

# Standard Generalized Markup Language: Mathematical and Philosophical Issues <sup>†</sup>

Derick Wood\*

Technical Report HKUST-CS95-37  
July 1995

\*Department of Computer Science  
The Hong Kong University of Science & Technology  
Clear Water Bay, Kowloon, Hong Kong

## Abstract

The Standard Generalized Markup Language (SGML), an ISO standard, has become the accepted method of defining markup conventions for text files. SGML is a metalanguage for defining grammars for textual markup in much the same way that Backus–Naur Form is a metalanguage for defining programming-language grammars. Indeed, HTML, the method of marking up a hypertext documents for the World Wide Web, is an SGML grammar. The underlying assumptions of the SGML initiative are that a logical structure of a document can be identified and that it can be indicated by the insertion of labeled matching brackets (start and end tags). Moreover, it is assumed that the nesting relationships of these tags can be described with an extended context-free grammar (the right-hand sides of productions are regular expressions).

In this survey of some of the issues raised by the SGML initiative, I reexamine the underlying assumptions and address some of the theoretical questions that SGML raises. In particular, I respond to two kinds of questions. The first kind are technical: Can we decide whether tag minimization is possible? Can we decide whether a proposed content model is legal? Can we remove exceptions in a structure preserving manner? Can we decide whether two SGML grammars are equivalent?

The second kind are philosophical and foundational: What is a logical structure? What logical structures may a document have? Can logical structures always be captured by context-free nesting?

---

<sup>†</sup>This research was supported by grants from the Natural Sciences and Engineering Research Council of Canada and from the Information Technology Research Centre of Ontario.



# Standard Generalized Markup Language: Mathematical and Philosophical Issues \*

Derick Wood<sup>1</sup>

Department of Computer Science  
Hong Kong University of Science & Technology  
Clear Water Bay, Kowloon  
Hong Kong

**Abstract.** The Standard Generalized Markup Language (SGML), an ISO standard, has become the accepted method of defining markup conventions for text files. SGML is a metalanguage for defining grammars for textual markup in much the same way that Backus–Naur Form is a metalanguage for defining programming-language grammars. Indeed, HTML, the method of marking up a hypertext documents for the World Wide Web, is an SGML grammar. The underlying assumptions of the SGML initiative are that a logical structure of a document can be identified and that it can be indicated by the insertion of labeled matching brackets (start and end tags). Moreover, it is assumed that the nesting relationships of these tags can be described with an extended context-free grammar (the right-hand sides of productions are regular expressions). In this survey of some of the issues raised by the SGML initiative, I reexamine the underlying assumptions and address some of the theoretical questions that SGML raises. In particular, I respond to two kinds of questions. The first kind are technical: Can we decide whether tag minimization is possible? Can we decide whether a proposed content model is legal? Can we remove exceptions in a structure preserving manner? Can we decide whether two SGML grammars are equivalent? The second kind are philosophical and foundational: What is a logical structure? What logical structures may a document have? Can logical structures always be captured by context-free nesting?

## 1 Introduction

The Standard Generalized Markup Language (SGML) came into being as an ISO Standard [18] in 1987 without fanfare and without hyperbole. It has changed forever the way we view and approach textual documents. Its impact in the textual world is greater than the impact that BNF had in the programming-language community thirty years earlier. The similarity does not end there since SGML is also a metalanguage that we can use to specify grammars. But it is also much more.

---

\* This work was supported under Natural Sciences and Engineering Research Council of Canada grants.

We explain a little of SGML's intended use and then explore some issues that are raised by the SGML standard. In this way we explore some fundamental issues stemming from the approach taken by SGML's creators.

Each SGML grammar defines a class of structured documents and also a method, called **markup**, of indicating the structure in the documents using interleaved text, called **tags**. When using typesetting systems such as GML, Scribe, L<sup>A</sup>T<sub>E</sub>X, troff, and T<sub>E</sub>X [25] authors also interleave. with the source text. typesetting commands specific to the typesetting system; such interleaving is called **procedural markup**. Procedural markup is powerful yet restrictive.

It is now axiomatic that markup should identify the component elements of text; that is, it should be **logical** and **descriptive**, rather than procedural. Such marked-up documents are more useful, because not only can they be typeset by compiling them into, for example, T<sub>E</sub>X marked-up documents, but also they can be used directly in textual databases and, hence, searched and manipulated. In addition, if the legal markup is defined by some document grammar, then we can communicate a document (and its structure) that is marked up according to a grammar by communicating both the grammar and the marked-up document. The ISO standard for SGML [18] provides a syntactic metalanguage for the definition of textual markup systems. Each SGML-markup system is specified, essentially, by an **extended context-free grammar (ECFG)** that specifies the legal placement and type of the markup in documents. SGML is a syntactic meta-language for document grammars just as BNF, BNF's various extensions, and syntax diagrams are for programming-language grammars.

The most well-known example of the use of SGML is in the World Wide Web. Pages for inclusion in WWW are marked up with HTML (HyperText Markup Language), which is defined by a specific SGML grammar. The explosive growth of WWW has exposed more people to SGML in a day, than were ever exposed to it before. A second example is the CALS initiative of the U. S. Department of Defense; the use of a specific SGML grammar to define a system-independent markup language for contracts, tenders, and other uses. In addition, organizations such as the American Association of Publishers have defined specific SGML grammars for, in their case, the book-publishing industry.

We begin the discussion of SGML by giving a very brief introduction to it by example and by its similarity with ECFGs. In the following sections we survey the status of current research on the issues of content-model unambiguity, DTD equivalence, exceptions, and omitted tag minimization. Lastly, we discuss some philosophical issues of documents, logical structures, and markup.

The work reported here is only a beginning, new standards have been developed that build on SGML; for example, the style specification standard DSSSL [20] and the hypermedia standard HyTime [19, 14]. While SGML and these additional Standards are important attempts to bring order out of chaos, at the same time, it is crucial that they do not add to the chaos. As a theoretician who has become an occasional practitioner, I believe that while taking advantage of these and similar standards, we must also take them seriously enough to investigate them theoretically. Only in this way can we ensure well designed standards

that are well defined.

## 2 An SGML Primer

SGML [18, 16] promotes the interchangeability and application-independent management of electronic documents by providing a syntactic metalanguage for the definition of textual markup systems. An SGML document consists of an SGML prolog and a marked-up document instance. The prolog contains a **document type definition (DTD)**, which is, essentially, an ECFG. An example of a simple SGML DTD is given in Fig. 1.

```
<!DOCTYPE message [  
<!ELEMENT message          (head, body)>  
<!ELEMENT head             (from & to & subject)>  
<!ELEMENT from             (person)>  
<!ELEMENT to               (person)+>  
<!ELEMENT person           (alias | (forename?, surname))>  
<!ELEMENT body             (paragraph)*>  
<!ELEMENT subject, alias, forename, surname, paragraph  
                        (#PCDATA)> ]>
```

Fig. 1. An SGML DTD for e-mail messages.

The productions of a DTD are called **element type definitions (ETDs)**. The right hand sides of ETDs are extended regular expressions called **content models**. The DTD in Fig. 1 is a DTD for e-mail messages, which consist of a **head** and a **body**, in that order; note the use of ‘,’ to mean catenation or followed by. The element **head** comprises subelements **from**, **to**, and **subject** appearing in any order; note the use of ‘&’ to mean unordered catenation. The element **from** is defined to be a **person**, which consists of either an **alias** or an optional **forename** followed by a **surname**; note the use of ‘|’ for alternation and of ‘?’ for an optional item. The element **to** consists of a nonempty list of **persons**. The **body** of a **message** consists of a (possibly empty) sequence of **paragraphs**. Finally, the last element definition specifies that the elements **subject**, **alias**, **forename**, **surname**, and **paragraph** are unstructured strings, denoted by the keyword **#PCDATA**.

The structural elements of a document instance (sentences in grammatical terminology) are marked up by enclosing them in matching pairs of **start tags** and **end tags**. A document that is marked up according to the DTD of Fig. 1 is shown in Fig. 2.

We model SGML DTDs with ECFGs. In ECFGs the right-hand sides of productions are regular expressions. Let  $V$  be an alphabet; then, the language  $L(E)$

```

<message>
  <head>
    <from><person><alias>Boss</alias></person></from>
    <subject>Tomorrow's meeting...</subject>
    <to><person><surname>Franklin</surname></person>
      <person><alias>Betty</alias></person></to>
  </head>
  <body><paragraph>...has been cancelled.</paragraph></body>
</message>

```

**Fig. 2.** An SGML document instance for the DTD of Fig. 1.

denoted by a regular expression  $E$  over  $V$  is defined inductively as follows.

$$\begin{aligned}
 L(\emptyset) &= \emptyset. & L(\epsilon) &= \{\epsilon\}. \\
 L(a) &= \{a\} \text{ for } a \in V. & L(FG) &= \{vw \mid v \in L(F), w \in L(G)\}. \\
 L(F \cup G) &= L(F) \cup L(G). & L(F^*) &= \{v_1 \cdots v_n \mid n \geq 0, v_1, \dots, v_n \in L(F)\}.
 \end{aligned}$$

We denote the set of symbols of  $V$  appearing in  $E$  by  $\text{sym}(E)$ .

An ECFG  $G$  consists of two disjoint finite alphabets of **nonterminal symbols** and **terminal symbols**, denoted by  $N$  and  $\Sigma$ , respectively, a finite set  $P$  of **production schemas**, and a **sentence symbol**,  $S$ . Each **production schema** has the form  $A \rightarrow E$ , where  $A$  is a nonterminal and  $E$  is a regular expression over  $V = N \cup \Sigma$ . When  $\beta = \beta_1 A \beta_2 \in V^*$ ,  $A \rightarrow E \in P$  and  $\alpha \in L(E)$ , we say that the string  $\beta_1 \alpha \beta_2$  can be **derived** from string  $\beta$  and denote the derivation by  $\beta \Rightarrow \beta_1 \alpha \beta_2$ . The **language  $L(G)$  defined by an ECFG  $G$**  is the set of terminal strings derivable from the sentence symbol of  $G$ . Formally,  $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^+ w\}$ , where  $\Rightarrow^+$  denotes the transitive closure of the derivability relation.

Even though a production schema may give rise to an infinite number of ordinary context-free productions, it is well-known that ECFGs and CFGs describe exactly the same languages [33].

SGML DTDs are similar to ECFGs, yet different from them. In particular, content models are more general than regular expressions in that they have the additional operators: '?', '&', and '+'. In a DTD the alphabet  $V$  comprises **generic identifiers** that are names of **elements** (nonterminals) and **#PCDATA**. The language  $L(E)$  defined by a content model  $E$  is defined inductively as follows:

$$\begin{aligned}
 L(a) &= \{a\} \text{ for } a \in V. & L(FG) &= \{vw \mid v \in L(F), w \in L(G)\}. \\
 L(F|G) &= L(F) \cup L(G). & L(F^*) &= \{v_1 \cdots v_n \mid v_1, \dots, v_n \in L(F), n \geq 0\}. \\
 L(F\&G) &= L(FG|GF). & L(F^+) &= \{v_1 \cdots v_n \mid v_1, \dots, v_n \in L(F), n \geq 1\}. \\
 L(F?) &= L(F) \cup \{\epsilon\}.
 \end{aligned}$$

### 3 Unambiguous Content Models

The following discussion of unambiguity is based on the work of Brüggemann-Klein and Wood [4, 5, 6, 7, 8, 9].

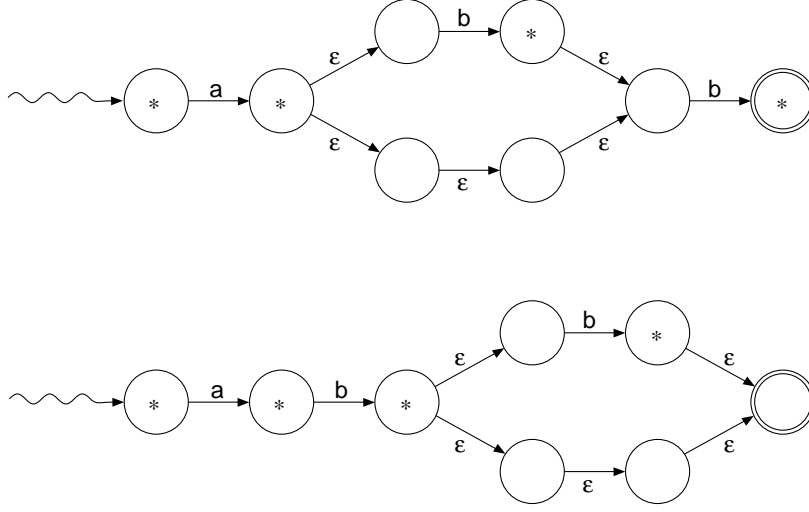
The SGML standard does not allow arbitrary regular expressions as content models, it says that they should be “unambiguous” in the sense that, using formal language terminology, “a symbol that occurs in a string accepted by the regular expression  $E$  must be able to satisfy only one occurrence of this symbol in  $E$  without looking ahead in the string.” This condition is illustrated by the expression  $E = a(b|\epsilon)b$  and the strings  $ab$  and  $abb$ . After the first symbol  $a$  has been matched with the  $a$  in the expression  $E$ , the second symbol  $b$  of the string can either be satisfied by the first or the second  $b$  in  $E$ , depending on the continuation of the string. So, if the lookahead in the string is confined to the current symbol  $b$ , we cannot decide which of the  $b$ 's in  $E$  it should match. Thus,  $E$  is “ambiguous” in the sense of the Standard. In this case, there is an equivalent expression  $E' = ab(b + \epsilon)$  that is “unambiguous.”

The intent of the authors of the Standard is twofold: They wish to make it easier for humans to write and understand expressions that can be interpreted unambiguously and, at the same time, to provide a means of generating parsers that do not have exponential blow up in size. (Such restrictions are familiar in other settings; for example, with `vi` and `grep`.) We call these expressions **1-deterministic**, since it is deterministic parsing with one-symbol lookahead that is implied by the standard, rather than unambiguity per se [9, 6, 8, 7]. Indeed, this view is reflected in Annex H of the standard, where the notion is explained, informally, in terms of the nondeterministic finite-state machine with  $\epsilon$ -transitions obtained from a regular expression. A regular expression is *1-deterministic* if its strings can be recognized deterministically, with one-symbol lookahead, by the corresponding nondeterministic finite-state machine. For example, the lower finite-state machine in Fig. 3 is 1-deterministic, whereas the upper machine in Fig. 3 is not.

Rather than discussing SGML content models directly, we first consider traditional regular expressions. Content models are more general in that they have the additional operators: ‘?’, ‘&’, and ‘+’.

To indicate different **positions** of the same symbol in an expression, we mark symbols with subscripts. For example,  $(a_1|b_1) * a_2(a_3b_2) *$  and  $(a_4|b_2) * a_1(a_5b_1) *$  are both **markings** of the expression  $(a|b) * a(ab) *$ . For each expression  $E$  over  $\Sigma$ , a marking of  $E$  is denoted by  $E'$ . If  $H$  is a subexpression of  $E$ , we assume that markings  $H'$  and  $E'$  are chosen in such a way that  $H'$  is a subexpression of  $E'$ . A **marked expression**  $E$  is an expression over  $\Pi$ , the alphabet of subscripted symbols, where each subscripted symbol occurs at most once in  $E$ .

The reverse of marking is the dropping of subscripts, indicated by  $\natural$  and defined as follows: If  $E$  is an expression over  $\Pi$ , then  $E^\natural$  is the expression over  $\Sigma$  that is constructed from  $E$  by dropping all subscripts. Thus, a marked expression  $H$  is a **marking** of expression  $E$  if and only if  $H^\natural = E$ . Unmarking can also be extended to strings and languages: For a string  $w$  over  $\Pi$ , let  $w^\natural$  denote the string over  $\Sigma$  that is constructed from  $w$  by dropping all subscripts. For a lan-



**Fig. 3.** Thompson's construction [30, 1] for the expression  $a(b|\epsilon)b$  that is not 1-deterministic and for the 1-deterministic expression  $ab(b|\epsilon)$ .

guage  $L$  over  $\Pi$ , let  $L^\sharp$  denote  $\{w^\sharp | w \in L\}$ . Then, for each expression  $E$  over  $\Pi$ ,  $L(E^\sharp) = L(E)^\sharp$ .

Now we can give a concise definition of the SGML notion of unambiguity.

**Definition 1.** An expression  $E$  is **1-deterministic** if and only if, for all strings  $u, v, w$  over  $\Pi$  and all symbols  $x, y$  in  $\Pi$ , the conditions  $uxv, wyw \in L(E')$  and  $x \neq y$  imply that  $x^\sharp \neq y^\sharp$ . A regular language is **1-deterministic** if it is denoted by some 1-deterministic expression.

In other words, for each string  $w$  denoted by a 1-deterministic expression  $E$ , there is exactly one marked string  $v$  in  $L(E')$  such that  $v^\sharp = w$ . Furthermore,  $v$  can be constructed incrementally by examining the next symbol of  $w$  which must match the next position of  $v$ . It is not hard to see that this definition is independent of the marking  $E'$  chosen for  $E$ . We now give an alternative definition in terms of the pairs of positions that follow each other in a string of  $L(E')$ .

**Definition 2.** For each language  $L$ , we define the following sets:

$$first(L) = \{b | \text{there is a string } w \text{ such that } bw \in L\},$$

$$last(L) = \{b | \text{there is a string } w \text{ such that } wb \in L\},$$

$$follow(L, a) = \{b | \text{there are strings } v \text{ and } w \text{ such that } vabw \in L, \}$$

for each symbol  $a$ , and

$$followlast(L) = \{b | \text{there are strings } v \text{ and } w \text{ such that } v \neq \epsilon \in L \text{ and } vbw \in L\}.$$

Furthermore, for each expression  $E$ , we extend the sets to expressions  $E$  by defining  $first(E)$  to be  $first(L(E))$  and similarly for the other sets.

Using these sets we can give an alternative characterization of 1-deterministic expressions that is not difficult to prove.

**Lemma 3.** *An expression  $E$  is 1-deterministic if and only if the following two conditions hold:*

1. *For all  $x, y$  in  $first(E')$ ,  $x \neq y$  implies that  $x^\sharp \neq y^\sharp$ .*
2. *For all  $z$  in  $sym(E')$  and  $x, y$  in  $follow(E', z)$ ,  $x \neq y$  implies that  $x^\sharp \neq y^\sharp$ .*

Glushkov [15] and McNaughton and Yamada [23] were the first to construct finite-state machines from marked expressions using the functions  $first$ ,  $last$ , and  $follow$ ; the machines they construct are, however, deterministic. Motivated by the work of Glushkov, Book et al. [3] define a nondeterministic finite-state machine  $G_E$ , for each expression  $E$ , that we call the **Glushkov machine** of  $E$ . Pin [26] observes that a finite-state machine for an expression  $E$  can be constructed from the  $first$ ,  $last$ , and  $follow$  functions of  $E$  as opposed to a marking of  $E$ , provided that the language denoted by  $E$  is local. Berry and Sethi [2] argue that Glushkov machines are natural representations of regular expressions.

**Definition 4.** We define the Glushkov machine  $G_E = (Q_E, \Sigma, \delta_E, s, F_E)$  of an expression  $E$  as follows.

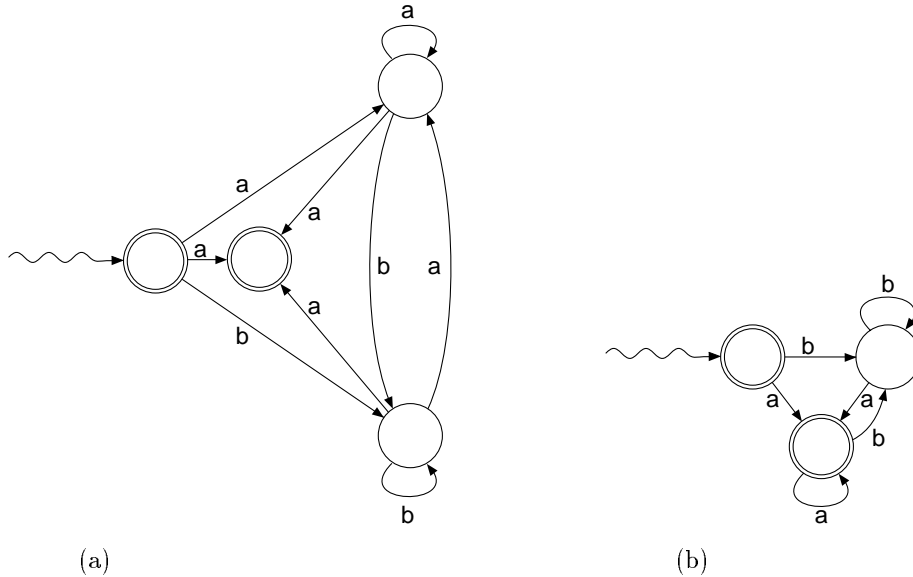
1.  $Q_E = sym(E') \dot{\cup} \{s\}$ ; that is, the states of  $G_E$  are the positions of  $E'$  together with a new start state  $s$ .
2.  $(s, a, x) \in \delta_E$  if  $x \in first(E')$  and  $x^\sharp = a$ , for some  $a \in \Sigma$ .
3.  $(x, a, y) \in \delta_E$  if  $y \in follow(E', x)$  and  $y^\sharp = a$ , for some  $x \in sym(E')$  and  $a \in \Sigma$ .
4.  $F_E = \begin{cases} last(E') \cup \{s\}, & \text{if } \epsilon \in L(E), \\ last(E'), & \text{otherwise.} \end{cases}$

Book et al. [3] established that  $L(G_E) = L(E)$ . In addition, the following simple observations emerge directly from the definition of the Glushkov machine. Let  $E$  be a regular expression; then:

1. The Glushkov machine  $G_E$  has no transitions that lead to the start state; that is, it is **nonreturning** [22].
2. Any two transitions that lead to the same state in  $G_E$  have identical input symbols.

Using Glushkov machines, we can give another characterization of 1-deterministic expressions that is a direct consequence of Lemma 3.

**Lemma 5.** *A regular expression  $E$  is 1-deterministic if and only if  $G_E$  is deterministic.*



**Fig. 4.** The Glushkov machines corresponding to (a)  $(a+b)^*a+\epsilon$  and (b)  $(b^*a)^*$ .

Fig. 4(a) demonstrates that  $(a|b)^*a|\epsilon$  is not a 1-deterministic expression. Nevertheless, the language denoted by  $(a|b)^*a|\epsilon$  is a 1-deterministic language, since it is also denoted by  $(b^*a)^*$ , which is a 1-deterministic expression; see Fig. 4(b).

The Glushkov machine  $G_E$  can be computed in time quadratic in the size of  $E$ , which is worst case optimal [4, 5, 12]. We can also construct the Glushkov machine  $G_E$  by induction on  $E$ . The inductive definition goes back to Mirkin [24] and Leiss [22]. Recently, Champarnaud has shown that both methods produce the same machine [11]; see also the taxonomy of Watson [32].

We can use the Glushkov machine to test for 1-determinism as follows. During the construction of the machine from an expression  $E$  we terminate the construction as soon as we identify two transitions from the same source state that have the same input symbol, but different target states. If we meet two such transitions the expression is not 1-deterministic; otherwise, it is 1-deterministic. We can implement this testing-by-construction algorithm to run in time linear in the size of  $E$ .

But what about the 1-determinism of content models? The major difference between regular expressions and content models is that content models have three additional operators: '?', '&', and '+'. They are defined as follows:  $E? \equiv E|\epsilon$ ,  $E+ \equiv EE^*$ , and  $E&F \equiv (EF|FE)$ . Clearly, we can remove the operators by using these equivalences to replace them with equivalent expressions; however, the removal of '&' and '+' may change the 1-determinism of the original expression (the removal of '?' does not change the 1-determinism of an expres-

sion). For example,  $(a?\&b)$  is 1-deterministic, but  $(a?b|ba?)$  is not 1-deterministic. Similarly,  $(a * |b)+$  is 1-deterministic, but  $(a * |b)(a * |b)*$  is not 1-deterministic. Brüggemann-Klein [6, 7] has demonstrated that the approach based on *first*, *last*, and *follow* sets can be modified to avoid the direct computation of the Glushkov machine. This observation is crucial since the corresponding Glushkov machine can have size exponential in the size of the expression (when ‘&’ occurs). For example, the minimal deterministic finite-state machine for the content model  $(a_1\&a_2\&\dots\&a_n)$  has size  $\Omega(2^n)$ . Similarly, using the preceding equivalence to remove ‘&’ from an expression can also lead to exponential blow up in the size of the expression. To summarize, we have the following result.

**Theorem 6.** *Given a regular expression or content model, we can decide whether it is 1-deterministic in time linear in its size.*

One important implication of this work is that it has been assumed that SGML DTDs are LL(1); however, this is only the case when the content models are 1-deterministic. Moreover, to construct an LL(1) parser from the DTD in this case means that we need to convert the content models into deterministic finite-state machines and they may have size exponential in the size of the content model.

Lastly, what if an expression or content model is not 1-deterministic? It turns out that the cycle structure of the Glushkov machine (called the **orbit property**) is preserved under minimization when it is deterministic [9, 8]. Thus, if the minimal deterministic machine for a regular expression does not satisfy the orbit property, then the corresponding language is not 1-deterministic. Using this result, it is not difficult to establish that the language of the regular expression  $(a|b) * (ac|bd)$  is not 1-deterministic [8] although it is LL(1) since every regular language is LL(1). For example, the following LL(1) grammar generates this language:

$$\begin{aligned} S &\rightarrow aC \mid bD \\ C &\rightarrow c \mid aC \mid bD \\ D &\rightarrow d \mid aC \mid bD. \end{aligned}$$

The condition can be modified to give a necessary and sufficient condition for a regular language to be 1-deterministic and, hence, give a worst-case exponential-time algorithm to construct a 1-deterministic expression from an expression for a 1-deterministic language.

## 4 Equivalence

Given two SGML DTDs a natural and fundamental question is whether they are equivalent in some sense [10, 21, 28]. The most basic equivalence is language equality: Two DTDs are equivalent if their languages are the same. For SGML DTDs, because document instances include the start and end tags of the DTD, the basic equivalence is a very strong equivalence: Two SGML DTDs are equivalent if and only if they define exactly the same sets of syntax trees. Moreover,

this property holds if and only if the two DTDs have exactly the same elements and the content models of identical elements are language equivalent. Since we can decide language equivalence for content models, we can decide SGML-DTD equivalence.

Although the decidability of SGML-DTD equivalence is of practical interest, SGML-DTD equivalence is too restrictive. For example, if we rename some of the elements in an SGML DTD, then the resulting DTD is not equivalent to the original DTD with respect to the preceding definition of equivalence. Thus, we wish to generalize the notion of equivalence such that it is still decidable, yet can capture such examples. **Structural equivalence** is such an equivalence. Two SGML DTDs are structurally equivalent if, when we replace all occurrences of start and end tags with `<>` and `</>`, respectively, in every document instance, the resulting languages are the same. We are, essentially, stripping the element labels from the syntax trees. A result of Thatcher [29] demonstrates, by way of tree machines, that structural equivalence is decidable for ECFGs. Cameron and Wood [10] reprove this result directly for ECFGs. They provide normalizations of two ECFGs such that the resulting grammars are structurally equivalent if and only if they are isomorphic. Although structural equivalence is decidable, the known algorithms take time doubly exponential in the total size of the two grammars. Two open problems are: Prove a nontrivial lower bound for the structural-equivalence problem for ECFGs and determine restricted versions of structural equivalence that are useful for SGML DTDs and investigate their time complexities.

## 5 Exceptions

SGML allows **exceptions** in content models that modify the corresponding contents. **Inclusion exceptions** allow named elements to appear anywhere within the content, whereas **exclusion exceptions** preclude them from the content. For example, we might want to allow notes to appear anywhere in the body of messages, except within notes themselves. This extension of the DTD of Fig. 1 could be defined by adding an inclusion exception to the EDT of `body`; namely,

```
<!ELEMENT body          (paragraph)* +(note)>
```

To prevent notes from appearing within notes we could then add the following ETD that has an exclusion exception for `note`:

```
<!ELEMENT note          (#PCDATA) -(note)>
```

A similar example is the specification of the legal positions for comments in the programs of a programming language. Typically, comments are defined precisely, but their placement is defined informally. The reason is a simple one, the element ‘comment’ would have to be added everywhere in the right-hand sides of the productions of the programming-language grammar. Such additions make the grammar unreadable and provide a justification for the use of exceptions.

Exclusion exceptions seem to be an especially useful concept, but their exact meaning is unclear from the Standard. We provide rigorous definitions for the meaning of both inclusion and exclusion exceptions and also discuss the transformation of grammars with exceptions to grammars without exceptions. This approach is based on the work of Kilpeläinen and Wood [21] who show that exceptions do not increase the expressive power of SGML DTDs.

Most existing SGML parsers deal with exceptions. The Amsterdam SGML parser [31] handles them in an *interpretive manner*. The names of excluded elements are kept in a stack, which is consulted whenever the parser encounters a new element. Inclusions are handled through the error routine. Whenever an input element that does not match the current content model is encountered, the parser enters its error mode. If the element is an allowed inclusion, the parser calls itself recursively with the included element as the root symbol of the parse.

We *compile* exceptions to produce a DTD that is “equivalent” to the original DTD but does not use any exceptions. In the worst case this transformation may increase the number of elements by a factor that is exponential in the number of the exceptions. An application that requires the elimination of exceptions from content models is the translation of DTDs into static database schemas. This method of integrating text documents into an object-oriented database has been suggested by Christofides *et al.* [13].

The SGML standard requires content models to be unambiguous (or 1-deterministic, see Section 3) and the methods of eliminating exceptions preserve the 1-determinism of the original content models. In this respect exception elimination extends the work of Brüggemann-Klein and Wood [9, 6, 8]. The Standard also gives vague restrictions on the applicability of exclusion exceptions. We propose a simple and rigorous definition of the applicability of exclusions, and present an optimal algorithm for testing applicability.

We first discuss exceptions in the simplified setting of ECFGs (see Section 2) before dealing with the SGML specific problems of how to preserve 1-determinism and how to check the applicability of exclusions.

An **ECFG with exceptions**  $G = (N, \Sigma, P, S)$  is similar to an ECFG, but now the production schemas in  $P$  have the form  $A \rightarrow E + I - X$ , where  $I$  and  $X$  are subsets of  $N$ . The intuitive idea is that a derivation of any string  $w$  from nonterminal  $A$  using the production schema  $A \rightarrow E + I - X$  must not involve any nonterminal in  $X$  but  $w$  may include, in any position, strings that are derivable from nonterminals in  $I$ . When a nonterminal is both included and excluded, its exclusion overrides its inclusion.

We first describe formally the effect of inclusions and exclusions on languages. Let  $L$  be a language over alphabet  $V$ , and let  $I, X \subseteq V$ . We define the **language  $L$  with inclusions  $I$**  as the set

$$L_{+I} = \{w_0 a_1 w_1 \cdots a_n w_n \mid a_1 \cdots a_n \in L, n \geq 0, \text{ and } w_i \in I^* \text{ for } i = 0, \dots, n\}.$$

Hence,  $L_{+I}$  consists of the strings in  $L$  interleaved with arbitrary strings over  $I$ . The **language  $L$  with exclusions  $X$** , denoted by  $L_{-X}$ , consists of the strings in  $L$  that do not contain any symbol in  $X$ . Notice that  $(L_{+I})_{-X} \subseteq (L_{-X})_{+I}$ , but the converse does not hold in general. We write  $L_{+I-X}$  for  $(L_{+I})_{-X}$ .

We describe formally the global effect of exceptions by attaching exceptions to nonterminals and by defining derivations from nonterminals with exceptions. We denote a nonterminal  $A$  with inclusions  $I$  and exclusions  $X$  by  $A_{+I-X}$ . When  $w$  is a string over  $\Sigma \cup N$ , we denote by  $w_{\pm(I,X)}$  the string obtained from  $w$  by replacing each nonterminal  $A$  by  $A_{+I-X}$ . Let  $\beta_1 A_{+I-X} \beta_2$  be a string over terminal symbols and nonterminals with exceptions. We say that the string  $\beta_1 \alpha' \beta_2$  can be derived from  $\beta_1 A_{+I-X} \beta_2$ , denoted by  $\beta_1 A_{+I-X} \beta_2 \Rightarrow \beta_1 \alpha' \beta_2$ , when the following two conditions are met:

1.  $A \rightarrow E + I_A - X_A$  is a production schema in  $P$ .
2.  $\alpha' = \alpha_{\pm(I \cup I_A, X \cup X_A)}$  for some string  $\alpha$  in  $L(E)_{+(I \cup I_A) - (X \cup X_A)}$ .

Finally, the **language  $L(G)$  described by an ECFG  $G$  with exceptions** is the set of terminal strings derivable from the sentence symbol with empty inclusions and exclusions. Formally,  $L(G) = \{w \in \Sigma^* \mid S_{+\emptyset-\emptyset} \Rightarrow^+ w\}$ .

Exceptions would seem to be a context-dependent feature: legal expansions of a nonterminal depend on the context in which the nonterminal appears. But exceptions do not extend the expressive power of ECFGs, as we show by presenting a transformation that produces an ECFG that is equivalent to an ECFG with exceptions. The transformation consists of propagating exceptions to production schemas and of modifying regular expressions to capture the effect of exceptions.

Let us first see how to modify regular expressions to capture the effect of exceptions. Let  $E$  be a regular expression over  $V = \Sigma \cup N$  and  $I = \{i_1, \dots, i_k\}$ . We can produce a regular expression  $E_{+I}$  from  $E$  such that  $L(E_{+I}) = L(E)_{+I}$  as follows. (We assume that either  $E$  does not contain  $\emptyset$  or  $E = \emptyset$ ; otherwise, we can transform  $E$  into an equivalent regular expression that satisfies this assumption by applying the rewrite rules  $F\emptyset \rightarrow \emptyset$ ,  $\emptyset F \rightarrow \emptyset$ ,  $F \cup \emptyset \rightarrow F$ ,  $\emptyset \cup F \rightarrow F$ , and  $\emptyset^* \rightarrow \epsilon$ .)  $E_{+I}$  can be obtained from  $E$  by replacing each occurrence of a symbol  $a \in V$  by  $(i_1 \cup i_2 \cup \dots \cup i_k)^* a (i_1 \cup i_2 \cup \dots \cup i_k)^*$ , and each occurrence of  $\epsilon$  by  $(i_1 \cup i_2 \cup \dots \cup i_k)^*$ . For a set of excluded elements  $X$  we obtain a regular expression  $E_{-X}$  such that  $L(E_{-X}) = L(E)_{-X}$  by replacing each occurrence of any symbol  $a \in X$  in  $E$  by  $\emptyset$ .

An algorithm for eliminating exceptions from an ECFG  $G$  with exceptions is given in Fig. 5. The algorithm produces an ECFG  $G'$  that is equivalent to  $G$ . The correctness of the algorithm is proved by Kilpeläinen and Wood [21]. The nonterminals of  $G'$  are of the form  $A_{+I-X}$ , where  $A \in N$  and  $I, X \subseteq N$ . A derivation step using a new production schema  $A_{+I-X} \rightarrow E$  in  $P'$  corresponds to a derivation step using an old production schema for nonterminal  $A$  under inclusions  $I$  and exclusions  $X$ .

The algorithm terminates since it generates at most  $2^{2^{|N|}}$  new nonterminals of the form  $A_{+I-X}$ . In the worst case the algorithm can exhibit this potentially exponential behavior. As an example consider the following ECFG with exceptions:

$$\begin{aligned} A &\rightarrow (A_1 \cup \dots \cup A_m) + \emptyset - \emptyset \\ A_1 &\rightarrow (a_1 \cup A) + \{A_2\} - \emptyset \end{aligned}$$



given string. We define the two sets:

$$first(L) = \{a \in V \mid au \in L \text{ for some } u \in V^*\}.$$

and

$$leftq(L, w) = \{u \in V^* \mid wu \in L\}, \text{ for every } w \in V^*.$$

Let  $I$  be a subset of  $V$ . We define the **SGML effect of inclusions I on language L** as the set

$$L_{\oplus I} = \{w_0 a_1 \cdots w_{n-1} a_n w_n \mid a_1 \cdots a_n \in L, n \geq 0, \\ w_i \in (I - first(leftq(L, a_1 \cdots a_i)))^*, i = 0, \dots, n\}.$$

For example, the language  $\{ab, ba\}_{\oplus \{a\}}$  consists of strings  $a^k b a^l$  and  $b a^k$  where  $k \geq 1$  and  $l \geq 0$ .

The basic idea of compiling the inclusion of symbols  $\{i_1, \dots, i_k\}$  in a content model  $E$  is to insert new subexpressions of the form  $(i_1 | \cdots | i_k)^*$  in  $E$ . Preserving the 1-determinism of the content model requires some extra care however. Consider through an example the complications caused by the  $\&$  operator. For example, the content model  $E = a?\&b$  is easily seen to be 1-deterministic. A content model that captures the inclusion of symbol  $a$  in  $E$  should give a string of  $as$  after  $b$ . A straightforward transformation would produce a content model of the forms  $F\&(ba^*)$  or  $(F\&b)a^*$ , with  $a \in first(L(F))$  and  $\epsilon \in L(F)$ . It easy to see that these content models are not 1-deterministic and that  $L(E)_{\oplus a} \neq L(F)$  since  $\epsilon \in L(F)$ .

Our strategy to handle such problematic subexpressions  $F\&G$  is first to replace them by an equivalent subexpression  $(FG|GF)$ . Notice that this may not suffice, since  $FG|GF$  can be ambiguous even if  $F\&G$  is 1-deterministic. For example, the content model  $(a?b|ba?)$  is ambiguous. To circumvent this problem we define a transformation of a content model  $E$  into a content model  $E^\circ$  that gives the same strings as  $E$  except for the empty string. Moreover, we can now apply the previously mentioned strategy to  $E^\circ$  to give a new content model  $E_{\oplus I}$  that has the desired properties. These properties can be summarized as follows:

1.  $E^\circ$  is 1-deterministic if and only if  $E$  is 1-deterministic.
2. The content model  $E_{\oplus I}$  is 1-deterministic if  $E$  is 1-deterministic.
3.  $L(E_{\oplus I}) = L(E^\circ)_{\oplus I} = L(E)_{\oplus I}$ .

Thus, for each 1-deterministic content model with inclusions we can construct an equivalent 1-deterministic content model without inclusions.

We now provide a precise definition of the meaning of exclusions, and present an efficient method of checking their validity and of modifying content models to capture the effect of exclusions.

Clause 11.2.5.2 of the SGML standard states that “exclusions modify the effect of model groups to which they apply by precluding options that would otherwise have been available.” The exact meaning of “precluding options” is not clear from the Standard. Our first task is therefore to formalize the intuitive

notion of exclusion rigorously. As a motivating example consider the exclusion of the symbol  $b$  from the content model  $E = a(b|c)c$  that defines the language  $L(E) = \{abc, acc\}$ . Element  $b$  is clearly an alternative to the first occurrence of  $c$  and we can realize its exclusion by modifying the content model to give  $E' = acc$ . Now, consider excluding  $b$  from the content model  $F = a(bc|cc)$ . The situation is less clear since  $b$  appears in a catenation subexpression. Observe that both  $E$  and  $F$  define the same language.

Let  $L \subseteq V^*$  and  $X \subseteq V$ . Motivated by the preceding example we define the effect of excluding  $X$  from  $L$ , denoted by  $L_{-X}$ , as the set of the strings in  $L$  that do not contain any symbol of  $X$ . As an example, the effect of excluding  $\{b\}$  from the languages of the preceding content models  $E$  and  $F$  is  $L(E)_{-\{b\}} = L(F)_{-\{b\}} = \{acc\}$ . Notice that an exclusion always gives a subset of the original language.

We now describe how we can compute, for a given content model  $E$  and a given set of symbols  $X$ , a content model  $E_{\ominus X}$  such that  $L(E_{\ominus X}) = L(E)_{-X}$ . The modified content model  $E_{\ominus X}$  is 1-deterministic if the original content model  $E$  is 1-deterministic. The computation of  $E_{\ominus X}$  takes time linear in the size of  $E$ .

For the transformation we extend content models with symbols  $\epsilon$  and  $\emptyset$ , which are constituents of standard regular expressions. Their addition extends the definition of the language  $L(E)$  represented by a content model  $E$  with the following two cases:

$$L(\epsilon) = \epsilon. \quad L(\emptyset) = \{\emptyset\}.$$

For a content model  $E$ , the extended content model  $E_{\ominus X}$  is defined inductively as follows:

$$\begin{aligned} a_{\ominus X} &= \begin{cases} \emptyset & \text{if } a \in X, \\ a & \text{otherwise.} \end{cases} \\ (FG)_{\ominus X} &= \begin{cases} \emptyset & \text{if } F_{\ominus X} = \epsilon \text{ or } G_{\ominus X} = \epsilon, \\ F_{\ominus X} & \text{if } G_{\ominus X} = \epsilon, \\ G_{\ominus X} & \text{if } F_{\ominus X} = \epsilon, \\ F_{\ominus X}G_{\ominus X} & \text{otherwise.} \end{cases} \\ (F|G)_{\ominus X} &= \begin{cases} F_{\ominus X} & \text{if } G_{\ominus X} = \emptyset, \\ G_{\ominus X} & \text{if } F_{\ominus X} = \emptyset, \\ \epsilon & \text{if } F_{\ominus X} = \epsilon \text{ and } G_{\ominus X} = \epsilon, \\ F_{\ominus X}? & \text{if } F_{\ominus X} \neq \epsilon \text{ and } G_{\ominus X} = \epsilon, \\ G_{\ominus X}? & \text{if } F_{\ominus X} = \epsilon \text{ and } G_{\ominus X} \neq \epsilon, \\ F_{\ominus X}|G_{\ominus X} & \text{otherwise.} \end{cases} \\ (F\&G)_{\ominus X} &= \begin{cases} \emptyset & \text{if } F_{\ominus X} = \emptyset \text{ or } G_{\ominus X} = \emptyset, \\ F_{\ominus X} & \text{if } G_{\ominus X} = \epsilon, \\ G_{\ominus X} & \text{if } F_{\ominus X} = \epsilon, \\ F_{\ominus X}\&G_{\ominus X} & \text{otherwise.} \end{cases} \\ (F?)_{\ominus X} &= \begin{cases} \epsilon & \text{if } F_{\ominus X} = \emptyset \text{ or } F_{\ominus X} = \epsilon, \\ F_{\ominus X}? & \text{otherwise.} \end{cases} \end{aligned}$$

$$(F^*)_{\ominus X} = \begin{cases} \epsilon & \text{if } F_{\ominus X} = \emptyset \text{ or } F_{\ominus X} = \epsilon, \\ (F_{\ominus X})^* & \text{otherwise.} \end{cases}$$

$$(F^+)_{\ominus X} = \begin{cases} \emptyset & \text{if } F_{\ominus X} = \emptyset, \\ \epsilon & \text{if } F_{\ominus X} = \epsilon, \\ (F_{\ominus X})^+ & \text{otherwise.} \end{cases}$$

The following properties justify the definition of  $E_{\ominus X}$ :

1.  $E_{\ominus X}$  is a content model (that is,  $E_{\ominus X}$  does not contain  $\emptyset$  or  $\epsilon$ ) if and only if  $E_{\ominus X} \notin \{\emptyset, \epsilon\}$ .
2.  $L(E_{\ominus X}) = \emptyset$  if and only if  $E_{\ominus X} = \emptyset$ .
3.  $L(E_{\ominus X}) = \{\epsilon\}$  if and only if  $E_{\ominus X} = \epsilon$ .
4.  $L(E_{\ominus X}) = L(E)_{-X}$ .
5. If  $E$  is 1-deterministic, then  $E_{\ominus X}$  is 1-deterministic.

As a restriction to the applicability of exclusions the Standard states that “an exclusion cannot affect a specification in a model group that indicates that an element is required.” The Standard does not specify how a model group (a subexpression of a content model) indicates that an element is required. A reasonable requirement for the applicability of excluding  $X$  from a content model  $E$  would be  $L(E)_{-X} \not\subseteq \{\epsilon\}$ . Note that an ordinary content model cannot denote a language that is either  $\emptyset$  or  $\{\epsilon\}$ . Intuitively,  $E_{\ominus X} = \emptyset$  or  $E_{\ominus X} = \epsilon$  means that excluding  $X$  from  $E$  precludes all elements from the content of  $E$ . On the other hand,  $E_{\ominus X} \notin \{\emptyset, \epsilon\}$  means that  $X$  precludes only elements that are optional in  $L(E)$ . Thus computing  $E_{\ominus X}$  is a reasonable and efficient test for the applicability of exclusions  $X$  to a content model  $E$ .

## 6 Omitted Tag Minimization

SGML allows us to specify that a tag is optional. It is crucial, however, that when a tag is omitted its position can be uniquely deduced from the context. For example, with the DTD

```
<!ELEMENT S          O -   (a, a?, S?)>
```

the start tag  $\langle S \rangle$  is optional. Observe that we have added two fields after the element name, the first one refers to the start tag and the second to the end tag. There are two possible values: ‘-’ meaning cannot be omitted and ‘O’ meaning may be omitted. Then with the string

```
<S>aaa</S></S> ,
```

we do not know whether we should insert the missing  $\langle S \rangle$  after the first ‘a’ or after the second ‘a’; that is we do not know whether the fully tagged string should be

```
<S>a<S>aa</S></S>
```

or should be

```
<S>aa<S>a</S></S>.
```

As with many other features of the Standard, omitted tag minimization is an unfortunate feature. It is clear that it is useful to allow marked-up texts that already exist to have omitted tags; for example  $\text{\LaTeX}$  has the start tag `\section` but it has no corresponding tag for the end of a section. Thus, there is a good reason for allowing tag minimization, although the implementation of the tag omission leaves much to be desired. Goldfarb's annotations [17] explain the strict rules under which omitted tags can actually be omitted. It is unclear from his discussion whether a tag that is marked for possible omission can be omitted in some documents but not in others. Can we decide whether a tag can always be omitted, sometimes be omitted and in what circumstances, or never be omitted? Currently we are investigating these and related problems.

As a final example of omitted tag minimization, we have a modified version of the e-mail DTD given in Fig. 6. Observe that we may omit the start tag of `head` and the end tags of `from`, `to`, and `body`. In all document instances we can always deduce the positions of the omitted tags.

```
<!DOCTYPE message [
<!ELEMENT message      - -   (head, body)>
<!ELEMENT head         0 -   (from & to & subject)>
<!ELEMENT from         - 0   (person)>
<!ELEMENT to           - 0   (person)+>
<!ELEMENT person       - -   (alias | (forename?, surname))>
<!ELEMENT body         - 0   (paragraph)*>
<!ELEMENT subject, alias, forename, surname, paragraph
- -   (#PCDATA)> ]>
```

**Fig. 6.** The example DTD with omitted tag minimization.

## 7 Documents, Markup, and Structure

A document is inherently structured at a fine-grained level since it can be defined as a structured collection of marks on a medium (as against a random collection of marks). This definition of a document indicates a truism: structure is in the mind of the author and the reader. The fine-grained concept of structure carries over to coarser-grained levels. For example, this paper has a typical report structure: a footnoted title, author, author's address, abstract, a sequence of sections, and a collection of references. This structure is logical (or descriptive) in that it is independent of content and, hence, of the content's meaning. The logical structure is identified by the author and it can be indicated with markup

in general and SGML markup in particular. It is not, however, the only logical structure of this paper. We have a page structure that is defined by the typeset layout of the paper; we have a version structure that indicates the different versions (or revisions) of the paper from its inception to its present form; we have a referential structure that indicates the cross-referencing. There are no footnotes in the body of the paper, but in legal reviews and papers in the humanities they are usually in abundance. *A document can have many logical structures and we cannot always consider only one of them.* The version and report structures are critical structures that need to be maintained in many situations. For example, consider the evolution of a user manual for some system. The user manual will have a report structure and one or two version structures. It is important, in this context, that users be informed of documentation changes for the same version of the system as well as of documentation changes when a new version of the system is distributed.

When faced with a mechanism such as SGML to define markup and markup systems we are invariably led to question its foundations, its *raison d'être* [27, 28]. Is this method, however attractive and however much used, the most appropriate? Are there similar, yet more widely embracing methods? Is SGML, or something very similar, an inevitable conclusion of the Western history of documents; for example, of manuscripts, books, and electronic documents? Is SGML an irrelevance, in that it is addressing the wrong problem?

We are discussing only textual documents here, although they may be small or large, simple or complex. Punctuation, sentences, and paragraphs are methods of markup (they were introduced in the eighth century) as are the reserved words of Pascal and the use of indentation in displayed Pascal programs. At a completely different level, the start and stop codons that delimit the aminoacid sequences that are genes in chromosomes are also markup (a DNA helix can be viewed as the medium on which the genes are the message). In each of these examples, markup is **embedded** in the text. Although markup need not be embedded (we discuss this issue later in this section), it is often most convenient to have it so and, historically, markup **is** embedded. Second, the markup is **distinguishable** from the non-markup. For example, punctuation is distinguishable as are start and stop codons and Pascal reserved words. Distinguishability is not necessarily easy to compute; thus, we can only separate Pascal reserved words by parsing a Pascal program. Third, **positioning** is also a feature of these kinds of markup in that none of the markup can be moved without changing a document's structure and meaning. To find a period, which is not associated with an abbreviation or with a decimal number, in the middle of a sentence would produce two sentences were only one was intended.

SGML markup is also embedded, distinguishable, and positioned. In addition, it is **separable**, **nested**, **context-free**, and **contiguous**. Separability is stronger than distinguishability in that the markup begins and ends with unique characters. It is nested in that it does not allow markup of the form

`<A>...<B>...</A>...</B>`.

It is context-free in that its legal nestings and adjacencies are specified with

a context-free grammar. Lastly, it is **contiguous** in that it is not interrupted and then resumed. Embedding, distinguishability, and positioning are required properties for the markup of most documents, whereas separability, nesting, context-freeness, and contiguity are not obviously necessary properties. Let us reexamine the latter four properties of SGML markup.

The use of the delimiters ‘<’ and ‘>’ to begin and end tags is not mandated by the SGML standard; we may choose different symbols for them. But, in all cases, we have separability—it is an inherent property of SGML. One reason for requiring this strong version of distinguishability is purely algorithmic—the ease of recognizing tags. Certainly, in a programming language such as ALGOL 60, the markup strings such as `while` and `do` can be detected as markup only when a full parse of a program is carried out because a programmer, however unwisely, can use them as variable names. The separation of markup in SGML documents is, in general, more straightforward; it requires only the power of a finite-state transducer. Therefore, separation (or tokenization) is fast.

Nesting and context-freeness are interrelated. SGML DTDs generate nested structures and every context-free language can be viewed as the image of nested structures (the Chomsky–Schützenberger Theorem). But not all nested structures can be obtained in this way. For example, consider a set of documents that exhibit structures of the forms

$$\langle A \rangle^i \dots \langle /A \rangle^i \dots \langle B \rangle^i \dots \langle /B \rangle^i,$$

for  $i \geq 1$ , such that for each  $i \geq 1$  there is a document with that structure and, in addition, a substructure of the form  $\langle A \rangle^i$  denotes a string of  $i$  copies of  $\langle A \rangle$  interleaved with text. Then, no SGML DTD can specify such markup. Indeed, even with documents of the form

$$\langle A \rangle^i \dots \langle /A \rangle^i,$$

where  $i = 2^k$ , for every  $k \geq 0$ , is not specifiable with any SGML DTD. The documents in each case are SGML-like, but they are not SGML documents or, to be more precise, they do not form a class of SGML documents. We can always give a DTD for finitely many documents of these forms, but not for **all** of them.

Next, context-freeness while not the most powerful grammatical mechanism has become the standard approach in the programming-language community. The reason is easy to understand; there are fast parsing methods available for context-free grammars and even faster methods for deterministic context-free grammars—SGML DTDs are LL(1).

To summarize, while nesting is an important property it can be too restrictive. The only argument for nesting as the sole bracketed structure in SGML is that, when there are no constraints among the different tags, the structures can be specified by a context-free grammar.

Finally, contiguity is too restrictive. It implies that a string between matching tags  $\langle A \rangle$  and  $\langle /A \rangle$ , belongs to  $A$  and only to  $A$ . But take the example of genes once more. Surprisingly, the aminoacid sequence that makes up a gene does not necessarily (indeed rarely) occur as a string; there are usually gaps between the

pieces of a gene. A similar example is seen in a conversation. When two or more people are conversing their utterances do not always occur sequentially and contiguously, an utterance begins, pauses while another person's utterance begins or continues, the original utterance continues, pauses again, and so on, until it ends. The parts of utterances are interleaved. When expressing this interaction of concurrent processes in a play, the utterances are also noncontiguous. Thus, we obtain structures such as

$$\langle A \rangle \dots \langle :A \rangle \langle B \rangle \dots \langle :B \rangle \langle /:A \rangle \dots \langle :A \rangle \dots \langle /A \rangle,$$

where I have used  $\langle :A \rangle$  as a start-pause tag and  $\langle /:A \rangle$  as an end-pause tag. In general the structures that we obtain in this way cannot be described by SGML DTDs.

Thus, the answer to the question, "Is this method, however attractive and however much used, the most appropriate?" is a qualified "Yes."

Let us return to the issue of embedding of markup. It is clear that having markup that is not only separable but also is separated from the text can be very useful. It is easier to define multiple markups of the same document; for example, a version markup that tracks the revision structure of a document, a structural markup that captures one of the logical structures of a document, and an annotational markup that provides a scholar's commentary on the text. Having said this, the issues of markup do not go away. Often markup is an integral part of the text; for example, Pascal's `while-do` construct and the punctuation in this sentence. We do not want to and often cannot separate the markup from the text.

Lastly, is SGML wrong headed? Is it addressing the wrong problem? In traditional relational databases, data representation is unimportant, rather the semantics of the query language is central. In other words, the same data in two different databases may be represented very differently, but the same query should give the same answer. The data representation should not interfere with the query mechanism. It is possible that we could approach electronic documents in the same way. For example, we could specify a text algebra that abstracts the notion and properties of text, including textual access and modification. Then, such an algebra could serve as a standard interface to higher-level applications. SGML does not specify such an algebra since it does not provide the algebraic operations that are needed. Raymond *et al.* [28] discuss the meta-semantic issues of SGML in more detail and at a higher level.

## 8 Acknowledgements

The research that I have surveyed is the result of a number of enjoyable and fruitful collaborations over the years with a small group of individuals: Anne Brüggemann-Klein (who introduced me to both typesetting and SGML as research areas and has been a continual mentor), Helen Cameron (who lived through my supervision of her PhD research), Pekka Kilpeläinen (who brought me his carefulness and persistence), Darrell R. Raymond (who broadened and

continues to broaden my horizons by his readings and reflections), and Frank W. Tompa (who always has had the patience to share his insights into text and his accumulated wisdom of databases). They are responsible for the results and arguments on which this survey is based; however, I alone am responsible for any errors, misdirections, and misinformation.

## References

1. A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Series in Computer Science. Addison-Wesley Publishing Company, Reading, MA, 1986.
2. G. Berry and R. Sethi. From regular expressions to deterministic automata. *Theoretical Computer Science*, 48:117–126, 1986.
3. R. Book, S. Even, S. Greibach, and G. Ott. Ambiguity in graphs and expressions. *IEEE Transactions on Computers*, C-20(2):149–153, February 1971.
4. A. Brüggemann-Klein. Regular expressions into finite automata. In I. Simon, editor, *Latin '92*, pages 87–98, Berlin, 1992. Springer-Verlag. Lecture Notes in Computer Science 583.
5. A. Brüggemann-Klein. Regular expressions into finite automata. *Theoretical Computer Science*, 120:197–213, 1993.
6. A. Brüggemann-Klein. Unambiguity of extended regular expressions in SGML document grammars. In Th. Lengauer, editor, *Algorithms—ESA '93*, pages 73–84, Berlin, 1993. Springer-Verlag. Lecture Notes in Computer Science 726.
7. A. Brüggemann-Klein. Compiler-construction tools and techniques for SGML parsers: Difficulties and solutions. To appear in *Electronic Publishing—Origination, Dissemination and Design*, 1995.
8. A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. To appear in *Information and Computation*, 1995.
9. A. Brüggemann-Klein and D. Wood. The validation of SGML content models. To appear in *Mathematical and Computer Modelling*, 1995.
10. H. Cameron and D. Wood. Structural equivalence of extended context-free and EOL grammars. Submitted for publication, 1995.
11. J.-M. Champarnaud. From a regular expression to an automaton. Unpublished Manuscript, 1992.
12. C.-H. Chen and R. Paige. New theoretical and computational results for regular languages. Technical report 587, Courant Institute, New York University, 1992. *Proceedings of the Third Symposium on Combinatorial Pattern Matching*.
13. V. Christofides, S. Christofides, S. Chuet, and M. Scholl. From structured documents to novel query facilities. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 313–324, 1994. SIGMOD Record, 23(2).
14. S. J. DeRose and D. G. Durand. *Making Hypermedia Work: A User's Guide to HyTime*. Kluwer Academic, Boston, 1994.
15. V. M. Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16:1–53, 1961.
16. C. F. Goldfarb. A generalized approach to document markup. *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, pages 68–73, June 1981. SIGPLAN Notices of the ACM.

17. C. F. Goldfarb. *The SGML Handbook*. Clarendon Press, Oxford, 1990.
18. ISO 8879: Information processing—Text and office systems—Standard Generalized Markup Language (SGML), October 1986. International Organization for Standardization.
19. ISO/IEC CD 10744: Information Technology—Hypermedia/Time-based structuring language (HyTime), 1991. International Organization for Standardization.
20. ISO/DIS 10179.2: Information processing—Text and office systems—Document style semantics and specification language (DSSSL), 1994. International Organization for Standardization.
21. P. Kilpeläinen and D. Wood. Exceptions in SGML document grammars. Submitted for publication, 1995.
22. E. Leiss. The complexity of restricted regular expressions and the synthesis problem of finite automata. *Journal of Computer and System Sciences*, 23(3):348–354, December 1981.
23. R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IRE Transactions on Electronic Computers*, EC-9(1):39–47, March 1960.
24. B. G. Mirkin. An algorithm for constructing a base in a language of regular expressions. *Engineering Cybernetics*, 5:110–116, 1966.
25. J. Nievergelt, G. Coray, Jean-Daniel Nicoud, and Alan C. Shaw. *Document Preparation Systems*. North Holland, Amsterdam, 1982.
26. J.-E. Pin. Local languages and the Berry-Sethi algorithm. Unpublished Manuscript, 1992.
27. D. R. Raymond, F. W. Tompa, and D. Wood. Markup reconsidered. *Principles of Document Processing*, 1992.
28. D. R. Raymond, F. W. Tompa, and D. Wood. From data representation to data model: Meta-semantic issues in the evolution of SGML. *Computer Standards and Interfaces*, to appear, July, 1995.
29. J.W. Thatcher. Characterizing derivation trees of a context-free grammar through a generalization of finite-automata theory. *Journal of Computer and System Sciences*, 1:317–322, 1967.
30. K. Thompson. Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.
31. J. Warmer and S. van Egmond. The implementation of the Amsterdam SGML parser. *Electronic Publishing—Origination, Dissemination and Design*, 2:65–90, 1989.
32. B. W. Watson. A taxonomy of finite automata construction and minimization algorithms. Manuscript, 1993.
33. D. Wood. *Theory of Computation*. John Wiley & Sons, New York, NY, 1987.