figure 5: Rules for a tree transducer

Figure 6 expresses the parse tree of the source grammar of Figure 2 without the content. Initial state `p1` of the transducer is assigned to the root of the source tree.

We use the rules of Figure 5 for transforming the document to be according to the target grammar of Figure 2. Rule 1 with the initial state `p1` is applied first. Figure 7 shows a parse tree after Rule 1 is applied.

Next we apply Rules 2 and 4 of Figure 5 to corresponding subtrees. Figure 8 shows the output tree after the rules are applied to the states of Figure 7.

For transforming the recursive structure, rules 2, 3 and 4 are applied until there are no states left in the result tree. Figure 9 expresses the result of the transformation which is a parse tree according to the target grammar of the Figure 2.

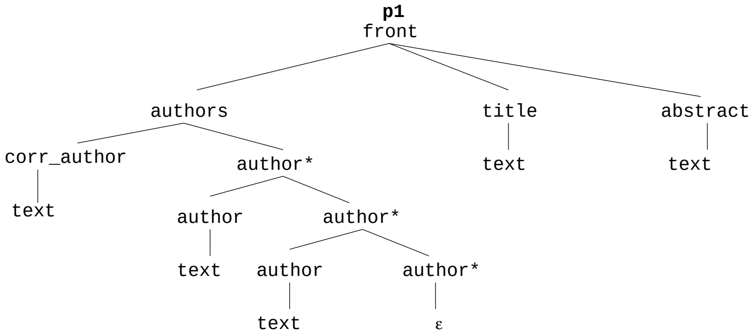

Figure 6: Parse tree of the source grammar

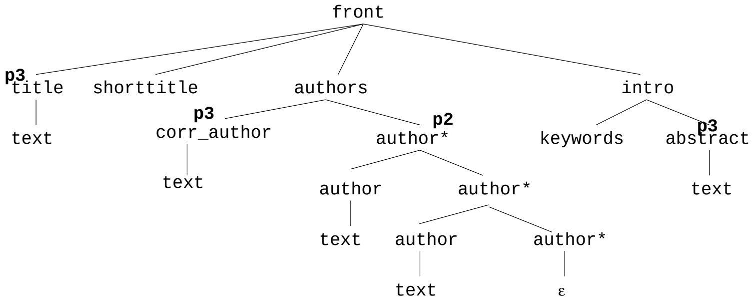

Figure 7: Transformation with tree transducer

4.3 Automatic transformation of documents

How many document trees must be transformed interactively before the system learns to transform all trees automatically? There are different strategies. One approach is the *lazy* strategy: Always try the automatic transformation by tree transducer first, and if it fails, use the interactive procedure and add rules to the transducer accordingly. The other approach is the *eager* strategy: transform first all (finitely many) characteristic trees interactively. After adding respective rules to the tree transducer, all trees can be transformed.

The advantage of the lazy strategy is that it is easy, but one has to return back to the interactive procedure every time a new type of document is met. The eager strategy produces a complete transducer at once, but after a big effort. Indeed, the effort may be overwhelmingly high, and some of the synthetically produced characteristic trees may be such that they practically never occur in real documents. Therefore we believe that the strategy should be lazy rather than eager.

We have a preliminary implementation of both transformations. They are still far from a real running transformation system, but prove that the procedures can be automatized.

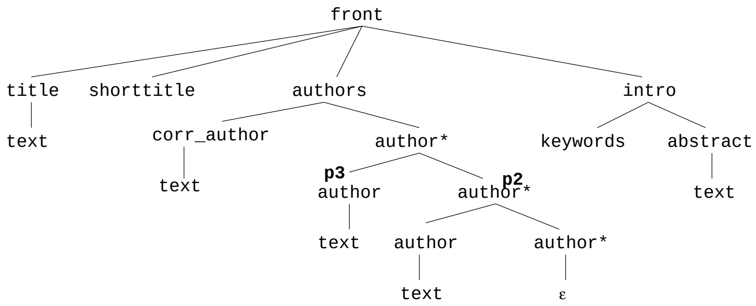

Figure 8: Transformation of the recursive structure

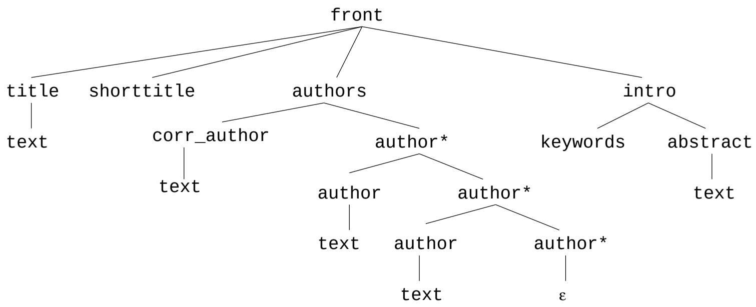

Figure 9: Parse tree of the target grammar

5 Conclusions

It is too difficult for the user to construct the rules for the tree transducer manually. By defining the association between the nonterminals of the grammars and applying it between the characteristic trees of the source grammar and corresponding structures of the target grammar it is possible to construct the rules at least semi-automatically with the computer if the grammars are closely related and the tree association is congruent with the characteristic trees of the source grammars and corresponding structures of the target grammar.

References

[AU72] A.V. Aho and J.D. Ullman. *The theory of parsing, translation, and compiling, Vol. I: Parsing*. Prentice-Hall, Engelwood Cliffs, N.J., USA, 1972.

[CK95] K. Chiba and M. Kyojima. Document transformation based on syntax-directed tree translation. *Electronic Publishing – Origination, Dissemination, and Design*, 8(1):15–29, 1995.

[Cla96] J. Clark. James' DSSSL engine. http://www.jclark.com/jade/ (May 31, 1999), 1996.

[Dra98] N. Drakos. The LaTeX2HTML translator. http://www-dsed.llnl.gov/files/programs/unix/ latex2html/manual/manual.html (June 1, 1999), 1998.

[FS88] R. Furuta and P.D. Stotts. Specifying structured document transformations. In J.C. van Vliet, editor, *Document Manipulation and Typography*, The Cambridge Series on Electronic Publishing, pages 109–120, Nice, France, 1988. Cambridge University Press.

[FW93] A. Feng and T. Wakayama. SIMON: A grammar-based transformation system for structured documents. *Electronic Publishing – Origination, Dissemination, and Design*, 6(4):361–372, 1993.

[GS84] F. Gécseg and M. Steinby. *Tree Automata*. Académiai Kiadó, Budabest, 1984.

[ISO86] ISO 8879. Information processing - Text and Office Systems - Standard Generalized Markup Language (SGML). ISO, Geneva, 1986.

[KLMN90] P. Kilpeläinen, G. Lindén, H. Mannila, and E. Nikunen. A structured document database system. In R. Furuta, editor, *EP90, Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography*, The Cambridge Series on Electronic Publishing, pages 139–151, Gaithersburg, Maryland, 1990. Cambridge University Press.

[KP95] E. Kuikka and M. Penttonen. Transformation of structured documents. *Electronic Publishing – Origination, Dissemination, and Design*, 8(4):319–341, December 1995.

[KPPM84] S.E. Keller, J.A. Perkins, T.F. Payton, and S.P. Mardinly. Tree transformation techniques and experiences. *SIGPLAN Notices*, 19(6):190–201, 1984.

[KPV94] E. Kuikka, M. Penttonen, and M.-K. Väisänen. Theory and implementation of SYNDOC document processing system. In *Proceedings of the Second International Conference on Practical Application of Prolog, PAP-94*, pages 311–327, London, UK, 1994.

[Kui96] E. Kuikka. *Processing of Structured Documents Using a Syntax-Directed Approach*. PhD thesis, Kuopio University Publications C. Natural and Environmental Sciences 53, 1996.

[KW97] P. Kilpeläinen and D. Wood. SGML and exceptions. In *Proceedings of the PODP96 Workshop on the Principles of Document Processing*, Lecture Notes in Computer Science, Vol. 1293, pages 39–49, Palo Alto, 1997. Springer-Verlag.

[Lam86] L. Lamport. *LaTeX, A Document Preparation System.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.

[MOB94] S. Mamrak, C.S. O'Connell, and J. Barnes. *Integrated Chameleon Architecture.* PTR Prentice Hall, Englewood Cliffs, N.J., USA, 1994.