Forming grammars for structured documents *

Helena Ahonen
University of Helsinki

Heikki Mannila
University of Helsinki

Erja Nikunen
Research Centre for Domestic Languages

April 1993

Abstract

We consider the problem of finding a small regular grammar that correctly describes the structure of a large text with named components. Examples of such texts are dictionaries, user manuals, business letters, and so on. A structural description in the form of the regular grammar can be used, e.g., to help in retrieving information from the document. We start by constructing for each named component of the document a weighted finite-state automaton that accepts all the structures for that component that are present in the original document. The weight of a transition shows how many times the transition is used in the original document. This automaton is generalized by merging states and updating the weights until the automaton satisfies a certain context condition. The automata corresponding to different components in the document are also generalized with respect to each other. The generalized weighted automata are transformed into a series of regular expressions corresponding to the heavy paths in the automata.

1 Introduction

Huge amounts of documents are created every day. Many of these documents have some kind of structure: consider for instance user manuals, business letters, technical documents, dictionaries, electronic letters, and so on. The structure of a document can be used to define transformations and queries with structural conditions. The structure also gives important knowledge of the data: what components the documents can have, which components can appear together, and so on.

In recent years, research on systems for writing structured documents has been very intensive. Recent surveys of the field are [3, 4, 19]. The interest in the area has led to the creation of several document standards, of which the best known are ODA and SGML [14, 6, 7, 10]. The common way to describe the structure of a document is to use regular or context-free grammars [11, 8, 9, 18]. In database terminology, grammars correspond to schemas, and parse trees to instances.

It is typical to use regular expressions in the right-hand sides of the productions of the grammar. For example, the following might describe the simplified structure of a dictionary entry:

Entry → Headword Sense*.

---

*This work was partially supported by TEKES and the Academy of Finland. Authors' addresses: Helena Ahonen, Heikki Mannila, Department of Computer Science, University of Helsinki, P.O. Box 26 (Teollisuuskatu 23), SF-00014 University of Helsinki, Finland. Erja Nikunen, Research Centre for Domestic Languages, Sörnäisten rantatie 25, SF-00500 Helsinki, Finland. e-mail: `hahonen@cs.helsinki.fi`, `mannila@cs.helsinki.fi`, `enikunen@domlang.fi` Part of the work of Heikki Mannila was done while he was visiting the Technical University of Vienna.

This tells us that the dictionary has components named Entry, Headword, and Sense. Each instance of an Entry component can consist of an instance of a Headword component, followed by zero or more instances of the component Sense. A more complicated example is

Entry → Headword [Inflection]
(Sense_Number Description [Parallel_Form | Preferred_Form] Example*)*,

which states that an instance of an Entry component consists of a Headword instance, followed by an optional Inflection instance and zero or more groups, each group consisting of an instance of Sense_Number, an instance of Description, a further optional part which is either an instance of Parallel_Form or of Preferred_Form, and a sequence of zero or more instances of Example. If there is no fear of confusion, we speak about components of a document instead of instances of components in the document.

Currently only few documents are created in structured form. Documents are written by numerous text processing systems, most of which are wysiwyg-oriented. However, existing documents can be transformed to structured documents, if

1. the instances of the components of the document can be identified, and
2. a simple and usable description of the structure of the document can be formed from the component structure.

The first problem, identification of components and their instances can be done if the instances are represented in a consistent way by wysiwyg features. These features are converted into structural tags, i.e. begin and end marks for the components. The conversion can be done using tools like AWK.

In this paper we consider the second problem, finding a small description of the structure of a document with a large number of named components. The problem is not trivial, since large documents can have a lot of different variations in their component structure. For example, the part A–J of a new Finnish dictionary [1] has 17385 articles with 1318 different structures. Thus one has to generalize the individual instances to obtain a useful description of the document's structure. Simple attempts to do this by hand do not succeed satisfactorily.

The method we have developed proceeds as follows.

1. Each instance of a component of the text is transformed to a production

   $A \rightarrow B_1, \ldots, B_n,$

   where A is the name of the component and $B_1, \ldots, B_n$ are the names of the components forming this instance of A. The production is given a weight which is the number of times this particular structure appears in the document. Such a production is also called an example in the sequel.
2. The productions are transformed into a set of finite automata, one for each nonterminal. These automata accept exactly the right-hand sides of the productions for the corresponding nonterminal. Every transition gets a weight which is the sum of the weights of the productions using this transition.
3. Each automaton is modified independently, so that it accepts a larger language. This language is the smallest one that includes the original right-hand sides, and has an additional property, called (k,h)-contextuality. This property states roughly that in the structure of the document what can follow a certain component is completely determined by the k preceding components at the same level. The modification is done by merging states. The weight of a merged transition is the sum of the two transitions that are merged.

4. The automata are modified further by considering them in pairs. These modifications guarantee that the resulting document structure is uniform. in the sense that a component is used in every position where all its subcomponents occur in the correct order.
5. The resulting automata are transformed to regular expressions, which form the right-hand sides of the productions for the corresponding nonterminals. The weights are used to construct a production which covers most of the examples for a nonterminal, and then several productions which cover the rare cases.

Steps 2 and 3 are similar to the synthesis of finite automata presented in [5, 16]. Specifically, our class of (k, h)-contextual regular languages is a modification of the classes of k-reversible [5] and k-contextual[16] languages.

Learning context-free and regular grammars from examples has been studied also in, e.g., [13, 20, 21]. However, these results are not directly applicable to our setting because they assume that positive and negative examples are available. Reference [17] makes the assumption that the examples are given to the system in lexicographic order. These assumptions are not valid in our case: it is unnatural to make up document structures which are not allowed, and to be practical the method has to be incremental, which excludes any ordering of the examples.

We have implemented our method in connection with the structured text database system HST [15]. Our preliminary empirical evidence indicates that the method is a useful tool for transforming existing texts to structured form.

The rest of this paper is organized as follows. Section 2 gives the basic definitions. Section 3 describes the construction of the initial automaton. In Section 4 we first describe the general method for generalizing the productions, and the particular inductive biases, k-contextuality and (k, h)-contextuality, we use in generalizing the examples. Section 5 considers the interaction between nonterminals and Section 6 the manipulation of weights in the automata. Section 7 describes the conversion to regular expressions. Empirical results are discussed in Section 8. Section 9 contains some concluding remarks.

2 Definitions

Our method uses finite automata to represent and manipulate the collection of examples. We assume that the reader is familiar with finite-state automata, context-free grammars, and regular expressions (see, e.g., [12] for details), and just give the basic definitions for reference. A finite-state automaton is a quintuple $(Q, \Sigma, \delta, S, F)$, where Q is the set of states, Σ is the set of input symbols, $\delta : Q \times \Sigma^* \to Q$ is the transition function, $S \in Q$ is the start state and $F \subseteq Q$ is the set of final states. For an automaton A the language accepted by A is denoted by $L(A)$.

Regular expressions are defined as follows:

1. $\emptyset$ is a regular expression.
2. ϵ is a regular expression.
3. For each $a \in \Sigma$, a is a regular expression.
4. If r and s are regular expressions, then $(r|s)$, (rs), and (r^*) are regular expressions.

A context-free grammar is a quadruple $G = (N, T, P, S)$, where N and T are finite sets of nonterminals and terminals, respectively, P is a finite set of productions, and S is the start symbol. Each production is of the form $A \to \alpha$, where A is a nonterminal and α is a regular expression over the alphabet $N \cup T$.

3 Prefix-tree automata

The right-hand sides of productions obtained from the document are represented by an automaton that accepts exactly those strings. This *prefix-tree automaton* is simply a trie that contains the right-hand sides. The transitions are weighted by counting how many times they are used in the construction of the automaton. For example, for the following productions the result is the automaton shown in Figure 1. For simplicity, we have left the weights out from the figure.

Entry → Headword Inflection Sense Sense
Entry → Headword Inflection Parallel_form Sense Sense Sense
Entry → Headword Parallel_form Sense Sense
Entry → Headword Preferred_form Sense
Entry → Headword Inflection Preferred_form Sense Sense

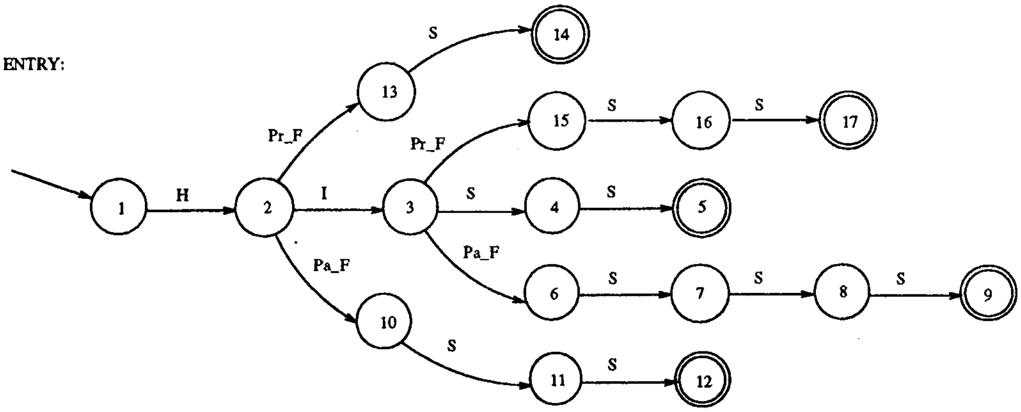

Figure 1: Prefix-tree automaton containing all the examples.

4 (k,h)-contextual languages

A prefix tree automaton accepts only the right-hand sides of the example productions. To obtain useful grammars, we need a meaningful way of generalizing the examples, and the automaton describing them.

In machine learning terms, the examples of productions are all *positive examples*, that is, the user gives no examples of forbidden structures. To learn from positive examples, one needs some restrictions on the allowed result of the generalization. Namely, a consistent generalization of a set of positive examples would be an automaton accepting all strings! Thus we have to define a class of automata that are allowed as results of the generalization.

By merging some of the states we get an automaton which accepts more strings, i.e., this automaton generalizes the examples. To merge states s_i and s_j we first choose one of them to represent the new state, say s_i. All the incoming arcs of s_j are then added to the set of incoming arcs of s_i , and all the outgoing arcs of s_j are added to the set of outgoing arcs of s_i.

How do we choose the states to be merged? Our assumption is that the grammars used in structured documents have only limited context in the following sense. Let k be an integer and consider two occurrences of a sequence of length k of component instances in the document. Then we assume that the subsequent components can be the same in both cases. Consider for example $k = 2$ and the production

Entry → Headword Example Example Example,

which says that an entry can contain three examples. Now the sequence Example Example of length 2 occurs two times on the right-hand side. Since the first occurrence is followed by Example, the structure should allow that also the second one is followed by Example. This means that an entry can contain also *four* examples. Continuing, we come to the conclusion that an entry can contain any number of examples, and thus we construct a production

Entry → Headword Example Example Example*.

A language satisfying the condition above is called *k-contextual* [16]. This property is defined formally as follows. For a language L, denote by $T_L(x)$ the set of strings that can follow x in a member of L, i.e.,

$$T_L(x) = \{v | xv \in L\}.$$

Definition 1 A regular language L is *k-contextual* if and only if for all strings u_1, u_2, w_1, w_2 and v, if u_1vw_1 and u_2vw_2 are in L and $|v| = k$, then $T_L(u_1v) = T_L(u_2v)$.

The condition of k-contextuality can be described simply in terms of automata.

Lemma 2 A regular language L is k-contextual if and only if there exists a finite automaton A such that $L = L(A)$, and for any states p and q of A and all sequences $a_1a_2 \dots a_k$ of input symbols we have: if there are states p_0 and q_0 of A such that $\delta(p_0, a_1a_2 \dots a_k) = p$ and $\delta(q_0, a_1a_2 \dots a_k) = q$, then $p = q$.

For a set of strings H, a k-contextual language L is called a *minimal k-contextual language including H* if

1. $H \subseteq L$ and
2. for all k-contextual languages M such that $H \subseteq M$ we have $L \subseteq M$

It can be shown [2] that for each H there exists a unique minimal k-contextual language containing a given set of strings. If A is an automaton such that $L(A)$ is k-contextual, we say that A is a *k-contextual automaton*. The above lemma gives a way of constructing, for an automaton C, a k-contextual automaton which accepts the smallest k-contextual language containing $L(C)$. States of C satisfying the conditions in the implication of the lemma are merged until no such states remain. For brevity, we omit the description of the algorithm.

The resulting 2-contextual automaton looks like the one in Figure 2. We can see that it generalizes the examples quite well. The automaton, however, accepts only entries which have two or more *Sense* nonterminals in the end. This is overly cautious, and therefore we need a looser generalization condition. In Figure 2, for example the states s_4 and s_5 could be merged.

The intuition in using k-contextuality is that if there are two occurrences of a sequence of components of length k then the subsequent components can be the same in both cases. We relax this condition and generalize the k-contextual languages further to (k, h)-contextual languages. In these languages two occurrences of a sequence of length k implies that the subsequent components are the same *already after h characters.*

Definition 3 A regular language L is *(k, h)-contextual* if and only if for all strings u_1, u_2, w_1, and w_2, and all input symbols $v_1, \dots, v_k$, if $u_1v_1 \dots v_kw_1$ and $u_2v_1 \dots v_kw_2$ are in L, then $T_L(u_1v_1 \dots v_i) = T_L(u_2v_1 \dots v_i)$ for every i, where $0 \leq h \leq i \leq k$.

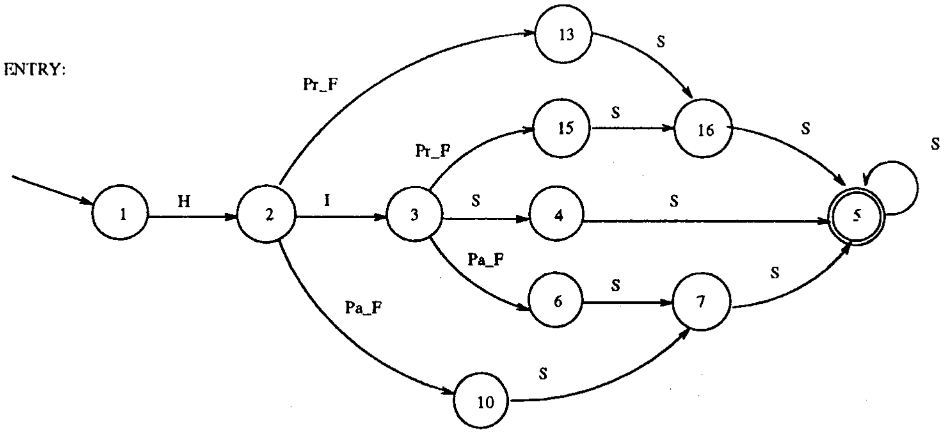

Figure 2: 2-contextual automaton.

Note that k-contextuality is equivalent to (k, k)-contextuality, and (k, h)-contextuality implies $(k, h + 1)$-contextuality. As for k-contextuality, we obtain an easy characterization in terms of automata.

Lemma 4 A regular language L is (k, h)-contextual if and only if there exists a finite automaton A such that $L = L(A)$, and for any two states p_k and q_k of A, and all input symbols $a_1a_2 \ldots a_k$ we have: if there are states p_0 and q_0 such that $\delta(p_0, a_1) = p_1, \delta(p_1, a_2) = p_2, \ldots, \delta(p_{k-1}, a_k) = p_k$ and $\delta(q_0, a_1) = q_1, \delta(q_1, a_2) = q_2, \ldots, \delta(q_{k-1}, a_k) = q_k$, then $p_i = q_i$, for every i, where $0 \leq h \leq i \leq k$.

The algorithm for producing the automaton that accepts the smallest (k, h)-contextual automaton is similar to the previous algorithm: one looks for states satisfying the conditions of the above lemma, and then merges states. If similar paths of length k are found, not only the last states but also some of the respective states along the paths are merged. If $h = k$ only the last states are merged. If $h < k$ the paths have a similar prefix of length h before they are joined, i.e $k - h + 1$ states are merged. In Figure 3 we can see the (2,1)-contextual automaton resulting from the set of example productions.

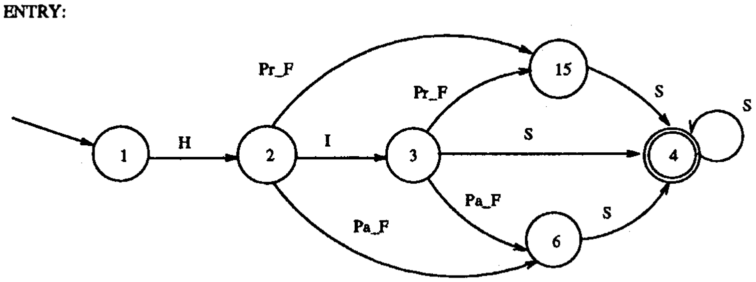

Figure 3: (2,1)-contextual automaton.

5 Interaction between nonterminals

The structure of the document can possibly be described much more concisely by taking into account the interaction between nonterminals. If, for instance, we had the examples

Entry → Headword Sense_number Description

and

Sense → Sense_number Description,

it would be sensible to replace Sense_number Description in the first production by Sense.

The interaction necessitates the redefinition of the concept of an automaton and its language. The labels of arcs are no more simple nonterminals but names of other automata. This kind of automata have slightly confusingly been called *recursive* [22].

Definition 5 Let $V = V_T \cup V_N$, where V_T is the terminal alphabet and V_N the nonterminal alphabet, and consider a set $S = \{A_X \mid X \in V_N\}$ of finite automata, one for each member of V_N. Then the *set of terminal strings accepted by A_X in context S*, denoted by $L(A_X, S)$, is defined by

$$\begin{aligned} L(A_X, S) = \{w_1 \dots w_n \mid\ & \text{there is a word } v_1 \dots v_n \in L(A_X), \\ & \text{and for each } i = 1, \dots, n \text{ either} \\ & \quad (1)\ v_i \in V_T \text{ and } w_i = v_i, \text{ or} \\ & \quad (2)\ v_i \in V_N \text{ and } w_i \in L(A_{v_i}, S)\}. \end{aligned}$$

Definition 6 Let M and N be regular languages. The language M is *N-closed*, if for any $w \in M$ such that $w = xvy$ for some x,v,y, with $v \in N$, we have $xv'y \in M$ for all $v' \in N$.

Thus M is N-closed, if for any string v of N occurring as a substring in a string w of M, we can replace v in w by an arbitrary string v' of N, and the resulting string is still in M. Note that if $\epsilon \in N$ and N contains a lot of strings, then the condition is fairly strong.

The inductive bias we adopt for handling the interaction of several nonterminals is as follows.

> Let $S = \{A_1 \dots A_n\}$ be the set of automata used. Then $L(A_i, S)$ has to be $L(A_j, S)$-closed for every $i \neq j$ with $1 \leq i, j \leq n$.

Again, the definition of closedness is transformed to automata. An automaton A is B-closed for an automaton B, if $L(A)$ is $L(B)$-closed.

Given regular languages M and N, we can make M N-closed as follows. Let A and B be automata such that $M = L(A)$ and $N = L(B)$. To make A B-closed we search for a path $p_1 \dots p_m$ in A, where $\delta(p_1, a_1) = p_2, \dots, \delta(p_{n-1}, a_{n-1}) = p_m$, such that B accepts the string $a_1 \dots a_{m-1}$. If such a path is found, an arc labeled B is added from p_1 to p_m.

6 Weights in the automata

In existing texts the structure of components can be fairly complicated, and even generalizing to (k, h)-contextual languages does not necessarily produce a simple expression of the structure. Therefore we use weights in the automata to quantify the importance of different types of structures for the component.

Adding weights to the prefix-tree automaton is easy: each transition is given a weight which is the number of examples in which this transition is used. When the automata are generalized, the weight of a merged transition is the sum of the weights of the two transitions that are merged.

7 Conversion into a regular expression

After the generalization steps presented in the previous sections have been performed, we have a collection of (k, h)-contextual automata that are closed with respect to each other. To obtain a useful description of the structure of the document, we still have to produce a grammar from these.

An automaton can be converted into a regular expression by using standard dynamic programming methods [12]. While this is useful in itself, we need something more refined. Namely, we want to produce one, hopefully simple, regular expression that describes most of the document instances correctly. This is done by considering a lower bound on the allowed transition weights and by pruning away all transitions whose weights are below the bound. This gives a smaller automaton that can then be transformed to a regular expression.

When this regular expression describing most of the document has been produced, the task is to describe the rest. This can be done either by considering a sequence of smaller and smaller bounds for the weights, and producing for each bound a regular expression using only transitions with weights greater or equal the bound.

With this approach, the sequence of regular expressions produced is monotonic in the sense that the language accepted grows as the bound decreases. To obtain a simpler description of the structures with smaller frequency, we can also use only one bound. For each transition whose weight is below that bound, we construct the automaton where that transition is mandatorily used. Each such automaton is converted to a regular expression and the results are simplified.

8 Experimental results

We have implemented the method described above in connection with the HST structured text database system [15]. We have experimented with several different document types, and the results are encouraging. Our experience shows that the method is a useful tool for finding the grammar for a structured document.

The most challenging document we experimented with was a part of a Finnish dictionary [1]. Originally the entries of the dictionary had only typographical tags (eg. begin bold – end bold). These tags were changed to structural tags (eg. begin headword – end headword). Then the text, and the end tags, were removed, and the structures of the entries were obtained, for instance:

<EN> → <H> <I> <CG> <S> <EX>.

The data consisted of 15970 entries. The total number of different entry structures was about 1000 but only 82 of them covered more than ten entries (see Appendix A for the productions and the meanings of the abbreviations). These 82 example structures, which together covered 15131 entries, were input to our learning program which generalized the examples and produced 13 productions. These productions were used to parse the whole data, and the coverages were counted.

The weight bound was 400, meaning that to be considered important, the structure needs to appear at least 400 times in the dictionary. The production corresponding only to transitions of at least this weight was

```
EN → H [I | (I CG | [I] [TF] ) S
          | (I CG S | [I] [TF] [S] ) EX
          | [I] TF]
```

and it covered 11274 examples. Out of the other productions, the most common were the complex

```
EN → H ( (I (CG S [ (EX S)* TF ( [S] (EX S)* TF)* [S] ]
             | [S (EX S)* ] TF ( [S] (EX S)* TF)* [S] | S)
          | S[ (EX S)* TF ( [S] (EX S)* TF)* [S] ] ) EX [ (S EX)* [S EX] ]
          | I (CG S [ ( (EX S)* TF ( [S] (EX S)* TF)* [ [S] EX (S EX)* ] | EX (S EX)* ) S]
             | [S (EX S)* ] TF ( [S] (EX S)* TF)* [ [S] EX (S EX)* ] S
             | S (EX S)* )
          | S [ ( (EX S)* TF ( [S] (EX S)* TF)* [ [S] EX (S EX)* ] | EX (S EX)* ) S] )
```

and the simpler

```
EN → H [I] ( [EX] TF (EX TF)* [EX] | EX)
          S [ (EX TF (EX TF)* [EX] S)* [EX [TF (EX TF)* [EX] ] S] ],
```

which covered 6536 and 1796 examples, respectively. Note that these productions could be easily simplified if, for instance, EX and [EX] were unified. The reason for these complicated structures is the flexibility of the dictionary. The TF-, S-, and EX-components can occur in any order.

The other ten productions were the following:

EN → H [I [CG]] TF (TF)* (S | R) (64 examples)
EN → H [I] PI (R | S) (163)
EN → H (PR | R) (428)
EN → H [I [CG]] R EX (63)
EN → H I [CG] SW EX (75)
EN → H I II S (15)
EN → H I [CG] PA [TF] S (68)
EN → H I (PR | ((II | PI) TF | PA [TF]) S | R) (330)
EN → H I CG (EX | PR | R | (PA | TF) S) (306)

These show how the generalization method performs fairly reasonably: for example, the 75 articles containing the elsewhere nonexistent component SW get their own production.

9 Conclusion and further work

In this paper we have presented a method for generating a regular grammar describing the structure of a large text. The method is based on identifying the components in the text, generating a production from each instance of a component, forming finite-state automata from the productions, generalizing the automata first in isolation and then with respect to each other, and then transforming the result to a regular expression for each component.

In the generalization of the examples we have first applied the idea of k-contextual languages and further developed them to (k, h)-contextual languages. The interaction of nonterminals is taken into account by introducing the concept of an $\mathcal{N}$-closed regular language. These conditions seem to describe quite natural constraints in text structures.

The emprical results we have so far seem fairly encouraging; the complexity in the resulting grammars seems largely due to the real complexity in the underlying document. Still, it seems that the generalization conditions should be slightly stronger to give smaller grammars. More experimentation is needed to verify this.

Acknowledgements

We thank Thomas Eiter, Pekka Kilpeläinen, and Greger Lindén for useful comments.

References

[1] *Suomen kielen perussanakirja. Ensimmäinen osa (A–K).* Valtion painatuskeskus, Helsinki, 1990.

[2] Helena Ahonen, Heikki Mannila, and Erja Nikunen. Interactive forming of grammars for structured documents by generalizing automata. Report C-1993-17, Department of Computer Science, University of Helsinki, April 1993.

[3] J. André, R. Furuta, and V. Quint. By way of an introduction. Structured documents: What and why? In J. André, R. Furuta, and V. Quint, editors, *Structured Documents*, The Cambridge Series on Electronic Publishing, pages 1–6. Cambridge University Press, 1989.

[4] J. André, R. Furuta, and V. Quint, editors. *Structured Documents.* The Cambridge Series on Electronic Publishing. Cambridge University Press, 1989.

[5] Dana Angluin. Inference of reversible languages. *Journal of the ACM*, 29(3):741–765, 1982.

[6] David Barron. Why use SGML? *Electronic Publishing*, 2(1):3–24, 1989.

[7] Heather Brown. Standards for structured documents. *The Computer Journal*, 32(6):505–514, December 1989.

[8] G. Coray, R. Ingold, and C. Vanoirbeek. Formatting structured documents: Batch versus interactive. In J. C. van Vliet, editor, *Text Processing and Document Manipulation*, pages 154–170. Cambridge University Press, 1986.

[9] R. Furuta, V. Quint, and J. André. Interactively editing structured documents. *Electronic Publishing*, 1(1):19–44, 1988.

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

[11] G.H. Gonnet and F.Wm. Tompa. Mind your grammar: A new approach to modelling text. In *VLDB '87, Proceedings of the Conference on Very Large Data Bases*, pages 339–346, 1987.

[12] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, Reading, MA, 1979.

[13] Oscar H. Ibarra and Tao Jiang. Learning regular languages from counterexamples. *Journal of Computer and System Sciences*, 43(2):299–316, 1991.

[14] Vania Joloboff. Document representation: Concepts and standards. In J. André, R. Furuta, and V. Quint, editors, *Structured Documents*, The Cambridge Series on Electronic Publishing, pages 75–105. Cambridge University Press, 1989.

[15] Pekka Kilpeläinen, Greger Lindén, Heikki Mannila, and Erja Nikunen. A structured document database system. In Richard Furuta, editor, *EP90 – Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography*, The Cambridge Series on Electronic Publishing, pages 139–151. Cambridge University Press, 1990.

[16] Stephen Muggleton. *Inductive Acquisition of Expert Knowledge*. Addison Wesley, Reading, MA, 1990.

[17] Sara Porat and Jerome A. Feldman. Learning automata from ordered examples. *Machine Learning*, 7(2):109–138, 1991.

[18] V. Quint and I. Vatton. Grif: An interactive system for structured document manipulation. In J. C. van Vliet, editor, *Text Processing and Document Manipulation*, pages 200–213. Cambridge University Press, 1986.

[19] Vincent Quint. Systems for the manipulation of structured documents. In J. André, R. Furuta, and V. Quint, editors, *Structured Documents*, The Cambridge Series on Electronic Publishing, pages 39–74. Cambridge University Press, 1989.

[20] Yasubumi Sakakibara. Learning context-free grammars from structural data in polynomial time. In D. Haussler and L. Pitt, editors, *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 330–344, 1988.

[21] Kurt Vanlehn and William Ball. A version space approach to learning context-free grammars. *Machine Learning*, 2(1):39–74, 1987.

[22] W. Woods. Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606, 1970.

A The document structures

The following page contains the original document structures used in the generalization process. The meanings of the abbreviations are the following:

EN = Entry
H = Headword
S = Sense
EX = Example
I = Inflection
TF = Technical field
CG = Consonant gradation
R = Reference
PR = Preferred form
PI = Pronunciation instructions
PA = Parallel form
SW = Stem word
II = Inflection instructions

2470 EN → H S
1787 EN → H EX
1325 EN → H
1122 EN → H I S
1056 EN → H S EX
1031 EN → H I S EX
995 EN → H TF S
574 EN → H I CG S EX
549 EN → H I TF S
387 EN → H I EX
352 EN → H I CG S
329 EN → H R
258 EN → H I TF S EX
232 EN → H TF S EX
195 EN → H TF
171 EN → H I R
138 EN → H I CG TF S
125 EN → H I
117 EN → H TF EX
100 EN → H PR
97 EN → H I CG TF S EX
94 EN → H I PI S
92 EN → H EX S
85 EN → H I CG R
84 EN → H TF R
66 EN → H I S EX TF EX
54 EN → H I PA S EX
53 EN → H I TF R
51 EN → H I CG S EX TF EX
47 EN → H I CG PR
46 EN → H I CG SW EX
45 EN → H I S EX TF S EX
44 EN → H I PR
44 EN → H PI S
42 EN → H I EX S
39 EN → H TF EX S
34 EN → H I PA S
34 EN → H I CG PA S EX
34 EN → H I PI TF S
31 EN → H I S TF S
30 EN → H I TF TF S
29 EN → H I II TF S
29 EN → H I S EX S
29 EN → H I SW EX
28 EN → H I CG S EX TF S EX
24 EN → H I CG EX
24 EN → H S EX S
22 EN → H I R EX
22 EN → H I PI R
22 EN → H TF TF S
21 EN → H R EX
21 EN → H S TF S EX
21 EN → H S EX TF EX
20 EN → H I CG R EX
20 EN → H EX TF S
19 EN → H I TF EX
18 EN → H I PA TF S
18 EN → H I S TF S EX
18 EN → H S TF EX
18 EN → H S EX TF EX S
18 EN → H EX TF EX
17 EN → H I S EX TF S
16 EN → H I TF TF S EX
16 EN → H EX TF S EX
15 EN → H I II S
15 EN → H PA EX
13 EN → H S TF S
12 EN → H I S EX TF EX S
12 EN → H I CG PA S
11 EN → H I S EX S EX
11 EN → H I TF S TF S
11 EN → H I TF S EX S
11 EN → H I CG TF R
11 EN → H PI TF S
10 EN → H I TF EX S
10 EN → H I CG S TF S
10 EN → H I CG S TF S EX
10 EN → H I CG S EX TF EX S
10 EN → H S TF EX S
10 EN → H EX S EX