Forming grammars for structured documents: an application of grammatical inference

Helena Ahonen
Heikki Mannila
Department of Computer Science
University of Helsinki
P.O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, Finland
E-mail: {hahonen,mannila}@cs.helsinki.fi

Erja Nikunen
Research Centre for Domestic Languages
Sörnäisten rantatie 25
FIN-00500 Helsinki, Finland
E-mail: enikunen@domlang.fi

Abstract

We consider the problem of generating grammars for classes of structured documents — dictionaries, encyclopedias, user manuals, and so on — from examples. The examples consist of structures of individual documents, and they can be collected either by converting typographical tagging of documents prepared for printing into structural tags, or by using document recognition techniques. Our method forms first finite-state automata describing the examples completely. These automata are modified by considering certain context conditions; the modifications correspond to generalizing the underlying language. Finally, the automata are converted into regular expressions, and they are used to construct the grammar. In addition to automata, an alternative representation, characteristic k-grams, is introduced. Some interactive operations are also described that are necessary for generating a grammar for a large and complicated document.

1 Introduction

In recent years, writing, storing, and retrieving documents in electronic form has become popular. Most of the non-fiction documents are somehow structured, i.e., they consist of parts that usually form a hierarchy. For instance, a scientific article consists of a title, an abstract, a text body, and references. The text body consists of a list of sections, every section consists of subsections or paragraphs, and so on. Other typical examples of structured documents are dictionaries, encyclopedias, user manuals, and annual reports. Recent surveys of the research concerning structured documents are [AFQ89, Qui89]. The interest in the area has led to the creation of several document standards, of which the best known are Open Document Architecture (ODA) and Standard Generalized Markup Language (SGML) [ODA89, SGM86, Bro89, Gol90].

The common way to describe the structure of a document is to use context-free grammars. It is typical to use regular expressions on 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 [Inflection]
(Sense_Number Description
[Parallel_form | Preferred_form] Example*)*,

which states that an entry consists of a headword followed by an optional inflection part and zero or more groups, each group consisting of a sense number, a description, a further optional part which is either a parallel form or a preferred form, and a sequence of zero or more examples.

Grammars can be used to facilitate transformations and queries that have structural conditions. The grammar also provides general knowledge of the text. It can be fairly complicated, however, to find the grammar that describes the structure of a given large text. A dictionary, for instance, may contain thousands of entries, and their structures may vary considerably.

Although it is difficult to construct a grammar for a set of documents, it is rather easy to name the parts of an individual document and construct simple productions from them. For instance the structure of the following dictionary entry:

delta: 1. a letter in the Greek alphabet. *2. (geographic)* land at the mouth of a river.

can be described with the simple productions:

Entry → Headword Sense Sense
Sense → Sense_number Description
Sense → Sense_number Technical_field Description

There are several ways to collect individual structures. When the sample is going to be rather small, or the user is just designing a new document structure, maybe the best way is to use an interactive program. Such a program is implemented in connection with the structured text database system HST [KLMN90]. It lets the user paint a part of a text with a pointing device and give a name to that part. The productions are constructed automatically from this information. The marking with a pointing device is inappropriate when the text considered is large and has a complicated structure. If this kind of text, like a dictionary, has been prepared for printing it is usually typographically tagged, i.e., parts of the text are marked by begin and end tags (e.g. begin bold – end bold). Since typographical means are used to make the structure clear to the reader, they can be used to make the structure explicit: tags can be changed to structural (SGML) tags (e.g. begin headword – end headword). The entries are further modified by removing the text and the end tags, and simple productions are formed. Nested tags form productions of their own. If the documents do not have any typographical tagging, it may be possible to use some document recognition techniques [TYS94, HI94, SEM94, SZ86] that try to recognize the parts using, for instance, keywords, fonts, or the physical layout of the text.

Since the simple productions are based on some specific parts of the text, they are overly restrictive and hence, they cannot be used as the grammar describing the structure of the whole set of documents. Thus, one should be able to generalize the productions in some meaningful way. For the generalization, we formulate the problem as a grammatical inference problem [AS83].

The method we have developed proceeds as follows.

1. The example productions are transformed to a set of finite automata, one for each nonterminal. These automata accept exactly the right-hand sides of the example productions for the corresponding nonterminal.

2. Each automaton is modified in isolation, 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. Steps 1 and 2 are based on the synthesis of finite automata presented in [Ang82, Mug90], specifically (k, h)-contextuality is a modification of k-reversibility [Ang82] and k-contextuality [Mug90].

3. The resulting automata are transformed to regular expressions, which form the right-hand sides of the productions for the corresponding nonterminals.

Practically our setting differs from the usual theoretical grammatical inference problems. The alphabet is not known beforehand. The user has not necessarily an underlying target language in mind: the target can change during the process, and the user can make errors. There are always a finite number of examples. In addition to (k, h)-contextuality, an important criterion of success is user's satisfaction that is affected by the clarity and compactness of the resulting grammar. Due to these practical reasons our automatic inference method is augmented by some interactive operations that allow the user to influence the learning process (See Section 6).

A related approach to the grammar generating problem is presented in [FX94]. While containing more developed environment for recognizing document structures it is not intended to be used with very large and complicated structures as our method is.

The rest of this paper is organized as follows. Section 3 describes our method of generalizing the right-hand sides of productions. Section 4 introduces an alternative representation to automata: a set of characteristic

k-grams. Section 5 describes the conversion of generalized automata into regular expressions, and Section 6 interactive operations necessary in generating a grammar for a large document. Section 7 gives some experimental results. Finally, Section 8 contains some concluding remarks.

2 Basic 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., [HU79] 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 regular expression $(r|\epsilon)$ can be rewritten as $[r]$.

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

3 Generalization of productions

The right-hand sides of productions obtained from the user's examples are represented by an automaton called a prefix-tree automaton. To construct a prefix-tree automaton we first take the set of sample productions that have the same left-hand side. The right-hand sides of these productions are added to the prefix-tree automaton one by one. For example, if the following productions are added into a prefix-tree automaton, the result is the automaton shown in Figure 1.

Entry $\to$ Headword Inflection Example Example
Entry $\to$ Headword Inflection Parallel_form Example Example Example
Entry $\to$ Headword Parallel_form Example Example
Entry $\to$ Headword Preferred_form Example
Entry $\to$ Headword Inflection Preferred_form Example Example

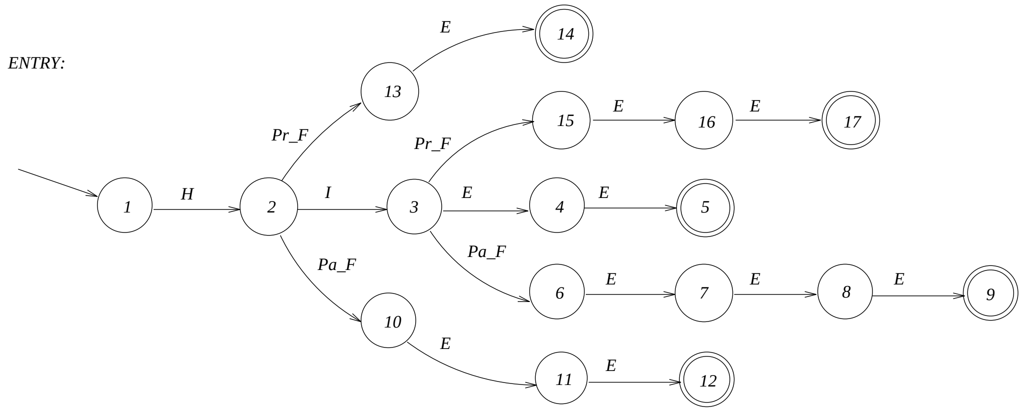

Figure 1: Prefix-tree automaton containing all the examples.

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

The examples of productions are all positive examples. 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. Our assumption is that the grammars used in structured documents have only limited context in the following sense. If a sufficiently long sequence of nonterminals occurs in two places, the components that can follow this sequence are independent of the position of the sequence in the document structure. A language satisfying this condition is called k-contextual [Mug90]. The condition of k-contextuality can be described simply in terms of automata.

Definition 1 A regular language L is k-contextual if and only if there is 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 $a_1 a_2 \dots a_k$ we have: if there are states p_0 and q_0 of A such that $\delta(p_0, a_1 a_2 \dots a_k) = p_k$ and $\delta(q_0, a_1 a_2 \dots a_k) = q_k$, then $p_k = q_k$.

The above definition gives a way of constructing a k-contextual automaton containing language $L(C)$ for an automaton C. States of C satisfying the conditions in the implication of the definition are merged until no such states remain.

The intuition in using k-contextuality is that two occurrences of a sequence of components of length k implies that the subsequent components can be the same in both cases. We have relaxed this condition and generalized 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 of these k characters. As for k-contextuality, we obtain an easy characterization in terms of automata.

Definition 2 A regular language L is (k,h)-contextual if and only if there is 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 $a_1 a_2 \dots a_k$ we have: if there are states p_0 and q_0 such that $\delta(p_0, a_1) = p_1, \delta(p_1, a_2) = p_2, \dots, \delta(p_{k-1}, a_k) = p_k$ and $\delta(q_0, a_1) = q_1, \delta(q_1, a_2) = q_2, \dots, \delta(q_{k-1}, a_k) = q_k$, then $p_i = q_i$, for every i, where $0 \le h \le i \le k$.

The algorithm for producing the automaton that acceps a (k,h)-contextual automaton is similar to the previous algorithm: one looks for states satisfying the conditions of the above definition, 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 2 we can see the final (2,1)-contextual automaton.

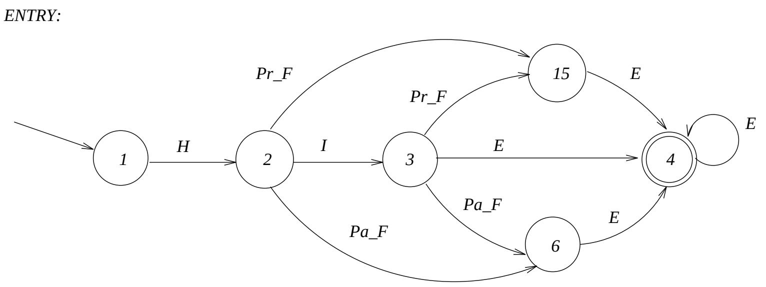

Figure 2: (2,1)-contextual automaton.

4 Characteristic k-grams

If an automaton is k-contextual, all the paths of length k that contain the same sequence of input symbols end at the same state. This feature illustrates the point that for any k-contextual language L there exists a finite set of strings of length $k+1$ that uniquely identifies L. We call these strings *characteristic k-grams*, and they are formally defined as follows.

Definition 3 Let L be any (k,h)-contextual language. Then the set of *characteristic k-grams*, $grams_k(L)$, for L is the set

$$\{u \mid u \text{ is a substring of } \#^k w\#,\ |u| = k+1,\ w \in L\}.$$

The definition extends to sets in a natural way. As any (k,h)-contextual language is also k-contextual, this definition applies also to (k,h)-contextual languages.

Example 4 The characteristic 2-grams of the language accepted by the automaton in Figure 2 are { # # H, # H Pr_F, # H I, # H Pa_F, H Pr_F E, I Pr_F E, H I Pr_F, H I E, H I Pa_F, H Pa_F E, I Pa_F E, Pr_F E E, Pa_F E E, I E E, E E E, Pr_F E #, Pa_F E #, I E #, E E # }

A set of k-grams can be used as a "machine", like automata and grammars.

Definition 5 Let CG be a set of k-grams over alphabet Σ. Then CG *generates* a language $L(CG) = \{w \mid grams_k(\{w\}) \subset CG\}$. A k-gram of the form $\#^k a$, where $a \in \Sigma$, is called an initial k-gram. Respectively, a k-gram of the form $u\#$, where $u \in \Sigma^k$ is called a final k-gram.

Any (k, h)-contextual automaton can be converted into a set of characteristic k-grams [Aho94]. Hence, the set of k-grams can be used as an efficient representation alternative to automata. Generalizing a set of examples to a k-contextual language can be done simply by adding all substrings of length $k+1$ to the set of k-grams. The (k, h)-contextual languages require additionally that the set is the set of k-grams for some (k, h)-contextual language.

5 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 or k-grams. 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 [HU79]. The expressions produced by the standard method are not always so short as they could be, and therefore they have to be simplified. The simplified regular expressions form the right-hand sides of the productions for the corresponding nonterminals. Sample productions in Section 3 generate the production:

Entry → Headword
(Inflection [Preferred_form | Parallel_form] |
Parallel_form | Preferred_form)
Example*

6 Interactive operations

Constructing a grammar for a large text may be quite a complex task. The sample text may, for instance, contain erroneous or rare structures, or the structures may be somehow insufficient. That is why it is important to offer — in addition to automatic learning — various interactive operations for influencing the learning process. These tools include displaying a list of examples of accepted structures and a possibility to remove some structures from this list. The user is also allowed to introduce more structure to make the resulting productions simpler. We also consider ways to use frequency information for separating exceptional cases from the more common ones.

If the user cannot understand the resulting grammar, it is often useful if the system can display a finite sample which somehow characterizes the grammar. The user can use the sample also to choose some structure to be removed.

Example 6 A characteristic sample constructed from the 2-grams of Example 4: H Pr_F E E E, H Pr_F E E, H Pr_F E, H Pa_F E E E, H Pa_F E E, H Pa_F E, H I E E E, H I E E, H I E, H I Pr_F E E, H I Pr_F E, H I Pa_F E E, H I Pa_F E.

It is quite usual that a large document contains some errors or very exceptional cases. Thus, the user may want to remove some structures. Some deletions, however, have to be refused, since the set of k-grams (or the automaton) has to be (k, h)-contextual also after operation.

If the document structure is very complicated, it is often useful to be able to introduce more nested structure than is present in the examples. For instance in our dictionary entries there are clearly nonterminals that form the first part of the entry (Headword, Inflection) and others that appear in the end (Parallel_form, Preferred_form, Example). Since both of these parts vary a lot, processing them separately reduces the complexity of the productions remarkably. The separation can be done by isolating a set of nonterminals to form a new nonterminal.

Generalizing to (k, h)-contextual languages does not necessarily produce a simple expression of the structure. Frequency information can be used for quantifying the importance of different types of structures for the component. It is desirable to get one or a few productions that cover most of the examples, and then several productions that correspond to the exceptions.

Frequency information can also be used in the following way. The user chooses the most common examples and let the learning program generate a grammar. The grammar is then used to parse rest of the example texts. If an example cannot be parsed, either the grammar is modified or the user changes the example. The latter gives the user a possibility to correct errors. The user should also be provided some statistical information, for instance the percentage of examples that the current grammar covers.

7 Experimental results

We have implemented the automatic generalization of the right-hand sides in C++ and, additionally, isolation of nonterminals and separation of common cases. As a representation of data we have used sets of $(2, 1)$-contextual 2-grams.

Our principal test data was the part $A - K$ of a Finnish dictionary [Suo90]. The dictionary was originally typographically tagged, i.e., parts of the text were marked by begin and end tags to show how certain parts should be

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 PrF
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 PaF S EX	53 EN → H I TF R
51 EN → H I CG S EX TF EX	47 EN → H I CG PrF
46 EN → H I CG BW EX	45 EN → H I S EX TF S EX
44 EN → H I PrF	44 EN → H PI S
42 EN → H I EX S	39 EN → H TF EX S
34 EN → H I PaF S	34 EN → H I CG PaF 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 BW 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	

Figure 3: Sample dictionary productions

printed (e.g. begin bold – end bold). We converted the typographical tags of our data, which consist of about 16000 entries, to structural tags, and obtained a set of 468 distinct productions. Every production also received a frequency, i.e., the number of entries that the production covers. We chose 55 of the most common productions (Figure 3), which together covered 14791 entries. As a result of the automatic generalization we got the production shown in Figure 4. The result is clearly unreadable and useless: some interactive operations are needed.

We assume that the user has some knowledge of the target document. In the case of the dictionary, for instance, we assume that the user knows or notices that some parts of a dictionary entry always appear in the beginning of the entry, and some parts in the end. Hence, as the next step in the test we isolated parts *Headword (H), Inflection (I), Consonant_gradation (CG), Inflection_instructions (II),* and *Pronunciation_instructions (PI)* and constructed a new nonterminal *Headword_part (HP)*. In Figure 5 we can see

EN → H[[I[CG]](PrF|[[S][EX(S EX)* [S]] TF (([S] EX (S EX)* [S] | S) TF)*] R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX [(S EX)* [[S] TF (([S] EX (S EX)* [S] | S) TF)* [S] EX [(S EX)* [S EX]] |S EX]] | [S] [[EX (S EX)* [S]] TF (([S] EX (S EX)* [S] | S) TF)* [S]] EX [(S EX)* [S EX]] | [[S] [EX (S EX)* [S]] TF (([S] EX (S EX)* [S] | S) TF)*] [R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX (S EX)* [[S] TF (([S] EX (S EX)* [S] | S) TF)* [[S] EX (S EX)*]] | [S] EX (S EX)*] S) |I [(II | [CG] (BW EX (S EX)* [S] | PaF S [EX (S EX)* [S]])) TF [((([S] EX (S EX)* [S] | S) TF)* [R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX (S EX)* [S] TF [((([S] EX (S EX)* [S] | S) TF)* [([S] EX (S EX)* [S] | S) TF]] | ([S] EX (S EX)* [S] | S) TF]] | II TF (([S] EX (S EX)* [S] | S) TF)* (R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX [(S EX)* [[S] TF (([S] EX (S EX)* [S] | S) TF)* [S] EX [(S EX)* [S EX]] | S EX]] | [S] EX [(S EX)* [S EX]]) | [CG] (BW EX [(S EX)* [[S] TF (([S] EX (S EX)* [S] | S) TF)* (R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX [(S EX)* [[S] TF (([S] EX (S EX)* [S] | S) TF)* [S] EX [(S EX)* [S EX]] |S EX]] | [S] EX [(S EX)* [S EX]]) |S EX]] | PaF S ([EX (S EX)* [S]] TF (([S] EX (S EX)* [S] | S) TF)* R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX [(S EX)* [[S] TF (([S] EX (S EX)* [S] | S) TF)* [S] EX [(S EX)* [S EX]] |S EX]] | [[EX (S EX)* [S]] TF (([S] EX (S EX)* [S] | S) TF)* [S]] EX [(S EX)* [S EX]])) | II TF (([S] EX (S EX)* [S] | S) TF)* [R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX (S EX)* [[S] TF (([S] EX (S EX)* [S] | S) TF)* [[S] EX (S EX)*]] | [S] EX (S EX)*] S | [CG] (BW EX (S EX)* [[S] TF (([S] EX (S EX)* [S] | S) TF)* R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX (S EX)*] [[S] TF (([S] EX (S EX)* [S] | S) TF)* [[S] EX (S EX)*]] S | PaF S [([EX (S EX)* [S]] TF (([S] EX (S EX)* [S] | S) TF)* [R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX (S EX)* [[S] TF (([S] EX (S EX)* [S] | S) TF)* [[S] EX (S EX)*]] | [S] EX (S EX)*] | EX (S EX)*) S])] | ([I] PI [([(S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)*] R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* | S) EX (S EX)*] [S] | [I [CG]] ([[S] [EX (S EX)* [S]] TF (([S] EX (S EX)* [S] | S) TF)*] R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX (S EX)* [S] | [S] [EX (S EX)* [S]])) TF [((([S] EX (S EX)* [S] | S) TF)* [([S] EX (S EX)* [S] | S) TF]] | (I (II | [CG] (BW EX (S EX)* [S] | PaF S [EX (S EX)* [S]])) TF (([S] EX (S EX)* [S] | S) TF)* | [I] PI [(S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)*] | [I [CG]] [[S] [EX (S EX)* [S]] TF (([S] EX (S EX)* [S] | S) TF)*]) R [(EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* [EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R]] | [I] PI ([(S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)*] R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX [(S EX)* [[S] TF (([S] EX (S EX)* [S] | S) TF)* [S] EX [(S EX)* [S EX]] |S EX]] | ((S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* [S] | S) EX [(S EX)* [S EX]] | ([(S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)*] R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX (S EX)* [[S] TF (([S] EX (S EX)* [S] | S) TF)* [[S] EX (S EX)*]] | (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* [[S] EX (S EX)*]) S | S (EX S)*)]

Figure 4: The result of the automatic generalization

the new production for HP and the production for EN where all the occurrences of H, I, CG, II, and PI are replaced by HP.

HP → H [I [II | CG] | [I] PI]

EN → HP [PrF| [([[PaF] S] [EX (S EX)* [S]] | BW EX (S EX)* [S]) TF (([S] EX (S EX)* [S] | S) TF)*] R [(EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* [EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R]] | (PaF S [EX (S EX)* [S]] | BW EX (S EX)* [S]) TF [(([S] EX (S EX)* [S] | S) TF)* [R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX (S EX)* [S] TF [(([S] EX (S EX)* [S] | S) TF)* [([S] EX (S EX)* [S] | S) TF]] | ([S] EX (S EX)* [S] | S) TF]] | ([[S] [EX (S EX)* [S]] TF (([S] EX (S EX)* [S] | S) TF)*] R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX (S EX)* [S] | [S] [EX (S EX)* [S]]) TF [(([S] EX (S EX)* [S] | S) TF)* [([S] EX (S EX)* [S] | S) TF]] | PaF S [[EX (S EX)* [S]] TF (([S] EX (S EX)* [S] | S) TF)* R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX [(S EX)* [[S] TF (([S] EX (S EX)* [S] | S) TF)* [S] EX [(S EX)* [S EX]] |S EX]] | [[EX (S EX)* [S]] TF (([S] EX (S EX)* [S] | S) TF)* [S]] EX [(S EX)* [S EX]] | ([EX (S EX)* [S]] TF (([S] EX (S EX)* [S] | S) TF)* [R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX (S EX)* [[S] TF (([S] EX (S EX)* [S] | S) TF)* [[S] EX (S EX)*]] | [S] EX (S EX)*] | EX (S EX)*) S] | BW EX [(S EX)* [[S] TF (([S] EX (S EX)* [S] | S) TF)* (R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX [(S EX)* [[S] TF (([S] EX (S EX)* [S] | S) TF)* [S] EX [(S EX)* [S EX]] |S EX]] | [S] EX [(S EX)* [S EX]]) |S EX]] | [[S] [EX (S EX)* [S]] TF (([S] EX (S EX)* [S] | S) TF)*] R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX [(S EX)* [[S] TF (([S] EX (S EX)* [S] | S) TF)* [S] EX [(S EX)* [S EX]] |S EX]] | [S] [[EX (S EX)* [S]] TF (([S] EX (S EX)* [S] | S) TF)* [S]] EX [(S EX)* [S EX]] | (BW EX (S EX)* [[S] TF (([S] EX (S EX)* [S] | S) TF)* R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX (S EX)*] [[S] TF (([S] EX (S EX)* [S] | S) TF)* [[S] EX (S EX)*]] | [[S] [EX (S EX)* [S]] TF (([S] EX (S EX)* [S] | S) TF)*] [R (EX (S EX)* [S] TF (([S] EX (S EX)* [S] | S) TF)* R)* EX (S EX)* [[S] TF (([S] EX (S EX)* [S] | S) TF)* [[S] EX (S EX)*]] | [S] EX (S EX)*]) S]

Figure 5: Headword_part (HP) has been isolated from Entry (EN).

The isolation process was continued further. In the next step we isolated parts *Sense, Example,* and *Technical_field*, and formed a new nonterminal *Sense_group (SG)*:

SG → ([S] ([EX (S EX)* [S]] TF [(([S] EX (S EX)* [S] | S) TF)* [([S] EX (S EX)* [S] | S) TF]] | [[EX (S EX)* [S]] TF (([S] EX (S EX)* [S] | S) TF)* [S]] EX [(S EX)* [S EX]]) | [[S] [EX (S EX)* [S]] TF (([S] EX (S EX)* [S] | S) TF)*] [[S] EX (S EX)*] S)

HP → H [I [II | CG] | [I] PI]

EN → HP [PrF | [[PaF | BW] SG] R (SG R)* | (PaF | BW) SG (R SG)* | [[SG] R (SG R)*] SG]

Sense_group looked still quite complicated, and there seemed to be repeating groups containing parts *Sense* and *Example* only. Therefore these nonterminals were isolated to form a new nonterminal *Sense_and_example (SE)*:

SE → ([S] EX (S EX)* | [[S] EX (S EX)*] S)

SG → ([SE] TF (SE TF)* | [[SE] TF (SE TF)*] SE)

HP → H [I [II | CG] | [I] PI]

EN → HP [PrF | [[PaF | BW] SG] R (SG R)* | (PaF | BW) SG (R SG)* | [[SG] R (SG R)*] SG]

Finally, some complexity of the production for *Entry* was possible to be removed by isolating the nonterminals *Sense_group* and *Reference*. They formed a new nonterminal *Sense_group_and_reference (SGR)*:

SGR → ([SG] R (SG R)* | [[SG] R (SG R)*] SG)

SE → ([S] EX (S EX)* | [[S] EX (S EX)*] S)

SG → ([SE] TF (SE TF)* | [[SE] TF (SE TF)*] SE)

HP → H [I [II | CG] | [I] PI]

EN → HP [PrF | [PaF | BW] SGR]

In a dictionary some structures are typically very common, and, on the other hand, most of the structures occur in one or two entries only. Hence, the rare cases can complicate the productions unnecessary. Therefore, the next step in the test was to separate the common cases. The threshold was 500, i.e., all the k-grams in a set describing the right-hand side had to appear at least 500 times in the examples. In the following the first production of each pair is the common one and the second one covers the exceptions.

SGR → (R | SG)
SGR → ([[SG] R (SG R)*] SG | [SG] R (SG R)*)

SE → ([S] EX | S)
SE → ([S] EX (S EX)* | [[S] EX (S EX)*] S)

SG → (TF | [TF] SE)
SG → ([[SE] TF (SE TF)*] SE | [SE] TF (SE TF)*)

HP → H [I [CG]]
HP → H (I II | [I] PI)

EN → HP [SGR]
EN → HP ((PaF | BW) SGR | PrF)

To check the correctness of the productions, the grammar was used to parse the examples, and the result was that it could parse all of them. Of the whole data of 15970 entries it could parse 15663 entries, which means that it covers 872 entries more than the original examples.

8 Conclusion

We have presented a method for generating a context-free grammar for a set of documents from the user's examples. The user provides names to the parts of existing texts. These names are used to form simple productions, which are then generalized and combined to form a grammar.

The following steps are taken:

1. The user describes the structure of the example text by extracting a part and giving it a name. These names form simple productions.
2. A prefix-tree automaton is built for every nonterminal that appears on the left-hand sides of the productions.
3. The automata are generalized by merging states.
4. The regular expressions are constructed which form the right-hand sides of the productions.

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. These conditions seem to describe quite natural constraints in text structures. We have also presented an alternative representation for (k, h)-contextual automata: a set of (k, h)-contextual characteristic k-grams.

Additionally, we have presented some interactive operations that are necessary in generating a grammar for a large and complicated document. These operations include listing a characteristic sample, deletion of examples, and introducing more nested structure by isolating nonterminals to form new nonterminals. The possibility to use frequency information to separate rare and erroneous cases from the more probable ones is also considered.

We have implemented parts of the described method in C++, and experimented with several document types. Most challenging of these experiments was generating a grammar for a part of a Finnish dictionary. The experiments show that if documents are rather simple, the automatic generalization gives sufficient results. However, interactive operations are necessary, if the structures are more complicated, and especially if they contain iterations of slightly different combinations of parts. Although in the dictionary some parts can appear almost in any order and automatic generalization gives totally unreadable productions, breaking the problem to smaller pieces using isolation of nonterminals yields a small and readable grammar.

References

[AFQ89] 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.

[Aho94] Helena Ahonen. *Generating grammars for structured documents using grammatical inference methods.* PhD thesis, University of Helsinki, Department of Computer Science, 1994. in preparation.

[Ang82] Dana Angluin. Inference of reversible languages. *Journal of the ACM*, 29(3):741–765, 1982.

[AS83] Dana Angluin and Carl H. Smith. Inductive inference: Theory and methods. *Computing Surveys*, 15(3):237–269, 1983.

[Bro89] Heather Brown. Standards for structured documents. *The Computer Journal*, 32(6):505–514, December 1989.

[FX94] Peter Fankhauser and Yi Xu. Markitup! An incremental approach to document structure recognition. *Electronic Publishing – Origination, Dissemination and Design*, 6(4):447–456, 1994.

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

[HI94] Tao Hu and Rolf Ingold. A mixed approach toward an efficient logical structure recognition from document images. *Electronic Publishing – Origination, Dissemination and Design*, 6(4):457–468, 1994.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, Reading, MA, 1979.

[KLMN90] 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.

[Mug90] Stephen Muggleton. *Inductive Acquisition of Expert Knowledge.* Addison Wesley, Reading, MA, 1990.

[ODA89] Information Processing – Text and Office Systems – Office Document Architecture (ODA) and Interchange Format. Technical Report ISO/IEC 8613, International Organization for Standardization ISO/IEC, Geneva/New York, 1989.

[Qui89] 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.

[SEM94] Giovanni Semeraro, Floriana Esposito, and Donato Malerba. Learning contextual rules for document understanding. In *Proceedings of the Tenth IEEE Conference on Artificial Intelligence for Applications*, pages 108–115, 1994.

[SGM86] Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML). Technical Report ISO/IEC 8879, International Organization for Standardization ISO/IEC, Geneva/New York, 1986.

[Suo90] *Suomen kielen perussanakirja. Ensimmäinen osa (A–K).* Valtion painatuskeskus, Helsinki, 1990.

[SZ86] S.N. Srihari and G.W. Zack. Document image analysis. In *Proceedings of the Eighth International Conference on Pattern Recognition*, pages 434–436. IEEE Computer Society Press, 1986.

[TYS94] Yuan Yan Tang, Chang De Yan, and Ching Y. Suen. Document processing for automatic knowledge acquisition. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):3–21, 1994.