

Document transformation based on syntax-directed tree translation

KAZUYA CHIBA AND MASAKI KYOJIMA

Fuji Xerox Co., Ltd.

Systems and Communications Laboratory

430 Sakai, Nakai-machi, Ashigarakami-gun, Kanagawa, 259-01 JAPAN

SUMMARY

We present a description system for transformation of structured documents based on Context Free Grammars (CFGs). The system caters to transformations between different document class descriptions, and is presented mainly in terms of logical structure transformation. Two requirements for transformation are proposed: the output document class must be explicitly representable, and inconsistency must be avoidable. First, a grammar for document class descriptions, called a T-CFG (Tree-preserving Context Free Grammar), is introduced, then SDTT (Syntax-Directed Tree Translation) is given for a document transformation. The SDTT transformation is formal, concise, and consistent with the above two requirements.

KEY WORDS Class-level document transformation Context Free Grammar Document transformation Structured document Syntax-Directed Translation

1 INTRODUCTION

In this paper we present a description system for structured document transformation based on Context Free Grammars (CFGs).

The structured documents considered here are similar to those in two international standards: the Open Document Architecture (ODA)[1] and the Standard Generalized Markup Language (SGML)[2]. These documents have tree structures; in particular, only leaf nodes are associated with contents such as texts, graphics, and mathematical expressions.

We use the term ‘document class’ in this paper in the same context as ‘document type’ in SGML. Formally, a document class is a set of documents. For example, document classes appearing in offices include business letters, invoices, and so on. A document class description represents constraints on document structures. Any document in a class must satisfy the constraints for that particular document class.

We define document transformation in terms of document class descriptions as follows:

1. The input document(s) to the transformation belongs to the input document class; the output document(s) from the transformation belongs to the output document class.
2. The transformation is described using input and output document class descriptions.

Within this framework, we can consider various types of document transformation, including transformation of layout structures and contents. However, we focus mainly on

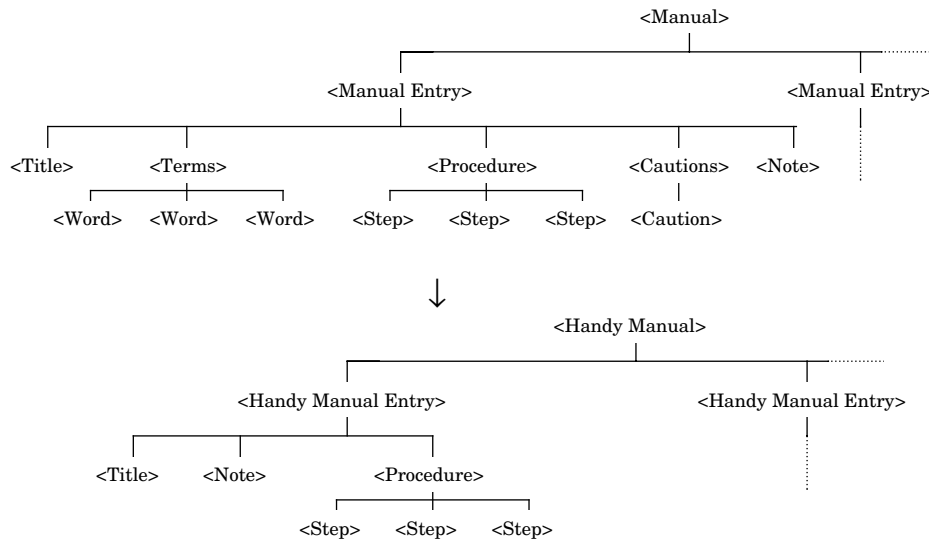


Figure 1. Transformation from a manual into a handy manual

transformation of logical structures because this illustrates the principles (and new layout structures can be generated, provided that appropriate formatters exist). Figure 1 shows a simple example of such a transformation: the transformation from the logical structure of a manual into that of a handy manual. It is a common requirement in offices to generate documents of one class from another. Our framework is applicable to doing this. Using automated systems for generating documents help reduce office work [3].

Let us now look at some other approaches related to the transformation of document logical structures. The Document Style Semantics and Specification Language (DSSSL)[4] is one of these. Its intent is to add formatting semantics to SGML documents. We believe DSSSL can also be used to describe transformations of logical structures. DSSSL specifications can be considered as a set of constraints, namely associations, based on SGML Document Type Definitions (DTD). An association relates an element in the input document instance to an element in the output document instance. Developing an execution mechanism for the descriptions of DSSSL, however, has not been accomplished. Moreover, associations conforming to DSSSL are not always executable. For example, two mutually inconsistent associations can exist.

Brown, et al., proposed a new description language [5], whose purpose is similar to that of DSSSL. They use a regular right-hand part grammar as a document class description, i.e., a DTD. In their description language, coordination was used to constrain structural correspondence between elements specified by source (input) and result (output) document class descriptions. We think that their descriptions of transformation look simpler and more compact than the descriptions of DSSSL. However, an execution mechanism has not been introduced. It is not clear whether their descriptions are always executable.

Akpotsui and Quint proposed a method where a comparator compares the old (input) and new (output) generic structures (document class descriptions) and outputs rules for doc-

ument instance conversion [6]. However, we cannot tell exactly its capability to transform because an algorithm of the comparison is not shown.

Cole and Brown proposed a method to change the classes of objects in structured documents during cut-and-paste editing [7]. Although their method can be considered one type of transformation, it does not manipulate structures.

In contrast to these approaches, the approach by Güting, et al. [8], is algebraic; it is similar to retrieving data from a relational database. They proposed the Nested Sequence of Tuples (NST) algebra which is an extended relational algebra. In their approach, operations are defined on schemata; in other words, document processing is described at the class level. This algebra allows the description of various transformation. Since any operation can be applied to a schema, their approach is quite capable of describing a transformation. Moreover, executing these descriptions is given, which would be simpler than the approaches mentioned earlier.

However, an output schema, i.e. a schema for outputs from the transformation, is not represented explicitly in the NST-algebra, unlike DSSSL or the approach of Brown, et al. In DSSSL, an output document class is represented by a DTD in the same way that an input document class is represented. If an output class is represented explicitly, it is easier to describe the transformation with a pre-defined form.

Similarly, in the method of Furuta and Stotts [9], an output document class is not represented explicitly; a transformation is specified by a sequence of operators, which are used to construct the output grammar (class) from the given input grammar.

Considering the above work, we determined that our transformation description should meet the following requirements:

- the output class must be explicitly representable;
- inconsistency must be avoidable: there is a well-defined syntax which guarantees that if a transformation description satisfies the syntax, the transformation is executable and outputs a document belonging to the output class.

In addition, we think that our transformation description should be formal and concise.

We now give an outline of our approach. We concentrate on transformation independent of contents. In other words, we discuss mainly document structure transformation. For convenience, hereinafter, by document transformation or simply transformation, we mean document structure transformation. For our document class descriptions, we use CFGs. For our structural transformation, we use a Syntax-Directed Translation (SDT)[10], a CFG-based translation between strings with declarative descriptions. We also introduce a string representation for document structures because CFGs and SDTs handle strings, not trees. Next, we obtain grammar for document class descriptions, called a *T-CFG (Tree-preserving Context-Free Grammar)*, and a tree transformation for document transformation, called an *SDTT (Syntax-Directed Tree Translation)*. Finally, we apply SDTTs to document transformation.

The rest of the paper is organized as follows: In Section 2, we first define a T-CFG and an SDTT, and then explain the document transformation using SDTTs. To illustrate its capability, we also show some examples of document transformation using SDTTs. In Section 3, we discuss some features of our description, including comparisons. Finally, in Section 4, we summarize our work and discuss areas that need further investigation in future work.

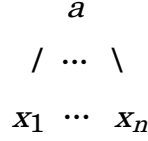


Figure 2. The tree structure corresponding to $a[x_1 \cdots x_n]$

2 DOCUMENT TRANSFORMATION USING SDTT

We define a T-CFG and an SDTT in Section 2.1. In Section 2.2, we apply SDTTs to document transformation and show some examples. Hereinafter, the term ‘structure’ means ‘logical structure’ unless otherwise stated, and notation related to SDTs follows [11].

2.1 T-CFG and SDTT

First, we give a string representation for a tree. We introduce two new special symbols, ‘[’ and ‘]’, and we define that string $a[x_1 \cdots x_n]$ corresponds to or is decoded into the tree structure illustrated in Figure 2. For example, string $a[b[cc]b[c]b[ccc]]$ corresponds to the tree shown in Figure 3. We introduce $T(A,B)$ to denote a set of strings which correspond to trees whose leaf nodes belong to B and whose non-leaf nodes belong to A . $T(A,B)$ is defined as follows

Definition 1 Let A and B be sets of symbols, and ‘[’ and ‘]’ be new special symbols to represent structures. A set of strings $T(A,B)$ is defined inductively as follows:

1. If $b \in B$, then $b \in T(A,B)$.
2. If $a \in A$ and $x_1, \dots, x_n \in T(A,B)$, then $a[x_1 \cdots x_n] \in T(A,B)$ ($n \geq 0$).

Next, we define a CFG and $T^*(A,B)$, a sequence of trees belonging to $T(A,B)$.

Definition 2 (CFG) A CFG is a four-tuple $G = \langle N, \Sigma, P, S \rangle$ where

1. N is a finite set of nonterminals;
2. Σ is a finite set of terminals and $N \cap \Sigma = \emptyset$;
3. P is a finite set of productions of the form $A \rightarrow \gamma$ where $A \in N$ and $\gamma \in (N \cup \Sigma)^*$;
- and
4. $S \in N$ is the starting symbol.

Definition 3 $T^*(A,B) \equiv \{x_1 \cdots x_n \mid x_1, \dots, x_n \in T(A,B)\} \cup \{\epsilon\}$ (ϵ denotes an empty string.)

Now we introduce T-CFGs as a subclass of CFGs. A T-CFG is a grammar which can generate a string representing a tree. As discussed later, a T-CFG will correspond to a document class description for tree-structured documents.

Definition 4 (T-CFG) A CFG $G: \langle N, \Sigma \cup \{‘[’, ‘]’\}, P, S \rangle$ is called a T-CFG if, for every production $A \rightarrow \gamma$ in the finite set of productions P , $\gamma \in T^*(\Sigma, N \cup \Sigma)$.

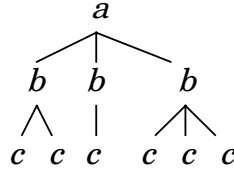


Figure 3. The tree corresponding to $a[b[cc]b[c]b[ccc]]$

An example of a T-CFG is $G_1 = \langle N, \Sigma \cup \{ '[', ']' \}, P, S \rangle$ as follows.

$$\begin{aligned}
 N &= \{ S, B, C \} \\
 \Sigma &= \{ a, b, c \} \\
 P &= \{ S \rightarrow a[B] \\
 &\quad B \rightarrow Bb[C] \\
 &\quad B \rightarrow b[C] \\
 &\quad C \rightarrow Cc \\
 &\quad C \rightarrow c \}
 \end{aligned}$$

G_1 generates the trees such that the root node is a ; node a has any number of children nodes, b 's; and each b has any number of children nodes, c 's. Figure 3 shows a tree generated by G_1 .

The following example is evidence that a T-CFG can generate a sequence of trees, not just a single tree. Only the productions are shown.

$$\begin{aligned}
 S &\rightarrow Sa[bc] \\
 S &\rightarrow a[bc]
 \end{aligned}$$

We give Theorem 1 stating that a T-CFG generates nothing but a sequence of trees.

Theorem 1 Let $L(G)$ be the language generated by a T-CFG

$$G = \langle N, \Sigma \cup \{ '[', ']' \}, P, S \rangle$$

Then $L(G) \subseteq T^*(\Sigma, \Sigma)$.

Proof. The proof is given in the Appendix. □

Here we consider the transformation description. By applying SDTs, translations between strings, to T-CFGs, we obtain SDTTs. Hence, an SDTT is a translation between strings corresponding to underlying tree structures. Now, we will define an SDTT.

Definition 5 (SDTT) An SDTT is a five-tuple $T = \langle N, \Sigma, \Delta, R, S \rangle$ where

1. N is a finite set of nonterminals and $N \cap (\Sigma \cup \Delta) = \phi$,
2. Σ is a finite set of symbols (the terminals of the input strings),
3. Δ is a finite set of symbols (the terminals of the output strings),
4. R is a finite set of rules of the form $A \rightarrow \beta, \gamma$
and
5. $S \in N$ is the starting symbol.

For each rule $A \rightarrow \beta, \gamma$, it must hold that $A \in N, \beta \in T^*(\Sigma, N \cup \Sigma), \gamma \in T^*(\Delta, N \cup \Delta)$, and the nonterminals in β must be a permutation of those in γ . Moreover, each instance of a nonterminal in β has an associated instance of the same nonterminal in γ . For repeated nonterminal symbols, superscripts are used as needed to indicate which nonterminals are associated.

The T-CFG $\langle N, \Sigma \cup \{', ''\}, P_i, S \rangle$ is the input grammar of this SDTT where $P_i = \{A \rightarrow \beta | A \rightarrow \beta, \gamma \in R\}$; the T-CFG $\langle N, \Delta \cup \{', ''\}, P_o, S \rangle$ is the output grammar of this SDTT where $P_o = \{A \rightarrow \gamma | A \rightarrow \beta, \gamma \in R\}$.

The idea of an SDTT is to use the input grammar for derivations from the input string and, simultaneously via the rules, use the output grammar for derivations from the output string. An SDTT T defines a translation set.

Definition 6 (translation form) Let $T = \langle N, \Sigma, \Delta, R, S \rangle$ be an SDTT. $\pi(T)$, the set of translation forms of T , is defined as follows:

1. $(S, S) \in \pi(T)$ and the S 's are associated.
2. If $(\beta' A \beta'', \gamma' A \gamma'') \in \pi(T)$ with the A 's associated and $A \rightarrow \beta, \gamma \in R$, then $(\beta' \beta \beta'', \gamma' \gamma \gamma'') \in \pi(T)$.

Definition 7 (translation set) Let $T = \langle N, \Sigma, \Delta, R, S \rangle$ be an SDTT, G_i be an input grammar of T , and G_o be an output grammar of T . The translation set of T is

$$\tau(T) = \{(x, y) | x \in L(G_i), y \in L(G_o), (x, y) \in \pi(T)\}$$

which is a subset of binary relation $L(G_i) \times L(G_o)$.

We say a string σ_i is transformed into a string σ_o by an SDTT T if $(\sigma_i, \sigma_o) \in \tau(T)$. Note that one string can be transformed into different forms by one SDTT. However, any string in $L(G_i)$ is transformed into at least one form, as stated in Theorem 2.

Theorem 2 Let $T = \langle N, \Sigma, \Delta, R, S \rangle$ be an SDTT, G_i be an input grammar of T , and G_o be an output grammar of T . Then, for any string $x \in L(G_i)$, there exists a string $y \in L(G_o)$ such that $(x, y) \in \tau(T)$.

Proof. The proof is given in the Appendix. □

We now give an example of a transformation. This transformation inputs one of the trees generated by T-CFG G_1 described above, and outputs the tree in which the order of occurrence of b 's is reversed. By this transformation, the tree in Figure 3 is transformed into the tree in Figure 4 We can describe this transformation by the following SDTT T_1 :

$$\begin{aligned} T_1 &= \langle N, \Sigma, \Delta, R, S \rangle \\ N &= \{ S, B, C \} \\ \Sigma &= \{ a, b, c \} \\ \Delta &= \{ a, b, c \} \\ R &= \left\{ \begin{array}{ll} S \rightarrow a[B], & a[B] \\ B \rightarrow Bb[C], & b[C]B \\ B \rightarrow b[C], & b[C] \\ C \rightarrow Cc, & Cc \\ C \rightarrow c, & c \end{array} \right\} \end{aligned}$$

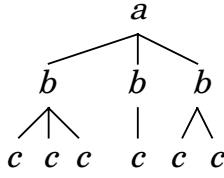


Figure 4. An output tree of transformation T_1

The input grammar of T_1 is T-CFG G_1 . Input grammar always indicates the domain of the transformation.

For the output side, the output grammar of T_1 is the following T-CFG with the starting symbol S .

$$\begin{aligned}
 S &\rightarrow a[B] \\
 B &\rightarrow b[C]B \\
 B &\rightarrow b[C] \\
 C &\rightarrow Cc \\
 C &\rightarrow c
 \end{aligned}$$

2.2 Document transformation using SDTT

In this section we give our document transformation description using SDTTs. First, by the definitions given in Section 2.1, we model a document, a document class, and a document transformation. Next, we give some examples to explain document transformation using SDTTs, and to show its characteristics and capability to describe.

The documents which we model here are the structured documents mentioned in Section 1. An example is shown in Figure 5. Each node of a tree, called an *object*, has a name. In Figure 5, a , b , d , e , and g are names of objects. In particular, each leaf node of a tree is associated with a container of contents, called a *content portion*. In Figure 5, C_1 , C_2 , and C_3 are content portions.

We model such documents in the following way. We represent an object as a symbol, and a document as a string σ which corresponds to the tree structure of the document. For example, the document shown in Figure 5 is represented as $a[b[e]d[gg]]$. Now we define a document formally.

Definition 8 Let A and B be sets of symbols. A string σ is called a document if $\sigma \in T(A,B)$.

Next, we use T-CFGs to model document classes. We define that a T-CFG is a document class description: if a T-CFG G generates a string σ , the document σ belongs to the document class G .

A T-CFG can generate a sequence of trees as mentioned above. If a T-CFG G generates a sequence of strings $\sigma_1 \cdots \sigma_n$, the sequence of documents $\sigma_1, \cdots, \sigma_n$ belongs to the document class G . Here we extend the concept of document classes: a document class described by a T-CFG is a set of sequences of documents, not just a set of single documents. However, we mainly discuss single documents.

Finally, we consider document transformation using SDTTs. The transformation between documents can be described as the transformation between the strings corresponding

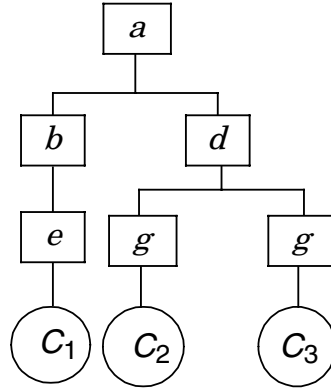


Figure 5. An example of structured documents

to the underlying documents. We say a document σ_i is transformed into a document σ_o by an SDTT T if $(\sigma_i, \sigma_o) \in \tau(T)$. SDTT can also transform multiple documents into multiple documents all at one time, as mentioned above.

Now we show two examples of a document transformation using SDTTs.

Example 1

This example shows that an SDTT can divide repeated objects into small pieces. See Figure 6. Words enclosed with brackets $\langle \rangle$ are symbols, for example $\langle Name \rangle$. The input document (a), or its string representation (c), is a participants list for a meeting. This list is a sequence of the names and addresses of members. C_1, C_2, \dots, C_6 are content portions.

Then, we explain the transformation. We divide the members into pairs. If the number of members is odd, the last member becomes a singleton. The output document is (b), or its string representation (d). Each content portion of a name or address in the input document becomes the content portion of the corresponding name or address in the output document.

This transformation can be described using the following SDTT T_2 . Only the rules are shown.

$$\begin{array}{l}
 S \rightarrow \langle List \rangle [PO], \quad \langle PairList \rangle [PO] \\
 P \rightarrow M^1 M^2 P, \quad \langle Pair \rangle [M^1 M^2] P \\
 P \rightarrow \epsilon, \quad \epsilon \\
 O \rightarrow M, \quad \langle Pair \rangle [M] \\
 O \rightarrow \epsilon, \quad \epsilon \\
 M \rightarrow \langle Member \rangle [\langle Name \rangle_1 \langle Address \rangle_2], \\
 \quad \quad \quad \langle Member \rangle [\langle Name \rangle_1 \langle Address \rangle_2]
 \end{array}$$

We now describe notations for transferring content portions. The subscripts in the last rule of SDTT T_2 are such notations. Subscripts are added to terminal symbols in the rules to indicate associations between a terminal symbol (an object) in the input document and that

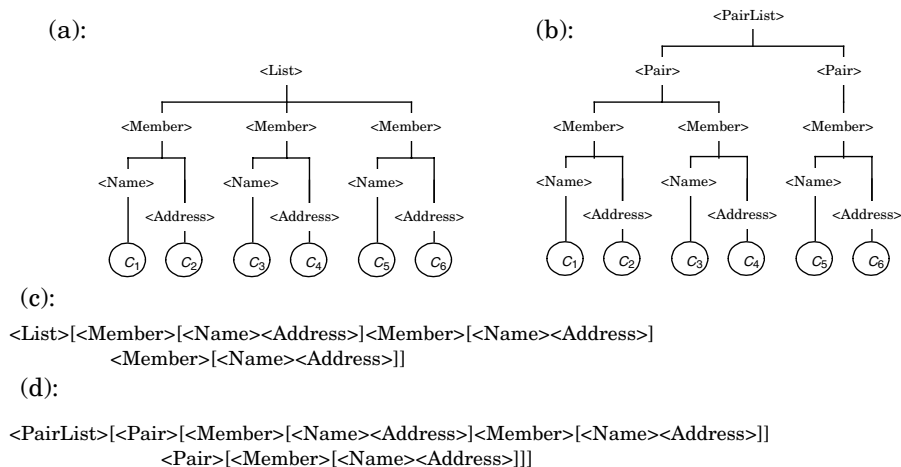


Figure 6. An example of document transformation (Example 1)

in the output document; the associations are called *terminal-associations*. These subscripts indicate that the content portion of an object should be transferred to the content portion of another object which has the same subscript. Such subscripts can be omitted if there are no repeated terminal symbols in the output part of a rule and each terminal symbol appears only once in the input part.

Then, we explain the execution of our transformation description using this example. First, from the input document, the corresponding string expression (c) in Figure 6 is derived. The string is parsed by a CFG parser with the input grammar of the SDTT, and the parsing tree (e) in Figure 7 is derived. Note that this parsing can be performed at worst in $O(n^3)$ time in terms of the length of the input string [12]. Next, using the input parse tree and the rules of the SDTT, the output parse tree (f) in Figure 7 is obtained. The terminal-associations are shown by dotted lines in Figure 7. From the output string (d) in Figure 6 obtained from the output parse tree, the output tree structure is constructed. Finally, the content portions in the input document are extracted, to form the output document (b) in Figure 6. If this parsing fails, the execution stops and it is judged that the input document does not belong to the input document class.

Example 2

This example shows that we can apply SDTTs to recursive transformation. See Figure 8. Each section in the input document (i) has a title and has subsections or a body. Similarly, each subsection can also have subsections. The input document class should be defined recursively, as the ODA expression (iii).

Next, we describe the transformations which pick up titles from the input document. We consider two cases: (a) to pick up titles as a flat sequence, and (b) to pick up titles with the tree structure of the input document. For each case, we show the output document (ii) (a) or (b) in Figure 8, and the SDTT, shown in Figure 8 (iv) (a) or (b), describing the transformation.

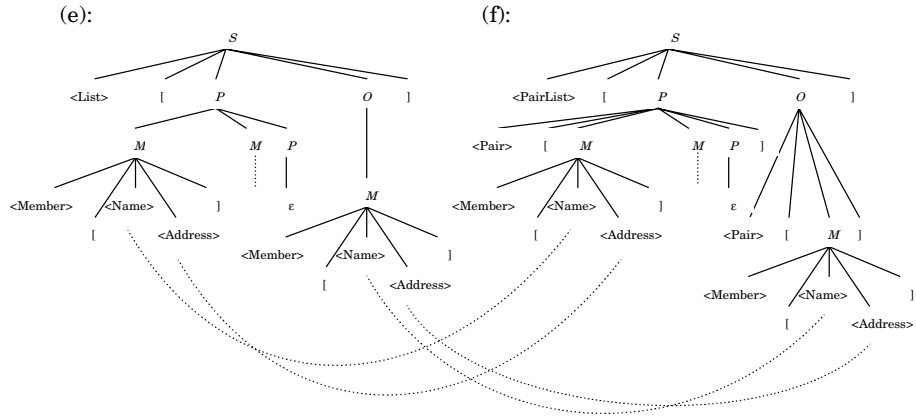


Figure 7. The input and the output parsing trees (for Example 1)

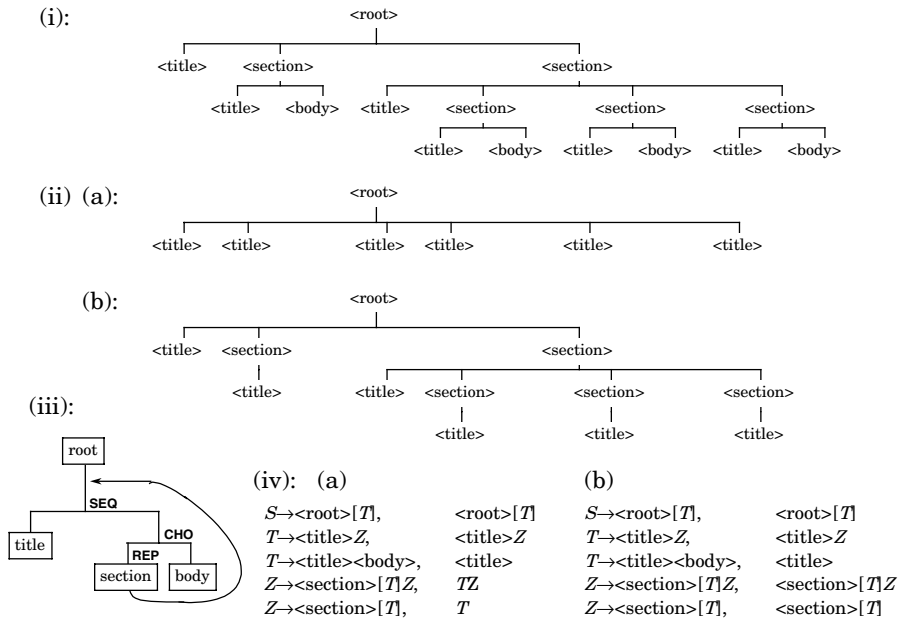


Figure 8. An example of document transformation (Example 2)

3 DISCUSSION

In this section we discuss some features of our transformation description. First, we confirm that our description meets the following two requirements mentioned in Section 1:

- the output class must be explicitly representable;
- inconsistency must be avoidable.

Our description satisfies the first requirement since the output grammar of the SDTT indicates the output document class explicitly. It also satisfies the second requirement because SDTTs are so well defined that we can syntactically check whether or not a description is an SDTT. Furthermore, an SDTT always outputs a document belonging to the output class as Theorem 2 states.

Here, we consider T-CFG's capability to represent document classes. T-CFGs can represent choice structures and recursive structures, none of which the NST data model can represent. Thus, T-CFGs can represent generic structures specified by any complete generator set of ODA. Similarly, T-CFGs can represent generic structures specified by any content models excluding 'exceptions'¹ in DTDs of SGML, except structures introduced by references. These characteristics are due to the fact that T-CFGs are based on CFGs. Note that various existing techniques for CFGs can also be applied to T-CFGs.

Let us now look at other features of our transformation description. SDTT descriptions are formal. Additionally, they are concise, especially compared with those of DSSSL. A description of DSSSL is a set of associations; an example of an association is as follows:

$$\begin{aligned} & \textit{Document/Body/Chapter/Para} \\ & \rightarrow \textit{TDocument/TBody/TChapter/ContPort(new)} \end{aligned}$$

Concerning execution, we introduced an execution mechanism for SDTT descriptions, in contrast to the two constraint-based approaches, i.e., DSSSL and the approach by Brown, et al. In general, it is difficult to find answers that satisfy the constraints. In comparison, the execution mechanism for SDTTs would be simpler.

Another characteristic of our document transformation is that a document structure is different from a parse tree, unlike other grammar-based approaches such as [5]. For example, the tree structure of document (a) in Figure 6 is quite different from parse tree (e) in Figure 7 for the document. This characteristic makes our description more capable, especially to describe transformations of repeated objects such as reversing their order, dividing a sequence, and selecting the n -th object.

We have mentioned that SDTTs can transform multiple documents belonging to one document class. We can also describe transformation of multiple documents belonging to multiple document classes using SDTTs with little enhancement. For example, suppose that some different document classes are represented by T-CFGs G_1, \dots, G_n whose starting symbols are S_1, \dots, S_n , respectively. Then we can get one unified T-CFG G' whose production rules consist of $S \rightarrow S_1, \dots, S_n$ and productions of G_1, \dots, G_n , where some nonterminals in productions of G_1, \dots, G_n are modified, so that each nonterminal appears in only one of G_1, \dots, G_n .

¹ T-CFGs can also represent $AGG('&')$ structures, by choice constructor (or group) of sequence constructor (or group) permutations. For example, $(a\&b)$ is represented by $(a,b)/(b,a)$.

Moreover, an SDTT is symmetric with respect to input and output. We can obtain the inverted SDTT by changing each rule $A \rightarrow \beta, \gamma$ into $A \rightarrow \gamma, \beta$. This inverted SDTT is the inverse transformation. Thus, we can use SDTTs to describe mutual conversion between two forms, for example, the forms in Example 1. In general, SDTTs used in two directions should be one-to-one transformations. The transformations in Example 2 are not one-to-one transformations.

Although we have regarded SDTTs as transformations for documents having tree structures, we can also apply them to flat documents, i.e., documents consisting of a flat sequence of objects. For example, even if a document $r[ax[bc]bc]ax[bc]ax[bc]bc$ is replaced by a flat document with the same contents $r[abcabcabc]bc$, SDTTs transforming the document change only slightly; the change is as slight as the difference between (a) and (b) in Figure 8 (iv). In other words, the SDTT-based approach does not extensively depend on hierarchical structures, as might be important from the point of view that the tree structure in a structured document can be considered to be additional. We can also use SDTTs to structure a flat document.

Next, we discuss SDTT's capability to describe. As the above examples and the preceding discussion show, SDTTs can describe various transformations, some of which other approaches cannot describe. However, we can list some transformations that SDTTs cannot describe:

- duplication of substructures,
- transformation of or into tree structures whose string representation is not in a context-free language, for example $a^n b^n c^n (\{abc, aabbcc, \dots\})$,
- sorting, grouping, or joining, in the sense of [6], and so on.

Since we have concentrated on transformation independent of contents, the following are also undecidable:

- contents-dependent transformation of structures,
- transformation between contents and structures, and
- transformation of contents.

We plan to extend our document transformation to currently undecidable transformations.

If one input document class description is already given, it is not very difficult to describe an SDTT from the input class into a new class. However, if input and output document class descriptions are given, describing an SDTT between the two classes may cause difficulty because the two classes must be combined into one SDTT by hand.

4 CONCLUSIONS AND FUTURE WORK

We have proposed a transformation description for class-level transformation of document logical structures. The given transformation description is formal, concise and consistent with our requirements: the output document class must be explicitly representable, and inconsistency must be avoidable. An execution mechanism for SDTT description has been introduced that would be simpler than that of other constraint-based approaches such as DSSSL. SDTTs can describe various transformations:

- transformation of or into multiple documents,
- various transformation of repeated objects,

- recursive transformation,
- transformation including choices of substructures, and
- transformation from a flat document into a document with a hierarchical structure, and so on.

Therefore, we think that this approach based on CFGs or SDTs is useful to describe class-level document transformation.

We now describe some directions to extend our approach to currently undecidable transformations. Attribute grammar [13] is probably suitable for describing transformations between contents and structures. We think that an attribute value can represent information of contents. For transformations which attribute grammar approaches cannot cover, variations of CFGs, currently existing or newly modified, may be needed. We may also need to describe manipulation of contents, not content portion as a whole. For future work, it is also important to characterize the transformations describable by SDTTs.

ACKNOWLEDGEMENTS

We would like to thank Yasunori Koda for his valuable comments on earlier versions of this paper.

REFERENCES

1. ISO. Information processing - text and office systems - office document architecture (ODA) and interchange format - ISO/IS 8613, 1988.
2. ISO. Information processing systems – text and office systems – standard generalized markup language (SGML). ISO 8879, 1986.
3. M. Kyojima, N. Kamibayashi, and N. V. Gopal, ‘Document programming’, *IPSI SIG Notes(In Japanese)*, (DPHI-23), (1989).
4. ISO. Information technology - text and office systems - document style semantics and specification language(DSSSL) ISO/IEC DIS 10179, 1991.
5. A. L. Brown Jr., T. Wakayama, and H. A. Blair, ‘A reconstruction of context-dependent document processing in SGML’, in *Electronic Publishing '92*, pp. 1–25, Cambridge University Press, (1992).
6. E. Akpotsui and V. Quint, ‘Type transformation in structured editing systems’, in *Electronic Publishing '92*, pp. 27–41, Cambridge University Press, (1992).
7. Fred Cole and Heather Brown, ‘Editing structured documents - problems and solutions’, *Electronic Publishing Origination, Dissemination, and Design*, **5**(4), 209–216, (December 1992).
8. R. H. Guting, R. Zicari, and D. M. Choy, ‘An algebra for structured office documents’, *ACM Transactions on Office Information Systems*, **7**(4), 123–157, (1989).
9. R. Furuta and P. D. Stotts, ‘Specifying structured document transformations’, in *Electronic Publishing '88*, pp. 109–120, Cambridge University Press, (1988).
10. P. M. Lewis II and R. E. Stearns, ‘Syntax-directed transduction’, *Journal of the ACM*, **15**(3), 465–488, (1968).
11. R. C. Gonzalez and M. G. Thomason, ‘Syntax-directed translations’, in syntactic pattern recognition’, in *Syntactic Pattern Recognition*, pp. 57–60, Addison-Wesley Publishing Company, (1978).
12. D. H. Younger, ‘Recognition and parsing of context-free languages in time n^3 ’, *Information and Control*, **10**(2), 189–208, (1967).
13. D. E. Knuth, ‘Semantics of context-free languages’, *Mathematical Systems Theory*, **2**(2), 127–145, (1968).

APPENDIX

We first define that \Rightarrow_G is the derivation relation for a grammar G , and that \Rightarrow_{G^*} is the reflexive-transitive closure of \Rightarrow_G .

Proof of Theorem 1.

Before proving the theorem, we prove that $S \Rightarrow_{G^*} x$ implies $x \in T^*(\Sigma, N \cup \Sigma)$, where $S \Rightarrow_{G^*} x$ means that there exists a derivation $S = x_0 \Rightarrow_G x_1 \Rightarrow_G \cdots \Rightarrow_G x_n = x (n \geq 0)$. This is proved by induction on n .

Suppose first that $n = 0$: no derivations is made. Since $S \in N$, it holds that $S \in T^*(\Sigma, N \cup \Sigma)$.

Next suppose that $S \Rightarrow_{G^*} x_{n-1}$ implies $x_{n-1} \in T^*(\Sigma, N \cup \Sigma)$ for derivations of length $n - 1$. Consider derivations of length n . Suppose that $S \Rightarrow_{G^*} x_n$ and let y be x_{n-1} . Then, by the induction hypothesis, y is a sequence of trees $y_1, \dots, y_j \in T(\Sigma, N \cup \Sigma)$. The production $A \rightarrow \gamma$ which is applied to y rewrites a nonterminal A in a tree, namely $y_k (1 \leq k \leq j)$, as γ . We consider two cases:

1. If $A = y_k$, then $x_n = y_1 \cdots y_{k-1} \gamma y_{k+1} \cdots y_j$. Since $\gamma \in T^*(\Sigma, N \cup \Sigma)$, it holds that $x_n \in T^*(\Sigma, N \cup \Sigma)$.
2. Otherwise, A appears in y_k . Since a nonterminal in y_k appears only as a leaf node of the tree, by rewriting a nonterminal in the tree $y_k \in T(\Sigma, N \cup \Sigma)$ as a sequence of trees $\gamma \in T^*(\Sigma, N \cup \Sigma)$, the resultant tree is still in $T(\Sigma, N \cup \Sigma)$. It thus holds that $x_n \in T^*(\Sigma, N \cup \Sigma)$, proving our assertion.

We have proved that $S \Rightarrow_{G^*} x$ implies $x \in T^*(\Sigma, N \cup \Sigma)$. By the definition of $L(G)$, for any $x \in L(G)$, $S \Rightarrow_{G^*} x$ and $x \in (\Sigma \cup \{', '\})^*$. Therefore we have that for any $x \in L(G)$, $x \in T^*(\Sigma, N \cup \Sigma)$ and $x \in (\Sigma \cup \{', '\})^*$. Hence $L(G) \subseteq T^*(\Sigma, N \cup \Sigma) \cap (\Sigma \cup \{', '\})^* = T^*(\Sigma, \Sigma)$.

Proof of Theorem 2.

Before proving the theorem, we prove that for any string x such that $S \Rightarrow_{G^*} x$, that is, $S = x_0 \Rightarrow_{G_i} x_1 \Rightarrow_{G_i} \cdots \Rightarrow_{G_i} x_n = x (n \geq 0)$, there exists a string y such that $S \Rightarrow_{G_o^*} y$, $(x, y) \in \pi(T)$, and the nonterminals in y is a permutation of those in x . This is proved by induction on n .

Suppose first that $n = 0$. This means that $x = S$. Let y be $S \in N$. It clearly holds that $S \Rightarrow_{G_o^*} S$, $(S, S) \in \pi(T)$, and the nonterminal in S is a permutation of that in S .

Next suppose that for any string x_{n-1} such that $S \Rightarrow_{G_i^*} x_{n-1}$ where the length of the derivation is $n - 1$, there exists a string y_{n-1} such that $S \Rightarrow_{G_o^*} y_{n-1}$, $(x_{n-1}, y_{n-1}) \in \pi(T)$, and the nonterminals in x_{n-1} is a permutation of those in y_{n-1} . We consider derivations of length n . Suppose that $x_{n-1} \Rightarrow_{G_i} x_n$. Hence we can write x_{n-1} and x_n as follows: $x_{n-1} = \beta' A \beta''$, $x_n = \beta' \beta \beta''$, and $A \rightarrow \beta, \gamma \in R$. By the induction hypothesis, y_{n-1} contains an A associated with that in x_{n-1} . Using this A , we can write y_{n-1} as $y_{n-1} = \gamma' A \gamma''$. Let y_n be $\gamma' \gamma \gamma''$. Hence, $y_{n-1} \Rightarrow_{G_o} y_n$, showing that $S \Rightarrow_{G_o^*} y_n$. Since $(x_{n-1}, y_{n-1}) \in \pi(T)$, we also have $(x_n, y_n) \in \pi(T)$ by the definition of translation forms. Since nonterminals in y_{n-1} is a permutation of those in x_{n-1} and those in γ is a permutation of those in β , nonterminals in y_n is a permutation of those in x_n . Therefore

we have that for any string x_n such that $S \Rightarrow_{G_i^*} x_n$ where the length of the derivation is n , there exists a string y_n such that $S \Rightarrow_{G_o^*} y_n$, $(x_n, y_n) \in \pi(T)$, and the nonterminals in y_n is a permutation of those in x_n .

We have proved that for any string x such that $S \Rightarrow_{G_i^*} x$, there exists a string y such that $S \Rightarrow_{G_o^*} y$, $(x, y) \in \pi(T)$, and the nonterminals in y is a permutation of those in x . Since $x \in L(G_i)$, x has no nonterminal. Hence neither does y , concluding that $y \in L(G_o)$. Thus, we also have that $(x, y) \in \tau(T)$ because $(x, y) \in \pi(T)$.