# Generating grammars for SGML tagged texts lacking DTD *

Helena Ahonen
University of Helsinki

Heikki Mannila
University of Helsinki

Erja Nikunen
Research Centre for Domestic Languages

**Abstract**

We describe a technique for forming a context free grammar for a document that has some kind of tagging — structural or typographical — but no concise description of the structure is available. The technique is based on ideas from machine learning. It forms first a set of finite-state automata describing the document completely. These automata are modified by considering certain context conditions; the modifications correspond to generalizing the underlying languages. Finally, the automata are converted into regular expressions, which are then used to construct the grammar. An alternative representation, characteristic $k$-grams, is also introduced. Additionally, the paper describes some interactive operations necessary for generating a grammar for a large and complicated document.

**Keywords:** structured documents, SGML, grammatical inference, inductive inference, context-free grammars

*This work was partially supported by the Academy of Finland. Authors' addresses: Helena Ahonen, Heikki Mannila, Department of Computer Science, University of Helsinki, P.O. Box 26 (Teollisuuskatu 23), FIN-00014 University of Helsinki, Finland. Erja Nikunen, Research Centre for Domestic Languages, Sörnäisten rantatie 25, FIN-00500 Helsinki, Finland. e-mail: {`hahonen,mannila`}`@cs.Helsinki.FI`, `enikunen@domlang.FI`

# 1 Introduction

There are a number of documents that have Standard Generalized Markup Language (SGML) [1] tags but no Document Type Definition (DTD), i.e., a grammar. There are also documents that have typographical tagging to be used in printing. If a large document, like a dictionary, has been prepared for printing, it is usually typographically tagged, i.e., parts of the text are circled by begin and end marks (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 SGML tags (e.g. begin headword – end headword). While tags can be quite easily converted into SGML tags, finding a DTD for the document is not that easy since the structure can be very complicated.

Our method constructs a context free grammar from SGML tagged text. A grammar can be transformed quite straighforwardly to a DTD. The DTD can then be used to facilitate transformations and queries that have structural conditions. Additionally, most structure editors require a document to have a DTD. The structure also provides general knowledge of the document. The problem of DTD generation has also been studied in [2] and [3] but the approaches differ from the one presented here.

**Example 1** *This example illustrates the conversion process described above. Part of a Finnish dictionary [4] containing Word Perfect commands for e.g. understriking and highlighting (in Finnish):*

```
<A><L>kaame/a<a><l><Y>15<y> kammottava, kamala,
kauhea, karmea, hirveä, pelottava.
<A>Kaamea onnettomuus, verityö. Tuliaseet
tekivät kaameaa jälkeä. Kertoa kaameita
kummitusjuttuja.<a> <L>Ark.<l> <A>Kaamea hattu.
On kaamean kylmä.<a> <A><L>kaameasti<a><l>
<A>Sireenit ulvoivat kaameasti.<a>
<A><L>kaameus<a><l><Y>40<y> <A>Sodan kaameus.<a>
```

*The same part with WP commands converted to structural SGML tags:*

```
<Entry><Headword>kaame/a</Headword>
<Inflection>15</Inflection>
<Sense> kammottava, kamala, kauhea, karmea,
hirveä, pelottava </Sense>. <Example_block>
<Example>Kaamea onnettomuus, verityö. </Example>
<Example> Tuliaseet tekivät kaameaa jälkeä.
</Example>
<Example> Kertoa kaameita kummitusjuttuja.
</Example> </Example_block> <Sense_structure>
<Technical_field>Ark.</Technical_field>
<Example_block><Example>Kaamea hattu.</Example>
<Example> On kaamean kylmä.</Example>
</Example_block> </Sense_structure> </Entry>

<Entry><Headword>kaameasti</Headword>
<Example_block><Example>Sireenit ulvoivat kaameasti.
</Example> </Example_block></Entry>

<Entry><Headword>kaameus</Headword>
<Inflection>40</Inflection> <Example_block>
<Example> Sodan kaameus.</Example></Example_block>
</Entry>
```

*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:*

```
Entry -> Headword Inflection Sense Example_block
         Sense_structure
Entry -> Headword Example_block
Entry -> Headword Inflection Example_block
Sense_structure -> Technical_field Example_block
Example_block -> Example Example Example
Example_block -> Example Example
Example_block -> Example
```

In addition to the trivial grammar, which allows any structure anywhere, the set of the example productions is one obvious grammar for a document. This *de-facto* grammar, however, is often quite large and, moreover, it is overly restrictive with respect to updates. Thus, one should be able to generalize the productions in some meaningful way.

For the generalization, we use techniques from machine learning [5, 6] and formulate the problem as a grammatical inference problem (see Section 3). 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 ideas of synthesis of finite automata presented in [7, 5]. Specifically $(k, h)$-contextuality is a modification of $k$-reversibility [7] and $k$-contextuality [5].

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

The rest of this paper is organized as follows. Section 2 gives the basic definitions. Section 3 describes grammar generation as a grammatical inference problem and section 4 our method of generalizing the right-hand sides of productions. Section 5 describes interactive operations necessary in generating a grammar for a large document. Finally, Section 7 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., [8] 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, $\Sigma$ 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. $\epsilon$ 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 $\alpha$ is a regular expression over the alphabet $(N \cup T)^*$. S is a special nonterminal called the start symbol.

## 3 Grammar generation as a grammatical inference problem

A grammatical inference problem can be specified by giving the following items [9]:

1. the class of languages,
2. the hypothesis space, i.e., a set of representations for the languages,

3. for each language, the set of positive and negative examples, and the admissible sequences of examples,
4. the class of inference methods,
5. the criteria for a successful inference.

Let us take as an example the following inference problem. We want to infer a regular language from positive and negative data. The class of languages is regular sets over the alphabet, say $\{0, 1\}$, and the hypothesis space is either deterministic finite automata, nondeterministic finite automata, regular expressions, or regular grammars over the same alphabet. An example of a regular language $L$ is of the form $\langle s, d \rangle$, where $d$ is 'yes' if string $s$ belongs to $L$ and 'no' otherwise. An admissible sequence of examples contains every string over the alphabet at least once. An inference method is any method that can be represented by a computer program that takes a finite initial sequence of examples of any regular language as input, always halts, and produces as output a representation of a chosen hypothesis space. The most important criterion of success is *identification in the limit* which is defined as follows [10]:

**Definition 2** *Method $M$ identifies language $L$ in the limit if, after a finite number of queries, $M$ makes a correct guess and does not alter its guess thereafter. A class of languages is identifiable in the limit if there is a method $M$ such that given any language of the class and given any allowable example sequence for this language, $M$ identifies the language in the limit.*

In machine learning terms, the example productions of our application are all *positive examples*. That is, the user gives no examples of illegal document structures. This is natural for the user, but it can make the learning problem undecidable:

**Theorem 3** *(Gold [10]) Any class of languages containing all the finite languages and at least one infinite language cannot be identified in the limit from only positive samples.*

From this theorem it follows that the class of context-free languages and even the class of regular languages cannot be learned from positive samples. In fact, this is quite natural: a consistent generalization of a set of positive examples would be an automaton accepting all strings! Hence, to learn from positive examples, one needs some restrictions on the allowed result of the generalization, or some background knowledge. Methods presented in [11, 12] use, as extra knowledge, unlabelled derivation trees of sample data. Although SGML tags form derivation trees the methods of [11, 12] are inappropriate for us, because the terminals, i.e., the text of the sample document, do not generally determine the nonterminals. The text varies too much to be used in the inference process. Additionally, we want to generate grammars that are in the abbreviated form, that is, the right-hand sides of productions are regular expressions. For these reasons, our method reduces the inference of context-free grammars to the inference of regular expressions that can appear on the right-hand sides of productions.

# 4 Generalizing right-hand sides of productions

A number of inference methods for regular languages use the schema introduced by [13]: Construct first a prefix-tree automaton that accepts exactly the examples, and generalize the automaton by merging states. Some methods [14] are heuristic in the sense that their result does not belong to any particular class of languages. Some methods [7, 5], however, are characteristic: they guarantee that the result belongs to some specified subclass of regular languages. In the following we consider two such subclasses: $k$-contextual [5] and $(k, h)$-contextual languages.

## 4.1 Prefix-tree automaton

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 → Headword Inflection Example Example
Entry → Headword Inflection Parallel_form
            Example Example Example
Entry → Headword Parallel_form Example Example
Entry → Headword Preferred_form Example
Entry → Headword Inflection Preferred_form Example Example

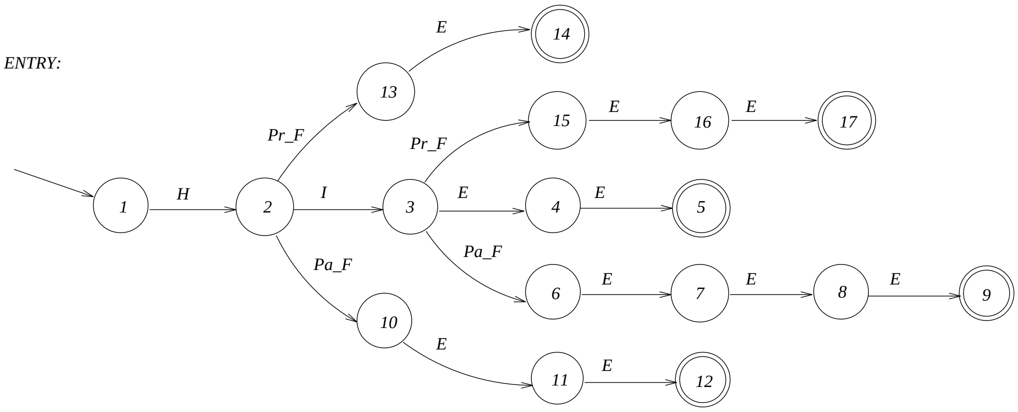


Figure 1: Prefix-tree automaton containing all the examples.

## 4.2 Generalizing automata

A 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. 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 $s_i$. The generic algorithm is the following:

**Algorithm 4** *Generalizing a set of productions using some criterion for merging states.*
**Input:** *A criterion for merging states and a sample*

$$I = \{A \to \alpha \mid A \in N,\ \alpha \in (N \cup T)^*\}$$

*consisting of productions for some nonterminals.*
**Output:** *A set*

$$O = \{A \to \alpha' \mid A \in N,\ \alpha' \text{ is a regular expression over } (N \cup T)\}$$

*of generalized productions such that for all $A \to \alpha \in I$ there is a production $A \to \alpha' \in O$ such that $\alpha$ is an instance of $\alpha'$.*
**Method:**

*1.* **for** *each nonterminal $A$*
*2.* *Construct a prefix-tree automaton $M_A$ from the productions of $I$ with left-hand side $A$*
*3.* **repeat**
*4.* **for** *each pair $p, q$ of states of $M_A$*
**if** *$p$ and $q$ fulfill the generalization condition*
**then** *modify $M_A$ by merging $p$ and $q$*
*5.* **until** *no more states can be merged*
*6.* *Convert $M_A$ to an equivalent regular expression $E_A$*
*7.* *Output the production $A \to E_A$*

## 4.3 (k,h)-contextual languages

How do we choose the generalization condition? 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 in the examples, 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* [5]. The property of $k$-contextuality can be described simply in terms of automata.

**Definition 5** *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 $a_1 a_2 \ldots a_k$ we have: if there are states $p_0$ and $q_0$ of $A$ such that $\delta(p_0, a_1 a_2 \ldots a_k) = p_k$ and $\delta(q_0, a_1 a_2 \ldots a_k) = q_k$, then $p_k = q_k$.*

For a set of strings $H$, a $k$-contextual language $L$ such that

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

is called a *minimal $k$-contextual language including $H$*.

It can be shown that there exists a unique minimal, i.e., the smallest, $k$-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*. Definition 5 and Algorithm 4 give a way of constructing a $k$-contextual automaton that accepts the smallest $k$-contextual language containing $L(C)$ for an automaton $C$. States of $C$ satisfying the conditions in the implication of the definition are merged until no such states remain.

In our example (Figure 1), assuming $k = 2$, we can choose first the paths $s_2 s_{13} s_{14}$ and $s_3 s_{15} s_{16}$ and merge states $s_{14}$ and $s_{16}$; the result is shown in Figure 2. Finally the 2-contextual automaton looks like the one in Figure 3. We can see that it generalizes the examples quite well. The automaton, however, accepts only entries that have two or more *Example* nonterminals in the end. This is overly cautious, and therefore we need a looser generalization condition. In Figure 3, for example the states $s_4$ and $s_5$ could be merged.

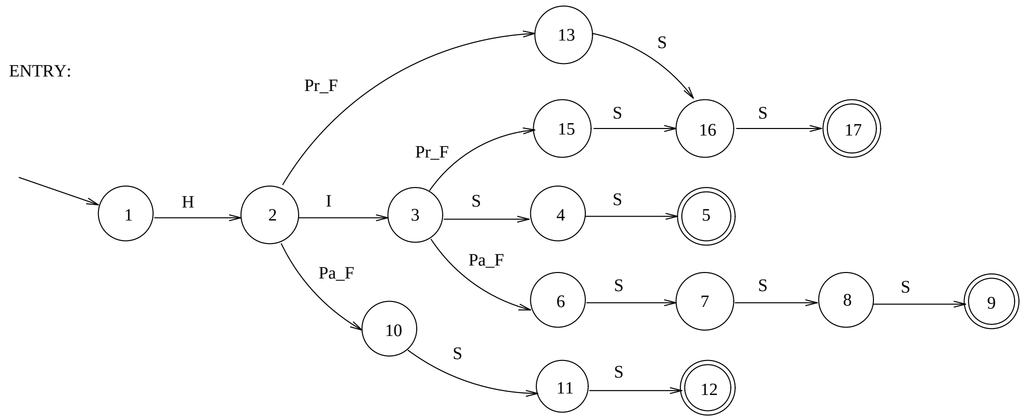


Figure 2: After merging states $s_{14}$ and $s_{16}$.

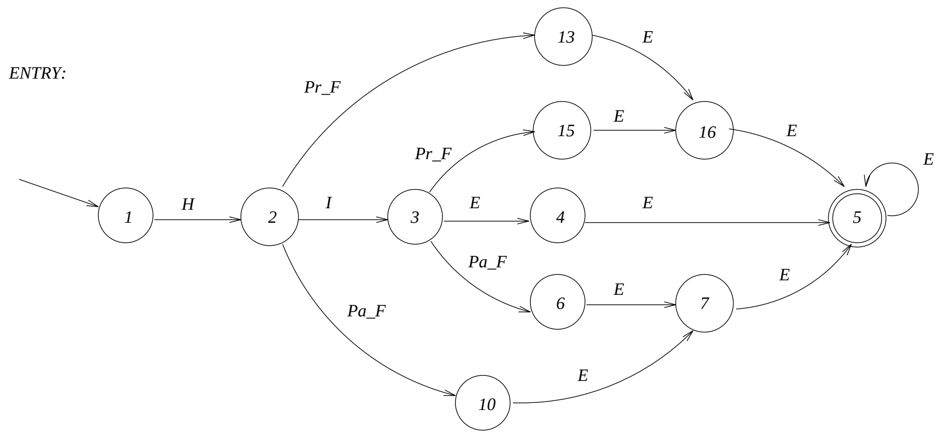


Figure 3: 2-contextual automaton.

The intuition in using $k$-contextuality is that two occurrences of a sequence of components of length $k$ imply that 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 $k$ imply that the subsequent components are the same *already after* $h$ *characters*. As for $k$-contextuality, we obtain an easy characterization in terms of automata.

**Definition 6** *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* $a_1 a_2 \ldots a_k$ *we have: if there are states* $p_0$ *and* $q_0$ *such that* $\delta(p_0, a_1) = p_1$*,* $\delta(p_1, a_2) = p_2$*, …,* $\delta(p_{k-1}, a_k) = p_k$ *and* $\delta(q_0, a_1) = q_1$*,* $\delta(q_1, a_2) = q_2$*, …,* $\delta(q_{k-1}, a_k) = q_k$*, then* $p_i = q_i$*, for every* $i$*, where* $0 \leq h \leq i \leq k$*.*

The algorithm for producing the automaton that accepts a $(k, h)$-contextual language 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 4 we can see the final (2,1)-contextual automaton.

## 4.4 Characteristic k-grams: an alternative representation

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.

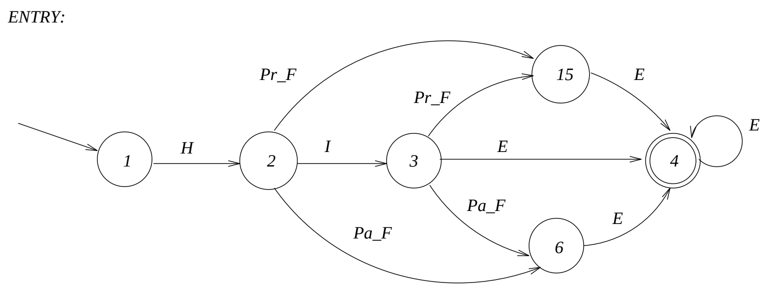


Figure 4: (2,1)-contextual automaton.

**Definition 7** *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 8** *The characteristic* 2*-grams of the language accepted by the automaton in Figure 4 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 9** *Let $CG$ be a set of $k$-grams over alphabet $\Sigma$. 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 [15]. 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. The algorithm is the following:

**Algorithm 10** *Constructing a set of characteristic $k$-grams for the smallest $(k, h)$-contextual language containing a given regular language.*
**Input:** *A regular language represented by a finite set of strings $S$, and positive integers $k$ and $h$.*
**Output:** *A set of characteristic $k$-grams for language $L$ such that $L$ is the smallest $(k, h)$-contextual regular language including $S$.*
**Method:**

*1.* **for** *each* $g \in grams_k(S)$
*2.* $\quad CG := CG \cup \{g\}$
*3.* **if** $CG$ *is not* $(k, h)$*-contextual*
*4.* **then** *Convert it to be* $(k, h)$*-contextual*

A set of characteristic $k$-grams can be made $(k, h)$-contextual by adding $k$-grams as in Algorithm 12. Adding $k$-grams corresponds merging states in automata. First we have to define the concepts of the predecessors and the successors of a $k$-gram.

**Definition 11** *Let $a_i, b,$ and $c \in \Sigma, i > 1$. A $k$-gram $u_1$ is said to be a* predecessor *of a $k$-gram $u_2 = a_1a_2 \cdots a_k$ if $u_1 = ba_1 \cdots a_{k-1}$ or $u_1$ is a predecessor of $ba_1 \cdots a_{k-1}$. Respectively, a $k$-gram $u_1$ is said to be a* successor *of a $k$-gram $u_2 = a_1a_2 \cdots a_k$ if $u_1 = a_2a_3 \cdots a_kc$ or $u_1$ is a successor of $a_2a_3 \cdots a_kc$.*

**Algorithm 12** *Converting a set of $k$-grams to be $(k, h)$-contextual.*
**Input:** *A set of $k$-grams $CG$ for language $L$.*
**Output:** *A set of $k$-grams $CG$ for the smallest $(k, h)$-contextual language $L'$ such that $L \subseteq L'$.*
**Method:**

**for** *each $k$-gram* $g = ab_1 \ldots b_k \in CG$
    **if** *$k$-gram* $f = cb_1 \ldots b_k \in CG$
        **for** *each $k$-gram* $f' = cb_1 \ldots b_h w_1$
            **if** *$f$ and $f'$ have a common predecessor* $wcb_1 \ldots b_h$
                $CG := CG \cup \{ab_1 \ldots b_h w_1\}$
        **for** *each $k$-gram* $g' = ab_1 \ldots b_h w_2$
            **if** *$g$ and $g'$ have a common predecessor* $w'ab_1 \ldots b_h$
                $CG := CG \cup \{cb_1 \ldots b_h w_2\}$

In Section 5 we assume that the representation of characteristic $k$-grams is used instead of automata.

## 4.5 Conversion into a regular expression

After the generalization steps presented in the previous sections have been performed, we have a collection of either $(k, h)$-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 [8].

One of our goals was to obtain a readable grammar. The regular 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 4.1 generate the production:

Entry $\rightarrow$ Headword
(Inflection [Preferred_form | Parallel_form] |
Parallel_form | Preferred_form)
Example*

# 5 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.

## 5.1 Listing a sample

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

**Example 13** *A characteristic sample constructed from the* 2-*grams of Example 8: 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.*

A sample is constructed from the characteristic $k$-grams by the following algorithm.

**Algorithm 14** *Constructing a characteristic sample $S$ from a set of characteristic $k$-grams $CG$.*
**Input:** *Set of characteristic $k$-grams $CG$.*
**Output:** *Characteristic sample.*
**Method:**

*1.* **if** $\#^{k+1} \in CG$ *output* $\epsilon$
*2.* **repeat**
*3.*     *Choose some initial $k$-gram $g = \#^k b \in CG$, and mark it*
*4.*     *Output_rest($\#^{k-1}b, b$)*
*5.* **until** *no unmarked $k$-grams*

*Output_rest($a_1 \cdots a_k, p_1 \cdots p_i$)*
*1.* **for** *each $g = a_1 \cdots a_k b \in CG$*
*2.*     **if** *$g$ is a final $k$-gram*
*3.*         *Mark it and output path $p_1 \cdots p_i$*
*4.*     **else if** *$g$ is already marked*
*5*         *Find the closest final $k$-gram and*
            *output the corresponding path*
*6.*     **else**
*7.*         *Mark($g$)*
*8.*         *Output_rest($a_2 \cdots a_k b, p_1 \cdots p_i b$)*

## 5.2 Removing examples

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. Additionally, it is not allowed to remove a string that is composed of $k$-grams everyone of which is used by some other string that is not supposed to be removed.

**Example 15** *Assume we have a language $L = \{abcd, ebcd, ebg, abg\}$ and a corresponding set of $k$-grams*

$$\{\#\#a, \#\#e, \#ab, \#eb, abc, ebc, bcd, abg, ebg, cd\#, bg\#\}.$$

*If we remove string ebg, a $k$-gram ebg has to be removed. But then the set is not $(k,h)$-contextual anymore, since $k$-grams $abc, ebc$, and abg imply that also ebg has to belong to the set.*

**Algorithm 16** *Removing an example from a $(k,h)$-contextual set of $k$-grams.*
**Input:** *$(k,h)$-contextual set of $k$-grams $CG$ and a set of strings $S$.*
**Output:** *$CG$ that does not accept any $s \in S$, if possible.*
**Method:**

**for** *each* $s \in S$
    **if** $s = a_1 \cdots a_n \in L(A)$
    **then**
        **for** *each $k$-gram $g$*
            *such that $g$ is a substring of $\#^k s\#$*
            *Mark($g$)*
**for** *each marked $k$-gram $g$*
    **if** *$g$ is not a substring of any string $s' \in L \setminus S$*
    **then** $CG := CG - g$

**if** *$CG$ is not $(k,h)$-contextual*
    *Converse it to be $(k,h)$-contextual*

## 5.3 Isolating nonterminals

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, Consonant_gradation) and others that appear in the end (Sense, Example_block). 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.

**Example 17** *Consider the following generalized production:*

Entry → Headword
  [ Example_block |
  Inflection [Example_block | Reference] |
  [Inflection [Consonant_gradation]]
    [Technical_field] [Sense] [Example_block] |
  Reference |
  Preferred_form ]

*If the nonterminals appearing in the beginning and in the end are isolated we get the following simpler productions:*

Entry → Headword_part [Rest_of_the_entry]
Headword_part → Headword [Inflection [Consonant_gradation]]
Rest_of_the_entry → Example_block | Reference | Preferred_form
  | [Technical_field] [Sense] [Example_block]

## 5.4 Separating common and exceptional cases

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.

Adding frequency information to the set of characteristic $k$-grams is easy: each $k$-gram is given a weight that is the number of examples in which this $k$-gram occurs. In our method the user gives a threshold $b$, which means that the program constructs a production which covers at least all the structures that appear $b$ times in data. In addition to this production several exception productions are constructed as well.

Frequency information can also be used in the following way. The user chooses the most common examples and lets the learning program generate a grammar. The grammar is then used to parse the 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.

# 6 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. We have experimented with several different document types, and the results are encouraging.

Most challenging of the documents was the part $A - K$ of a Finnish dictionary [4]. We converted the typographical tags of the dictionary, which consists 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 5), which together covered 14791 entries.

After automatic generalization the result was rather unreadable, but after the following isolations the productions became simpler. First, *Headword (H), Inflection (I), Consonant_gradation (CG), Inflection_instructions (II)*, and *Pronunciation_instructions (PI)* were isolated from *Entry (EN)* to form a new nonterminal *Headword_part (HP)*. Second, *Sense (S), Example (EX),* and *Technical_field (TF)* were isolated from *Entry (EN)*, and a new nonterminal *Sense_group (SG)* was formed. In the next step *Sense_or_Example (SE)*, containing the nonterminals *Sense* and *Example*, was isolated from *Sense_group*, and finally *Sense_group_or_Reference (SGR)* was isolated from *Entry*. The last 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.

| | |
|---|---|
| 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 5: Sample dictionary productions

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.

# 7 Conclusion

In this paper we have presented a method for generating a context-free grammar from a SGML tagged document. The tags are used to form simple productions, which are then generalized and combined to form a grammar. The resulting grammar can then be used to construct a DTD for a document.

The method forms finite-state automata from the right-hand sides of the productions, generalizes the automata, and finally transforms them to regular expressions. In the generalization of the automata we have first applied the idea of $k$-contextual languages and further extended them to $(k, h)$-contextual languages. An alternative representation, the set of characteristic $k$-grams, has also been considered.

We have also 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 isolation of nonterminals to form new nonterminals. The possibility to use frequency information to separate rare and erroneous cases from the more probable ones was also considered.

We have implemented parts of the described method in C++, and experimented with several document types. 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


[1] 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.

[2] Jinhua Chen. Grammar generation and query processing for text databases. Research proposal, University of Waterloo, January 1991.

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

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

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

[6] Balas K. Natarajan. *Machine Learning: A Theoretical Approach.* Morgan Kaufmann Publishers, May 1991.

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

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

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

[10] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[11] Yasubumi Sakakibara. Efficient learning of context-free grammars from positive structural examples. *Information and Computation*, 97(1):23–60, 1992.

[12] Timo Knuutila. Inference of k-testable tree languages. In H. Bunke, editor, *Proceedings of the International Workshop on Structural and Syntactic Pattern Recognition*, pages 109–120, 1993.

[13] Jerome A. Feldman, James Gips, James J. Horning, and Stephen Reder. Grammatical complexity and inference. Report TR CS 125/1969, Stanford University, 1969.

[14] A. W. Biermann and J. A. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Transactions on Computers*, 21(6):592–597, 1972.

[15] Helena Ahonen. *Generating grammars for structured documents using grammatical inference methods.* Ph. Lic. thesis, University of Helsinki, Department of Computer Science, 1994.