ELECTRONIC PUBLISHING, VOL. 6(4), 373–383 (DECEMBER 1993)

Transformation of structured documents with the use of grammar

EILA KUIKKA
University of Kuopio
P. O. Box 1627
SF–70211 Kuopio, Finland

email: kuikka@cs.uku.fi

MARTTI PENTTONEN
University of Joensuu
P. O. Box 111
SF–80101 Joensuu, Finland

email: penttonen@cs.joensuu.fi

SUMMARY

In structured text processing systems the need for transformation of document instances is obvious if the structure definition of the document type changes. This article presents a transformation method with the use of an extended syntax-directed translation schema and its implementation to certain modifications in a syntax-directed document processing system created by the authors. The method uses grammars to define both the structure of documents and transformation between structures.

KEY WORDS Type transformations Structured documents Syntax-directed translation schema

1 INTRODUCTION

One of the difficulties encountered in structured document processing systems is how to transform the corresponding existing document instances whenever a user has to change the generic structure definition of these documents. So far only few studies and implementations concerning these *static structure transformations* of documents have been carried out. These modifications are employed in a range of documents with similar structure. In References [1,2], and [3] transformations can be considered as type transformations because they define a new form for elements of the structure for future processing of the document. In References [4,5], and [6] transformations group all the existing elements of the document in a new way and they add new elements to the structure for the use of the layout or view processing of this document. Another group of structure modifications are *dynamic transformations* that occur when the structured editing only changes a document instance, not its generic structure. They can be perceived as a structured editing of a single document.

The aim of our research is to develop a syntax-directed document processing system that uses grammars and their parse trees for inputting, updating, and outputting as well as storing and retrieving documents. The system is meant to be declarative in the sense that the user is asked only to define what he wants and does not have to know how to achieve it. Hence, the system should support the user as much as possible. In the research carried out so far as well as in the development of the whole system, the target has been to focus on the principle of the grammar-based processing as far as possible throughout the implementation without any *ad hoc* extensions. In the first part of the study, a prototype of a syntax-directed document processing system was designed and tested (see Reference [7]).

CCC 0894–3982/93/040373–11
©1993 by John Wiley & Sons, Ltd.
© 1998 by University of Nottingham.

Received 15 August 1993
Revised 1 December 1993

Its aim was to input and output structured documents with the use of grammars and a syntax-directed translation schema.

The present report describes the initial step in the study: how static structure modifications such as type transformations should be made in the syntax-directed document processing system mentioned in the previous paragraph. On the basis of the examination of a range of documents in a variety of environments, for example articles in different journals and letters in sample offices, it has been observed that the following modifications of document instances are most commonly found: reordering of elements, adding a new element, deleting or renaming an element, and changing features of elements. More complex transformations, such as relocating a nonterminal in a parse tree or adding a layer to and removing a layer from a parse tree are also required as well as grouping elements in a new way. The methods based on pair grammars and rewrite rules (see Reference [1]), coordinate grammars (see References [4] and [5]), attribute grammars (see Reference [6]) or tree operations (see Reference [3]) have been used for complex modifications. This report is restricted only to the simpler modifications; the implementation of more complex transformations is the next step in our research.

2 SYNDOC: SYNTAX-DIRECTED DOCUMENT PROCESSING SYSTEM

The prototype of the syntax-directed document processing system, called SYNDOC and created by the authors, uses *modified context-free grammars* similar to grammars in Reference [8] to define the hierarchical logical structure of a class of structured documents. Nonterminals of the grammar represent the elements of the document, whereas terminals comprise the content of the document. In addition to BNF of the context-free grammar, it is possible to define whether an optional element in the structure appears once or not at all and also to define two kinds of iterations. In one of the iterations, elements appear zero or more times while in the other one or more times. Grouping of elements as in grammars of Reference [8] is possible with the use of an additional nonterminal.

The processing of the document is implemented with the use of a *syntax-directed translation schema*. A grammar, the input grammar in this case, is used in the input phase of the text to expand incrementally the parse tree of the document. Another grammar, the output grammar, is used to generate the layout of the document. These two grammars are coordinated. They have the same nonterminals arranged in the same order. Many output grammars can be defined for one input grammar, which makes it possible to get different kinds of output of the same document.

At present, the internal as well as the external representation of the document is the parse tree for the input grammar. Because the implementation of SYNDOC is made in Prolog, the parse tree of the document is represented by a Prolog term whose nesting structure describes the hierarchy of the document structure.

3 DEFINITION OF THE STRUCTURE TRANSFORMATION

The aim of the study reported in this article was to implement static structure transformations with the use of the idea of compiling with a syntax-directed translation schema (Figure 1). First, the old and new input grammars are compared to form a translation grammar. The process would need the user's interruption only when the user renames nonterminals. The order of rules of the translation grammar is coordinated by the old input grammar which

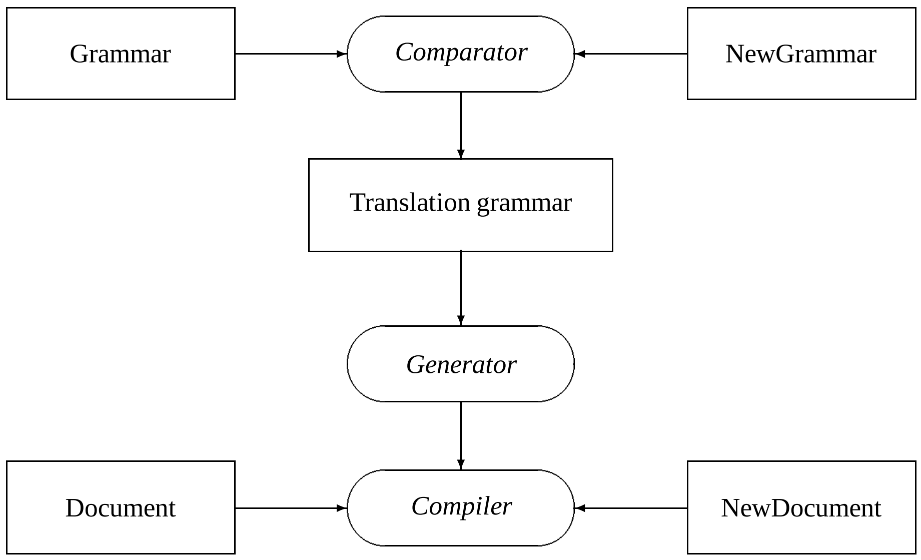


Figure 1. The architecture of the modification system

helps to avoid the problems of nondeterminism in parsing. Second, the compiler which is to perform the actual transformation is generated from the translation grammar. Third, the actual transformations of all the similar types of documents are made by the compiler.

The input grammar is a modified context-free grammar as was mentioned in Section 2. The translation grammar is an *extended syntax-directed translation schema* (ESDTS). The extension has been made by changing the notation defined in Reference [9] so that on the left side of the ESDTS rule there are two nonterminals, the names of the nonterminals on the left sides of the new and old grammars, separated by a semicolon and at the end of the grammar rule there can be semantic actions like in attribute grammars [10] to provide the renaming of nonterminals. Semantic actions are given in braces. On the right side of ESDTS rule there are the right sides of the new and old grammars separated by a semicolon. If any rule in either the old or new grammar has no corresponding rule in the other grammar, the missing part placed on the right side of ESDTS rule is marked as an empty string by "".

In order to clarify the above description let us consider the following example. The original input grammar rules for a letter with the use of our notation are:

```
letter -> receiver content sender.
receiver -> text.
content -> text.
sender -> text.
```

and the new input grammar rules are:

```
letter -> sender receiver body sign.
sender -> text.
receiver -> text.
body -> text.
sign -> text.
```

The transformation is meant to change the place of the nonterminal `sender`, rename `content` to `body` and add a nonterminal `sign` as a new element to the letter. The ESDTS rules for the transformation of letter instances are as follows:

```
      letter ; letter -> receiver content sender ; sender receiver body
sign { content = body }. sender ; sender -> text ; text. receiver ;
receiver -> text ; text. content ; body -> text ; text. sign ; sign -> "" ;
text.
```

From the ESDTS grammar we generate definite clause grammar (DCG) rules (defined, for example, in Reference [11]) that form our compiler program. The following DCG is generated from the above ESDTS.

```
      letter(letter(A,B,C),letter(D,E,F,G)) -->
sender(C,D),receiver(A,E),body(B,F),sign(H,G). sender(sender(A),sender(B))
--> text(A,B). receiver(receiver(A),receiver(B)) --> text(A,B).
body(content(A),body(B)) --> text(A,B). sign(sign(A),sign(B))
--> {B = sign}. text(text(A),text(B)) --> {A = B}.
```

Let us consider, for example, the first DCG rule. The nonterminal `letter` on the left side of the DCG rule has two parse tree arguments, namely `letter(A,B,C)`, and `letter(D,E,F,G)`, the former one for the old parse tree and the latter one for the new parse tree. These parse tree arguments have arguments A, B, and C and further D, E, F, and G. The arguments A, B, and C correspond to the nonterminals on the right side of the rule in the original input grammar and the arguments D, E, F, and G to the nonterminals on the right side of the rule in the new input grammar in the order defined in the original grammar rule. The old input grammar rule of this example comprises three nonterminals (`receiver`, `content` and `sender`) and the new input grammar rule four nonterminals (`sender`, `receiver`, `body` and `sign`). Thus, the argument A corresponds to the nonterminal `receiver`, B corresponds to `content`, and so on, whereas the argument D corresponds to the nonterminal `sender`, E corresponds to `receiver`, and so on. The nonterminals on the right side of the DCG rule correspond to the nonterminals of the right hand side in the new input grammar (`sender`, `receiver`, `body` and `sign`), and further their arguments are partly the same as arguments of the parse tree arguments on the left side of the DCG rule. A pair of arguments on the right side of the DCG rule informs which of the nonterminals of the old and new input grammars correspond to each other. For example, a pair C, D indicates that the third argument of the old parse tree corresponds to the first argument of the new parse tree and produces the reordering of the `sender` nonterminal. If an argument does not have a corresponding argument on the other side of the DCG rule, for example H, then the other argument of the pair corresponds to a nonterminal that is either added, like `sign` in this example, or removed.

If there are optional nonterminals or list of identical nonterminals in the input grammars, extra rules are generated into the DCG to overtake their processing as can be seen in the example given in the next section.

4 THE TRANSFORMATION OF A SIMPLE ARTICLE INSTANCE

In this section a more difficult example to transform a simple article from one structure to another will be demonstrated. The original grammar rules for a simple article could be as follows:

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

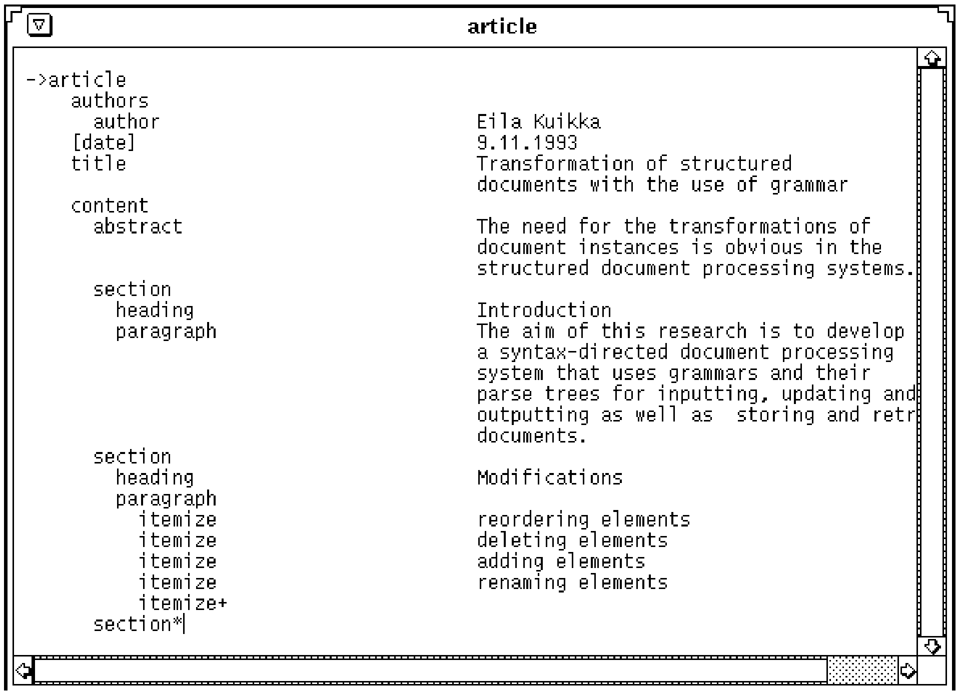
In the hierarchical representation of this sample article, `article`, the root element of the document, has such components as `authors`, `date`, `title`, and `content`. The `date` is optional and is placed in brackets. The element `authors` contains a component `author` and the element `content` comprises components `abstract` and a list of none or more components called `section`. Further, the element `section` contains components `heading` and `paragraph`. The element `paragraph` can be a component `text` or a list of one or more components called `itemize`. The component `text` implicitly is a character list, the actual content of an element. Figure 2 represents an article instance according to the above grammar displayed in the document window on the screen of SYNDOC system. The left hand side of the screen informs about the structure of the document, the right hand side indicates the contents of the corresponding elements of the document.

The parse tree (without full contents) of this document instance is the following Prolog term:

```
article(authors(author(text(‘Eila...’)),
        ’[date]’(text(‘9.11..’)),
        title(text(‘Transformations...’)),
        content(abstract(text(‘The need...’)),
               [section(heading(text(‘Introduction’)),
                        paragraph(text(‘The aim...’))),
                section(heading(text(’Modifications’)),
                        paragraph([itemize(text(’reordering...’)),
                                   itemize(text(’deleting...’)),
                                   itemize(text(’adding...’)),
                                   itemize(text(’renaming...’)),
                                   ’itemize+’])),
               ’section*’])).
```

The lists are represented as Prolog lists. The new grammar rules could be as follows:

```
     article -> title writers date body bibliography. writers ->
(author)+. author -> text. date -> text. title -> text. body -> abstract
(section)+ acknowledgement. abstract -> text. section -> [heading]
(paragraph)+. acknowledgement -> text. heading -> text. paragraph -> text.
paragraph -> (enumerate)+. itemize -> text. enumerate -> text.
bibliography -> (item)+. item -> text.
```



```
->article
    authors
      author                            Eila Kuikka
    [date]                              9.11.1993
    title                               Transformation of structured
                                        documents with the use of grammar
    content
      abstract                          The need for the transformations of
                                        document instances is obvious in the
                                        structured document processing systems.
      section
        heading                         Introduction
        paragraph                       The aim of this research is to develop
                                        a syntax-directed document processing
                                        system that uses grammars and their
                                        parse trees for inputting, updating and
                                        outputting as well as  storing and retr
                                        documents.
      section
        heading                         Modifications
        paragraph
          itemize                       reordering elements
          itemize                       deleting elements
          itemize                       adding elements
          itemize                       renaming elements
          itemize+
      section*
```

Figure 2. A document instance of a simple article

There are many differences in the grammars represented above. The transformation is supposed to perform the following actions:

- change the place of the nonterminal `title`,
- rename and and change the place of the nonterminal `authors`, the new name being `writers`,
- rename the nonterminal `content` with the new name `body`,
- change the optional nonterminal `date` to a compulsory one,
- add a new nonterminal `bibliography` to the article (this nonterminal is a list of one or more `item` nonterminals that contain the text of the item),
- change the single nonterminal `author` to a list of one or more nonterminals `author`,
- change a list of null or more `section` elements in such a way that at least one element has to be on the list,
- add a new nonterminal `acknowledgement` to the body of the article,
- change the compulsory nonterminal `heading` of the section to an optional one,
- change the nonterminal `paragraph` to a list of one or more `paragraph` nonterminals, and
- change a list of one or more `itemize` nonterminals to a list of one or more `enumerate` nonterminals.

The ESDTS rules for the transformation of this simple article instance are as follows:

```
article ; article -> authors [date] title content ;
                     title writers date body bibliography
                       {authors = writers, content = body }.
authors ; writers -> author ; (author)*.
author ; author -> text ; text.
date ; date -> text ; text.
title ; title -> text ; text.
content ; body -> abstract (section)* ;
                  abstract (section)+ acknowledgement.
abstract ; abstract -> text ; text.
section ; section -> heading paragraph ;
                     [heading] (paragraph)+.
acknowledgement ; acknowledgement -> "" ; text.
heading ; heading  -> text ; text.
paragraph ; paragraph -> text ; text.
paragraph ; paragraph -> (itemize)+ ; (enumerate)+
                           {itemize = enumerate }.
itemize ; enumerate -> text ; text.
bibliography ; bibliography -> "" ; (item)+.
item ; item -> "" ; text.
```

From the ESDTS rule DCG rules are generated and they form the compiler program for all document instances of type similar to the document in Figure 2. The whole compiler program is represented in the Appendix.

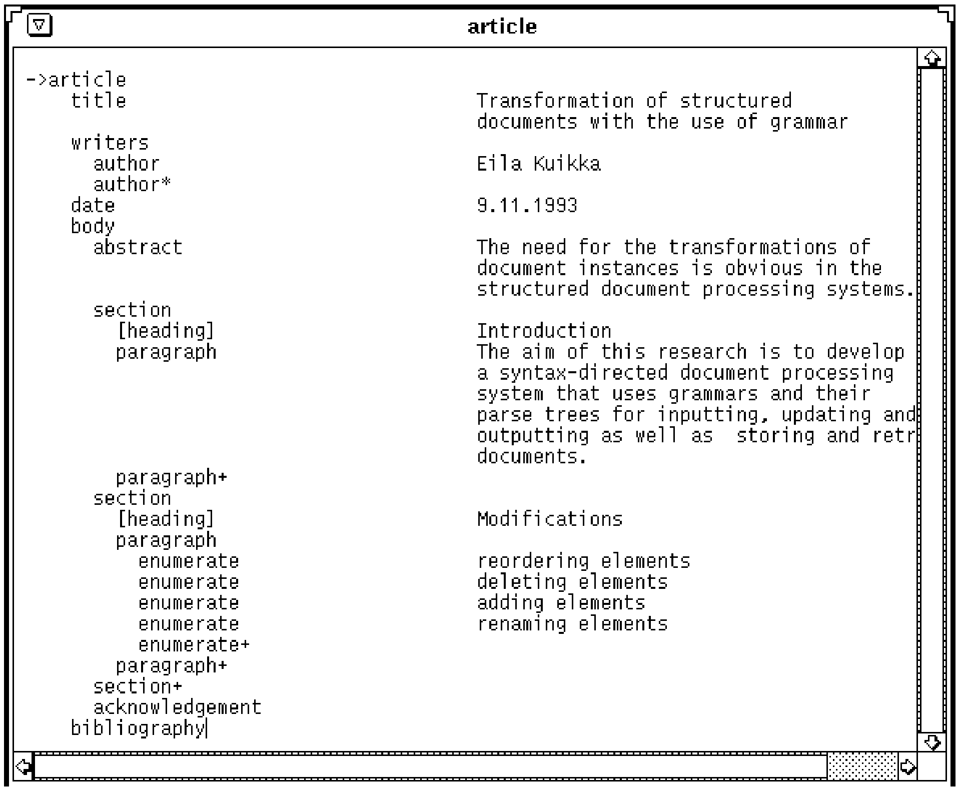
To illustrate the processing of optional and list elements let us consider two examples. First, as it can be seen in the Appendix, from the rule:

```
article ; article -> authors [date] title content ;
                     title writers date body bibliography
                     { authors = writers , content = body }.
```

the following three DCG rules are generated:

```
article(article,article)-->!.
date('[date]'(_27173),date(_27171))-->
    date(date(_27173),date(_27171)).
article(article(_16548,_16614,_16729,_16721),
       article(_16835,_16898,_16961,_17024,_17016))-->
    title(_16729,_16835),writers(_16548,_16898),
    date(_16614,_16961),body(_16721,_17024),
    bibliography(_26808,_17016).
```

The first rule is responsible for transforming an `article` element without subtrees in the parse tree. The second one changes the `date` element from an optional element to a compulsory one. The third one is responsible for the transformation of the `article` element with subtrees.



```
->article
    title                              Transformation of structured
                                       documents with the use of grammar
    writers
      author                           Eila Kuikka
      author*
    date                               9.11.1993
    body
      abstract                         The need for the transformations of
                                       document instances is obvious in the
                                       structured document processing systems.
      section
        [heading]                      Introduction
        paragraph                      The aim of this research is to develop
                                       a syntax-directed document processing
                                       system that uses grammars and their
                                       parse trees for inputting, updating and
                                       outputting as well as  storing and retr
                                       documents.
        paragraph+
      section
        [heading]                      Modifications
        paragraph
          enumerate                    reordering elements
          enumerate                    deleting elements
          enumerate                    adding elements
          enumerate                    renaming elements
          enumerate+
        paragraph+
      section+
      acknowledgement
    bibliography
```

Figure 3. A document instance of a simple article according to the new structure

Second, it can also be seen in the Appendix that from the rule:

```
content ; body -> abstract (section)* ;
                  abstract (section)+ acknowledgement.
```

the following five DCG rules are generated:

```
body(content,body)-->!.
'sectionlist+'(['section*'],[section,'section+'])-->!.
'sectionlist+'([_17430,'section*'],[_17426,'section+'])-->
     section(_17430,_17426).
'sectionlist+'([_17536|_17537],[_17534|_17535])-->
     section(_17536,_17534),'sectionlist+'(_17537,_17535).
body(content(_11738,_11730),body(_11917,_11980,_11972))-->
     abstract(_11738,_11917),'sectionlist+'(_11730,_11980),
    acknowledgement(_16950,_11972).
```

The first rule transforms a `content` element without subtrees. The second, third and fourth ones are responsible for the processing of a list of sections with the use of the name `sectionlist+` and at the same for changing the type of the list. The fifth one processes the old `content` element and changes the name to `body`. On the right side of the last rule, the whole list of sections corresponds to a nonterminal `sectionlist+`.

In order to produce the modified article the parse tree of the document instance in Figure 2 has been processed with the program represented in the Appendix. Figure 3 represents this new article on the screen of the SYNDOC system. The old content of the article is represented with the use of the new required structure. All the modifications have been done. The program has generated elements without content for the new elements `acknowledgement` and `bibliography`. Their contents can be added later.

5 DISCUSSION

With the above simple example it has been demonstrated how the transformation of a document instance to a new structure is made with the use of our method. From the user's point of view, the method is easy to use and understand as part of the SYNDOC system. The user is supposed to define structures in other parts of the system, thus, it is natural that (s)he is able to use the same method also in the structure transformations. At present the grammar used for the transformation is generated manually, but the aim is to implement a user-friendly comparator program.

The modifications have been done in the parse tree among the children of a parent node. Although this is enough in most practical situations, the modifications between different layers of the parse tree are also needed. The ESDTS as such does not allow this possibility. Some kind of new elements are needed for these operations. The implementation of these complex transformations will be a future part of the research.

Transformations needed in the usual document processing are probably not much more complicated than those described in this report. However, the basic fact about transformations is that they as theirselvesas such do not add information. Sometimes, for example, it is necessary to enrich the structure of the text. In the model presented above, this should be done in two stages: first, by refining the generic structure of the original document manually or with the help of a special tool, and then by transforming the document as described in this report. The process of refining involves increasing structural information of the document and therefore requires intelligence. Transforming, on the other hand, increases entropy (in physical terms) and can be automated.

REFERENCES

1. R. Furuta and P.D. Stotts, ‘Specifying Structured Document Transformations’, in *Proceedings of the International Conference on Electronic Publishing, Document Manipulation and Typography (EP88),* ed., J.C. van Vliet, 109–120, Cambridge University Press, Cambridge, UK, (1988).
2. E. Akpotsui and V. Quint, ‘Type Transformation in Structured Editing Systems’, in *Proceedings of Electronic Publishing, 1992 (EP92),* eds., C. Vanoirbeek and G. Coray, 27–41, Cambridge University Press, Cambridge, UK, (1992).
3. E. Blake, T. Bray, and F.W.M. Tompa, ‘Shortening the OED: Experience with a Grammar-Defined Database’, *ACM Transactions on Information Systems*, **10**(3), 213–232, (1992).
4. A.L. Brown and H.A. Blair, ‘A Logic Grammar Foundation for Document Representation and Document Layout’, in *Proceedings of the International Conference on Electronic Publishing, Document Manipulation, and Typography (EP90),* ed., R. Furuta, 47–64, Cambridge University Press, Cambridge, UK, (1990).
5. A.L. Brown, T. Wakayama, and H.A. Blair, ‘A Reconstruction of Context-Dependent Document Processing in SGML’, in *Proceedings of Electronic Publishing, 1992 (EP92),* eds., C. Vanoirbeek and G. Coray, 1–25, Cambridge University Press, Cambridge, UK, (1992).

6. P. Franchi-Zannettacci and D.S. Arnon, ‘Context-Sensitive Semantics as a Basis for Processing Structured Documents’, in *Proceedings of the Workshop on Object-Oriented Document Manipulation (WOODMAN’89)*, 135–146, (1989).
7. E. Kuikka and M. Penttonen, ‘Designing a Syntax-Directed Text Processing System’, in *Proceedings of the Second Symposium on Programming Languages and Software Tools*, eds., K. Koskimies and K.-J. Räihä, 191–204, University of Tampere, Department of Computer Science, A-1991-5, Tampere, Finland, (1991).
8. R. Furuta, ‘A Grammar for Representing Documents’, Technical Report UMIACS–TR–87–67, University of Maryland, Maryland, (1987).
9. A.V. Aho and J.D. Ullman, *The Theory of Parsing, Translation, and Compiling*, volume 1, 2, Prentice–Hall, Englewood Cliffs, NJ, 1972.
10. *Attribute Grammars, Applications and Systems*, eds., H. Alblas and B. Melichar, Lecture Notes in Computer Science, No. 545, Springer, Berlin, 1991.
11. F.C.N. Pereira and D.H.C. Warren, ‘Definite Clause Grammars for Language Analysis – A Survey of the Formalism and a Comparison with Augmented Transition Networks’, *Artificial Intelligence*, **13**, 231–278, (1980).

APPENDIX A: THE COMPILER PROGRAM OF A SIMPLE ARTICLE

```
article(article,article)-->!.
date('[date]'(_27173),date(_27171))-->
  date(date(_27173),date(_27171)).
article(article(_16548,_16614,_16729,_16721),
     article(_16835,_16898,_16961,_17024,_17016))-->
  title(_16729,_16835),writers(_16548,_16898),
  date(_16614,_16961),body(_16721,_17024),
  bibliography(_26808,_17016).
writers(authors,writers)-->!.
'authorlist*'(['author*'],['author*'])-->!.
'authorlist*'([_10960|_10961],[_10958|_10959])-->
  author(_10960,_10958),'authorlist*'(_10961,_10959).
writers(authors(_8102),writers(_8087))-->
  'authorlist*'([_8102,'author*'],_8087).
author(author,author)-->!.
author(author(_7284),author(_7269))-->text(_7284,_7269).
date(date,date)-->!.
date(date(_6856),date(_6841))-->text(_6856,_6841).
title(title,title)-->!.
title(title(_7070),title(_7055))-->text(_7070,_7055).
body(content,body)-->!.
'sectionlist+'(['section*'],[section,'section+'])-->!.
'sectionlist+'([_17430,'section*'],[_17426,'section+'])-->
  section(_17430,_17426).
'sectionlist+'([_17536|_17537],[_17534|_17535])-->
  section(_17536,_17534),'sectionlist+'(_17537,_17535).
body(content(_11738,_11730),body(_11917,_11980,_11972))-->
  abstract(_11738,_11917),'sectionlist+'(_11730,_11980),
  acknowledgement(_16950,_11972).
abstract(abstract,abstract)-->!.
abstract(abstract(_7712),abstract(_7697))-->text(_7712,_7697).
```

```
section(section,section)-->!.
'paragraphlist+'([_15859,'paragraph+'],[_15855,'paragraph+'])-->
  paragraph(_15859,_15855).
'paragraphlist+'([_15965|_15966],[_15963|_15964])-->
  paragraph(_15965,_15963),'paragraphlist+'(_15966,_15964).
'[heading]'(heading(_16100),'[heading]'(_16098))-->
  heading(heading(_16100),heading(_16098)).
section(section(_10568,_10560),section(_10674,_10666))-->
  '[heading]'(_10568,_10674),
  'paragraphlist+'([_10560,'paragraph+'],_10666).
acknowledgement(acknowledgement,acknowledgement)-->!.
acknowledgement(_8782,_8780)-->{_8780=acknowledgement}.
heading(heading,heading)-->!.
heading(heading(_7532),heading(_7517))-->text(_7532,_7517).
paragraph(paragraph,paragraph)-->!.
paragraph(paragraph(_7926),paragraph(_7911))-->text(_7926,_7911).
paragraph(paragraph,paragraph)-->!.
'enumeratelist+'([_14435,'itemize+'],[_14431,'enumerate+'])-->
  enumerate(_14435,_14431).
'enumeratelist+'([_14541|_14542],[_14539|_14540])-->
  enumerate(_14541,_14539),'enumeratelist+'(_14542,_14540).
paragraph(paragraph(_11215),paragraph(_11200))-->
  'enumeratelist+'(_11215,_11200).
enumerate(itemize,enumerate)-->!.
enumerate(itemize(_7712),enumerate(_7697))-->text(_7712,_7697).
bibliography(bibliography,bibliography)-->!.
bibliography(_8312,_8310)-->{_8310=bibliography}.
item(item,item)-->!.
item(_6428,_6426)-->{_6426=item}.
text(text(_4410),text(_4400))-->{_4410=_4400}.
```