Transformation of structured documents

Eila Kuikka*
Department of Computer Science, University of Waterloo, Canada
email: ekuikka@watsol.uwaterloo.ca

Martti Penttonen
Department of Computer Science, University of Joensuu, Finland
email: penttonen@cs.joensuu.fi

October, 1995
Technical Report CS-95-46

Abstract

Structure definitions of documents have been used successfully for inputting and formatting in text processing systems. This report considers transformations between different representations of structured documents and studies possibilities to extend the use of structure definitions to document transformations and to discover algorithmic methods for carrying out transformations. Documents are presented as parse trees for context-free grammars and transformations are made from parse tree to parse tree. First, the report describes differences of manuscript styles required by various scientific journals and presents a declarative classification for structure differences between two parse trees. Second, a set of tree transformation methods are described and their suitability for transformations between documents having a structure difference in each defined class is analyzed. For each class several methods may or must be used and only certain kinds of differences can be managed automatically. Finally, instead of designing a system where a method accommodates for all kinds of differences or where different methods are used in various transformations, the report presents a model for a document transformation system that presents a possibility of using various methods according to differences in document representations. The system is divided two modules. In the first one transformations are made automatically and they do not change the hierarchical structure of a document. In the second one transformations are made semiautomatically or nonautomatically and the hierarchical structure changes. Differences between the existing and the required representation of a document are analyzed and methods selected according to the classified differences.

*Author's permanent address: University of Kuopio, Department of Computer Science and Applied Mathematics, P.O.Box 1627, Kuopio, Finland, email: kuikka@cs.uku.fi

1 Introduction

We consider electronic document processing as transformations between different representations. We do not mean only those which can be seen on the computer screen or on paper. Other representations are, for example, many external representations for saving the document to a file on a disk or CD-ROM or for exchanging documents between different environments (perhaps using some standard coding methods like SGML [ISO86b, Gol90], ODA [ISO86a, FFK92], or HTML [BLCLS94]). From our point of view, documents are structured and the most important representation is the representation that contains only the actual content of the document in the structured form, for example, as a tree. We consider this representation as an internal representation of a document as an opposite to external representations which are generated from it and used by the user of the system. The internal representation can be a single unit or it can be created by collecting its parts from a set of documents.

A document processing system must work between all these different representations. In our syntax-directed paradigm for structured document processing [Kui90] transformations between different document representations are the basis of the system. But instead of trying to build special software for each generated framework we would like to find some typical differences that occur in representations and find algorithmic methods to carry out transformations between representations. Each representation of a structured document consists of three types of information, the content, the structure and the layout. When documents are transformed from one representation to another, some of these or all of them will change. In the transformation, the information of a document will either stay unchanged, decrease or increase. We shall respect the fact that information cannot be created from nothing. If the transformation decreases or preserves information then it should be eligible for automatization. If the transformation adds information some additional input from the user is needed.

As a motivation for the coming discussion of transformation methods for structured documents we will consider manuscript styles of an article in eight scientific journals in the field of Computer Science. We want to demostrate structure differences in a situation when authors have written documents using a specific representation which they would later need in another form. We will deal with an article in scientific journals because journals usually give quite strict rules about the structure and format of a manuscript and suppose or require that the authors will follow them. To avoid the modification work the ideal situation would be if the authors would know which format to use when they start to write a manuscript. Because this is only seldom possible it is interesting to know what kinds of differences exist and how the differences in formats could be managed. We selected a manuscript for an article, but we are sure that the situation is common for other types of documents, such as letters, manuals, books, etc. A competitive example would also be the same text in different forms, as a report, as an article, as a chapter of a book or as an online document. However, it is not necessary to discuss all these examples because we want to find methods for classes of differences not for all detailed differences in structures documents.

There are many systems and languages [FS88, AQ92, AQR93, RA94, MKNS87, Kae87, BJB90, BJWB92, FW93, CP89, CC93, Arn93, DIS94, Ins92, Pla93, KN94] that are able to execute transformations between structured documents. These systems use many meth-

ods and techniques very often with some kinds of extensions specific for documents. They are usually suitable only for some types of transformations or can be applied only to documents which use a particular method to define the structure. Instead of using ad-hoc implementations for all different types of transformations, we will study the suitability of tree transformation methods to manage various kinds of differences between tree-structured documents.

Trees are very common data structures in many applications and techniques for tree transformations are found in theories for graphs, trees and terms and their languages as well as in the research of tree pattern matching/replacement methods in many application areas. For grammar-based transformations, compilation theory for formal languages offers a set of well-defined algorithmic techniques that can be used to process translations in parsed trees [AU72, AU73]. Translations are defined with the use of paired grammar productions and made production by production. Transformations one structure after another can be implemented by tree transducers [Rou70, Tha73, Bak78b, GS84, NP92], by term rewriting systems [Klo92, SPvE93] and by methods and languages based on a tree pattern matching and replacement [KPPM84, HO82, SF83, Kil92, KM93, CCB95, NBY95, BCD+95, JK95]. The user will define the transformation and the method will direct the transformation according the definitions. In manual methods like tree editors [Des88] the definition and the control are directed by the user and translations are made one element after another.

The aim of this work it to find tree transformation methods

1. whose definition for a translation can be made either automatically from the structure definitions for documents or at least interactively with the help of the user;

2. whose translator can be generated automatically or at least semiautomatically from the definition of a translation;

3. which are applied to parse trees for a grammar and produce parse trees for another grammar accompanied by its grammar, (thus, they are not applied to string representations of documents); and

4. which are semi-deterministic, which means that they process the translation automatically or at least interactively with the help of the user to locate the modified nodes in a tree.

The rest of the report is organized as follows. After defining some concepts used throughout the report in the second section, we present a brief survey to related works. In the fourth section we analyse what kinds of different representations would be needed when a document exists in different environments. We have selected a set of scientific journals in Computer Science and analyzed differences in representations for a manuscript submitted to these journals. Rather than merely listing differences in documents the fifth section presents a declarative classification for the differences in parse trees. Parse trees are used as internal representations for tree-structured documents. The sixth section defines and describes a set of tree translation methods. After that, next three sections analyze the suitability of different methods for document transformations, present a model for a flexible transformation, and briefly describe the implementation of parts of such a transformation system in our syntax-directed document processing system. The last section concludes the report and outlines our future research.

2 Definitions

The following definitions are mainly taken from [AU72, Woo87].

A *context-free grammar* [AU72] is a 4-tuple $G = (\mathrm{N}, \Sigma, P, S)$ where N is a finite set of *nonterminals*, Σ is a finite set of *terminals*, disjoint from N, P is a finite subset of $\mathrm{N} \times (\mathrm{N} \cup \Sigma)^*$, and S is a distinguished symbol in N called a *start symbol*. An element (A, α) in P will be written $A \to \alpha$ and called a *production*. It is a common notation to define the grammar only by giving its productions.

The *derivation relation* used to define a language of a context-free grammar is as follows: Let G = (N, Σ, P, S) be a context-free grammar. A relation $\Rightarrow$ (to be read as *directly derives*) on $(\mathrm{N} \cup \Sigma)^*$ is defined as follows: If $\alpha A \beta$ is a string in $(\mathrm{N} \cup \Sigma)^*$, $A \in$ N and $A \to \gamma$ is a production in P, then $\alpha A \beta \Rightarrow \alpha \gamma \beta$. Strings $\alpha A \beta$ and $\alpha \gamma \beta$ are called *sentential forms* of the grammar G. The start symbol S is also a sentential form. A sequence $S = \alpha_0, \ldots, \alpha_n$, where $\alpha_i \Rightarrow \alpha_{i+1}, (i = 0, \ldots, n-1)$ is a *derivation* of α_n from S in the grammar G, denoted by $S \stackrel{*}{\Rightarrow} \alpha_n$. If a derivation is applied to the leftmost nonterminal of a sentential form in every step, then the derivation is said to be a *leftmost derivation*. Correspondingly, if the nonterminal is selected to be the rightmost nonterminal of a sentential form then the derivation is a *rightmost derivation*.

Let A be a finite set of nodes, R a binary relation on A. An *unordered directed graph* G is a pair (A, R). A pair $(a, b) \in R$ is called an *edge* of G, and it is said to leave node a and enter node b. Node a is a *direct predecessor* of node b and node b is a *direct successor* of node a. A node a_n is said to be *accessible* from node a_0 if there is a sequence of nodes $a_0, a_1, \ldots, a_n$, $n \geq 0$, called a *path* of length n from node a_0 to node a_n so that there is an edge which leaves node a_{i-1} and enters node a_i for $1 \leq i \leq n$. For an empty path $n = 0$.

A *tree* T is an ordered, directed graph $G = (A, R)$ with a specified node r in A called the *root* such that r has no predecessor, all other nodes of T has one predecessor and are accessible from r. Since each node except the root in tree has exactly one predecessor, a node has a node called a *parent* and an ordered (from left to right) sequence of nodes called *children*. Similarly a child node has *siblings* to its left and right. The *depth of a node* in a tree is the length of the path from the root to that node. The *depth of the tree* is the length of the longest path.

A labelled ordered tree D is a *derivation tree or parse tree* for a context-free grammar $G(S) = (\mathrm{N}, \Sigma, P, S)$ if

1. The root of D is labelled S.

2. If $D_1, \ldots, D_k$ are subtrees of the direct successor of the root and the root of D_i is labelled X_i, then $S \to X_1 \ldots X_k$ is a production in P. D_i must be a derivation tree for $G(X_i) = (\mathrm{N}, \Sigma, P, X_i)$ if X_i is a nonterminal, and D_i is a single node labelled X_i if X_i is a terminal.

3. Alternatively, if D_1 is the only subtree of the root of D and the root D_1 is labelled ϵ, where ϵ is an empty string, then $S \to \epsilon$ is a production in P.

The derivation tree defines a graphical representation of an equivalence class of derivations for context-free grammars. For a derivation $S = \alpha_0, \ldots, \alpha_n$ of α_n in a context-free grammar $G = (\mathrm{N}, \Sigma, P, S)$ there is a derivation tree D for G such that D has *a frontier* α_n.

For a derivation tree D for a context-free grammar $G = (\mathrm{N}, \Sigma, P, S)$ with frontier α there is a derivation of α from S.

3 Related works

Before structure transformations have been studied concerning documents, the same problem has been dealt with other programming tasks, for example, in syntax-directed programming environments and in databases. Before describing systems or methods for structured documents we briefly mention some examples from other areas, too.

3.1 Structure-oriented environments

Structure-oriented programming environments, such as supported by Gandalf [HN86] and the Synthesizer Generator [Rep89], represent their generated programmes as parse trees. If grammars which define these programmes will change also these trees should be changed. Thus, the situation is same as in document transformations. Syntax-directed methods which will be described in this work as well as methods based on attribute grammars [Bar93] can be used for transformations. They require to define the relationship between input and output grammars. However, TransformGen [GKL94], a system used in Gandalf to generate automatic converters for abstract syntax trees, uses another approach. In TransformGen the user changes the grammar to a new one using commands which generate rules forming the converter for trees.

3.2 Databases

Transformations in databases are caused from two factors: changes in users' functional requirements and changes in their performance requirements. The conversion must process two kinds of modifications: translation of the data and translation of programmes that process the data. A survey to conversion technology applied to database transformations in 1970s in presented in [FE81]. The article represents a model where data transformations are made using an intermedia form which can be produced from different representations and from which different representations can be generated. In applications the form was a file with records having fields and transformations were made batch-oriented. Data conversions are needed especially in object-oriented databases which consist of objects being instances of some type. A type definition consists of a set of operations, a set of properties and a set of constraints. Types are related to each other by an *is-a* property. The structure of the database defined using this property may be represented as a directed acyclic graph with exactly one node containing no outgoing edges. Two different approaches for type conversions may be used: either to transform the objects according to the new structure or to support version control mechanism for types. The conversion approach was used in [ABB$^+$84, BKKK87, PS87]. The methods represent mappings to compute an object of one type version to another, rules for keeping the type graph consistent and for converting existing objects to correspond to the new graph, or a set of operations to be applied to objects and types whenever structure differences exist. The versioning approach is used in

[Bor85, SZ87]. Methods associate a set of error handlers with the different versions of a type, either focusing on instance-level variations or on type-level changes.

3.3 Structured documents

The methods for document transformations are usually suitable for a specific definition of a document representation and for a specific process, and either they are grammar-based (i.e. a transformation definition is a grammar) or pattern matching -based (i.e. patterns define document instances that are transformed).

If a document representation is defined by a context-free grammar Furuta and Stotts [FS88] describe a method where a transformation is defined by a grammar transformation. Using a set of commands like described above for TransformGen, the user defines steps how the grammar can be rewritten to a new grammar and these steps define the transformation grammar which is used in document transformations. In the structured document processing system Grif [AQ92, AQR93, RA94] the structure is defined by a context-free grammar using a special language. Transformations are achieved by comparing grammars to produce the transformation definition. Document transformations according to these definitions are made in static transformations using additional renaming and relocating rules and in dynamic transformations automatically requiring embedded matching structures but no identical labels. Integrated Chameleon Architecture (ICA) [MKNS87, Kae87, MKNS89, MBO93, GM93, MOB94] is a software environment for automatic generation of translators among certain text formatters. Transformations are made by parsing a document string representation according to the old grammar, by transforming an intermediate tree to a tree structure suitable for the new form of the document and by generating a new string representation according to the new grammar. Structure transformations allow to reorder children of a node in a tree. The parsing is made with the user's help if needed.

For documents whose structure is defined by an attribute grammar [Bar93], the method to generate layout representation for a formatter, developed by Brown et.al. [BJB90, BJWB92], defines structure transformations using Definite Clause Grammar rules [PW80] for the element coordination and attribute grammars for the parsing. Similarly, in SIMON [FW93], a system for restructuring documents, a transformation is defined by an extended attribute grammar and externally defined programmes and the document transformation is made by a transformation engine of the system. SIMON suits document assembly, view generation, document retrieval and document type evolution and it supports composition of transformations.

Rest of the examples are based on the tree pattern matching and replacement. Scrimshaw [Arn93] is a language which uses hierarchical regular expressions and variables in patterns and "try-again" paradigm for transformation of documents which are represented as terms using a specific notation. For only SGML documents, specific languages for the transformation and formatting are defined in a standard for Document Style Semantics and Specification Language (DSSSL) [DIS94]. These languages can be used to make filters for a single document, or for a set of similar type of documents. A transformation is defined by a set of transformation specifications and made by generating a new structure into an output tree for every node of the input tree. The input tree is processed in the preorder. An example about a specific transformation is the generation of the shorter form from the

whole Oxford English Dictionary (OED). The transformation was written in the GOEDEL language [BBT92] to execute operations in so called p-strings (parsed strings) [GT87]. TXL language [CP89, CC93] is designed especially for transformations of Turing type programming languages, but can be used also to transform trees which represent documents. Every transformation programme is defined for its own context-free grammar and in addition to context-free transformations, context-sensitive transformations can be executed. The string representation of a document is parsed, then the parse tree is transformed and after that a new string representation is generated.

Transformations between a pair of particular representations of formatters and/or text processing systems can be made using filter programmes such as those generated using qwertz/FORMAT [Ins92] to transform SGML documents to LaTeX or troff documents, or many specific filters, for example, from FrameMaker documents to SGML documents, from Word documents to LaTeX documents, from WordPerfect documents to FrameMaker documents, etc. A set of commercial products for transforming structured documents have been listed in [KN94].

4 Example: Manuscript styles in scientific journals

To study differences in document representations we chose the following journals in Computer Science: ACM Transactions of Information Systems (ACMTOIS); Journal of American Society of Information Science (JASIS); The Computer Journal (CJ); Communications of the ACM (CACM); Journal of Computer and System Sciences (JCSS); Information Systems (IS); Electronic Publishing - Origination, Dissemination and Design (EPODD); and SOFTWARE - Practice and Experience (SPE). Although the journals have been selected from a single field there exist many differences, as we shall see. We are convinced that similar differences are to be expected if journals from different disciplines were to be selected. This is because formats of scientific articles usually do not depend on the subject at the level of our consideration. In this section we will illustrate structure and layout differences in certain forms or parts of manuscripts. Appendices 1 - 4 give more detailed descriptions.

4.1 Delivery form

Manuscripts can be transmitted on paper or electronically (for details see Appendix 1, Table 1). Only ACMTOIS encourages sending an initial manuscript electronically and does not require a paper copy of a manuscript at all. All the others require the initial submission to be made on paper. After a paper has been accepted, most of the journals require or encourage an electronic form either with or without a paper copy and give a list of methods or systems to use. Listed possibilities for an electronic submission (Appendix 1, Table 2) are formatting languages like LaTeX [Lam94], troff [Sil83], RTF [Mic91, Mic94] or MIF [Fra93], the markup language SGML [Gol90], word processing systems like Word [Mic94] or WordPerfect [Sim91, Wor90], plain ASCII text or PostScript page description language [Add90].

From the author's point of view the requirement for a predefined electronic representation can mean that special programmes are needed to make a conversion from the format of one formatter to the format of another formatter or from the format of a text processing

LaTeX	troff	SGML
\documentstyle{article}	.TL	<article><front>
\begin{document}	A paper for Journal	<title>
\title{A paper for Journal}	.AU	A paper for Journal</title>
\author{Eila Kuikka}	Eila Kuikka	<author>
\affiliation{Finland}	.AI	Eila Kuikka</author>
\maketitle	Finland	<affiliation>
\begin{document}	.AB	Finland</affiliation></front>
The paper starts with	The paper starts with	<body><abstract>
a summary.	a summary	<paragraph>The paper starts
\end{abstract}	.AE	with a summary</paragraph>
\keywords{document}	.OK	</abstract><keyword>
\section*{Introduction}	document	document</keyword>
First paragraph.	.SH A	<section><heading>
	Introduction	Introduction</heading>
Second paragraph.	.PP	<paragraph>
	First paragraph.	First paragraph.</paragraph>
....	.PP	<paragraph>
	Second paragraph	Second paragraph.</paragraph>
	...	....

Figure 4.1. The format of an article using LaTeX, ms for troff and SGML

system to the format of a formatter. Usually commercial products offer a set of conversions or conversion programmes to make modifications to most common systems. These programmes, if they exist at all, usually perform only an approximate transformation because document representations do not contain enough information for the deterministic transformation [MKNS87, MKNS89]. If a plain ASCII text is required, then all the extra coding defining the layout and interleaved with the content must be replaced by appropriate white spaces. Text processing systems like Word and WordPerfect use their specific hidden coding systems and usually offer the possibility to generate ASCII text. When formatters are used the document is already represented as an ASCII text and the additional coding must be replaced by a special programme or manually. For formatters, such as LaTeX or troff, for markup language SGML or for styles used in Word text processing system journals usually define their style definitions and the user only creates correct codes for layout. As can be seen in Figure 4.1 codes differ for different languages. If a PostScript language is required then a document can be created with the use of any system that produces a PostScript code. However, PostScript files cannot be edited, thus some other electronic form is needed for editing. The requirement for a Postscript or a camera-ready form means that the author has to follow accurately the rules for the layout of a manuscript and format the text in all its detail.

The journals give descriptions for a page layout of a submitted manuscript (for details, see Appendix 1, Table 3). Margins should be adequately wide or more than 25 mm. Line spacing usually should be double or even triple. Some of journals prefer page numbering, some do not. A page number is either a number or the name of the first author and a

number. Some journals require that authors define a running header, some do not.

4.2 Overall structure and actual content

The structure of an article in all these journals follows approximately the same structure at the highest hierarchy level and contains the following main elements:

- a front part that contains the title, author information and grants,
- a main body that consists of an abstract, a list of keywords and the content,
- a bibliography with references, and
- a back part, if needed, that contains floating elements, such as figures, footnotes, and tables, or information about authors and grants.

However, the structure within these elements varies considerably.

Because the journals give very detailed descriptions for the front part and the bibliography we shall discuss their differences in later sections. However, for the bibliography part there is one exception that influences the overall structure of a document. If a LaTeX or troff formatter is used it is possible that the bibliography is not even part of the main document; it is presented separately in a database file. The actual document contains only citations, the name of a style definition file for references, and the name of a bibliography database file. A separate ASCII file containing references has its own structural representation (see Figures 4.2 and 4.3 in Section 4.4) and is merged with a document using the BibTeX [Lam94] or refer [Tut83] programme.

For the actual text, JASIS, CACM, JCSS and EPODD define the writing style (Appendix 2, Table 1) concerning mainly the content and layout of some parts of the content. These style rules dictate, for example, that a number of an equation be before or after an equation, what kinds of layouts are used for tables (how the lines and footnotes are represented in tables), how references to footnotes are done and how to represent bibliographic references and their citations. Some journals give additional rules for the numbering of sections and, if used, to which level the numbering is allowed.

Floating elements like figures and footnotes are placed either in the main body of a manuscript or on different pages at the end. ACMTOIS uses the first alternative, JCSS uses the latter and other journals use either one of these or some mixture of these extremes (for details, see Appendix 2, Table 2).

Concerning keywords there are two kinds of differences. Keywords in ACMTOIS, CACM, EPODD and SPE form a part of the main body of a manuscript; in JASIS, CJ, JCSS and IS they are missing.

4.3 Title, author information and grants

Appendix 3 describes how the title, author information and grants should be represented in various journals.

In addition to the actual title, some journals demand a shorter form of the title for the running headers of pages. CJ and IS require that this shorter title be after the actual title; JCSS wants it to be represented after the author information together with the address of

the corresponding author on the second page of a manuscript. Other journals do not require the shorter title.

The author information of a manuscript consists of the name, affiliation and address, specifying particularly the author to whom any correspondence should be delivered. We find the following structural differences (for details, see Appendix 3). ACMTOIS requires the authors to give their names and affiliations after the article title and their names, affiliations and addresses in a footnote. JASIS, CJ and SPE situate names, affiliations and addresses after the article title. CACM dictates that names be placed after the article title and descriptions, affiliations and addresses of authors at the end of a manuscript. JCSS requires names, affiliations and addresses of all authors to be after the article title, and the information for the corresponding author as the last element of the front part of the manuscript. IS groups these same elements quite differently: names of all authors are given after the title and their affiliations and addresses are after their names. EPODD uses similar grouping to IS and requires a list of the names of all authors followed by a list of their affiliations and addresses. In all journals, if the first author is not the contact author, that should be noted.

In all journals names of authors are represented as a character string containing the first and last names. However, ACMTOIS represents names of authors also in the footnote, and its form contains only initials for the first names.

An affiliation contains information about the organization of the author, usually the name of the university or company and the name of the department. ACMTOIS requires the affiliation of an author in two places. Only the name of the university or company is given after the name of the author and both the name of the university or company and the name of the department is in the footnote. Other journals represent the affiliation in one place only. IS requires the name of the university or company to be followed by the name of the department. JASIS, CJ, CACM, JCSS and SPE want the name of department first followed by the name of the university or company. In EPODD the affiliation consists of a list of the names of departments followed by a list of names of the universities or companies.

The addresses for the contact author are almost identical in these journals, but more variations apply for the other authors. The postal address of some other authors contains a street address, a telephone number, a fax number and an email address in JASIS, a street address and an email address in ACMTOIS, CACM, EPODD and SPE, but only a street address in CJ, IS and JCSS.

The footnote texts connected to the author information are represented on the first page of a manuscript. In ACMTOIS footnote texts are used for grants and detailed addresses of authors, without footnote citations connected to any other element in the manuscript. Because ACMTOIS represents also other information in the footnote, different footnotes elements are nested together. JASIS, JCSS and IS require that information about grants be in the footnote and that footnote citations be used. JASIS requires that the footnote citation be placed together with the title, whereas JCSS and IS ask them to be placed together with the name of the author.

4.4 Bibliographic references

Citations to references in the bibliography of a manuscript are represented in two ways in these journals. The citation should include the name of the first two authors or if there are more than two authors the name of the first author and a character string et al. and the publishing year, or a sequence number of the list of references. For example, in ACMTOIS, JASIS and CJ a citation to an article written by Kuikka and Penttonen in 1994 is given in a form (Kuikka and Penttonen 1994). Whereas, in CACM, JCSS, IS, EPODD, SPE, if the same article is the ninth reference of the bibliography list, it is given in the form [9]. The value of a citation depends either on the values of other citations or the value of some elements in the bibliographic part of a document.

The reference list is found either in alphabetical order, as dictated by ACMTOIS, JASIS, CJ, CACM, JCSS and IS, or in the order of citation occurrences in the actual text, as dictated by EPODD and SPE.

Appendix 4 shows representations in bibliographical references grouped by their types. The discussed reference types are an article, a book and an article in a conference proceedings. Each of these has typical elements. A book reference has different elements compared to an article reference and to a conference article reference. All the journals use the same elements for a book and the same elements for a conference proceedings article, but elements are not the same for an article. For an article reference CACM uses the number of the first page only, whereas, others use an element which has as its subelements the first and last page number of a referenced paper. CACM and CJ do not use an issue number of a volume of the journal. The placement of the year of publication differs. ACMTOIS, JASIS and CJ place the year after names of authors in all reference types. For other journals the place of a publishing year depends on the bibliographic type. For an article reference CACM and JCSS place a year before the page information, whereas IS, EPODD and SPE place it as the last element. For a book reference all these journals place a year at the end of the reference. For an article in a conference proceedings CACM and SPE place a year before the page information, JCSS after the conference name and before the conference place, and IS and EPODD as the last element. Another difference in the order of elements concerns the last name of an author: exists before the initials of an author (ACMTOIS, JASIS, CJ, CACM) or after the initials of an author (JCSS, IS, EPODD, SPE).

Journals also give rules for formatting of references according to their types. The references are represented in Appendix 4 using the required layout. These kinds of layouts can be created in two different ways. Either the author explicitly creates the correct formatting (by writing the formatter codes or by using different text processing systems) or by using BibTeX [Lam94] or refer [Tut83] bibliographic databases and their style definitions. In the latter case the journals will give the author style definition files, and the user only creates databases. The structures of article references in these databases are represented in Figures 4.2 and 4.3. The order of fields is free, only their format is fixed. BibTeX uses style files for formatting definitions; refer uses macros. Both of these systems allow certain automatic modifications to the structure and the content of the original database entry. It is possible to reverse initials and the last name of an author, to capitalize or lower case some fields, or to select either the number or alphabetical representation for reference identifications and for their citations in the content of a manuscript.

```
%A R. Furuta and V. Quint and J. Andre\(aa
%T Interactively editing structured documents
%J Electronic Publishing
%D 1988
%V 1
%N 1
%P 19-44
```

Figure 4.2. An article reference in a refer file

```
@article{furutaQA88,
author="Furuta, R. and Quint, V. and Andr\'{e}, J.",
title="Interactively editing structured documents",
journal="Electronic Publishing",
year="1988",
volume="1",
number="1",
pages="19-44"
}
```

Figure 4.3. An article reference in a BibTEX file

As we can see from the Appendix 4 (Example 1) none of the representations for article references is identical. Names of authors are written either using upper or upper and lower case letters; the last author is separated by a string AND, and or &; and the names can end either with a comma or a period. The article title can be in single quotes or plain, and it can end either with a comma or by a period. The journal name can be either a common abbreviation of the name or the whole name, and the name can end with a comma or a space. All the journals use italic font for the journal title. A volume number is represented either by italic or bold font. The number of the journal, if it exists, is in parentheses or is separated by a comma from the volume number. The publishing year is given in parentheses or plain, and is followed by a period, a space or a comma.

For a book reference, as we can see from Appendix 4 (Example 2), EPODD and SPE use exactly the same format. Similarly to an article reference, names of authors are written either using upper or upper and lower case letters; the last author is separated by a string AND, and or & and names can end either with a comma or a period. The book title is written in italic font or Roman font, and given inside double quotes or plain. The publishing year is represented in parentheses or plain.

Finally, any of the representations for an article in a conference proceedings, as depicted in Appendix 4 (Example 3), vary. Similarly to an article and a book reference, names of authors are written either using upper or upper and lower case letters; the last author is separated by a string AND, and or & and names can end either with a comma or a period. Also in the same way as an article reference, the article title can be in single quotes or plain and end either with a comma or a period. The name of the conference is presented either in

italic or Roman font. It is represented in double quotes and separated from the title of the article by a period, by a period and the word 'In', by a comma and the word 'in' which is in italic or Roman font. It can end with a comma or a space. The publishing year is given in parentheses or plain, or it can be in parentheses together with the name of the conference. Page numbers are represented in parentheses or plain and may have a prefix 'pp.'

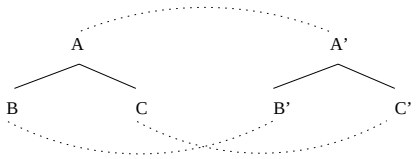
5 A classification of differences

5.1 Bases for the classification

The main question in the transformation from one representation to another is what kind of modifications should be made to which parts of a document and how. Dissimilarities of two trees can be measured by the distance from one tree to another tree defined as a minimum cost sequence of edit operations needed to effect the transformation [Tai79]. Edit operations allow changing one node of a tree into another node, deleting one node from a tree, or inserting one node into a tree. A classification of differences in two trees using different values of the distance, however, does not support our declarative approach in document transformations. It is based on how a transformation should be carried out, rather than describing where the difference is and of what type it is.

Any transformation method needs to specify how to locate transforming elements, how to define the modification with the use of located elements, and how to control the transformation process. Hence, we will base our classification of differences found in structured documents on areas which contain differing elements of two structures, and the nature of a difference between elements in these areas. As the example in the previous section describes a difference can be in the actual content, in separators of parts of the text, in parts that are related to other parts, etc. On the other hand, the difference may be, for example, in orders, existences, or names of elements. The classification based on these criteria has a relation to the power of transformation methods. The wider the differing area containing transforming elements is and the more complex the difference, the more powerful method is required. Notably, a more powerful method needs more information for the transformation. Part of this information is contained within the structure definitions, part of it should be given by the user.

The fundamental assumption for classifying differences in trees is that the *associations between the nodes of two trees* under consideration are given. The association is defined by a mapping which describes unambiguously which nodes in one tree correspond to which nodes in the other tree. For example, in the following figure the mapping is defined by dotted lines between nodes of trees. This means that nodes with the labels A and A' are associated and are roots of trees. These nodes have two children. The node with the label B is the first child of the node with the label A and it is associated with the node labelled by B' that is the first child of the node labelled by A'. Similarly, nodes labelled by C and C' are associated and are the second children of A and A', respectively.



We will consider the classification of differences in parse trees for context-free grammars. In the following discussion we will call nodes that are labelled by nonterminals *nonterminal nodes* and leaf nodes that are labelled by terminals *terminal nodes*. We will also talk about *nonterminal roots* and *nonterminal children*. Further, our classification refers to specific parts of parse trees which we call local parttrees, global parttrees, and basic parttrees. Let T by a parse tree and t a node in T. A *parttree* PT rooted at a nonterminal node t is defined to be a subset of T connected to t. The *depth of the parttree* PT is the longest path from the node t to a leaf of PT. A *local parttree* is a parttree whose depth and number of nodes are smaller than a small given constant. A parttree is considered *global* if it is very deep (the depth is greater than a small integer) or if it is very wide (although the depth is small, there are many nodes in the parttree). A *basic parttree* is a specific parttree which consists of a nonterminal node and all its nonterminal children only. To highlight the relationship between parse trees and nonterminals in the grammar, we will classify types of structure differences between parse trees by considering differences in a pair of basic parttrees whose roots are associated. The classification according to areas of differences between two trees is defined using two parttrees contained in trees and called *associated parttrees*. The associated parttrees

1. contain basic parttrees whose roots are associated and which have a structure difference of a type, and

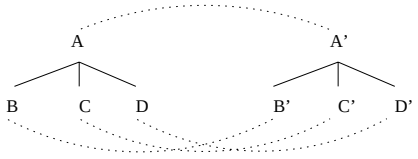
2. have the roots of the basic parttrees as their roots, and other associated nodes are restricted to be only among the leaves but not among the internal nonterminal nodes of the parttrees.

The pair of associated parttrees is a pair of smallest parttrees with associated nonterminal roots where association of nonterminal nodes is defined by the association mapping between nodes of trees.

5.2 Same structure, differences in terminals

Let BT and BT' be two basic parttrees. Let h_b be a mapping that defines the association between nodes of BT and BT'. If the trees are identical when the mapping h_b is used for labels there is *no structure difference* between basic parttrees BT and BT'.

In the following basic parttrees h_b is defined by relations h_b(A) = A', h_b(B) = B', h_b(C) = C', h_b(D) = D' (denoted by dotted lines). Hence, the trees are identical; there is no structure difference.



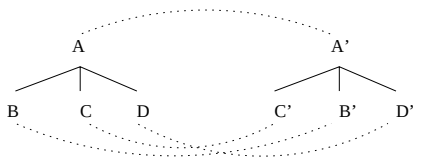
Let T and T' be two parse trees and h the mapping that defines how nodes of T and T' are associated. If for each basic parttree BT in T there is a basic parttree BT' in T' so that roots of BT and BT' are associated and there is no structure difference between BT and BT' then the trees T and T' have the *same internal structure* (as defined by nonterminals). Let $a_1, \ldots, a_n$ be the labels of terminal nodes of T in order from left to right; the resulting sequence $F = a_1 \ldots a_n$ is then a (terminal) frontier of T. Let $a'_1, \ldots, a'_m$ similarly be the labels of terminal nodes of T' and $F' = a'_1 \ldots a'_m$. If the internal structures of T and T' are the same and $F \neq F'$, then the difference between T and T' is classified as *the same structure and differences in terminals.*

5.3 Local differences in the structure

5.3.1 Difference in the order of nonterminal children of a nonterminal node

Let BT and BT' be two basic parttrees. Let h_b be a mapping that defines the association between nodes of BT and BT'. Let $r \in BT$ and $r' \in BT'$ be associated roots and $a_1 \ldots a_n$ be a string formed by concatenating the labels of children of the node r in order from left to right. If a string formed similarly from the labels of the children of r' is a permutation of $h_b(a_1) \ldots h_b(a_n)$, the basic parttrees BT and BT' have an *order difference.*

In the following basic parttrees h_b is defined by relations h_b(A) = A', h_b(B) = B', h_b(C) = C', h_b(D) = D' (denoted by dotted lines). In these trees the children of A labelled by B, C and D, are in a different order than their associated nodes labelled by B', C' and D', respectively, as children of A'.



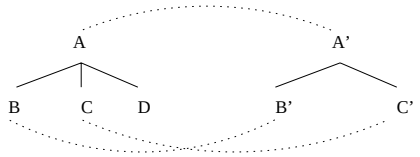
Let T and T' be two parse trees. Let h be a mapping which defines association between nodes of T and T'. Let $T_n \subseteq T$ and $T'_n \subseteq T'$ be parse trees formed from T and T', so that their terminal nodes are removed. If at least for a basic parttree BT_n in T_n there is a basic parttree BT'_n in T'_n such that the roots of BT_n and BT'_n are associated and BT_n and BT'_n have an order difference and if otherwise for a basic parttree in T_n there exists the basic parttree in T'_n such that the roots are associated and the two basic parttrees have no structure difference then the parse trees T and T' have the *difference in the order of nonterminal children.*

The order difference of nonterminal children between two trees is classified *local* because the associated parttrees in trees containing basic parttrees with an order difference are local parttrees formed from a nonterminal node and all its children. The lengths of all the paths in associated parttrees are one. The local parttree of this type can be described by a production of a grammar: the nonterminal on the left side of a production corresponds to the root of the local parttree and the nonterminals and terminals of the right side of a production correspond to the children of the root.

5.3.2 Differences in the existence of nonterminal children of a nonterminal node

Let BT and BT' be two basic parttrees. Let h_b be a mapping that defines the association between nodes of BT and BT'. Let $r \in BT$ and $r' \in BT'$ be associated roots, $a_1, \ldots, a_n, n \geq 1$, nonterminal children of r and $a'_1, \ldots, a'_m, m \geq 1$, nonterminal children of r'. If there exists at least one node a_k $(1 \leq k \leq n)$ in BT which has no associated node in $\{a'_1, \ldots, a'_m\}$, or at least one node a'_k $(1 \leq k' \leq m)$ in BT' which has no associated node in $\{a_1, \ldots, a_n\}$, then basic parttrees BT and BT' have an *existence difference*.

In the following basic parttrees h_b is defined by relations h_b(A) = A', h_b(B) = B', h_b(C) = C' (denoted by dotted lines). For the node labelled D as a child of the node labelled by A in the left-hand side basic parttree there is no associated child in the right-hand side basic parttree.



Let T and T' be two parse trees and h be a mapping which defines how nodes of T and T' are associated. Let $T_n \subseteq T$ and $T'_n \subseteq T'$ be trees formed from T and T', so that the terminals are removed. If at least for a basic parttree BT_n in T_n there is a basic parttree BT'_n in T'_n such that the roots of BT_n and BT'_n are associated and BT_n and BT'_n have an existence difference and otherwise for a basic parttree in T_n there exists a basic parttree in T'_n such that their roots are associated and the two basic parttrees have either an order difference or no structure difference then the parse trees T and T' have the *difference in the existence of nonterminal children.*

The existence difference of nonterminal children between two trees is classified *local* if the associated parttrees in trees containing basic parttrees with an existence difference are local parttrees. If a node n as a child of the root r of a local parttree pt in T has no associated node as a child of the root r' of the associated local parttree pt' in T' we can find three different cases when we consider to which node in T' the node n can be associated:

1. A nonterminal child n of the root r of a local parttree pt and nodes of subtrees of n in a tree T have no associated nodes in a tree T'. The associated parttrees are defined by associated nonterminal roots and their children.

2. A nonterminal child n of the root r of a local parttree pt in a tree T has an associated node as a leaf of the local parttree pt' in a tree T' but not as a child of the root r' which is associated to r. The difference is recognized in a pair of local associated parttrees whose roots are associated and at least one of parttrees has a depth which is greater than one.

3. A nonterminal child n of the root r of a local parttree pt in a tree T has no associated node in another tree T' but leaves (successors of n) of the local parttree pt have associated nodes as leaves of a local parttree pt' of another tree T'. The difference is recognized in a pair of local associated parttrees whose roots are associated and at least one of parttrees has a depth which is greater than one.

5.4 Global differences in the structure

The difference between two parse trees is global if it occurs in large associated parttrees where "large" means that the depths or widths of parttrees are not bounded by small given constants. Large associated parttrees exist, if associated nonterminal nodes as leaves of associated parttrees are in very different places in the parse trees. Also, large associated parttrees exists if associated nonterminal nodes being leaves of associated parttrees are scattered in the whole tree or a wide part of one tree and are collected together into a list in another tree. Further, large associated parttrees arise also, if trees are represented using very different structures, meaning that, in the worst case only leaves of parse trees can be associated. A further example of global differences is a situation when many trees are compared. As a structure difference this means that a tree or their subtrees should be associated or merged with subtrees of many trees.

6 Transformation methods

Many transformation methods suitable for trees can be found in the literature. The following subsections represent brief descriptions of syntax-directed translation methods, tree transducers, tree pattern matching/replacement methods and manual methods. The representations contain definitions of methods and their extensions which are important for transformations of structured documents. In addition to algorithms for methods found in the literature, we have made a set of algorithms which are needed when methods are used in document transformations.

6.1 Syntax-directed translation

Used in compilers a syntax-directed translation model [AU72, AU73, ASU86] combines syntax analysis and code generation in the compilation of formal languages. The translation schema is defined using pairs of productions of two grammars. Whenever a production is used in the derivation of the input parse tree, the translation element of a production is used to compute a portion of the output parse tree associated with a portion of the input tree generated by the production. A translation schema corresponds in translations from a tree to a tree same as a grammar in translations from a string to a tree. The association of nodes in portions of input and output trees is an essential part of the translation. According to the association of nonterminals in paired productions, different kinds of translation schemas are defined. In the coming subsections we will define first a translation method that is usually called a syntax-directed translation, then, its restricted form called a simple syntax-directed translation, its generalized form called a generalized syntax-directed translation, finally, our extended form called an extended syntax-directed translation.

6.1.1 Syntax-directed translation schema

Definition 6.1. A *syntax-directed translation schema* (SDTS) [AU72] is a 5-tuple $T = (\mathrm{N}, \Sigma, \Delta, \mathrm{R}, S)$, where N is a finite set of nonterminals, Σ is a finite input alphabet, Δ is a finite output alphabet, R is a finite set of rules of the form $A \rightarrow \alpha, \beta$, where $\alpha \in (\mathrm{N} \cup \Sigma)^*$,

$\beta \in (\mathrm{N} \cup \Delta)^*$ and the nonterminals in β are a *permutation of the nonterminals* in α, and S is a distinguished nonterminal in N, the start symbol.

In a rule $A \to \alpha, \beta$ to each nonterminal of α there is associated an identical nonterminal of β. If a nonterminal occurs only once the association is obvious. If the same nonterminal occurs more than once then integer superscripts are used to indicate the association.

The definition for the derivation of a transformation [AU72] from a pair of start symbols (S, S) for T corresponds to the definition of the derivation for a grammar represented in Section 2. The derivation starts from (S, S). In a single derivation step associated nonterminals A in a pair of translation forms $(\alpha A \beta, \alpha' A \beta')$ are derived using an SDTS rule $A \to \gamma, \gamma'$ to get a pair $(\alpha\gamma\beta, \alpha'\gamma'\beta')$.

If T is an SDTS, then the grammar $G_i = (\mathrm{N}, \Sigma, \mathrm{P}, S)$, where $\mathrm{P} = \{A \to \alpha \mid A \to \alpha, \beta \in \mathrm{R}\}$, is called the *input grammar* of T and the grammar $G_o = (\mathrm{N}, \Delta, \mathrm{P}, S)$, where $\mathrm{P} = \{A \to \beta \mid A \to \alpha, \beta \in \mathrm{R}\}$ the *output grammar* of T. Thus, the rules of SDTS can be formed from productions of the input and output grammars with the use of the following algorithm.

Algorithm 6.1. Comparing of an input and output grammar to form a SDTS T. Productions of grammars are paired and nonterminals in paired productions are associated by an identity and using superscripts.
Input: Grammars G_i and G_o.
Output: The SDTS T.
Method:

1. Apply 2 to all productions in G_i starting at the first production p_i.
2. The production p_i is of form $\mathrm{A} \to \alpha$.
   (a) Search productions of the G_o whose left-hand side nonterminal is A. If there are more than one production, the production which has on its right-hand side the same nonterminals as in α on the right-hand side of the production is selected as the production p_o corresponding to p_i. The production p_o is of of form $\mathrm{A} \to \beta$.
   (b) Add the rule $\mathrm{A} \to \alpha, \beta$ to T.
   (c) Select the next production p_i in G_i, if any, and apply 2 to it.

Aho and Ullman [AU72] give an algorithm for the transformation via an SDTS from a derivation tree in an input grammar into a derivation tree in an output grammar. The algorithm starting from the root of a tree changes the terminal children of a node and reorders the nonterminal children according to rules of a translation schema and processes the tree from the root to leaves using the depth-first traversal of nodes of modified tree. In [AU72] and [Nik90] it has been proved that the algorithm executes a translation from a tree with a frontier to another tree with the different structure and with a different frontier. A single transformation step concerns a node and all its children. If the transformation definition is semantically unambiguous (i.e. for each structure there is only one translation) this algorithm gives a deterministic technique to carry out transformations when differences between an input and output tree are in the frontier and in the order of nonterminal children of a node.

In the SDTS rules the association of nonterminals is made for identical nonterminals. If nonterminal labels differs we will need a string homomorphism on nonterminals. Let Σ_1^* and Σ_2^* be the alphabets of two languages. A string homomorphism is a mapping $h : \Sigma_1^* \to \Sigma_2^*$ for which $h(\epsilon) = \epsilon$ (ϵ is an empty word) and $h(ax) = h(a)h(x)$ for $a \in \Sigma_1$ and all $x \in \Sigma_1^*$.

Another way to implement a syntax-directed translation is to use the Szilard-word of the context-free grammar.

Definition 6.2. Let G be a grammar and let each rule of G have an associated label. For a (terminal) derivation

$$D : S \stackrel{r_1}{\Rightarrow} x_1 \stackrel{r_2}{\Rightarrow} x_2 \stackrel{r_3}{\Rightarrow} \ldots \stackrel{r_k}{\Rightarrow} x_k \in L(G)$$

using the rules with labels $r_1, r_2, \ldots, r_k$ the word $\mathrm{Sz}(D) = r_1 r_2 \ldots r_k$ is called the *Szilard word* of the derivation D. The set

$$\mathrm{Sz}(G) = \{\mathrm{Sz}(D) | D \text{ is a (terminal) derivation in } G\}$$

is called the *Szilard language* associated to G.

Each context-free grammar has a corresponding Szilard language. The *leftmost Szilard-word* is the sequence of production labels in the leftmost derivation. The translation of parse trees can be defined by the following algorithm.

Algorithm 6.2. Tree transformation via leftmost Szilard-word.
Input. Two context-free grammars G_1 and G_2 with labelled productions, the derivation tree T for a grammar G_1, and a homomorphic mapping h that defines the association between production labels of G_1 and G_2.
Output. A derivation tree T' for a grammar G_2.
Method.

1. Form a leftmost Szilard-word $\mathrm{Sz}(D_l(T))$ for a derivation tree T. Let $\mathrm{Sz}(D_l(T)) = p_1 \ldots p_n$, $n \geq 1$ where $p_i, 1 \leq i \leq n$, is a production label of G_1.
2. Make the mapping $h(p_1 \ldots p_n) = h(p_1) \ldots h(p_n) = q_1 \ldots q_n = \mathrm{Sz}(D'_l(T'))$.
3. Generate a tree T' by applying the productions of G_2 according their labels in $q_1 \ldots q_n$ from the left.

The algorithm allows also that labels of parse trees are changed because the association of rules in the input and output grammar is made according to the rule label not according to the nonterminals on the left sides of the productions.

The leftmost Szilard-word [Sal73] can be used to a transformation via an SDTS from a leftmost derivation tree of the input grammar to a derivation tree of the output grammar. The output tree of an SDT is not the leftmost derivation tree of the output grammar because nonterminals in an SDTS rule are allowed to be permuted.

6.1.2 Simple syntax-directed translation schema

In a *simple syntax-directed translation schema* (SSDTS) [AU72] the difference to the definition 6.1 of SDTS is in the definition of the set R which must be following:

R is a finite set of rules of the form $A \rightarrow \alpha, \beta$, where $A \in \mathrm{N}$, $\alpha \in (\mathrm{N} \cup \Sigma)^*$, $\beta \in (\mathrm{N} \cup \Delta)^*$, and the nonterminals in β are the nonterminals in α *in the same order* from the left.

Each nonterminal of α in a rule $A \rightarrow \alpha, \beta$ has an associated identical nonterminal of β in the order from left to right. Thus, nonterminals of α and β are the same and in the same order and the input and output grammars define same structures for parse trees. If nonterminal labels of parse trees differ we will need a string homomorphism on nonterminals in SSDTS rules.

The SSDTS can be generated with the use of the algorithm 6.1. Further, the same algorithm as mentioned for the SDTS can be used also to implement the transformation using an SSDTS. Actually, the algorithm could also be represented in a simpler form because a single translation step of an SSDT does not need to execute the reordering of nonterminal nodes. Only terminal nodes are deleted and inserted according to schema rules.

The leftmost Szilard-word [Sal73] can be used also to a transformation via an SSDTS from a leftmost derivation tree of the input grammar to a leftmost derivation tree of the output grammar.

6.1.3 Generalized syntax-directed translation schema

A generalized syntax-directed translation schema allows to define several output forms corresponding one input form in a rule using an extra translation symbols. Each translation depends on translations of the various direct successors of the node in question. The translation elements can be arbitrary strings of output symbols and translation symbols representing the translation in its successors. Thus, translation symbols can be repeated.

A *generalized syntax-directed translation schema* (GSDTS) [AU73] is a 6-tuple T=(N, Σ, Δ, Γ, R, S), where N, Σ, Δ and S are defined as in Definition 6.1 and

- Γ is a finite set of translation symbols of the form A_i, where $A \in \mathrm{N}$ and i is an integer and it is assumed that $S_1 \in \Gamma$, and
- R is a finite set of rules of the form

$$A \rightarrow \alpha, A_1 = \beta_1, A_2 = \beta_2, \ldots, A_m = \beta_m$$

subject to following constraints:

1. $A \in \mathrm{N}$.
2. $\alpha \in (\mathrm{N} \cup \Sigma)^*$.
3. $A_j \in \Gamma$ for $1 \leq j \leq m$.
4. Each symbol of $\beta_1, \ldots, \beta_n$ is either in Δ or symbol $B_k \in \Gamma$ such that B is a nonterminal which appears in α.
5. If α has more than one symbol B, then each B_k in the β's is associated by a superscript to one of these instances of B.

In a rule $A \rightarrow \alpha, A_1 = \beta_1, A_2 = \beta_2, \ldots, A_m = \beta_m$ A_i is called a *translation* of A and $A_i = \beta_i$ a *translation element* associated with a production for A in T. With each interior node n of a parse tree (in the input grammar) labelled by A it is associated one string of symbols for each A_i in Γ. The transformation is made bottom-up and each value is computed by substituting the values defined at direct successors of n for the translation symbols of the translation element for A_i. Thus, the generalized syntax-directed form can be used to compute semantic translations, not only syntactic.

6.1.4 Extended syntax-directed translation schema

Definition 6.3. An *extended syntax-directed translation schema* (ESDTS) is a 6-tuple T =(N, M, Σ, Δ, R, S), where N is a finite set of input nonterminals, M is a finite set of output nonterminals, $\mathrm{N} \cap \mathrm{M} \neq \emptyset$, Σ is a finite input alphabet, Δ is a finite output alphabet, $\mathrm{S} \in \mathrm{N} \cap \mathrm{M}$ is a distinguished nonterminal, a start symbol, and R is a finite set of rules of one of the types:

1. $A \rightarrow \alpha, \beta$, where $\mathrm{A} \in \mathrm{N} \cap \mathrm{M}$, $\alpha \in (\mathrm{N} \cup \Sigma)^*$, $\beta \in (\mathrm{M} \cup \Delta)^*$.
2. $A \rightarrow \alpha, \varepsilon$, where $\mathrm{A} \in \mathrm{N}$, $\mathrm{A} \notin \mathrm{M}$, $\alpha \in (\mathrm{N} \cup \Sigma)^*$, ε is an empty string.
3. $A \rightarrow \varepsilon, \beta$, where $\mathrm{A} \notin \mathrm{N}$, $\mathrm{A} \in \mathrm{M}$, $\beta \in (\mathrm{M} \cup \Delta)^*$, ε is an empty string.

The association of nonterminals differs from the association defined for the SDTS and SSDTS because in a rule $A \rightarrow \alpha, \beta$ each nonterminal of α does not have an associated nonterminal in β, and vice versa. Thus, only a part of nonterminals in α has an associated identical nonterminal in β. If the same nonterminal occurs more than once on the right-hand side of the ESDTS rule superscripts are used to indicate the association. The ESDTS can be further extended to carry out transformations that need to change labels of associated nodes in the similar way as we explained for the SDTS.

Also, the definitions for an input and an output grammar as well for the derivation of translation forms differ for the same reason. The input grammar of T is $G_i = (\mathrm{N}, \Sigma, \mathrm{P}_i, S)$ where $\mathrm{P}_i = \{A \rightarrow \alpha | A \rightarrow \alpha, \beta\} \cup \{A \rightarrow \alpha | A \rightarrow \alpha, \varepsilon\}$ and the output grammar of T is $G_o = (\mathrm{M}, \Sigma, \mathrm{P}_o, S)$ where $\mathrm{P}_o = \{A \rightarrow \beta | A \rightarrow \alpha, \beta\} \cup \{A \rightarrow \beta | A \rightarrow \varepsilon, \beta\}$. Let (s, s') be a translation form derived from a pair of start symbols (S, S') for an ESDTS T. In the input form s there exists explicit nonterminals that are not associated to any nonterminals in the output form s', and vice versa. A single derivation step from the translation form $(s, s') = (\alpha A \beta, \alpha' A \beta')$ where A's are associated is made according to the rule $\mathrm{A} \rightarrow \gamma, \gamma'$ of type 1 in Definition 6.3 in the similar way as by the SDTS. This produces a translation form $(\alpha\gamma\beta, \alpha'\gamma'\beta')$, where a part of nonterminals of γ and γ' are associated. When a rule is applied to a nonassociated nonterminal either in the input form or in the output form, an empty word is derived from an empty word in the output and input form, respectively. Thus, a single derivation step from the translation form $(\alpha A \beta, \delta')$ where A is a non-associated nonterminal is made according to the rule of type 1 or 2 in Definition 6.3 so that in δ' from an empty word an empty word is derived. Thus, if there is a rule of type 1, $\mathrm{A} \rightarrow \gamma, \theta$, $\theta \in (\mathrm{M} \cup \Delta)^*$ or a rule of type 2, $\mathrm{A} \rightarrow \gamma, \epsilon$, in R then the derivation produces a translation form $(\alpha\gamma\beta, \delta')$, where any of nonterminals of γ has no associated nonterminal in δ'. A single derivation step in a pair $(\delta, \alpha' A \beta')$ where A is a non-associated nonterminal is made in the

similar way according to the rule of type 1 or 3 and any of the nonterminals derived from A has no associated nonterminal in δ. Let us have an ESDTS, with the start symbol `A` and rules

```
A --> BCDE, BDE
B --> b,b
C --> F,""
F --> f,f
D --> F,F
E --> e,e
```

Then we will get a following leftmost derivation for (`A`,`A`)

```
(A,A) ==> (B[C]DE,BDE) ==> (b[C]DE,bDE) ==> (b[F]DE,bDE) ==>
          (bfDE,bDE) ==> (bfFE,bFE) ==> (bffE,bfE) ==>
          (bffe,bfe)
```

where nonassociated nonterminals are denoted by brackets.

The following algorithm generates an ESDTS from the input and output grammar of the translation. The difference to Algorithm 6.1 is caused by the fact that some productions of these grammars do not have a corresponding production in the other grammar.

Algorithm 5.3. Comparing of two grammars to form a ESDTS T. A part of grammar productions is paired and a part of nonterminals in a production of the input grammar is associated with a part of nonterminals in the paired production by an identity and using superscripts.
Input: Grammars G_i and G_o.
Output: The ESDTS T.
Method:

1. Apply 2 to all productions in G_i starting at the first production p_i.
2. The production p_i is of form A $\rightarrow \alpha$.
   (a) Search productions of the G_o, whose left-hand side nonterminal is A. If there is more than one production ask the user to select one of these as the production p_o corresponding to p_i.
   (b) If a corresponding production p_o of form A $\rightarrow \beta$ was found, add the rule A $\rightarrow \alpha, \beta$ to T.
   (c) If no corresponding production for p_i is found, add the rule A $\rightarrow \alpha, \epsilon$ to T.
   (d) Apply 2 to the next production p_i of G_i.
3. For all productions p_o in G_o of form A $\rightarrow \beta$ for which there are no corresponding productions in G_i, add the rule A $\rightarrow \epsilon, \beta$ to T.

The algorithm for a tree transformation via an ESDTS is a modification from the corresponding SDTS algorithm [AU72]. A single step of the algorithm will cut input terminals and non-associated nonterminal children (and their subtrees) from the input tree and reorder

the associated nonterminal children, and finally, add nodes labelled by output terminals and nonassociated nonterminals (as nonterminal leaves).

Algorithm 6.4. Tree transformation via an ESDTS.
Input. An ESDTS $T = (\mathrm{N}, \mathrm{M}, \Sigma, \Delta, \mathrm{R}, S, S')$, with input grammar $G_i = (\mathrm{N}, \Sigma, \mathrm{P}_i, S)$, output grammar $G_o = (\mathrm{M}, \Delta, \mathrm{P}_o, S')$, and a derivation tree D in G_i, with frontier in Σ^*.
Output. Some derivation tree D' in G_o such that if x and y are the frontiers on D and D', respectively, then y is a translation of x.
Method.

1. Apply step (2), recursively, starting at the root of D.
2. Let this step be applied to node n. It will be the case that n is an internal node of D. Let n have direct successors $n_1, \ldots, n_k$.
   (a) Let in R be a rule $A \to \alpha, \beta$ such that n has a label A and its direct successors have the labels from which α is formed by concatenating the labels of $n_1, \ldots, n_k$. For direct successors of n, if any:
   - Delete those direct successor nodes which are leaves (i.e. have terminal or $\epsilon - label$).
   - Delete those direct successor nodes (and subtrees denominated by these nodes) that have a nonterminal in α but no associated nonterminal in β.
   - Reorder remaining direct successors, if any, according to the permutation between associated nonterminals in α and β, (any subtree dominated by these nodes remains in fixed relationship to the direct successors of n).
   - Insert direct successor leaves of n if there is a nonterminal in β but no associated nonterminal in α. The insertion is made so that for each leaf a label and a place in left to right order is defined by nonterminals in β.
   - Insert direct successor leaves of n so that the labels of its direct successors form β.

   (b) Apply step (2) to direct successor of n which are not leaves (i.e. have terminal or ϵ-label or nonterminal label in a set of output nonterminals), in order from the left.
3. The resulting tree is D'.

6.2 Tree transducer

A tree transducer [Rou70, Tha73, Bak78b, GS84, NP92] is a device that takes as its input a finite labelled tree and outputs a finite labelled tree so that every transition function is defined for a structure in the input tree and a state of the transducer. A tree transducer [Rou70, Tha73, Eng77a] is a descending tree transducer if it reads trees from the root towards the leaves, or an ascending tree transducer if it reads trees from the leaves towards the root. In this work we will use only a descending tree transducer and give the definition here. The ascending tree transducer is defined correspondingly.

A *ranked alphabet* is a finite set of symbols each of which has a nonnegative integer *arity* or *rank*. If Σ is a ranked alphabet, then for any $m \geq 0$, Σ_m denotes a set of m-ary symbols in Σ. Let X be an (ordinary) alphabet disjoint from Σ. Then the set $\mathrm{T}_\Sigma(\mathrm{X})$ of all Σ-*terms over* X is the smallest set T for which $\mathrm{X} \subseteq \mathrm{T}$ and $\sigma(t_1, \ldots, t_m) \in \mathrm{T}$ whenever $m \geq 0$, $\sigma \in \Sigma_m$ and $t_1, \ldots, t_m \in \mathrm{T}$. The ΣX-terms are regarded as formal representation of labelled left-to-right ordered finite trees, and hence they can be called ΣX-*trees* or just trees. In the following definition Ξ_m consists of the first $m \geq 0$ elements of a countably infinite set $\Xi = \{\xi_1, \xi_2, \ldots\}$.

Definition 6.4. A *descending tree transducer* is a 8-tuple $D = (\Sigma, \mathrm{X}, \mathrm{Q}, \Lambda, \mathrm{Y}, \mathrm{R}, \mathrm{Q}')$, where X and Y are the frontier alphabets, Σ and Λ are ranked alphabets, Q is a ranked alphabet of rank 1, the *state set* of D (disjoint from all other sets in definition of D, except Q'), $\mathrm{Q}' \subseteq \mathrm{Q}$ is a set of initial states, and R is a finite set of productions of the following two types:

1. $q(x) \to t$ $(q \in \mathrm{Q}, x \in \mathrm{X}, t \in \mathrm{T}_\Lambda(\mathrm{Y}))$,
2. $q(\sigma(\xi_1, \ldots, \xi_m)) \to t$ $(q \in Q, \sigma \in \Sigma_m, m \geq 0, t \in \mathrm{T}_\Lambda(\mathrm{Y} \cup \mathrm{Q}\Xi_m)$.

The left-hand side of a rule defines the input pattern to be located in the input tree and the right-hand side an output pattern replacing the input pattern in the input tree to create the output tree. The variables ξ_i in this definition represent the subtrees hanging under the node labelled by σ, and a set $\mathrm{Q}\Xi_m$ is a set of terms of a form $q_j(\xi_i)$ where q_j in Q and ξ_i in Ξ_m. The definition allows input patterns to consist of a state node, its only child with all the children. The transducer is *left linear* if each ξ_i exists only once on the left-hand side of a rule and *deterministic* if there is only one initial state and there is no distinct rules in R with the same left-hand side.

A descending *tree transducer with output* [Tha73] processes an input tree from the root to leaves, combines transforming nodes of an input tree with their states and outputs an output tree a structure after a structure. Input trees are trees over $\Sigma \cup \mathrm{X} \cup \mathrm{Q}$ and output trees over $\Lambda \cup \mathrm{Y} \cup \Xi$ where variables $\xi_i \in \Xi$, $(i \geq 1)$, can occur only in leaves. During the process unprocessed nodes (as a child of a state) in the input tree are linked to corresponding unprocessed nodes (variables) in the output tree.

A *tree homomorphism* is a one-state tree transducer [GS84] and it extends a string homomorphism to the trees. The tree homomorphism does not modify the structure of the tree, only terminal nodes and labels of nonterminal nodes will be changed. A left linear tree transducer $\tau \subseteq \mathrm{T}_\Sigma(\mathrm{X}) \times \mathrm{T}_\Lambda(\mathrm{Y})$ can be defined by a *tree bihomomorphism* $B = (h, \mathrm{R}, h')$ [AD76, Ste84, Ste90] where h and h' are two tree homomorphisms

$$h : \mathrm{T}_\Omega(\mathrm{Z}) \to \mathrm{T}_\Sigma(\mathrm{X})$$

$$h' : \mathrm{T}_\Omega(\mathrm{Z}) \to \mathrm{T}_\Lambda(\mathrm{Y})$$

and R is a regular set of ΩZ trees (ΩZ tree language). The transformation $\tau(B) = h^{-1}\rho h'$ where ρ on a transformation of a regular tree to its self and composition of relations is made from left to right. Because for every regular tree ΣX language can be generated a tree transducer, different kinds of transformations can be implemented using different tree homomorphisms.

To increase the transformational capability, more powerful transducers are introduced as well. A descending tree transducer with regular look-ahead [Eng77b] allows to inspect

the subtrees of a node before processing it, however, restricting extracted information to be finite and regular. The rules in macro tree transducers [EV85] are in the form $q(\sigma(\xi_1,\ldots,\xi_m),y_1,\ldots,y_m) \rightarrow t$ where the first parameter defines the syntax for a tree similarly to a descending tree transducer and other parameters $y_1,\ldots,y_m$ define conditions for the context of a tree. On the right-hand side of rules, operations on the tree are allowed. A basic tree transducer defined in [Vog87] is similar to a macro tree transducer except forbidding nesting of states. In attributed tree transducers [CFZ82], applied when attribute grammars for trees are used, the context attributes of a macro tree transducer are used to compute attributes. High level tree transducers [EV88] is an extension of descending transducers such that if the level $n = 0$ it defines a descending tree transducer and if the level $n = 1$ it defines a macro tree transducer. Thus, the context of a tree can contain more than one level. The rules of a modular tree transducer [EV91] are defined by a special left-linear, non-overlapping term rewriting system where a set of rules is partitioned into numbered modules and a module with number i may call modules with numbers not less than i. A modular tree transducer, as an generalization of a macro tree transducer, allows also to specify operations on trees in a modular way.

Definition 6.6. A *term rewriting system* [Klo92, Rao93, SPvE93] is a pair $R = (\Sigma, \mathrm{P})$ where Σ is a finite ranked alphabet of function symbols and P is a set of rewrite rules over a set of terms $\mathrm{T}_\Sigma(\Xi)$ of the form $t \rightarrow t'$ where t and t' are terms called an *input pattern* and an *output pattern*, respectively, so that the input pattern t is not a variable, and the variables in the output pattern t' are already contained in t. A *rewrite rule* $t \rightarrow t'$ determines a set of rewrites $s(t) \rightarrow s(t')$ for all substitutions s. This defines a single-step rewriting relation (t, t') generated by it. Concatenating rewriting steps we have a rewriting sequence $t_1 = t_0 \rightarrow \ldots \rightarrow t_n = t_2$ and t_2 is a rewriting of t_1.

In the previous definition a *substitution* s is a (partial) mapping on a set of terms $\mathrm{T}_\Sigma(\Xi)$ which satisfies $s(\sigma(t_1,\ldots,t_n)) = \sigma(s(t_1),\ldots,s(t_n))$ for every n-ary function symbol σ, $(n \geq 0)$.

We will restrict currently to a descending tree transducer for which transition rules are simpler and thus generated at least interactively from the grammar definitions of parse trees. However, instead of using Definition 6.4 we represent two extensions for its rule definition. Firstly, because we want to allow more complex tree structures on left-hand sides of rules we use an extension which is defined, for example, in [GS84, page 144]. A single transformation step is applied to a tree and its hanging subtrees and not only to a node and its hanging subtrees as according to Definition 6.4. In this case, we present the rules of a descending tree transducer as a term rewriting system. Secondly, we want to allow transformations which add atomic nonterminal nodes to an output tree which according to Definition 6.4 it is not possible. Thus, a set R in Definition 6.4 is redefined as follows:

R is a finite set of rules of the following two types:

1. $q(x) \rightarrow t$ $(q \in \mathrm{Q}, x \in \mathrm{X}, t \in \mathrm{T}_\Lambda(\mathrm{Y}))$,
2. $q(t') \rightarrow t$ $(q \in \mathrm{Q}, t' \in \mathrm{T}_\Sigma(\mathrm{X} \cup \Xi_m), m \geq 0, t \in \mathrm{T}_\Lambda(\mathrm{Y} \cup \mathrm{Q}\Xi_m \cup \Lambda)$.

When two grammars define structures for an input and output tree and an association between nodes of trees is defined by the association mapping, transition rules for a tree transducer can be formed with the use of the following algorithm. The algorithm is

semiautomatic because the user is asked to give additional information for the transducer (select productions and construct term rewriting rules). States will inform which are the processing nodes in the output tree. Thus, a state is defined for every different nonterminal of the output grammar in the beginning of the algorithm. In addition, a state is used for terminals in rules of type 1. Start symbols of the output grammar define the initial states of the tree transducer. Rules of type 2 are needed for all states which exist in the right-hand side of some rewrite rules. A production that defines alternative forms for its left-hand side nonterminal is regarded to be several productions and a rule is defined for each alternative.

Algorithm 6.5. Comparing two grammars to define states, initial states and transition rules for a descending tree transducer. The association of nonterminals of grammars is defined by a mapping.
Input: Grammars G_i and G_o and mapping h.
Output: States Q, initial states Q' and transition rules R.
Method:

1. Let $p_1, ..., p_n$ be states corresponding to all left-hand side nonterminals in productions of G_o. Let p_{n+1} be a state for all terminals.

2. For all terminals x in the input grammar add a rule $p_{n+1}(x) \rightarrow t$ to R; t is a tree in $T_\Lambda(Y)$ defined by the user.

3. Start symbols of G_o define initial states in the set $\{p_1 \ldots p_n\}$. They form a set Q'. Add initial states to a list of processing states.

4. Starting at the first state in the list of processing states apply 5 to every state in the list and for each state to every production pr_o in G_o whose left-hand side nonterminal corresponds to the state (or to every alternative form in pr_o).

5. The state is q_B and the nonterminal of the left-hand side of pr_o is B.

   (a) Ask the user to select a corresponding production pr_i in G_i to pr_o. If no production is selected, continue from (f). The left-hand side of pr_i is A.

   (b) Ask the user to derive a parttree pt_i rooted at A using G_i and the associated parttree pt_o rooted at B using G_o. According to the definition for associated parttrees (see the definition in Section 5.1), among leaves of pt_i labelled by $A_1, \ldots, A_n$ from the left there exist leaves of pt_o labelled by $A'_1, \ldots, A'_m$ from the left so that the mapping h defines the association between $A_1, \ldots, A_n$ and $A'_1, \ldots, A'_m$ and internal nodes of pt_i and pt_o are not associated.

   (c) Let variables $\xi_1, \ldots, \xi_n$ correspond labels $A_1, \ldots, A_n$. Let $q_1, \ldots, q_m$ be states from a set $\{p_1 \ldots p_n\} \cup \{p_{n+1}\}$ corresponding to the labels $A'_1, \ldots, A'_m$.

   (d) Replace labels $A_1, \ldots, A_n$ in pt_i with corresponding variables $\xi_1, \ldots \xi_n$. Replace labels $A'_1, \ldots, A'_m$ in pt_o with corresponding terms $q_j(\xi_i)$ where q_j and ξ_i are defined according to the association defined by h between labels $\{A_1, \ldots, A_n\}$ and $\{A'_1, \ldots, A'_m\}$, and add states q_j occurring in pt_o to the list of processing states.

(e) Add the term rewrite rule $q_B(pt_i) \rightarrow pt_o$ to R representing parttrees pt_i and pt_o as terms.

(f) Apply 5 to the next production pr_o in G_o whose left-hand side nonterminal corresponds the state q_B (or the next alternative in pr_o). If there is not such a production (or an alternative), apply 5 to the next state in the list of processing states and the first production pr_o in G_o whose left-hand side nonterminal corresponds the state (or to the first alternative in pr_o, if pr_o has alternative forms), if there is any.

6. Separate states in a list of processing states list form the set Q.

A descending tree transducer according to the following algorithm processes an input tree from the root to leaves, adds a node for an initial state as a parent of the root of the input tree and replaces an input pattern with an output pattern controlled by states of the transducer. The match of an input pattern against to the input tree is made according to the following definition.

Definition 6.7. The input pattern t with occurrences $\xi_1, \ldots, \xi_k$ in leaves *matches* a tree t'' in T rooted at node n of the tree T if there exist trees $t_1, \ldots, t_k$ in T such that the tree t' obtained from t by substituting t_i for the ith occurrence of ξ_i in t, is equal to the subtree of t'' rooted at n.

State nodes are added to and deleted from the input tree during the transformation to show the unprocessed places of the tree. The process continues as long as there are states in the input tree.

Algorithm 6.6. Tree transformation with the use of a modified descending tree transducer.
Input: A tree T, a tree transducer D.
Output: A tree T'.
Method:

1. Add a node with a label q_i of the initial state as parent of the root of T. For the selected initial state q_i there exists a rule $q_i(t) \rightarrow t'$ in the rule set R of D such that t matches the tree T.

2. Starting at the initial state node of T apply 3 to every state node n of T; its label is q_n.

3. The child node n' of n will be a leaf or an internal node of T.

   (a) If n' is a (terminal) leaf, select a rule $q_n(x) \rightarrow t'$ from the set R of D and replace the node n with t'.

   (b) If n' is an internal (nonterminal) node, select a rule $q_n(t) \rightarrow t'$ from the rule set R of D such that t matches a subtree t'' whose root is n'. Variables ξ_i in t are associated with subtrees of t'' according to their occurrence in t. Replace the occurrence t in t'' with t' so that every variable ξ_i in t' is replaced by the subtree of t'' whose root was associated with the variable ξ_i of t. Replace the node n with the modified subtree t''.

(c) Apply 3 to a state node of T, if there are any.

4. The resulting tree is T'.

6.3 Tree pattern matching/replacement methods

Transformations with the use of a tree pattern matching and a tree replacement consist according to [HO82] of three steps. The first one, a pattern matching, is the process of locating substructures of a larger structure by comparing them to a given form of a pattern. The second one is a process to decide between different replacements. The third one, pattern replacement, is a process that takes a substructure and replaces it with a new one. Languages for tree pattern matching/replacement methods for trees specify a tree-to-tree transformation as sets of

```
pattern { replacement }
```

rules. The pattern and the replacement in rules can be specified as syntactical structures of trees and expressed with the use of the semantics for the labels of the trees defined by a grammar. On the other hand, the pattern specification can use only structural information in tree instances, and must in this case be formed so that the pattern and the replacement depict the real structure of a part of the tree. Because the replacement of a part of the tree with another structure is usually a straightforward process, the most important phase when considering the effectiveness of transformations comes from the repeated searching for the next subtree to be replaced.

The tree pattern matching/replacement methods have been used in a number of programming tasks, for instance, in building pretty-printers and designing interpreters for non-procedural languages as well as in code optimatization, symbolic computation and context search and replacement in structure editors. In these cases the tree pattern can be defined using syntactical structures and the transformations can be implemented with methods such as generalized syntax-directed translation [Bak78a], tree transducers and term rewriting systems. The pattern matching/replacement method represented in [KPPM84] reminds the syntax-directed translation because it specifies the transformation using a tree transformation grammar. This grammar defines patterns using sets of productions from the grammars for the input and output trees. The transformation programme is generated from this grammar definition. Different algorithms have been developed for the tree pattern matching problem. In [HO82] they are mainly extensions to string pattern matching algorithms and in [Kil92] based on a tree inclusion. Basic sequential algorithms traverse the input tree in the preorder and compare the pattern to each subtree of the input tree in turn. Usually efficient tree pattern matching algorithms require some kind of a preprocessing for patterns and/or for input trees producing additional information (tables, indices, etc.). Pattern matching is also an essential part of query languages for structured text. For example languages described in [KM93, CCB95, NBY95, BCD+95, JK95], use patterns to specify conditions which integrate content and structure constraints.

6.4 Manual transformations

In manual transformations the user modifies individual trees and defines both the rules and the control flow for transformation. Only the environment for the user (a specification language or an editor) can be generated as a tool for the user. To this group we classify batch-oriented programmes made using some programming language and tree editors that are able to modify a tree interactively.

6.4.1 Filters

As a filter we understand a sequence of filtering operations whose execution is manually controlled. The process is implemented usually by a programme that is made using a common or a specific programming language. Single transformations are specified as a set of

```
pattern { action }
```

rules. These rules are contained in the control flow sentences of the programme.

The filter programme takes as its input a tree and produces as its output a new tree. The programme uses variables to store intermediate information like deleted parts of the tree that should be inserted into another place in the tree or parts that should be sorted, etc. By writing a programme the user controls the definition for the transformation but also the process that makes the transformation. The programme can search the modified parts of the tree in two different ways. Either it travels the tree in some predefined order and makes a change when it finds a part that needs to be modified, or it makes a sequence of modifications according to some specification and searches the places for these separately for every change as long as places for modifications can be found.

Transformations implemented by filter programmes are often specific for a set of documents, for example for SGML as in [DIS94], for a specific document as in [BBT92] for the New Oxford English Dictionary, or for a tree structured representation using a special technique like in Scrimshaw [Arn93] and TXL [CC93]. The specifications of the used languages contain a possibility to define content and structure based conditions for nodes of a tree for the pattern matching. Thus, they can be used also as query languages for specified parts of a tree in the retrieval of documents.

6.4.2 Tree editors

An interactive tree editor, for example such as in [Des88], allows the user to make modifications in any single part of a tree. The editor represents a tree on the screen and the user modifies the tree with the use of editor commands or a mouse. The user can point to any part of a tree, thus, there is no need for any common specification for the change in the tree. This allows any modifications, it depends only upon the intelligence of the editor. The editor can allow any kinds of modifications, but it can be built so that it ensures, for example, that the new structure is according to a new grammar. There is no way to carry out similar modifications to all trees of one type using an editor for tree transformations. All the trees are processed individually.

7 Tree transformations in parse trees for structured documents

7.1 Grammars in transformation definitions

The notation of a context-free grammar in Backus-Naur form (BNF) [AU72] used in transformation methods in the previous section is not always powerful enough to define all representations of structured documents [Fur87]. The most used form for a document grammar, for example in the document type definition (DTD) of SGML documents, is an extended BNF which allows regular expressions defined in the right-side of the production. Special symbols are used for *optional*, *alternative* and *iterative* nonterminals (separately for zero or more and one or more elements), and *unnested groupings* of grammar elements. When symbols for these are used in grammars and grammars are used by transformations or transformation programmes are generated from grammars their semantics has to be defined unambiguously. For example, a missing element for an optional element and alternative elements have to be processed without side effects. The notation for lists makes it possible to process all elements of a list only as an undivided part of a document. It means that elements of a list cannot be located separately; they are processed similarly from the first to the last element in the same order. If the transformation needs to add some separators between elements of a list there has to be a special notation for them and the transformation have to care of the right position of these separators in the parse tree. A group of elements should be processed as if it would be an extra implicit nonterminal on the right-hand side of a grammar production.

In structured documents defined by context-free grammars two kinds of terminals are used. Transformation methods need handle them differently. Terminals which are represented as character strings in a grammar are called *syntactic terminals*. They remain same for every document for a grammar and define, for example, codes for formatters and separators of the content. A term "terminal" in document grammars means usually them and very often transformations will change them. Terminals for variable data in a document are called *user terminals*. The actual content text for them is supposed to be given by the user when a document instance is created. Their content is not fixed in the grammar and they will usually remain unchanged in transformations. Usually in grammars user terminals are represented by special nonterminals called *content nonterminals* for every different type of data, for example, for character strings, bitmaps, tables, etc. There are no rules for content nonterminals in the grammar, they are considered as undivided parts of the content. The content for these user nonterminals can also be generated by the system.

Names of nonterminals of a context-free grammar are used to define structural elements of documents. For the flexible use of structured documents the user has to be allowed to define the names of elements so that they suit his or her purposes. This possibility induces to a situation that in different documents *different labels* mean the same thing in different parse trees. If the transformation method does not allow different labels for associated nonterminals a string homomorphism is needed on nonterminals.

Transformation methods using grammars usually suppose that the parse tree of a document does not contain nonterminal leaves. Thus, the structures of documents has to be fully derived. If *uncomplete* documents will be transformed methods need an extra action

for every nonterminal of a grammar to process an atomic nonterminal.

An other used definition for structured documents is an attribute grammar [AM91, Bar93] that allows each node of a parse tree for a context-free grammar to be attached with *attributes* whose values are computed during the processing of the tree. Attributes express the context-sensitivity relations in parse trees for a context-free grammar. Attributes are used in structured documents to express references between elements, to contain system generated values (for example chapter numbers) and to define default values for nodes or content or context dependent formatting codes for nodes. For example, an SGML document contains attributes for default values and for internal and external references. Transformation methods that are specific for attribute grammars can be used in cases were processing of attribute values is necessary. However, if the transformation does not change attributes, instead of using attributes connected to nodes of parse trees, attributes can be expressed as nodes in trees [BCD+95] and defined as nonterminals of a context-free grammar. In this case, transformations can be defined using methods suitable only for context-free grammars.

7.2 Methods for differences in terminals

A simple syntax-directed translation schema (SSDTS) can be used to define transformations between structured documents having differences in terminals. These transformations need to add and delete terminal nodes. The terminal nodes are defined by syntactic terminals of grammars. Other nodes which define the actual content text and the structure of a document are supposed to stay same and be in the same order. Thus, grammars for documents differ only according to their syntactic terminals and each production of a grammar have a corresponding production in another grammar and productions contain same nonterminals in the same order. All the information needed for the transformation of a parse tree is contained in an SSDTS. To define the schema two grammars, an input grammar for existing documents and an output grammar for transformed documents are needed. An output grammar, if one does not exist, is made from an input grammar by a tree editor which allows only the modification of terminals. This guarantees that the structure remains same. A programme made for Algorithm 6.1 in Section 6 will generate the SSDTS from the two grammars. A batch-oriented transformation programme can be made according to the algorithm in [AU72], Algorithm 6.2 or by making a compiler which compiles a transformation programme from an SSDTS. The methods generate programmes that make document transformations automatically. The SSDT can also be used interactively to execute incremental transformations in the input phase of the structured text. The input tree is generated syntax-directed on the screen of a computer and an internal representation of a document is generated according to an SSDTS. The transformation via SSDTS is well defined, difficulties in applying it to documents depend on how unambiguously locations of terminals can be defined in grammars.

Example 7.1. Transformation via SSDTS. The example shows how to add separators for structure elements and codes for different formatters to an internal representation of a bibliographic article reference in ACMTOIS. The aim is to produce the layout representation for the LaTeX formatter. The parse tree for an article reference in a reference list of a manuscript for ACMTOIS is represented in Appendix 4 (Example 1) and defined by the structure grammar in Appendix 5. From this parse tree we can generate the layout

represented in Appendix 4 (Example 1) using the following SSDTS rules

```
article  --> bauthors year title journal vol number pages,
             bauthors " " year ". " title ". {\it " journal " "
             vol "}, " number ", " pages "."
bauthors --> bname+, bname+{", ","," AND "}
bname    --> last ini+, last ", " ini+{"."} "."
pages    --> firstp lastp, firstp "-" lastp
```

Terminals for separators and codes are represented in double quotes. Separators in lists are represented in braces after the name of the nonterminal, those characters will be placed only between list elements, not after the last element. To produce the journal name using a slanted font we have used the codes for the LaTeX formatter. The name of authors should have been written in capital letters originally.

A document tree can also be represented as an external string representation for the document. The string representation contains unambiguous definitions of the structure with the use of interleaved tags in the content. These tags correspond to terminals of the grammar. In this case the transformation for documents with terminal differences can be implemented also by a filter programme. The filter is a string homomorphism on terminals of the input and output grammar and the filter transformation only substitutes terminals for new terminals.

7.3 Methods for order differences in nonterminal children of a node

A syntax-directed translation schema (SDTS) offers a method to define transformations between structured documents whose all elements are the same but where subelements of associated elements can be in a different order. The transformation moves subelements from a place to another. In this case the orders of same nonterminals on right-hand sides of corresponding grammar productions are different and the moving of elements is made among children of a node in a parse tree. The automatic and interactive implementation of transformations managing order differences can be done as explained for the SSDTS in the previous subsection. A tree editor to create an output grammar from an input grammar allows the user to move nonterminals in the right-hand side of a grammar production, but not to delete them. An SDTS can define also transformations for terminal differences if those exist in documents.

Example 7.2. Transformation via SDTS. Let us consider how the internal representation of an article reference in a bibliography of our manuscript (grammar in Appendix 5) can be transformed to an article reference of a manuscript for EPODD (grammar in Appendix 5). The structures differ in two places. Subelements of the `article` elements and subelements of the `name` elements are in a different order. The transformation can be defined using the following SDTS rules:

```
article  --> cite bauthors title journal year vol number pages,
             cite bauthors title journal vol number pages year
bauthors --> bname+, bname+
bname    --> ini+ last, last ini+
pages    --> firstp lastp, firstp lastp
```

If we need to add separators and formatting codes for the LaTeX formatter to form the layout structure for an article reference in EPODD to our manuscript we can define transformations using only an SDTS, not first using an SDTS for order differences and then an SSDTS for terminal differences. The rules will be in this case as follows:

```
article  --> cite bauthors title journal year vol number pages,
             cite ". " bauthors ", '"  title "', {\it " journal
             "}, {\bf " vol "}(" number "), " pages " (" year ")."
bauthors --> bname+{", ",", and "}, bname+
bname    --> ini+ last, last ini+{"."} ". "
pages    --> firstp lastp, firstp "-" lastp
```

The denotations has been explained in Example 7.1.

7.4 Methods for existence differences in nonterminal children of a node

The transformation between structured documents having local existence differences in their structures needs either to insert or delete elements, move or copy elements from a place to another place, add or remove levels of nested elements, or reparse elements with a new grammar. At the same there can be terminal and order differences, too. As we defined in Section 5 local existence differences of nonterminal children of nodes will influence inside restricted parts of trees that we called local associated parttrees. These kinds of local differences can be managed using different methods depending on the nature of the needed modification.

An extended syntax-direct translation schema (ESDTS) allows the definition of transformations that will delete, insert and reorder nonterminal children of a node as well as change terminal nodes. A deletion of an element must mean the deletions of its subelements, and an insertion of an element must mean its addition as an atomic element, without subtrees. The ESDTS needs an output grammar for the transformation and guarantees that the new structure is according to the new grammar. To generate an output grammar for a document from an input grammar a similar tree editor as for SSDTS or SDTS can be implemented. However, this editor allows also the deletion and insertion of nonterminals. ESDTS rules are defined from a pair of productions of the input and output grammar with the use of an interactive programme generated for Algorithm 6.3. The batch-oriented transformation using an ESDTS can be implemented by a programme for Algorithm 6.4 or by making a compiler which generates the transformation programme from the ESDTS. The programme processes the parse tree in the preorder.

A descending tree transducer allows the modification of the structure of a tree locally adding, deleting, copying and duplicating nonterminal nodes, renaming labels of nonterminal nodes and replacing terminal nodes. Subtrees of a deleted node can be deleted or moved to be subtrees of other nodes. An inserted node can be without subtrees or subtrees for it can be subtrees of other nodes or their copies. A tree transducer does not use grammars to define the transformation, thus, to guarantee that the structures are according to grammars, the rules must be made using grammars. For a descending tree transducer, local associated parttrees can be defined using two sets of productions from the input and output grammar, respectively. A rule generation programme made according to Algorithm

6.5. needs an interactive rule editor that allows the user to represent the sets of grammar productions for local parttrees as trees and associate their leaves. Later in this subsection we will describe how this kind of an editor would work. A batch-oriented descending tree transducer can be implemented by making a programme from Algorithm 6.6 but also by making a compiler that compiles the term rewriting rules to a transformation programme. The tree transducer traverses an input tree in the preorder and processes only nodes which have associated nodes in an output tree.

Tree pattern matching/replacement methods modify structures of trees similarly to a descending tree transducer but are able to recognize any local tree structure in a document instance according to its structure and content and to replace it by a new instance. Tree pattern matching/replacement methods do not use grammars to define transformations. In the tree pattern matching/replacement methods, local associated parttrees are defined as parts of parse trees of document instances, possibly also requiring that some content conditions are true. Patterns are described using the notation of the selected method. To guarantee that structures are according to grammars, structure grammars can possibly be used as a help. Anyway, the user is responsible for the forms of patterns. The user is also responsible for the order in which the rules are applied in the input tree. If a control is not defined, rules are applied to a parse tree of a document as long there are matching structures.

A filter programme allows to apply operations to the searched elements. A filter programme for local differences are needed only when the transformation concerns the content of a document and a modification based on the content is required. Thus, filters are needed, for example, if local elements should be sorted or reparsed. The user is responsible for the definition of the location of the elements and for the writing of the needed procedures.

Using an ESDTS and a descending tree transducer the processing of lists for the same nonterminal is possible only as a whole lists. Separate list elements cannot be defined differently. For this reason, lists in local associated parttrees have to exist in leaves and must be associated to other lists. Because lists cannot be expanded into single elements in parttrees according to these methods another element in the parttree cannot be associated with a single element of a list. These kinds of specifications can be made by tree pattern matching/replacement methods or manual methods which can select specified elements (in the beginning or at the end of the list, or according to a place or the content of an element).

If transformations delete a part of the content of a document the original document cannot be recreated from the transformed document. In some cases the reverse transformation is needed too. In [Nik90] it has been represented a pair of transformations based on a partial SDTS and its complement SDTS that allows to delete nodes from the parse tree and reconstruct the original parse tree.

Rule editor for a tree transducer. A programme to generate rules for a tree transducer according to Algorithm 6.5 needs an editor to form local associated parttrees and term rewriting rules. Such an editor allows the user to display input and output parttrees parallel. The structure of a parttree is defined as a tree that describes a set of grammar productions. In the visualization of a parttree the label of each node corresponding to a grammar element of a production can be written on its own line and the parent-child relationships are expressed by indentations. The rule generation editor starts by displaying a pair of productions of the input and output grammar for the root nonterminals of parttrees.

For example, if we consider the grammars for the front part of the manuscript of this work (the grammar in Appendix 5) and a manuscript in EPODD (the grammar in Appendix 5) and want to generate a rule for the `front` nonterminal, the editor will select productions for `front` in grammars and display them in the following form.

```
OURS                          EPODD

front                         front
   title                         title
   authors                       authors
   [footnote]
```

To represent the requested structure the user selects a leaf of a parttree and expands it according to a grammar production for the label of a leaf. For example, if the user has expanded the `authors` leaf in the EPODD tree the result parttrees would be following:

```
OURS                          EPODD

front                         front
   title                         title
   authors                       authors
   [footnote]                       names
                                    affiliations_addresses
```

The user adds new structures to parttrees according the grammar productions until he/she can find trees where nonterminals can be associated in the requested way. The editor can check that the associated parttrees do not grow too big. In this example, no further expansions are needed and the system asks the user to mark the associated leaves of trees as depicted below.

```
OURS                          EPODD

front                         front
   title        1                title                      1
   authors      2                authors
   [footnote]                       names                   2
                                    affiliations_addresses  2
```

The editor generates a term rewrite rule for the `front` nonterminal from this representation. The unmarked leaf node in the parttree for our grammar is `footnote`; its content is supposed to be removed. If we denote by $X_i, (i \geq 1)$ the variables for all leaves in the parttree for our manuscript in the order from the left (in the above picture from top to down) and if we denote the states of associated output nonterminals by $\mathrm{q}_{nonterminal}$ we will get the following rule:

$$\mathrm{q}_{front}(\mathrm{front}(X_1,X2,[X3])) \rightarrow$$
$$\mathrm{front}(\mathrm{q}_{title}(X_1),\mathrm{authors}(\mathrm{q}_{names}(X_2),\mathrm{q}_{affiliations\_addresses}(X_2))$$

Example 7.3. Local transformations. Section 4 represented structure differences in manuscripts of eight journals. In this example we illustrate how we transformed parts of the internal representation of the manuscript of this report to parts of the internal representation of manuscripts of those journals. We will discuss the front and back parts and bibliography. Structures of parse trees for the internal representations of those parts for different journals are represented in Appendices 3 and 4. Grammars for our work and some of the journals are represented in Appendix 5.

An ESDTS was enough to define the transformation to a manuscript for CJ and SPE. The differences were only in the order or existence of nonterminals on the right-hand side of grammar productions. As an example, let us consider the transformation definitions from the manuscript of this report to a manuscript for CJ. When we transform front parts, elements to be removed are the footnote with all its contents and a reference to it. Added elements of the front part are a short title, postal addresses of authors and telephone and fax numbers of the correspondence author (the author to whom all connections from the journal will be made). We generated the following rules for the ESDTS automatically from the grammars:

```
front       --> title authors [footnote],
                title shorttitle authors
authors     --> corr_author author*, corr_author author*
corr_author --> name [fref] affiliation email,
                     name affiliation postal email tel fax
author      --> name [fref] affiliation email,
                name affiliation postal
affiliation --> dept university, dept university
footnotes   --> grant+,""
```

In the transformation of bibliographic references some elements are deleted (number of a journal for an article, editors for a conference article), added (publishing place for a book or a conference article) or reordered (year element in all types of references). The following ESDTS will define the transformation for CJ:

```
article            --> cite bauthors title journal year vol
                       number pages,
                       bauthors year title journal vol pages
book               --> cite bauthors title publisher year,
                       bauthors year title publisher place
conference_article --> cite bauthors title conference year beditors
                       pages,
                       bauthors year title conference place pages
bauthors           --> bname+, bname+
bname              --> last ini+, last ini+
pages              --> firstp lastp, firstp lastp
beditors           --> bname+,""
```

A descending tree transducer was used to carry out transformations of the front part of a manuscript of this report to the front parts of manuscripts for JASIS, CACM, JCSS,

IS and EPODD. The tree transducer was used because associated nonterminals exist in different grammar productions. The generation of grammar rules was made by hand, but was quite easy. (An example about the tree transducer using EPODD as the target journal is given in Appendix 6.) An issue serves, however, to be mentioned. As we can see, for example, from the productions of our grammar and EPODD grammar in Appendix 5, in our grammar `name` and `email` are represented as subelements of an `author` element and `dept` and `university` as subelements of an `affiliation` element but in EPODD grammar each of them is represented as an element of their own lists which are nested by an `affiliations_addresses` element. The transformation from our manuscript to a manuscript in EPODD was possible with the use of a tree transducer because in the rule for a `front` nonterminal a nonterminal `authors` in our grammar can be associated both with the nonterminal `names` and the nonterminal `affiliations_addresses` in the EPODD grammar (see the example in the description about the rule editor for a tree transducer above). Then in the further rules only one nonterminal from the nonterminals `name`, `dept`, `university` and `email` in the right-hand side of the `author` production can be associated to the nonterminals `name`, `dept`, `university` or `email` respectively. Thus, the transformation was able to be carried out using a tree transducer, although it divided a list into several lists. If we had needed to make a transformation to the opposite direction, a tree transducer is not powerful enough. In this case, the transformation needs to collect subelements of an element in one list from several lists. The bibliography part of our manuscript can be transformed to the bibliography parts of JASIS, CACM, JCSS, IS and EPODD manuscript also with the use of a tree transducer. Actually, bibliography parts for all the other journals except CACM could be transformed also with the use of an ESDTS.

The transformation to a manuscript for ACMTOIS should parse names of authors in our manuscript to the initials and the last name of the author when names were copied to the back part of a manuscript for ACMTOIS. This would need a filter programme to apply reparsing operation to the content of `name` element. We made the transformation with the use of a tree transducer, in which case the transformation generated new atomic elements for them. The user should add the right content later.

7.5 Methods for global differences

The transformation between structured documents having global differences in their structures needs either to reorder elements, delete or insert a set of elements, and move or copy elements or a set of elements from one place to another which does not situate inside a local part of a tree. Typical global modifications concern elements that are scattered among other elements and should be collected into a certain part of a document, certain parts of a document should be spread among other elements, and structures that should be flattened removing hierarchical elements or heightened by adding hierarchical levels. Also the modifications will concern elements of a list which should be divided to several lists according to its elements or content of elements, or elements of several lists which are merged into one list, possibly sorting into a new order, and modifications which cut a tree to many trees.

Filter programmes and tree editors allow modifications that are not restricted to some parts of parse trees and can be used in previous kinds of transformations. In addition, the modifications can be made by local methods (the SDTS, the ESDTS, tree transducers,

or tree pattern matching/replacement methods) if the modification can be processed incrementally applying the methods sequentially so that the output of previous transformation is the input of the next transformation. Anyway, global transformations are always made only in the control of the user. Either the user edits every single document separately, writes the filter programme that makes the transformation in its entirety, or creates grammars or transformation rules for the incremental steps of the transformation for an SDTS, an ESDTS, a tree transducer, or a tree pattern matching/replacement method. A filter programme must define the control flow of the transformation as well as procedures to change the matched parts. The pattern matching should be based on the structure, on the content, or on the integration of these as well as in the order or quantity of elements. The filter must be able to save structures temporally and replace them later and process a document more than one time. In addition, the filter programme should test that the output document is valid for the output grammar, if such exists, and inform the user about errors.

A tree editor for a structure document should check the validity of document against the input grammar and produce valid documents for the output grammar if such exists. Errors should be marked to the document and allowed the user to correct them. The validity checkings during the editing process should be made when the user wants and always at the end of the editing. The editor should be possible to spread an executed single modification in a tree to all similar structures, thus, taking care of the situations that the same parts have the same structures in the document.

If the user does not have an output grammar after making a transformation using filter programmes or editors, the system could use methods that can generate the grammar of a document from the transformed document [AMN93, AMN94, Aho94].

8 Model for a document transformation system

The previous sections showed that according to a difference between the representations for structured documents we are able to decide what type of a transformation method can be used, and further, for which types of differences the transformation can be carried out automatically. As we saw, if the structure does not change the transformation can be made automatically. If the hierarchical structure changes, transformations can be managed automatically only in specific cases which, however, need the user's help to define the transformation rules.

A system where the common use of different transformation methods is possible needs an universal representation for documents. A parse tree for a context-free grammar in extended BNF [AU72, Fur87] can be selected as such. The internal representation of a document is a parse tree for a grammar that is called a skeleton grammar. It contains only nonterminals which define the hierarchical structure of a document. The skeleton grammar is not used, for example, to generate or parse the string representation of the parse tree; it is used only by the system. Other representations of a document are regarded as external representations used by the user of the system. Representations which the user sees in the input of a document or on the paper as well as representation for exchanging documents are such. Grammars for parse trees of external representations are skeleton grammars added by terminals.

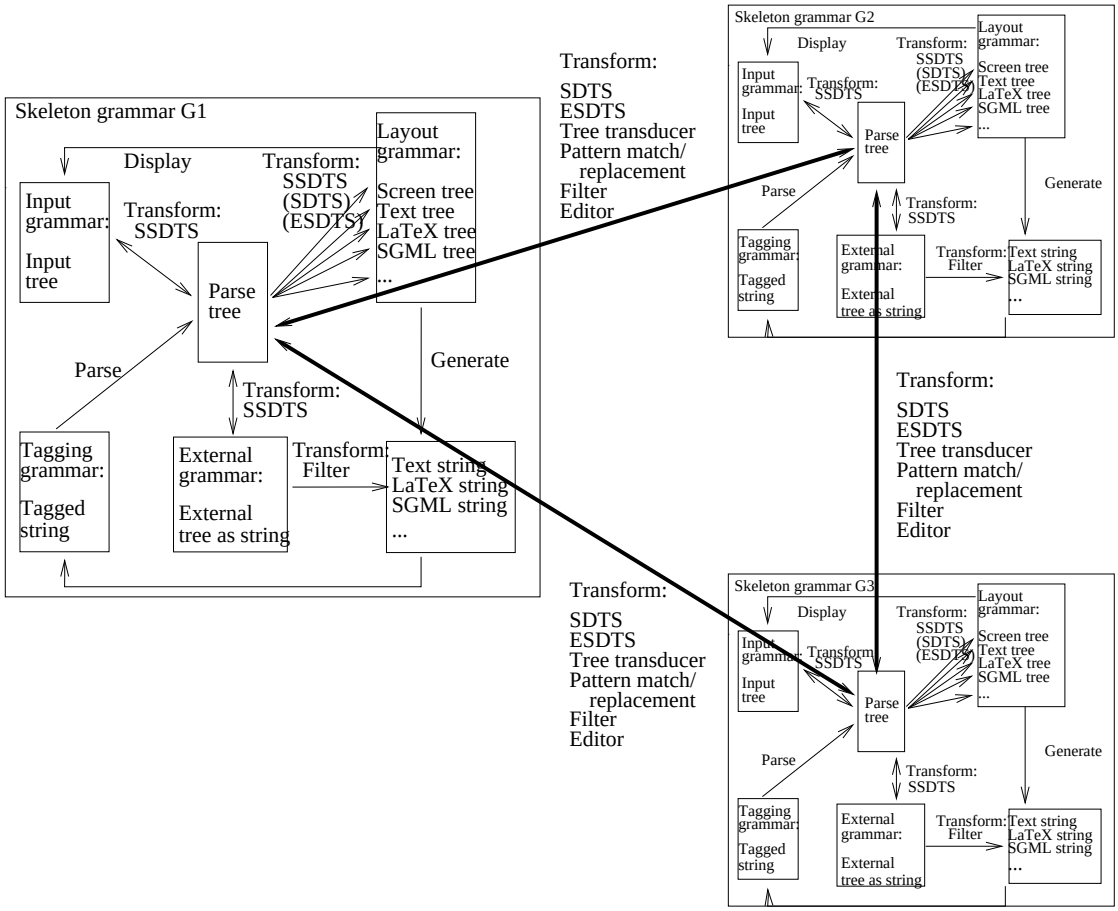


Figure 8.1. Transformations between different representations of documents.

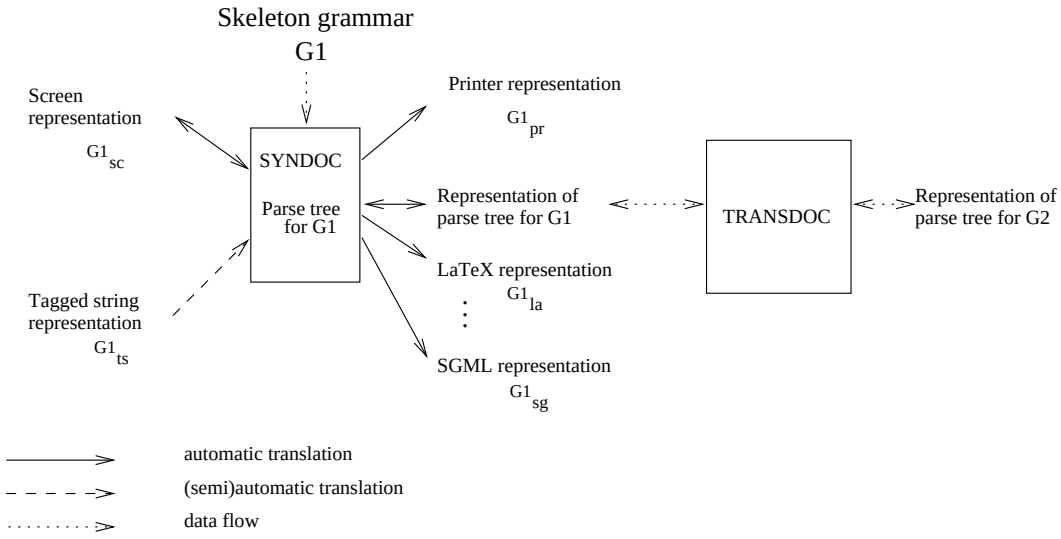


Figure 8.2. An model for a document transformation system.

A framework for transformations between different representations of a document is represented in Figure 8.1. Different document representations are grouped according to their common skeleton grammar; all representations for a same skeleton grammar form a cluster. In such a cluster all transformations between different representations are defined by simple syntax-directed translation schemata or made by a homomorphic filter. Some transformations are made by generating an output representation straight from a parse tree for the representation in the preorder, or by parsing a string representation according to a grammar to form a parse tree. The methods leave the actual structure unchanged. Transformations between representations in different cluster are executed by first transforming the internal representation for a skeleton grammar to a new internal representation according to a new skeleton grammar and then using methods available inside the cluster for the new skeleton grammar. In the case that the transformed representation is only meant to generate a layout representation for the printing of a document and not for further processing, and the transformation needs to reorder elements or delete elements, the transformation can be made by an SDTS and ESDTS which change the structure defined by the skeleton grammar.

Our model to a document transformation system separates transformations that are simple and easy to automatize from transformations that need more sophisticated methods. We will divide the system into two tools (Figure 8.2), one tool that allows to make only transformations that do not change the structure and another to make transformations from one structure to another. In the first tool a simple syntax-directed translation is used. The second tool contains a set of available methods and offers a possibility of analyzing differences between an old and new representation and by using different methods according to the results of the analysis.

The first tool called SYNDOC, a SYNtax-directed DOCument processing system, contains those transformations which produce representations for the user of the system. Inputting, outputting, storing, importing and exporting of a document in different forms

without changing the hierarchical structure need transformations in this group. In this tool transformations are always between the internal representation and an external representation. They are defined automatically from the grammars for the input and output representations and are executed automatically or semiautomatically. Figure 8.2 describes transformations that are automatical by solid arrows and transformations which may need help from the user (because of the ambiguous notations for the structure [MKNS89]) by dashed arrows. Dotted arrows describe data flows.

The second tool called TRANSDOC, a TRANSlation system for DOCuments, manages transformations which produce representation from an internal representation to another internal representation with a different structure. TRANSDOC gets as its input a representation for a parse tree and produces a new parse tree. The methods available in TRANSDOC can be based on the syntax-directed translation, but other methods are used, too. The only assumption made to methods is that they create a parse tree for a grammar from a parse tree for another grammar. Each available method contains its own tools to help the user to define the transformation definitions. TRANSDOC needs methods to compare grammars to find out their differences and to decide which method to use.

9 About implementation of a document transformation system

In our earlier works we have developed the prototype for the SYNDOC [KPV94, KS95]. The prototype is implemented in SICStus Prolog with the Graphics Manager library to create the X-windows user interface [Swe93] and in Tcl (version 7.4) and Tk toolkit (version 4.0) script language [Ous93]. A simple syntax-directed translation schema is used for the syntax-directed input of documents and for the generation of the outputs for documents [KP91, KPV94]. A grammar directs the input. Different layout representations are made by transformation programmes which are generated from corresponding transformation grammars by a compiler.

The implementation of transformation methods for TRANSDOC has started as a part of SYNDOC. The system does not support the selection of methods, the user selects a method her/himself. The implementation of transformations using an ESDTS is described in [KP93]. The implementation of the ESDT can be used also for transformations via an SDTS. The implementation consists of a generator that forms a transformation grammar from old and new skeleton grammars and a compiler that compiles a transformation grammar to a transformations programme. Different labels are allowed for associated nonterminals and the transformation processed also uncomplete documents. In this work we have implemented transformations with the use of a tree transducer. In the similar way as in the implementation of an ESDTS, a programme for a tree transducer is compiled from a term rewriting system of the tree transducer. The current implementation does not contain the generator programme for the transducer rules, the rules are generated manually. In tests of the method, tree transducers were used to transform the front part of this manuscript to front parts of manuscripts for JASIS, CACM, JCSS, IS and EPODD (journals mentioned in Section 4). All these transformations needed a tree transducer, CACM even a more powerful method as described in Example 7.3. Appendix 6 describes how the transformation

from our manuscript was made to a manuscript of EPODD journal.

10 Conclusion

Transformation of documents occurs very often in connection with the document processing. The reuse of documents has made it even more important because today's computer systems contain a huge amount of information in electronic form and using nets like Internet it is available worldwide. And this information is represented in very many ways. Automatic transformations of these many representations are only possible if the system has some information about documents that it processes. A structure defined by a grammar contains sufficient information.

This work has concentrated in the transformation methods which suit tree-structured documents. Structures are defined by context-free grammars and different document representations are parse trees for their grammars. Transformations either change the hierarchical structure of a document or leave it unchanged. For a purpose of developing a universal framework for document transformations the report first analyzed an example using manuscript styles in different scientific journals to find out the kinds of differences which exist in document representations. To get a basis for method selections we defined a classification for differences in parse trees. After a survey to different tree transformation methods we analyzed which methods could be used flexibly for each kind of difference classes and, finally, presented a model for a transformation system for structured documents.

In such a transformation system, methods like a simple syntax-directed translation schema, a syntax-directed translation schema, an extended syntax-directed translation schema, a descending tree transducer, and a language to generate tree pattern matching based filters as well as a tree editor, form a powerful environment for making modifications in documents whose structures are defined by context-free grammars. The implementation of each method contains its own tools to generate transformation programmes automatically or with help of the user from the grammars of different representations. The transformations in the system produce documents according to the output grammar of the transformation. Some of transformations between different document representations are such that they can be carried out using only a specific method, otherwise the user needs to select a method. For the selection the user would need a programme that analyses a pair of grammars and helps the user to select the suitable method according to differences in documents.

In this work we continued the implementation of this kind of a system as a part of our prototype for a syntax-directed document processing system. Current implementation contains the syntax-directed translation methods and some functionalities for transformations with the use of a tree transducer. The further development of a document transformation system will be the issue for our future research.

Acknowledgements

The research for this report was carried out as a subproject of the research project on Structured Documents and Text Databases financed by the Academy of Finland and supported by grants of the Saastamoinen Foundation and the Emil Aaltonen Foundation, which are

gratefully acknowledged. The authors would like to thank Magnus Steinby for his help during this work and Frank Tompa for reading the manuscript, for representing comments which remarkably improved the report, and for the possibility of the other author to work in the University of Waterloo.

References

[ABB+84] M. Ahlsen, A. Björnestedt, S. Britts, C. Hulten, and L. Söderlund. Making type changes transparent. Technical Report No. 22, University of Stockholm, SYSLAB, Sweden, 1984.

[AD76] A. Arnold and M. Dauchet. Bi-transductions de forets. In S. Michaelson and R. Milner, editors, *Automata, Languages and Programming, Third International Conference*, pages 74–86. Ediburgh University Press, Edinburgh, UK, 1976.

[Add90] Addison-Wesley. *PostScript language reference manual*, 1990.

[Aho94] H. Ahonen. Generating grammars for structured documents using grammatical inference methods. Master's thesis, Department on Computer Science, University of Helsinki, Finland, 1994.

[AM91] H. Alblas and B. Melichar, editors. *Attribute grammars, applications and systems.* Lecture Notes in Computer Science 545, Springer Verlag, Berlin, 1991.

[AMN93] H. Ahonen, H. Mannila, and E. Nikunen. Interactive forming of grammars for structured documents by generalizing automata. Technical Report C-1993-17, Deparment of Computer Science, University of Helsinki, Finland, 1993.

[AMN94] H. Ahonen, H. Mannila, and E. Nikunen. Generating grammars for SGML tagged texts lacking DTD. In *Second Workshop on Principles of Document Processing, PODP'94*, 1994.

[AQ92] E. Akpotsui and V. Quint. Type transformations in structured editing systems. In C. Vanoirbeek and G. Coray, editors, *EP92*, pages 27–41. Cambridge University Press, Cambridge, 1992.

[AQR93] E. Akpotsui, V. Quint, and C. Roisin. Type modelling for document transformation in structured editing systems. Submitted to Mathematical and Computer Modelling, 1993.

[Arn93] D.S. Arnon. Scrimshaw: A language for document queries and transformations. *Electronic Publishing - Origination, Dissemination and Design*, 6(4):385–396, 1993.

[ASU86] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers, Principles, Techniques, and Tools.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.

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

[AU73] A.V. Aho and J.D. Ullman. *The theory of parsing, translation, and compiling, Vol. II: Compiling.* Prentice-Hall, Inc., Englewood Cliffs, N.J., USA, 1973.

[Bak78a] B.S. Baker. Generalized syntax directed translation, tree transducers, and linear space. *SIAM J. of Computing*, 7(3):376–391, 1978.

[Bak78b] B.S. Baker. Tree transducers and tree languages. *Information and Control*, 37:241–266, 1978.

[Bar93] K. Barbar. Attributed tree grammars. *Theoretical Computer Science*, 119:3–22, 1993.

[BBT92] E. Blake, T. Bray, and F.W. Tompa. Shortening the OED: Experience with a grammar-defined database. *ACM Transactions on Information Systems*, 10(3):213–232, 1992.

[BCD+95] G.E. Blake, M.P. Consens, I.J. Davis, P. Kilpeläinen, E. Kuikka, P.-Å. Larson, T. Snider, and F.W. Tompa. Text / relational database management systems: Overview and proposed SQL extension. Technical Report CS-95-25, University of Waterloo, Computer Science Department, 1995.

[BJB90] A.L. Brown Jr. and H.A. Blair. A logic grammar foundation for document representation and document layout. In R. Furuta, editor, *EP90, Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography*, pages 47–64. Cambridge University Press, Cambridge, 1990.

[BJWB92] A.L. Jr. Brown Jr., T. Wakayama, and H.A. Blair. A reconstruction of context-dependent document processing in SGML. In C. Vanoirbeek and G. Coray, editors, *EP92*, pages 1–25. Cambridge University Press, Cambridge, 1992.

[BKKK87] J. Banerjee, H.-J. Kim, W. Kim, and H.F. Korth. Schema evolution in object-oriented persistent databases. *ACM SIGMOD Record*, 16(3):311–321, 1987.

[BLCLS94] T. Berners-Lee, R. Cailliau, H.F. Luotonen, A. Nielsen, and A. Secret. The world-wide web. *Communications of the ACM*, 37(8):76–82, 1994.

[Bor85] A. Borgida. Language features for flexible handling of exceptions in information systems. *ACM Transactions of Data Base Systems*, 10(4):565–603, 1985.

[CC93] J.R. Cordy and I.H. Carmichel. The TXL programming language syntax and informal semantics, version 7. Technical Report 93-355, Department of Computing and Information Science, Queen's University at Kingston, Canada, 1993.

[CCB95] C.L.A. Clarke, G.V. Cormack, and F.J. Burkowski. An algebra for structured text search and a framework for its implementation. *The Computer Journal*, 38(1):43–56, 1995.

[CFZ82] B. Courcelle and P. Franchi-Zannettacci. Attribute grammars and recursive program schemes, I and II. *Theoretical Computer Science*, 17:163–191, 235–257, 1982.

[CP89] J.R. Cordy and E. Promislow. Specification and automatic prototype implementation of object-oriented concepts using the TXL dialect processor. Technical Report 89-251, Department of Computing and Information Science, Queen's University at Kingston, Canada, 1989.

[Des88] P. Desain. Tree Doctor, a software package for graphical manipulation and animation of tree structures. In G. van der Veer and G. Mulds, editors, *Human-Computer Interaction: Psychonomic Aspects*, pages 223–236. Springer Verlag, Heidelberg, 1988.

[DIS94] DIS 10179.2. *Information technology - Text and office systems - Document Style Semantics and Specification Language (DSSSL)*, 1994.

[Eng77a] J. Engelfriet. Bottom-up and top-down tree transformations - a comparision. *Mathematical Systems Theory*, 9(3):198–231, 1977.

[Eng77b] J. Engelfriet. Top-down tree transducers with regular look-ahead. *Mathematical Systems Theory*, 10:289–303, 1977.

[EV85] J. Engelfriet and H. Vogler. Macro tree transducers. *Journal of Computer and System Sciences*, 31:71–146, 1985.

[EV88] J. Engelfriet and H. Vogler. High level tree transducers and iterated pushdown tree transducers. *Acta Informatica*, 26:131–192, 1988.

[EV91] J. Engelfriet and H. Vogler. Modular tree transducers. *Theoretical Computer Science*, 78:267–303, 1991.

[FE81] J.P. Fry (Ed.). Conversion technology, an assessment. *ACM SIGBDP Data Base*, 12&13(4&1):39–61, 1981.

[FFK92] H. Fanderl, K. Fischer, and J. Kämper. The open document architecture: From standardization to the market. *IBM System Journal*, 31(4):728–754, 1992.

[Fra93] Frame Technology Corporation. *FrameMaker*, 1993.

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

[Fur87] R. Furuta. A grammar for representing documents. Technical Report UMIACS-TR-87-67, University of Maryland, College Park, Maryland, USA, 1987.

[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.

[GKL94] D. Garlan, C.W. Krueger, and B.S. Lerner. TransformGem: Automating the maintenance of structure-oirented environments. *ACM Transactions on Programming Languages and Systems*, 16(3):727–774, 1994.

[GM93] J.A. Gawkowski and S.A. Mamrak. A universal framework for data transformation. Technical Report OSU-CICRC-11/93-TR39, Department of Computer and Information Science, The Ohio State University, 1993.

[Gol90] C.F. Goldfarb. *The SGML Handbook.* Oxford University Press, Oxford, UK, 1990.

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

[GT87] G.H. Gonnet and F.W. Tompa. Mind your grammar: a new approach to modelling text. In *Proceedings of the 13th International Conference on Very Large Data Bases*, pages 339–346, 1987.

[HN86] A.N. Habermann and D.S. Notkin. Gandalf: Software development environments. *IEEE Transactions on Software Engineering, SE-12*, 12:1117–1127, 1986.

[HO82] C.M. Hoffman and M.J. O'Donnell. Pattern matching in trees. *Journal of the ACM*, 29(1):68–95, 1982.

[Ins92] Institut for Applied Information Technology, German National Research Center for Computer Science, Schloss Birlinghoven, Germany. *The qwertz SGML Document Types, Version 1.2, Reference Manual*, October 1992.

[ISO86a] ISO 8813. *Office Document Architecture (ODA)*, 1986.

[ISO86b] ISO 8879. *Information processing - Text and office systems - Standard Generalized Markup Language (SGML)*, 1986.

[JK95] J. Jaakkola and P. Kilpeläinen. Sgrep - a tool to search structured text. Technical report, Department of Computer Science, University of Helsinki, Finland, 1995. Manuscript, unpublished.

[Kae87] M.J. Kaelbling. *Braced languages and a model of translation for context-free strings: Theory and practice.* PhD thesis, The Ohio State University (UMI 8804059), 1987.

[Kil92] P. Kilpeläinen. *Tree matching problems with applications to structured text databases.* PhD thesis, Department of Computer Science, University of Helsinki, Report A-1992-6, 1992.

[Klo92] J.W. Klop. Term rewriting systems. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, pages 1–116. Clarendon Press, Oxford, 1992.

[KM93] P. Kilpeläinen and H. Mannila. Retrieval from hierarchical texts by partial patterns. In *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 214–222. ACM Press, 1993.

[KN94] E. Kuikka and E. Nikunen. Rakenteisten tekstien käsittelyjärjestelmistä (Systems for struc-tured documents). Technical Report A/1994/4 (in Finnish, English version avaible in WWW address `http://www.cs.kuopio.fi/~kuikka/systems.html`), University of Kuopio, Department of Computer Science and Applied Mathematics, Finland, 1994.

[KP91] E. Kuikka and M. Penttonen. Designing a syntax-directed text processing system. In K. Koskimies and K.-J. Räihä, editors, *Proceedings of the Second Symposium on Programming Languages and Software Tools*, pages 191–204. University of Tampere, Department of Computer Science, A-1991-5, Tampere, Finland, 1991.

[KP93] E. Kuikka and M. Penttonen. Transformation of structured documents with the use of grammar. *Electronic Publishing - Origination, Dissemination and Design*, 6(4):373–383, 1993.

[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*, pages 311–327, London, UK, April 1994.

[KS95] E. Kuikka and A. Salminen. Two-dimensional filters for structured text. Submitted for publication, 1995.

[Kui90] E. Kuikka. Syntax-directed text processing. Master's thesis, Department of Computer Science, University of Kuopio, Report A/1990/5, 1990.

[Lam94] L. Lambort. *LaTeX*. Addision-Wesley Publishing Company, 1994.

[MBO93] S.A. Mamrak, J. Barnes, and C.S. O'Connell. Benefits of automatic data translation. *IEEE Software*, pages 82–88, 1993.

[Mic91] Microsoft Press. *Microsoft Windows Programmar's referecne*, 1991.

[Mic94] MicroSoft Corporation. *MicroSoft Word, User's Manual, Version 6.0*, 1993-94.

[MKNS87] S.A. Mamrak, M.J. Kaelbling, C.K. Nicholas, and M. Share. A software architecture for supporting the exchange of electronic manuscripts. *Communications of the ACM*, 30(5):408–413, 1987.

[MKNS89] S.A. Mamrak, M.J. Kaelbling, C.K. Nicholas, and M. Share. Chameleon: A system for solving the data-translation problem. *IEEE Transactions on Software Engineering*, 15(9):1090–1108, 1989.

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

[NBY95] G. Navarro and R. Baeza-Yates. A language for queries on structure and contents of textual databases. In *Proceedings of the 18th SIGIR Conference on Research and Development in Information Retrieval*, pages 93–101, Seattle, WA, 1995.

[Nik90] E. Nikunen. Views in structured text databases. Master's thesis, Department of Computer Science, University of Helsinki, C-1990-60, 1990.

[NP92] M. Nivat and A. Podelski, editors. *Tree Automata and Languages.* North-Holland, Amsterdam, 1992.

[Ous93] J.K. Ousterhout. Tcl and the tk toolkit. Draft version of ISBN 0-201-63337-X, 1993.

[Pla93] Thomas Plass. *fb2sgml, a Filter for FrameBuilder Documents.* MID Information Logistics Group, 1993.

[PS87] D.J. Penney and J. Stein. Class modification in the GemStone object-ooriented DBMS. *SIGPLAN Notices*, 22(12):111–117, 1987.

[PW80] F.C.N. Pereira and D.H.C. Warren. Definite clause grammars for language analysis - a survey of the formalism and a comparision with augmented transition networks. *Artificial Intelligence*, 13:231–278, 1980.

[RA94] C. Roisin and E. Akpotsui. Implementating the Cut-and-Paste operation in structured editing system. In *Second Workshop on Principles of Document Processing, PODP'94*, 1994.

[Rao93] J.-C. Raoult. A quick look at tree transductions. In M.R. Sleep, M.J. Plasmeijer, and M.C.J.D. van Eekelen, editors, *Term Graph Rewriting, Theory and Practice*, pages 215–228. Wiley & Sons, Chichester, UK, 1993.

[Rep89] T.W. Reps, editor. *The Synthesizer Generator: a system for constructing language-based editors.* Springer-Verlag, New York, USA, 1989.

[Rou70] W.C. Rounds. Mappings and grammars on trees. *Mathematical Systems Theory*, 4(3):257–287, 1970.

[Sal73] A. Salomaa. *Formal languages.* Academic Press, NewYork, N.Y., USA, 1973.

[SF83] J.-M. Steyaert and P. Flajolet. Patterns and pattern-matching in trees: An analysis. *Information and Control*, 58:19–58, 1983.

[Sil83] P.P. Silvester. *The Unix System Guidebook.* Springer Verlag, New York, 1983.

[Sim91] A. Simpson. *Mastering WordPerfect 5.0.* SYBEX, San Fransisco, 1991.

[SPvE93] M.R. Sleep, M.J. Plasmeijer, and M.C.J.D. van Eekelen, editors. *Term Graph Rewriting, Theory and Practice.* Wiley & Sons, Chichester, UK, 1993.

[Ste84] M. Steinby. Some decidable properties of σ-rational and σ-algebraic tree transformations. Technical Report Ann. Univ. Turkuensis, Serie A I 184, University of Turku, Turku, Finland, 1984.

[Ste90] M. Steinby. A formal theory of errors in tree representations of patterns. *Journal of Information Processing and Cybernetics, EIK 26*, 1(2):19–32, 1990.

[Swe93] Swedish Institute of Computer Science. *SICStus Prolog User manual and SICStus Prolog Library manual (Version 2.1)*, 1993.

[SZ87] A.H. Skarra and S.B. Zdonik. Type evolution in an object-oriented database. In B. Shriver and P. Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 393–416. The MIT Press, Cambridge, Massachusetts, 1987.

[Tai79] K.-C. Tai. The tree-to-tree correction problem. *Journal of the Association for Computing Machinery*, 26(3):422–433, 1979.

[Tha73] J.W. Thatcher. Tree automata: An informal survey. In A.V. Aho, editor, *Currents in the Theory of Computing*, pages 143–172. Prentice-Hall, Inc., Englewood Cliffs, N.J., USA, 1973.

[Tut83] B. Tuthill. Refer - A bibliography system. Technical report, Computing Services, University of California, Berkeley, California, 1983.

[Vog87] H. Vogler. Basic tree transducers. *Journal of Computer and System Sciences*, 34:87–128, 1987.

[Woo87] D. Wood. *Theory of Computation.* Harper & Row, Publisher, New York, 1987.

[Wor90] WordPerfect Corporation. *WordPerfect, Reference Manual, Version 5.1*, 1989-90.

Appendix 1. General rules for a manuscript

Table 1: Delivery form of a manuscript

Journal	Initial manuscript	Accepted article
ACMTOIS	electronic (prefer) or paper	electronic (prefer)
JASIS	paper	paper
CJ	paper or electronic	paper and electronic
CACM	paper	paper and electronic
JCSS	paper	electronic and paper
IS	paper	electronic and paper
EPODD	paper	electronic
SPE	paper	electronic and paper

Table 2: Formats for an electronic form of a manuscript

Journal	LaTeX	troff	PostScript	RTF	MIF	SGML	Ventura	ASCII	Word/WP	Others
ACMTOIS	x	x	x	x	x	x				
JASIS										
CJ	x		x[a]							
CACM								x		
JCSS[b]										
IS								x		
EPODD	x	x	x				x	x		
SPE	x		x						x	x

[a] only for the manuscript
[b] many software, not prespecified

Table 3: Page layout of a manuscript

Journal	Page size	Page number	Margin	Running header	Line space
ACMTOIS					double
JASIS	21*28cm	author nb	adequate		double
CJ	A4	nb	>25mm	yes	double
CACM					double
JCSS	8.5*11inch	nb		yes(<35char)	triple/double
IS	8.5*11inch-A4		1.5inch		double
EPODD				yes	
SPE			wide		double

Appendix 2. The writing style and overall structure of a manuscript

Table 1: Writing style for a manuscript

Journal	Writing style
ACMTOIS	-
JASIS	Publication Manual of the American Psychological Association
CJ	-
CACM	The Chicago Manual of Style
JCSS	A Manual for Authors of American Mathematical Society
IS	-
EPODD	The Chicago Manual of Style
SPE	-

Table 2: Places and existence of certain elements in a manuscript

Journal	Footnotes	Figures	Figure legends	Keywords
ACMTOIS	in text	in text	in text	yes
JASIS	in text	in text	in text	no
CJ	in text	at end	at end	no
CACM	in text	in text	in text	yes
JCSS	at end	at end	at end	no
IS	at end	at end	at end	no
EPODD	in text	in text	in text	yes
SPE	in text	at end	at end	yes

Appendix 3. Examples about formats and structures for title, author and grant information of a manuscript

Example 1: ACM Transactions of Information Systems (ACMTOIS)

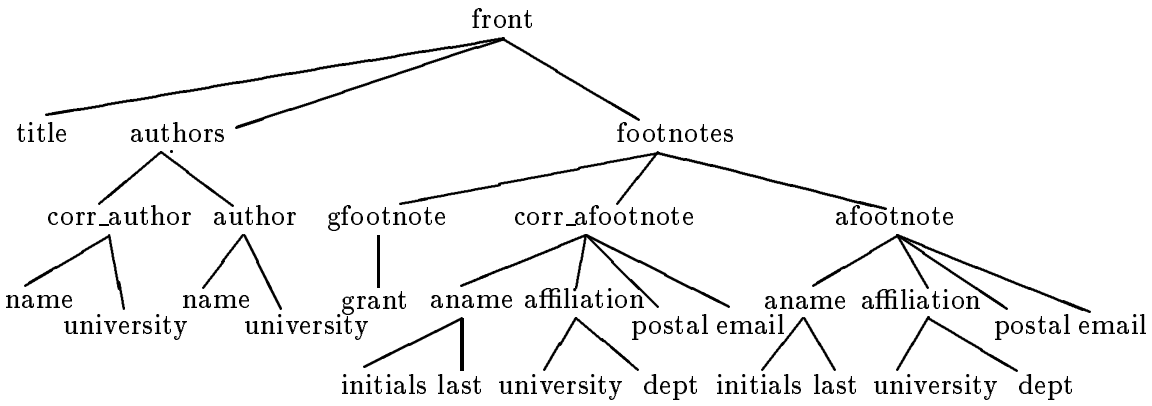
The form of title, author and grant information of a manuscript for ACMTOIS:

```
Transformations of structured documents with the use of grammar

EILA KUIKKA
University of Waterloo
and
MARTTI PENTTONEN
University of Joensuu
```

In a footnote at the first page:

```
This work was supported by a grant from University of Kuopio.
Authors's addresses: E. Kuikka, University of Waterloo, Department
of Computer Science, Waterloo, Ontario, N2L 3G1, Canada, email:
ekuikka@watsol.uwaterloo.ca. M. Penttonen, University of Joensuu,
Department of Computer Science, P.O.Box 111, 80101 Joensuu, Finland,
email: penttonen@cs.joensuu.fi.
```

A parse tree (without the content)



Example 2: Journal of American Society of Information Science (JASIS)

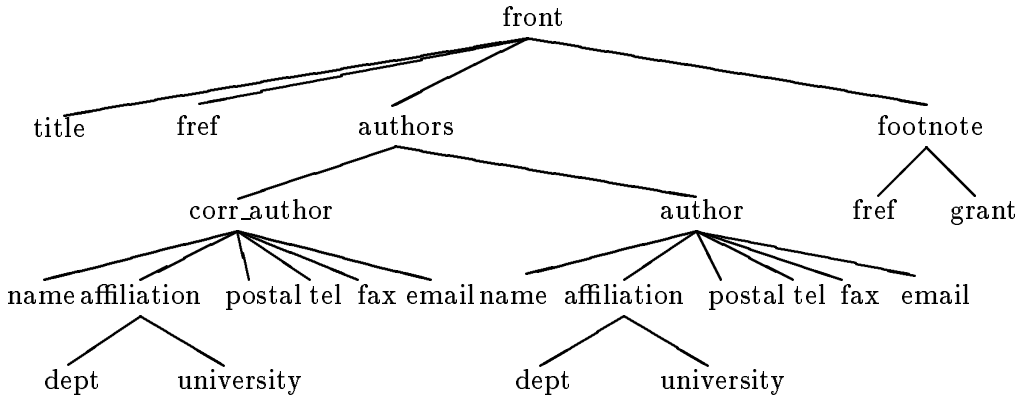
The form of title, author and grant information of a manuscript for JASIS:

```
Transformations of structured documents with the use of grammar*

Eila Kuikka
Department of Computer Science, University of Waterloo,
Waterloo, Ontario, N2L 3G1, Canada, tel: +1-519-888-4456,
fax: +1-519-885-1208, email: ekuikka@watsol.uwaterloo.ca
and Martti Penttonen
Department of Computer Science, University of Joensuu
P.O.Box 111, 80101 Joensuu, Finland, tel: +358-81-153305,
fax: +358-81-153301, email: penttonen@cs.joensuu.fi
```

In the footnote of the same page:

```
*This work was supported by a grant from University of Kuopio.
```

A parse tree (without the content)



Example 3: The Computer Journal (CJ)

The form of title, author and grant information of a manuscript for CJ:

Transformations of structured documents with the use of grammar

Transformation with grammars

EILA KUIKKA

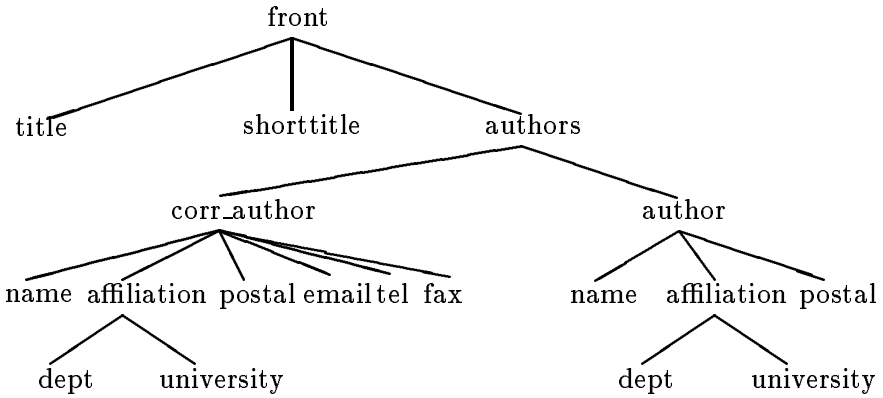
Department of Computer Science, University of Waterloo
Waterloo, Ontario, N2L 3G1, Canada, email: ekuikka@watsol.uwaterloo.ca
tel: +1-519-888-4456, fax: +1-519-885-1208

and

MARTTI PENTTONEN

Department of Computer Science, University of Joensuu
P.O.Box 111, 80101 Joensuu, Finland

A parse tree (without the content)



Example 4: Communications of the ACM (CACM)

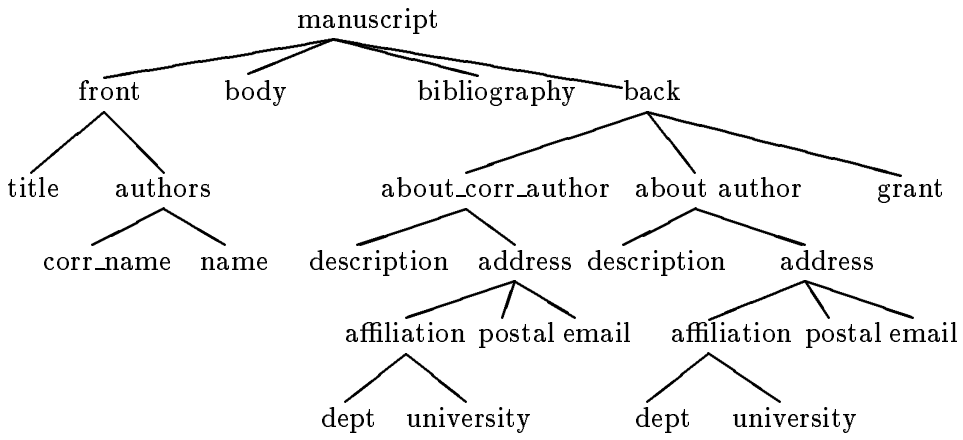
The form of title, author and grant information of a manuscript for CACM:

```
Transformations of documents using grammar

Eila Kuikka and Martti Penttonen
```

In "About the Authors" at the end of article:

```
Eila Kuikka is an assistant and post-graduate student of
Computer Science at the University of Kuopio. Her research
interest is structured document processing. Author's
present address: Department of Computer Science, University
of Waterloo, Waterloo, Ontario, N2L 3G1, Canada, email:
ekuikka@watsol.uwaterloo.ca.

Martti Penttonen is a professor of Computer Science
at the University of Joensuu. His research interests are
parallel computing and  theoretical computer science. Authors's
present address: Department of Computer Science, University of
Joensuu,  P.O.Box 111, 80101 Joensuu, Finland, email:
penttonen@cs.uku.fi.

This work was supported by a grant of the University of Kuopio.
```

A parse tree (without the content)



Example 5: Journal of Computer and System Science (JCSS)

The form of title, author and grant information of a manuscript for JCSS:

```
Transformations of structured documents with the use of grammar

Eila Kuikka*

Department of Computer Science, University of Waterloo
Waterloo, Ontario, N2L 3G1, Canada

and

Martti Penttonen

Department of Computer Science, University of Joensuu
P.O.Box 111, 80101 Joensuu, Finland
```

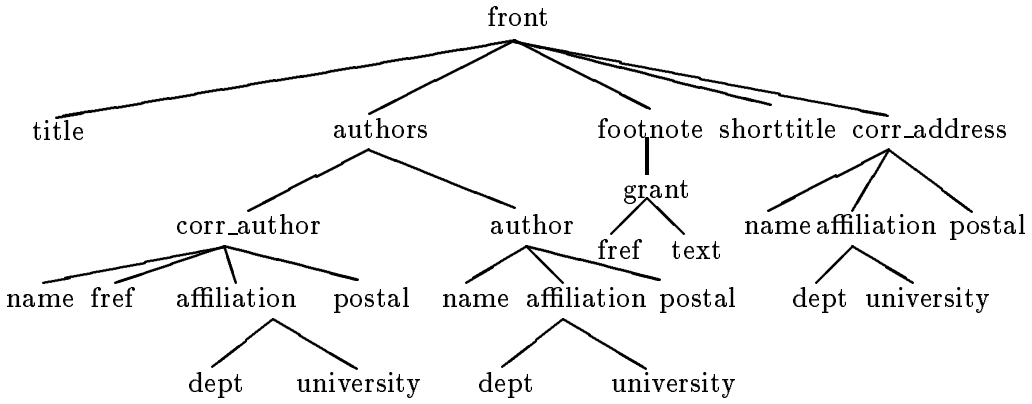
On the footnotes on the same page:

```
*This work was supported by a grant of the University of Kuopio.
```

On the next page running heading and correspondence address:

```
Transformations using grammar

Eila Kuikka
Department of Computer Science
University of Waterloo
Waterloo, Ontario, N2L 3G1, Canada
```

A parse tree (without the content)



Example 6: Information Systems (IS)

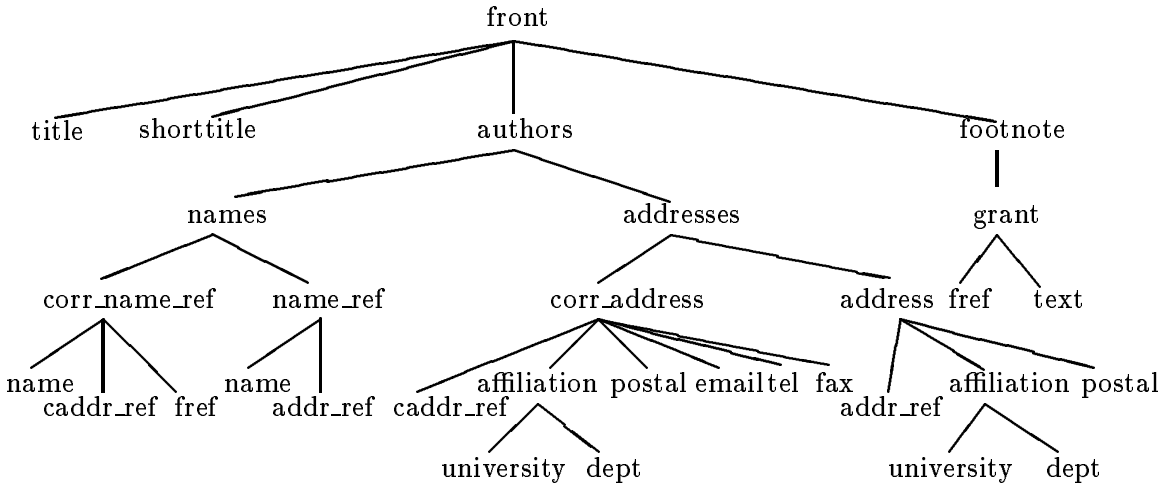
The form of title, author and grant information of a manuscript for IS:

```
Transformations of structured documents with the use of grammar

Transformation with grammars

EILA KUIKKA1* and MARTTI PENTTONEN2

1University of Waterloo, Department of Computer Science
Waterloo, Ontario, N2L 3G1, Canada
email: ekuikka@watsol.uwaterloo.ca
tel: +1-519-888-4456, fax: +1-519-885-1208

2University of Joensuu, Department of Computer Science
P.O.Box 111, 80101 Joensuu, Finland
```

In footnote on the same page:

```
*This work was supported by a grant form the University of Kuopio.
```

A parse tree

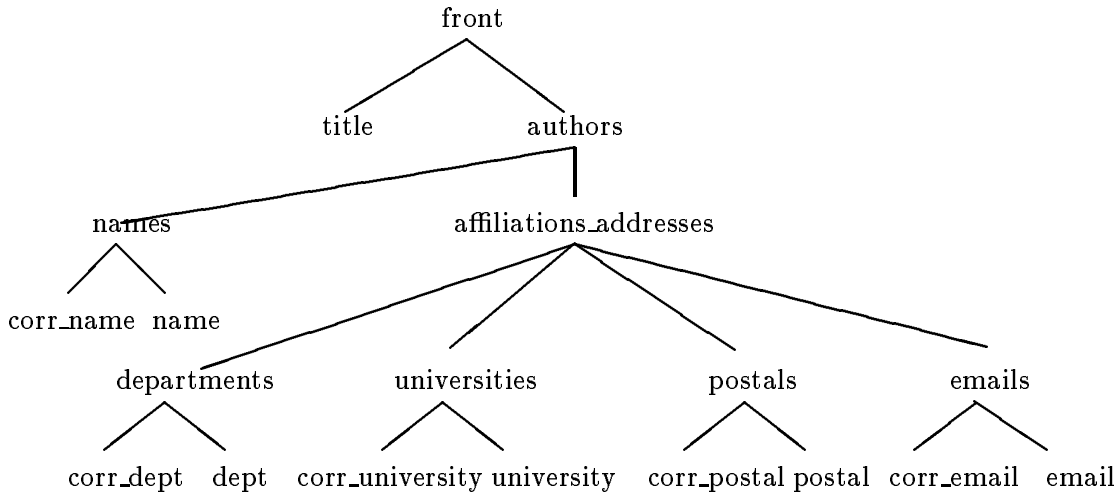


Example 7: Electronic Publishing - Origination, Dissemination and Design (EPODD)

The form of title, author and grant information of a manuscript for EPODD:

```
Transformations of structured documents with the use of grammar

EILA KUIKKA                          MARTTI PENTTONEN
Department of Computer Science       Department of Computer Science
University of Waterloo               University of Joensuu
Waterloo,Ontario,N2L3G1,Canada       P.O.Box 111,80101 Joensuu,Finland
email: ekuikka@watsol,uwaterloo.ca   email: penttonen@cs.joensuu.fi
```

A parse tree

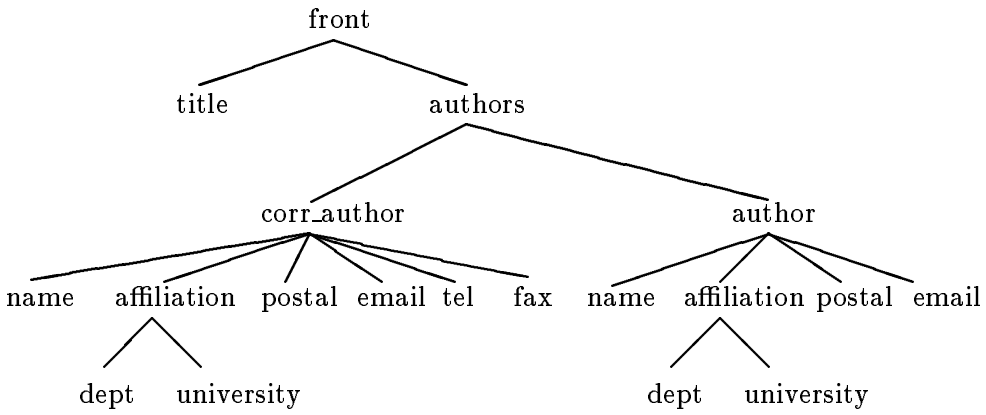


Example 8: SOFTWARE - Practice and Experience (SPE)

The form of title, author and grant information of a manuscript for SPE:

```
Transformations of structured documents with the use of grammar

                        Eila Kuikka
      Department of Computer Science, University of Waterloo
             Waterloo, Ontario, N2L 3G1, Canada
              (email: ekuikka@watsol.uwaterloo.ca)
          tel: +1-519-888-4456, fax: +1-519-885-1208

                            and

                       Martti Penttonen
      Department of Computer Science, University of Joensuu
               P.O.Box 111, 80101 Joensuu, Finland
                (email: penttonen@cs.joensuu.fi)
```

A parse tree (without the content)



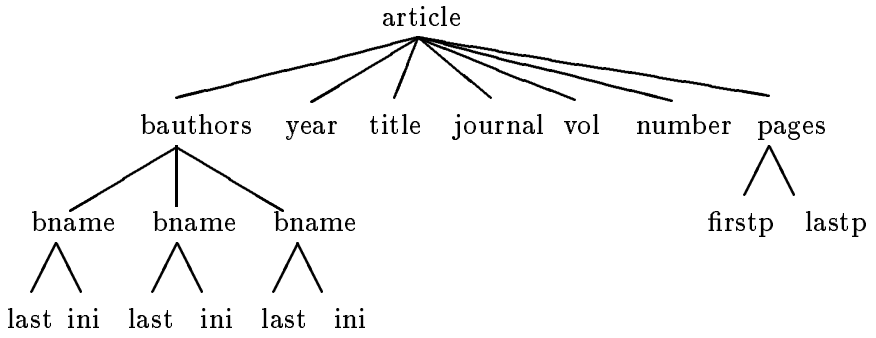
Appendix 4. Examples of bibliographic references in manuscripts

Example 1: Reference to an article

ACM Transaction of Information Systems (ACMTOIS):

FURUTA, R., QUINT, V., AND ANDRÉ, J. 1988. Interactively editing structured documents. *Electron. Pub. 1*, 1, 19-44.

A parse tree for an article reference in ACMTOIS:



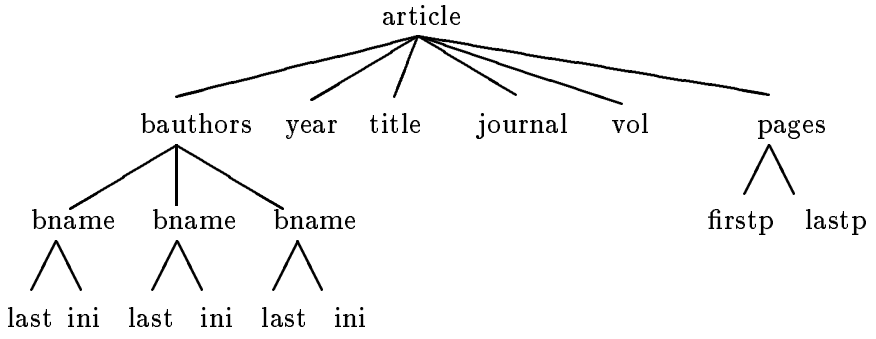
Journal of American Society of Information Science (JASIS):

Furuta, R., Quint, V., & André, J. (1988). Interactively editing structured documents. *Electronic Publishing, 1*, 19-44.

The Computer Journal (CJ):

Furuta, R., Quint, V., and André, J. (1988) Interactively editing structured documents. *Electron. Pub.*, **1**, 19-44.

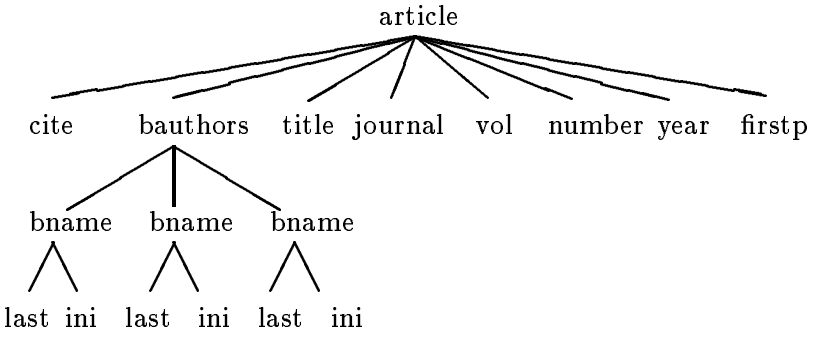
A parse tree for an article reference in JASIS and CJ:



Communications of the ACM (CACM):

1. Furuta, R., Quint, V. and André, J. Interactively editing structured documents. *Electron. Pub. 1*, 1, (1988), 19.

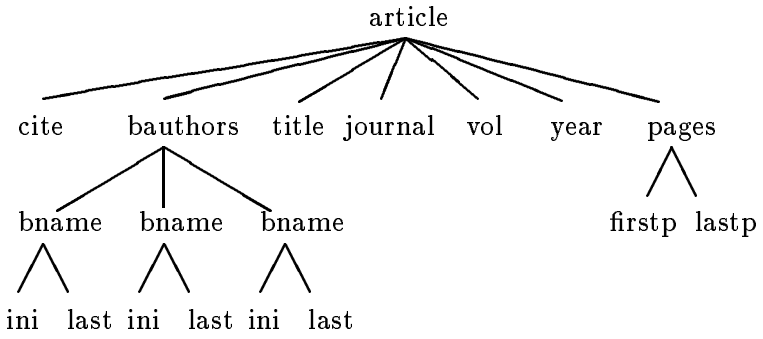
A parse tree for an article reference in CACM:



Journal of Computer and System Sciences (JCSS):

1. R. FURUTA, V. QUINT, AND J. ANDRÉ, Interactively editing structured documents, *Electron. Pub.* **1** (1988), 19-44.

A parse tree for an article reference in JCSS:



Information Systems (IS):

1. R. Furuta, V. Quint, and J. André. Interactively editing structured documents. *Electronic Publishing* **1**(1), 19-44 (1988).

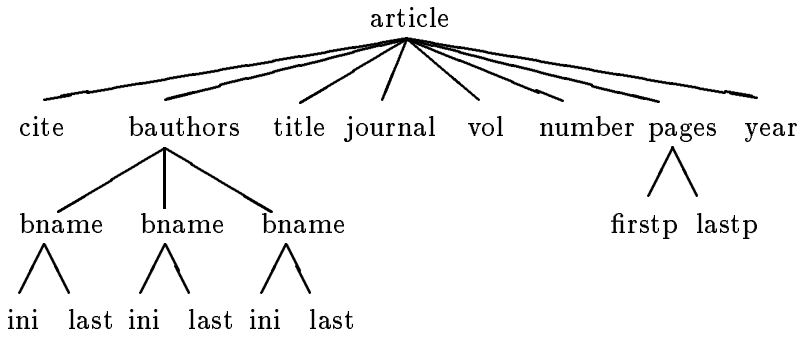
Electronic Publishing - Origination, Dissemination and Design (EPODD):

1. R. Furuta, V. Quint, and J. André, 'Interactively editing structured documents', *Electronic Publishing*, **1**(1), 19-44 (1988).

SOFTWARE - Practice and Experience (SPE):

1. R. Furuta, V. Quint, and J. André, 'Interactively editing structured documents', *Electronic Publishing*, **1**(1), 19-44 (1988).

A parse tree for an article reference in IS, EPODD and SPE:



Example 2: Reference to a book

ACM Transaction of Information Systems (ACMTOIS):

AHO, A. V., AND ULLMAN, J. D. 1972. *The Theory of Parsing, Translation and Compiling, Vol. I: Parsing.* Prentice-Hall, Inc., Englewood Cliffs, N. J.

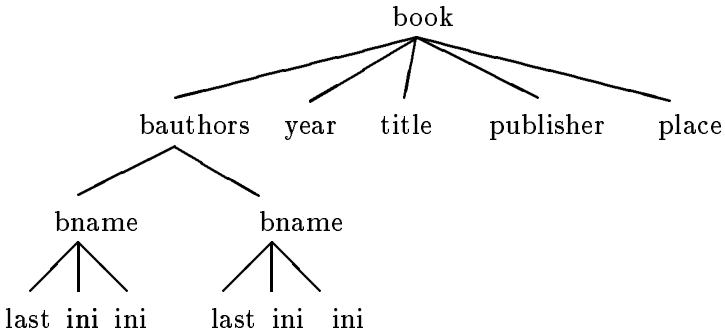
Journal of American Society of Information Science (JASIS):

Aho, A. V., & Ullman, J. D. (1972). The Theory of Parsing, Translation and Compiling, Vol. I: Parsing. Englewood Cliffs, N.J.: Prentice-Hall, Inc.

The Computer Journal (CJ):

Aho, A. V., and Ullman, J. D. (1972) *The Theory of Parsing, Translation and Compiling, Vol. I: Parsing.* Prentice-Hall, Inc., Englewood Cliffs, N. J.

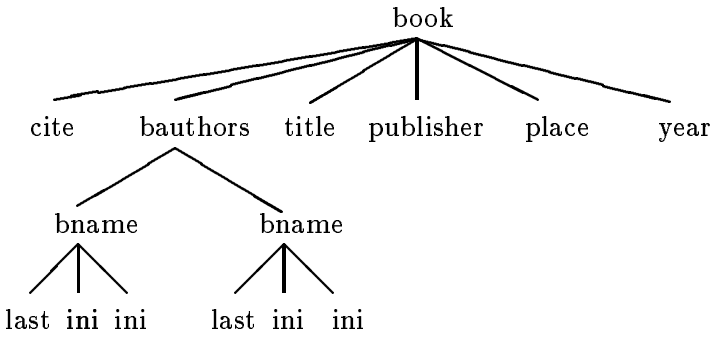
A parse tree for a book reference in ACMTOIS, JASIS and CJ:



Communications of the ACM (CACM):

1. Aho, A. V., and Ullman, J. D. *The Theory of Parsing, Translation and Compiling, Vol. I: Parsing.* Prentice-Hall, Inc., Englewood Cliffs, N. J., 1972.

A parse tree for a book reference in CACM:



Journal of Computer and System Sciences (JCSS):

1. A. V. AHO, AND J. D. ULLMAN, "The Theory of Parsing, Translation and Compiling, Vol. I: Parsing," Prentice-Hall, Inc., Englewood Cliffs, N. J., 1972.

Information Systems (IS):

1. A.V. Aho and J.D. Ullman. *The Theory of Parsing, Translation and Compiling, Vol. I: Parsing.* Prentice-Hall, Inc., Englewood Cliffs, N. J. (1972).

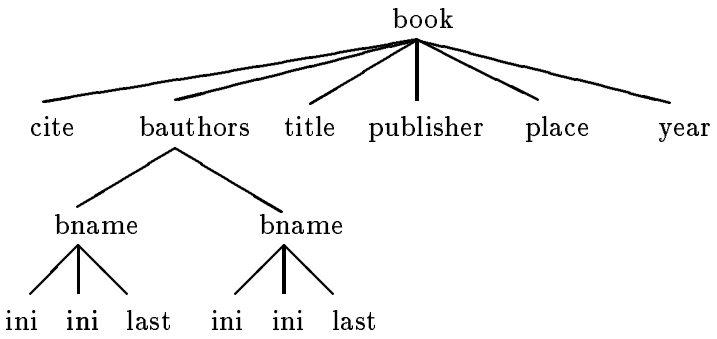
Electronic Publishing - Origination, Dissemination and Desing (EPODD):

1. A.V. Aho and J.D. Ullman, *The Theory of Parsing, Translation and Compiling, Vol. I: Parsing*, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1972.

SOFTWARE - Practice and Experience (SPE):

1. A.V. Aho and J.D. Ullman, *The Theory of Parsing, Translation and Compiling, Vol. I: Parsing*, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1972.

A parse tree for a book reference in JCSS, IS, EPODD and SPE:



Example 3: Reference to an article in conference proceedings

ACM Transaction of Information Systems (ACMTOIS):

FRANCHI-ZANNETTACCI, P., AND ARNON, D. S. 1989. Context-sensitive semantics as a basis for processing structured documents. In *WOODMAN'89, Workshop on Object-Oriented Document Manipulation*, Rennes, France, 135-146.

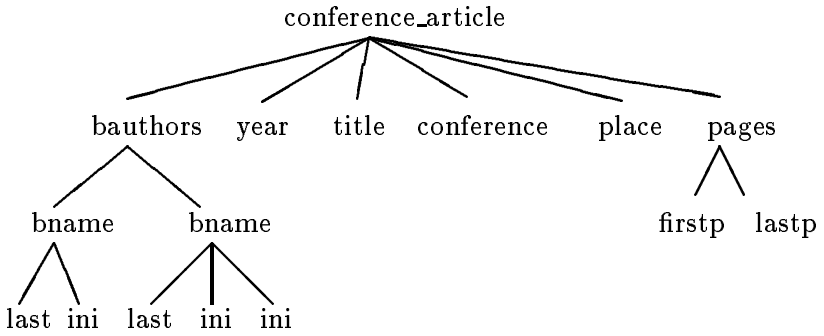
Journal of American Society of Information Science (JASIS):

Franchi-Zannettacci, P., & Arnon, D. S. (1989). Context-sensitive semantics as a basis for processing structured documents. In *WOODMAN'89, Workshop on Object-Oriented Document Manipulation*, Rennes, France, (pp. 135-146).

The Computer Journal (CJ):

Franchi-Zannettacci, P., and Arnon, D. S. (1989) Context-sensitive semantics as a basis for processing structured documents. *WOODMAN'89, Workshop on Object-Oriented Document Manipulation*, Rennes, France, pp. 135-146.

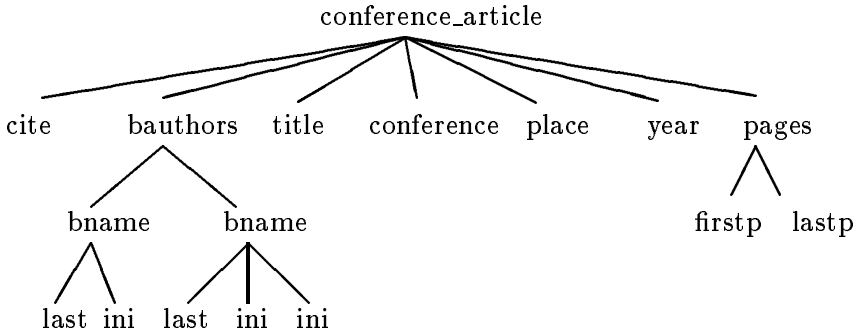
A parse tree for a reference to an article in conference proceedings in ACMTOIS, JASIS and CJ:



Communications of the ACM (CACM):

1. Franchi-Zannettacci, P., and Arnon, D. S. Context-sensitive semantics as a basis for processing structured documents. In *WOODMAN'89, Workshop on Object-Oriented Document Manipulation* (Rennes, France) 1989, pp. 135-146.

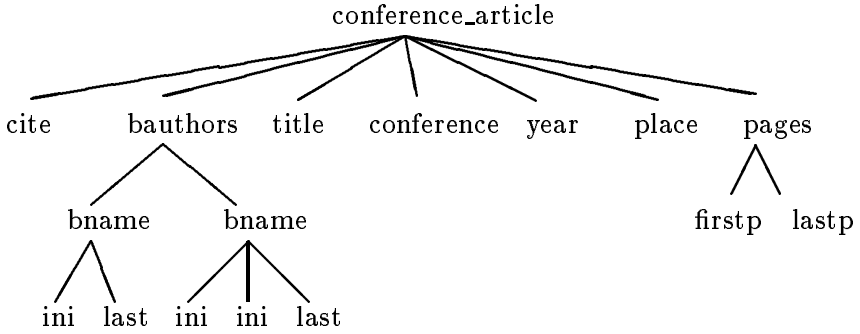
A parse tree for a reference to an article in conference proceedings in CACM:



Journal of Computer and System Sciences (JCSS):

1. P. FRANCHI-ZANNETTACCI, AND D.S. ARNON, Context-sensitive semantics as a basis for processing structured documents, *in* "WOODMAN'89, Workshop on Object-Oriented Document Manipulation, 1989", Rennes, France, pp. 135-146.

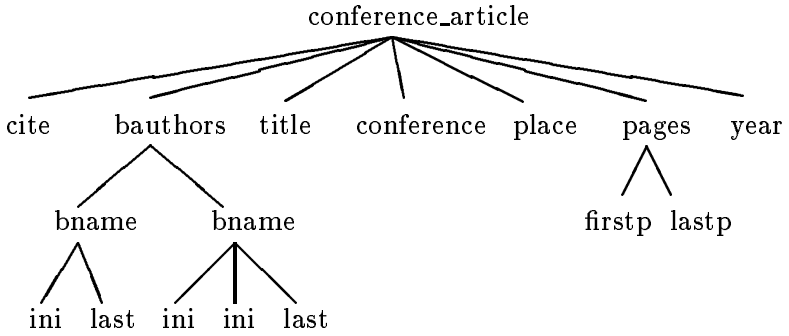
A parse tree for a reference to an article in conference proceedings in JCSS:



Information Systems (IS):

1. P. Franchi-Zannettacci and D.S. Arnon. Context-sensitive semantics as a basis for processing structured documents. *WOODMAN'89, Workshop on Object-Oriented Document Manipulation*, Rennes, France, 135-146, (1989).

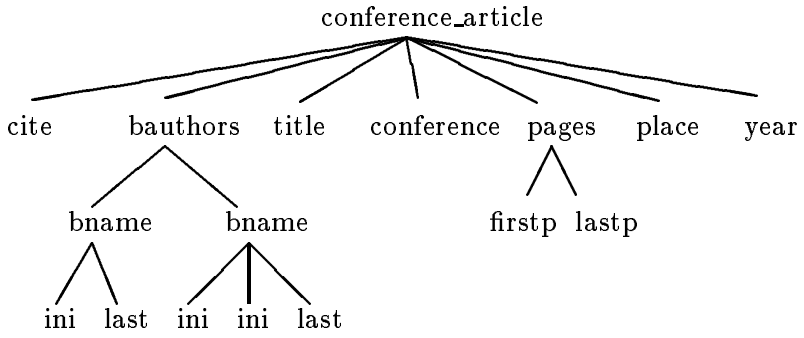
A parse tree for a reference to an article in conference proceedings in IS:



Electronic Publishing - Origination, Dissemination and Design (EPODD):

1. P. Franchi-Zannettacci and D.S. Arnon, 'Context-sensitive semantics as a basis for processing structured documents', in *WOODMAN'89, Workshop on Object-Oriented Document Manipulation*, 135-146, Rennes, France, (1989).

A parse tree for a reference for an article in conference proceedings in EPODD:



SOFTWARE - Practice and Experience (SPE):

1. P. Franchi-Zannettacci and D.S. Arnon, 'Context-sensitive semantics as a basis for processing structured documents', *WOODMAN'89, Workshop on Object-Oriented Document Manipulation*, Rennes, France, 1989, pp. 135-146.

A parse tree for a reference to an article in conference proceedings in SPE is the same as the parse tree used for CACM.

Appendix 5. Grammars for manuscripts

In the follwing grammars brackets define an optional nonterminal, | separates alternative nonterminals, * defines a list with zero or more elements and + a list of one or more elements. All the nonterminals for which a production does not exist define elements that contain character strings.

The grammar for the manuscript of this work is as follows. The grammar is represented in whole, in other grammars it is supposed that the body part is defined as in this grammar.

```
manuscript         --> front body bibliography
front              --> title authors [footnote]
authors            --> corr_author author*
corr_author        --> name [fref] affiliation email
author             --> name [fref] affiliation email
affiliation        --> dept university
footnotes          --> grant+
grant              --> fref text.
body               --> abstract content
content            --> section+ acknowledge
section            --> heading sectionbody+
sectionbody        --> paragraph | subsection
subsection         --> heading subsectionbody+
subsectionbody     --> paragraph | subsubsection
subsubsection      --> heading paragraph+
paragraph          --> textparagraph | equationparagraph
equationparagraph  --> number equation
bibliography       --> item+
item               --> article | book | conference_article
article            --> cite bauthors title journal year vol
                       number pages
book               --> cite bauthors title publisher year
conference_article --> cite bauthors title conference year
                       beditors pages
bauthors           --> bname+
bname              --> last ini+
pages              --> firstp lastp
beditors           --> bname+
```

Grammar for ACMTOIS:

```
manuscript         --> front body bibliography
front              --> title authors footnotes
authors            --> corr_author author*
corr_author        --> name university
author             --> name university
footnotes          --> [gfootnote] corr_afootnote afootnote*
```

```
gfootnote          --> grant+
corr_afootnote     --> aname affiliation postal email
aname              --> initials last
affiliation        --> university dept
afootnote          --> aname affiliation postal email
body               --> ...
bibliography       --> item+
item               --> article | book | conference_article
article            --> bauthors year title journal vol number
                       pages
book               --> bauthors year title publisher place
conference_article --> bauthors year title conference place
                       pages
bauthors           --> bname+
bname              --> last ini+
pages              --> firstp lastp
```

Grammar for CJ

```
manuscript         --> front body bibliography
front              --> title shorttitle authors
authors            --> corr_author author*
corr_author        --> name affiliation postal email tel fax
author             --> name affiliation postal
affiliation        --> dept university
body               --> ...
bibliography       --> item+
item               --> article | book | conference_article
article            --> bauthors year title journal vol  pages
book               --> bauthors year title publisher place
conference_article --> bauthors year title conference place
                       pages
bauthors           --> bname+
bname              --> last ini+
pages              --> firstp lastp
```

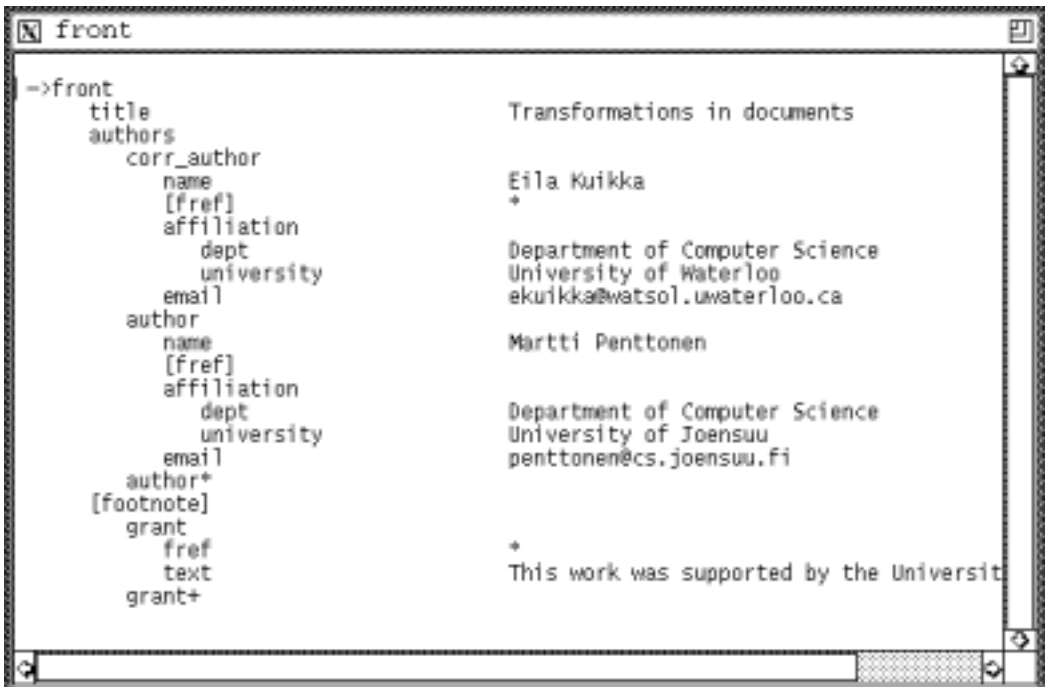
Grammar for EPODD:

```
manuscript         --> front body bibliography
front              --> title authors
authors            --> names affiliations_addresses
affiliations_addresses -->
                       departments universities postals emails
names              --> corr_name name*
departments        --> corr_dept dept*
universities       --> corr_university university*
postals            --> corr_postal postal*
```

```
emails             --> corr_email email*
body               --> ...
bibliography       --> item+
item               --> article | book | conference_article
article            --> cite bauthors title journal vol number
                       pages year
book               --> cite bauthors title publisher place year
conference_article --> cite bauthors title conference  pages
                       place year
bauthors           --> bname+
bname              --> ini+ last
pages              --> firstp lastp
```

Appendix 6. Transformation of a front part of this manuscript to a manuscript for EPODD

The front part of the manuscript of this work on the screen of a syntax-directed document processing system SYNDOC.



The transformation is made by a tree transducer. The grammars are represented in Appendix 5. The states for nonterminals of the EPODD grammar are as follows. The initial state is p1.

State	Nonterminal	State	Nonterminal	State	Nonterminal
p1	front	p2	authors	p3	names
p4	affiliations_addresses	p5	departments	p6	universities
p7	postals	p8	emails	p9	title
p10	corr_name	p11	name	p12	corr_dept
p13	dept	p14	corr_university	p15	university
p16	corr_postal	p17	postal	p18	corr_email
p19	email	p20	string		

Rules for the tree transducer that transforms previous parse to a parse for the grammar of EPODD. The form of rules is readable by the generator of transformation programme.

```
p1(front(X1,X2,[X3])) -> front(p9(X1),authors(p3(X2),p4(X2))).
p9(title(X1)) -> title(p20(X1)).
p3(authors(X1,(X2)*)) -> names(p10(X1),p11((X2)*)).
p4(authors(X1,(X2)*)) -> affiliations_addresses(
    departments(p12(X1),p13((X2)*)),universities(p14(X1),p15((X2)*)),
    postals,emails(p18(X1),p19((X2)*))).
p10(corr_author(name(X1),[X2],X3,X4)) -> corr_name(p20(X1)).
p11(author(name(X1),[X2],X3,X4)) -> name(p20(X1)).
p12(corr_author(X1,[X2],affiliation(dept(X3),X4),X5)) ->
    corr_dept(p20(X3)).
p13(author(X1,[X2],affiliation(dept(X3),X4),X5)) -> dept(p20(X3)).
p14(corr_author(X1,[X2],affiliation(X3,university(X4)),X5)) ->
    corr_university(p20(X4)).
p15(author(X1,[X2],affiliation(X3,university(X4)),X5)) ->
    university(p20(X4)).
p18(corr_author(X1,[X2],X3,email(X4))) -> corr_email(p20(X4)).
p19(author(X1,[X2],X3,email(X4))) -> email(p20(X4)).
p20(string(X1)) -> string(X1).
```

The transformation programme in Prolog generated automatically from the previous tree transducer rules.

```
p1(front(X1,X2,X3),front(A,authors(B,C))):-p9(X1,A),p3(X2,B),p4(X2,C).
p9(title(X1),title(A)):-p20(X1,A).
list(p11([A],['name*'])).
list(p11([A|B],[C|D])):-p11(A,C),list(p11(B,D)).
p3(authors(X1,X2),names(A,B)):-p10(X1,A),list(p11(X2,B)).
list(p13([A],['dept*'])).
list(p13([A|B],[C|D])):-p13(A,C),list(p13(B,D)).
list(p15([A],['university*'])).
list(p15([A|B],[C|D])):-p15(A,C),list(p15(B,D)).
list(p19([A],['email*'])).
list(p19([A|B],[C|D])):-p19(A,C),list(p19(B,D)).
p4(authors(X1,X2),affiliations_addresses(departments(A,B),universities
(C,D),postals,emails(E,F))):-p12(X1,A),list(p13(X2,B)),p14(X1,C),list(
p15(X2,D)),p18(X1,E),list(p19(X2,F)).
p10(corr_author(name(X1),X2,X3,X4),corr_name(A)):-p20(X1,A).
p11(author(name(X1),X2,X3,X4),name(A)):-p20(X1,A).
p12(corr_author(X1,X2,affiliation(dept(X3),X4),X5),corr_dept(A)):-p20(
X3,A).
p13(author(X1,X2,affiliation(dept(X3),X4),X5),dept(A)):-p20(X3,A).
p14(corr_author(X1,X2,affiliation(X3,university(X4)),X5),corr_universi
```

```
ty(A)):-p20(X4,A).
p15(author(X1,X2,affiliation(X3,university(X4)),X5),university(A)):-p2
0(X4,A).
p18(corr_author(X1,X2,X3,email(X4)),corr_email(A)):-p20(X4,A).
p19(author(X1,X2,X3,email(X4)),email(A)):-p20(X4,A).
p20(string(X1),string(X1)).
```

The front part of a manuscript in the format for EPODD on the screen of SYNDOC.