SGML and XML Document Grammars and Exceptions*

Pekka Kilpeläinen† Derick Wood‡

January 25, 1999

Abstract

The Standard Generalized Markup Language (SGML) and the Extensible Markup Language (XML) allow users to define document type definitions (DTDs), which are essentially extended context-free grammars expressed in a notation that is similar to extended Backus–Naur form. The right-hand side of a production, called a content model, is both an extended and a restricted regular expression. The semantics of content models for SGML DTDs can be modified by exceptions (XML DTDs do not allow exceptions). Inclusion exceptions allow named elements to appear anywhere within the content of a content model, and exclusion exceptions preclude named elements from appearing in the content of a content model.

We give precise definitions of the semantics of exceptions, and prove that they do not increase the expressive power of SGML DTDs when we restrict DTDs according to accepted practice. We prove the following results:

*The research of the first author was supported by the Academy of Finland and the research of the second author was supported by grants from the Natural Sciences and Engineering Research Council of Canada, from the Information Technology Research Centre of Ontario and from the Research Grants Council of Hong Kong. A preliminary version of this paper was presented at the Third Workshop on Principles of Document Processing (PODP '96) [15].

†Department of Computer Science, University of Helsinki, Helsinki, Finland. E-mail: kilpelai@cs.helsinki.fi.

‡Department of Computer Science, Hong Kong University of Science & Technology, Clear Water Bay, Kowloon, Hong Kong. E-mail: dwood@cs.ust.hk.

- Exceptions do not increase the expressive power of extended context-free grammars
- For each DTD with exceptions, we can obtain a structurally equivalent extended context-free grammar
- For each DTD with exceptions, we can construct a structurally equivalent DTD when we restrict the DTD to adhere to accepted practice
- Exceptions are a powerful shorthand notation—eliminating them may cause exponential growth in the size of an extended context-free grammar or DTD

1 Introduction

The Standard Generalized Markup Language (SGML) [12, 13] promotes the interchangeability and application-independent management of electronic documents by providing a syntactic metalanguage for the definition of textual markup systems. The Extensible Markup Language (XML) [2] is, essentially, a simplified and more restrictive version of SGML. The goal of XML is to allow SGML documents to be served, received and processed on the Web. It is the proposed syntactic metalanguage for the specification of document grammars for W3 documents.

Both SGML and XML allow users to define document type definitions (DTDs), which are essentially extended context-free grammars expressed in a notation that is similar to extended Backus–Naur form. The right-hand side of a production, called a **content model,** is both an extended and a restricted regular expression. The semantics of content models for SGML document grammars can be modified by exceptions (XML document grammars do not allow exceptions). Inclusion exceptions allow named elements to appear anywhere within the content of a content model, and exclusion exceptions preclude named elements from appearing in the content of a content model. In terms of (extended) context-free grammars, inclusion exceptions for a nonterminal A allow some specified nonterminals to appear anywhere in strings derivable from A whereas exclusion exceptions for A preclude some specified nonterminals from appearing in strings derivable from A. For example, comments can appear almost anywhere in most programming languages so they are usually not defined by a programming language's grammar. The reason is that the syntactic structure of the language would be obscured by the many appearances of a nonterminal for comments. Using an inclusion exception for comments is a simple solution that does not obscure the syntactic structure of the language.

The intent of this paper is to rigorously define the effect of exceptions on SGML DTDs and also to demonstrate that, for all practical purposes, they can be removed to produce structurally equivalent DTDs. We model SGML DTDs with extended context-free grammars with exceptions and prove that, in this case, we can always construct a structurally equivalent extended context-free grammar without exceptions.

The paper is organized as follows. In Section 2, we give a brief introduction to SGML (and XML) and exceptions. In Section 3, we introduce extended context-free grammars as a formal model of SGML DTDs and present the basic method of eliminating exceptions from extended context-free grammars by propagating them to the production schemas affected by them. In Section 4, we define content models and their languages. In Section 5, we show how we can modify content models to capture the local effect of inclusion exceptions. The modifications preserve the unambiguity of content models. In Section 6, we give a similar modification for the local effect of exclusion exceptions; we also propose a simple test for the applicability of exclusions. In Section 7, we explain how we can remove exceptions from SGML DTDs and show that the transformed DTDs are structurally equivalent to the original DTDs. On the other hand, the new DTDs contain new nonterminals; therefore, they are not directly applicable as SGML DTDs for the original document instances. We discuss in what circumstances our results can be applied to the design of DTDs and to the manipulation of SGML documents.

Lastly, in Section 8, we mention two unsolved problems, both of which are subjects of our current research.

2 A brief SGML primer

To simplify the presentation we explain and define notions only for SGML; most of the time the notions are identical in XML.

An SGML document consists of an SGML prolog and a marked-up document instance. The prolog contains a **document type definition (DTD)** that is an extended context-free grammar in which the right-hand sides of productions are both extended and restricted regular expressions. In this regard, DTDs are similar to grammars in extended Backus–Naur form. An example of a simple SGML DTD is given in Fig. 1; HTML is an example of a more complex SGML DTD.

```
<!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)>
```

Figure 1: An example SGML DTD that defines simple messages.

The productions of a DTD are called **element type definitions.** The right-hand sides of element type definitions are extended and restricted regular expressions called **content models.** The DTD in Fig. 1 defines a document type for messages, which consist of a `head` followed by a `body`. The element `head` consists of subelements `from`, `to`, and `subject` that can appear in any order. The element `from` is defined to be a `person` that can be denoted either by an `alias` or by an optional `forename` followed by a `surname`. The element `to` consists of a nonempty list of `person`s. The `body` of a `message` consists of a (possibly empty) sequence of `paragraph`s. Finally, the last element definition specifies that elements `subject`, `alias`, `forename`, `surname`, and `paragraph` are unstructured strings, denoted by the keyword #PCDATA.

The structural elements of a document instance are made visible by enclosing them in matching pairs of **start tags** and **end tags.** A possible instance of the message DTD in Fig. 1 is shown in Fig. 2. The DTD of Fig. 1 does not allow any begin or end tags to be omitted. The SGML Standard does allow, however, for DTDs to specify tag omissions. Since XML does not have this feature, we treat SGML DTDs in the same way. Another reason to make this assumption is that the complex rules of SGML that govern tag omission are there to ensure that an SGML parser can infer the omitted tags in valid documents. Therefore an additional reason for ignoring omitted tags is that we are not considering the parsing problem; that is, we assume that we have an SGML parser.

```
<!doctype message system "message.dtd">
<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>
```

Figure 2: A document instance of the SGML DTD of Fig. 1.

The semantics of content models can be modified by what the Standard calls **exceptions. Inclusion exceptions** allow named elements to appear anywhere within the content of a content model, and **exclusion exceptions** preclude named elements from the content of a content model. For example, with the DTD of Fig. 1, we could modify the definition of `body` to allow a new element `note`, defined as follows,

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

to appear anywhere in the bodies of messages. We accomplish this task by redefining `body` to be

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

The added inclusion exception `+(note)` allows notes to appear within notes. To prevent such recursive appearances of `note` we modify the definition of element type `note` by adding an exclusion exception to it:

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

Exclusion exceptions seem to be a useful concept, but their exact meaning is unclear from the Standard [13] and from Goldfarb's annotation of the Standard [12]. We first give, in Section 3, algorithms for transforming extended context-free grammars with exceptions into extended context-free grammars without exceptions. In Sections 5 and 6, we give rigorous definitions for the meaning of exceptions. The correctness proofs of these methods imply that exceptions are not necessary for the expressiveness of SGML DTDs that satisfy some technical restrictions.

3 Extended context-free grammars with exceptions

We first introduce extended context-free grammars as a model of SGML DTDs and discuss exceptions in this simplified setting. Then, in Sections 4 through 7, we consider the SGML-specific problems of unambiguity preservation and exclusion applicability.

Extended context-free grammars are context-free grammars in which the right-hand sides of productions are regular expressions. Let V be an alphabet. Then, we define a regular expression over V inductively as follows:

$\emptyset$	is a regular expression,
λ	is a regular expression,
$a \in V$	is a regular expression,
$F \cup G$,	where F and G are regular expressions over V, is a regular expression,
FG,	where F and G are regular expressions over V, is a regular expression,
F^*,	where F is a regular expression over V, is a regular expression,
(F),	where F is a regular expression over V, is a regular expression.

The language $L(E)$ described by a regular expression E over V is defined inductively as follows:

$$
\begin{aligned}
L(\emptyset) &= \emptyset, \\
L(\lambda) &= \{\lambda\}, \\
L(a) &= \{a\}, \text{ for } a \in V, \\
L(F \cup G) &= L(F) \cup L(G), \\
L(FG) &= \{vw \mid v \in L(F), w \in L(G)\}, \\
L(F^*) &= L(F)^*, \\
L((F)) &= L(F).
\end{aligned}
$$

The symbol λ denotes the null string and L^*, where L is a language, consists of all strings that are obtained as the catenation of zero or more strings from L. We denote by $sym(E)$ the set of symbols of V that appear in a regular expression E.

An **extended context-free grammar** G is specified by a tuple (N, Σ, P, S), where N and Σ are disjoint finite alphabets of **nonterminal symbols** and **terminal symbols,** respectively, P is a finite set of **production schemas,** and the nonterminal S is the **sentence symbol.** Each production schema has the form $A \to E$, where A is a nonterminal and E is a regular expression over $V = N \cup \Sigma$. When $\beta = \beta_1 A \beta_2 \in V^*$, $A \to E \in P$, and $\alpha \in L(E)$, the string $\beta_1 \alpha \beta_2$ can be derived from the string β and we denote this fact by writing $\beta \Rightarrow \beta_1 \alpha \beta_2$. The **language L(G) of an extended context-free grammar 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 correspond to an infinite number of ordinary context-free productions, it is known that extended and ordinary context-free grammars allow us to describe exactly the same languages; for example, see the text of Wood [19].

An **extended context free grammar G with exceptions** is specified by a tuple (N, Σ, P, S) and is similar to an extended context-free grammar except that the production schemas in P have the form $A \to E + I - X$, where A is in N, E is a regular expression over $V = N \cup \Sigma$, and I and X are subsets of N. The intuitive idea is that a derivation of any string w from the nonterminal A using the production schema $A \to E + I - X$ must not involve any nonterminal in X, yet w may contain, in any position, strings that are derivable from nonterminals in I. When a nonterminal is both included and excluded, its exclusion overrides its inclusion.

Existing SGML parsers have to deal with exceptions. For example, the Amsterdam SGML parser [18] 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 an error routine. Whenever an input element is encountered that does not match the current content model, the parser enters its error mode. If the element is an allowed inclusion exception, the parser calls itself recursively with the generic identifier of the included element as the root symbol of the parse.

We develop methods to *compile* exceptions; that is, we provide methods to produce a grammar that is structurally equivalent to the original one yet does not use any exceptions. In the worst case, this transformation may increase the number of productions by a factor which is exponential in the number of the exceptions.

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

language L with inclusions I as the language

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

Thus, L_{+I} consists of the strings in L with arbitrary strings from I^* inserted into them. The **language L with exclusions X** is defined as the language L_{-X} that 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. In the sequel we will write L_{+I-X} for $(L_{+I})_{-X}$.

We formally describe 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 of a regular expression over V, we denote by $w_{\pm(I,X)}$ the string obtained from w by replacing every appearance of a nonterminal A in w with A_{+I-X}. Let $\beta = \beta_1 A_{+I-X} \beta_2$ be a string over terminal symbols and nonterminal symbols with exceptions. We say that the string $\beta_1 \alpha' \beta_2$ can be derived from β, denoted by $\beta \Rightarrow \beta_1 \alpha' \beta_2$, when the following two conditions hold:

1. $A \to E + I_A - X_A$ is a production schema in P.

2. $\alpha' = \alpha_{\pm(I \cup I_A, X \cup X_A)}$ for some string α in $L(E)_{+(I \cup I_A)-(X \cup X_A)}$.

Observe that the second condition reflects the idea that exceptions are propagated and cumulated by derivations. Finally, the **language L(G) of an extended context-free grammar G with exceptions** consists of the 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 seem to be a context-dependent feature: *Legal expansions of a nonterminal depend on the context in which the nonterminal appears.* We show, however, that exceptions do not extend the descriptive power of extended context-free grammars by giving a transformation that produces an extended context-free grammar that is structurally equivalent to an extended context-free grammar with exceptions. The transformation propagates exceptions to production schemas and modifies their associated regular expressions to capture the effect of exceptions.

We now demonstrate how to modify regular expressions to capture the effect of exceptions. Let E be a regular expression over $V = \Sigma \cup N$ and let $I = \{i_1, \ldots, i_k\}$ be a set of inclusion exceptions. First, observe that we can remove the $\emptyset$ symbol from the regular expression E and maintain equivalence, if the language of the expression is not $\emptyset$. We do so by replacing the occurrences of $\emptyset$ using the following replacement rules until either it is identical to $\emptyset$ or there are no occurrences of $\emptyset$: $F\emptyset \to \emptyset$, $\emptyset F \to \emptyset$, $F \cup \emptyset \to F$, $\emptyset \cup F \to F$, and $\emptyset^* \to \lambda$. We now assume that either $E = \emptyset$ or E does not contain $\emptyset$. We modify E to obtain a regular expression E_{+I} such that $L(E_{+I}) = L(E)_{+I}$. We obtain E_{+I} from E by replacing each occurrence of a symbol $a \in V$ in E with

$$(i_1 \cup i_2 \cup \cdots \cup i_k)^* a (i_1 \cup i_2 \cup \cdots \cup i_k)^*$$

and each occurrence of λ with

$$(i_1 \cup i_2 \cup \cdots \cup i_k)^*.$$

For a set X of excluded elements, we obtain a regular expression E_{-X} such that $L(E_{-X}) = L(E)_{-X}$ by replacing each occurrence of a symbol $a \in X$ in E with $\emptyset$.

After this preparatory work, we give an algorithm for eliminating exceptions from an extended context-free grammar $G = (N, \Sigma, P, S)$ with exceptions. The algorithm, given in Fig. 3, propagates the exceptions in a production schema to the nonterminals in the schema. The algorithm produces an extended context-free grammar $G' = (N', \Sigma', P', S')$ that is structurally equivalent to G as we establish in the following. The nonterminals of G' have the form A_{+I-X}, where $A \in N$ and $I, X \subseteq N$. A derivation step using a new production schema $A_{+I-X} \to E$ 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, from each nonterminal A, 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 extended context-free grammar with exceptions:

$$\begin{array}{rcl}
A & \to & (A_1 \cup \cdots \cup A_m) + \emptyset - \emptyset, \\
A_1 & \to & (a_1 \cup A) + \{A_2\} - \emptyset, \\
A_2 & \to & (a_2 \cup A) + \{A_3\} - \emptyset,
\end{array}$$

Algorithm I:

$N' := \{A_{+\emptyset-\emptyset} \mid A \in N\};$
$S' := S_{+\emptyset-\emptyset};$
$\Sigma' := \Sigma;$
$Q := \{A_{+\emptyset-\emptyset} \to E + I - X \mid A \to E + I - X \in P\};$
for all $A_{+I_A-X_A} \to E + I - X \in Q$
do for all $(B \in (sym(E) \cup I) - X)$ **and** $B_{+I-X} \notin N'$
  do $N' := N' \cup \{B_{+I-X}\};$
    $Q := Q \cup \{B_{+I-X} \to E_B + (I \cup I_B) - (X \cup X_B)$
      $\mid B_{+\emptyset-\emptyset} \to E_B + I_B - X_B \in Q\};$
  od;
od;
$P' := \{A_{+I_A-X_A} \to E_A \mid A_{+I_A-X_A} \to E + I - X \in Q$ and
    $E_A = ((E_{+I})_{-X})_{\pm(I,X)}\};$

Figure 3: Exception elimination from an extended context-free grammar (N, Σ, P, S) with exceptions.

$$\vdots$$
$$A_m \quad \to \quad (a_m \cup A) + \{A_1\} - \emptyset.$$

Given this grammar the algorithm produces production schemas of the form

$$A_{+I-\emptyset} \to E$$

for every subset $I \subseteq \{A_1, \ldots, A_m\}$. We conjecture that this exponential behavior cannot be avoided.

We eliminate exceptions from an extended context-free grammar using the algorithm of Fig. 3 to produce a grammar that is structurally equivalent to the original grammar. By structural equivalence we mean that not only do the two grammars have the same language, but also they impose an isomorphic derivation or parse on each of their terminal strings. This property is important in the case of SGML, where applications define the semantics of the elements using their structural relationships.

We first define **structural derivations** with respect to (extended) context-free grammars as follows: Let $G = (N, \Sigma, P, S)$ be an extended context-free grammar and $G' = (N', \Sigma', P', S')$ be an extended context-free grammar with exceptions. Let '[' and ']' be new symbols that are not in $N \cup N' \cup \Sigma \cup \Sigma'$. When $\beta = \beta_1 A \beta_2$ is in $(N \cup \Sigma \cup \{[,]\})^*$, $A \to E$ is in P, and α is in $L(E)$, we say that the string $\beta_1[\alpha]\beta_2$ can be **structurally derived** from the string β using grammar G. We denote structural derivation by writing

$$\beta \underset{G,[,]}{\Rightarrow} \beta_1[\alpha]\beta_2.$$

Let $\beta' = \beta'_1 A_{+I-X} \beta'_2$ be a string that consists of symbols from $\Sigma' \cup \{[,]\}$ and of nonterminals from N' with exceptions. We say that the string $\beta'_1[\alpha']\beta'_2$ can be structurally derived from β', denoted by $\beta' \underset{G',[,]}{\Rightarrow} \beta'_1[\alpha']\beta'_2$, when the following two conditions hold:

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

Finally, the **structural languages described by G and G$'$,** denoted by $L(G, [,])$ and $L(G', [,])$, respectively, are the languages

$$\{w \in (\Sigma \cup \{[,]\})^* \mid [S] \underset{G,[,]}{\Rightarrow}^+ w\}$$

and

$$\{w \in (\Sigma' \cup \{[,]\})^* \mid [S_{+\emptyset-\emptyset}] \underset{G',[,]}{\Rightarrow}^+ w\}.$$

When $L(G, [,]) = L(G', [,])$ we say that G and G' are **structurally equivalent.**

We are now able to prove the following result as a byproduct of Lemma 3.2 and Theorem 3.3.

Theorem 3.1 *Given an extended context-free grammar $G = (N, \Sigma, P, S)$ with exceptions there is a structurally equivalent extended context-free grammar $G' = (N', \Sigma, P', S'_{+\emptyset-\emptyset})$. Moreover, G' can be constructed effectively.*

Let $G_1 = (N_1, \Sigma, P_1, S_1)$ be an extended context-free grammar with exceptions and let $G_2 = (N_2, \Sigma, P_2, S_2)$ be the extended context-free grammar obtained when we eliminate exceptions from G using the algorithm in Fig. 3.

Lemma 3.2 *Let* $S_1 \underset{G_1}{\Rightarrow}^+ \alpha B_{+I-X}\beta$ *for some* $\alpha, \beta \in (N_1 \cup \Sigma)^*$, $B \in N_1$ *and some* $I, X \subseteq N_1$.

Then, there is a nonterminal $B_{+I-X} \in N_2$ *and a production schema* $B_{+I-X} \rightarrow E_B$ *in* P_2 *if and only if there is a production schema* $B \rightarrow E + I_B - X_B$ *in* P_1 *such that* $E_B = (E_{+I\cup I_B - X\cup X_B})_{\pm(I\cup I_B, X\cup X_B)}$.

Proof. By inspection of the algorithm in Fig. 3. $\square$

Theorem 3.3 *G_1 and G_2 are structurally equivalent.*

Proof. We show by induction on the length n of derivations that $[S_{1+\emptyset-\emptyset}] \underset{G_1,[,]}{\Rightarrow}^n \alpha$ if and only if $[S_2] \underset{G_2,[,]}{\Rightarrow}^n \alpha$. The claim then follows immediately.

When $n = 0$, $\alpha = [S_{1+\emptyset-\emptyset}] = S_2$. When $n > 0$, $[S_{1+\emptyset-\emptyset}] \underset{G_1,[,]}{\Rightarrow}^n \alpha$ only if $[S_{1+\emptyset-\emptyset}] \underset{G_1,[,]}{\Rightarrow}^{n-1} \alpha_1 A_{+I-X}\alpha_2$ and there is a production schema $(A \rightarrow E + I_A - X_A) \in P_1$ such that $\alpha = \alpha_1[\beta_{\pm(I\cup I_A, X\cup X_A)}]\alpha_2$ for some $\beta \in L(E)_{+I\cup I_A - X\cup X_A}$. By induction this property holds only if $[S_2] \underset{G_2,[,]}{\Rightarrow}^{n-1} \alpha_1 A_{+I-X}\alpha_2$. Lemma 3.2 implies that there is a production schema $(A_{+I-X} \rightarrow E_A) \in P_2$, where $E_A = ((E_{+I\cup I_A})_{-X\cup X_A})_{\pm(I\cup I_A, X\cup X_A)}$. Now $\beta_{\pm(I\cup I_A, X\cup X_A)} \in L(E_A)$ and, therefore, $[S_2] \underset{G_2,[,]}{\Rightarrow}^n \alpha$. The converse direction can be shown in a similar manner. $\square$

Cameron and Wood [9] have proved that structural equivalence is decidable for extended context-free grammars.

4 Content models and their languages

Document type definitions (DTDs) are, essentially, extended context-free grammars that have restricted and generalized regular expressions on the right-hand sides of their productions. The right-hand sides of the productions are called **content models** in the ISO Standard [12, 13]. Content models are similar to regular expressions over an alphabet V. In a DTD the alphabet V consists of **generic identifiers** that are names of **elements** (nonterminal symbols) and #PCDATA. We refer to the members of V simply as **symbols.** Note that #PCDATA is the only "terminal symbol" in the usual grammatical sense, but it denotes all strings over some separate alphabet Σ. The set of all strings over the alphabet Σ is denoted by Σ^* in the usual way.

The inductive definition of content models[1] for an alphabet V is:

$a \in V$	is a content model,
$F \mid G$,	where F and G are content models, is a content model,
FG,	where F and G are content models, is a content model[2],
$F_1 \& \cdots \& F_n$,	where $F_1, \ldots, F_n$ are content models, is a content model,
F^*,	where F is a content model, is a content model,
F^+,	where F is a content model, is a content model,
$F?$,	where F is a content model, is a content model.

The **language L(E) defined by a content model E** is defined inductively as follows:

$$
\begin{aligned}
L(a) &= \{a\}, \text{for } a \in V - \#\text{PCDATA}, \\
L(\#\text{PCDATA}) &= \Sigma^*, \\
L(F \mid G) &= L(F) \cup L(G), \\
L(FG) &= \{vw \mid v \in L(F) \text{ and } w \in L(G)\}, \\
L(F_1 \& \cdots \& F_n) &= \{v_1 \cdots v_n \mid v_i \in L(F_{\Phi(i)}) \text{ for } i = 1, \ldots, n, \text{ where} \\
&\quad\ \Phi \text{ is a permutation of } \{1, \ldots, n\}\}, \\
L(F^*) &= \{v_1 \cdots v_n \mid v_1, \ldots, v_n \in L(F), n \geq 0\}, \\
L(F^+) &= \{v_1 \cdots v_n \mid v_1, \ldots, v_n \in L(F), n \geq 1\}, \\
L(F?) &= L(F) \cup \{\lambda\}.
\end{aligned}
$$

The SGML Standard requires content models to be **unambiguous** in the sense that each nonempty prefix of a string uniquely determines which symbols of the content model match the symbols of the prefix. We follow the approach of Brüggemann-Klein and Wood [4, 5, 6, 7, 8] in the technical treatment of unambiguity.

Let E be an expression or content model over an alphabet V. We often need to refer to different occurrences of symbols in E, which we call the **positions** of the expression E. We do so by **marking** expressions as follows: An expression E' over the alphabet $\Pi = \{a_i \mid a \in V, i \in \{1, 2, \ldots\}\}$ is a

[1]Actually an SGML content model consists of a **model group,** which is similar to a regular expression, and optional exceptions. We deviate from this parlance by calling model groups content models, and by considering exceptions as a separate addition to a content model.

[2]Note that we use the traditional syntax of regular expressions to denote catenation instead of the comma used in the SGML Standard.

marking of an expression E if E' is obtained from E by attaching a different subscript i to each appearance of symbols in E. The subscripted symbols are the *positions* of E'. We use the letters E, F and G to refer to expressions, the early lower-case letters a, b, ..., for symbols of V, and the late lower-case letters x, y, ..., for the subscripted symbols of Π. Finally, we use u, v and w for strings over V or over Π. For a subscripted symbol a_i, we denote its underlying letter a by $\chi(a_i)$, and for a set of subscripted symbols $A \subseteq \Pi$ we define $\chi(A) = \{\chi(x) \mid x \in A\}$. For a marking E' of an expression E we denote its underlying unmarked expression by $\chi(E')$. In the sequel, when we refer to the positions $\mathrm{pos}(E)$ of an expression E, we assume that we have some fixed marking for the expression E.

Let E be a content model and let E' be any marking of E. A content model E is **ambiguous** if there are strings u, v, and w over Π and symbols x and y of Π such that both uxv and uyw are in $L(E')$, $\chi(x) = \chi(y)$ and $x \neq y$. We call such symbols x and y of Π, for which both uxv and uyw are strings of $L(E')$ and $x \neq y$, **competing positions** of E. A content model E is **unambiguous** if it is not ambiguous.

The **subexpressions** of a content model E are defined inductively as follows:

1. E is a subexpression of E.
2. If $E = FG$ or $E = F \mid G$, then the subexpressions of F and of G are subexpressions of E.
3. If $E = F_1 \& \cdots \& F_n$, then the subexpressions of F_i are subexpressions of E, for each $i = 1, \ldots, n$.
4. If $E = F?$, $E = F^*$, or $E = F^+$, then the subexpressions of F are subexpressions of E.

We relate the unambiguity of a content model to the unambiguity of its subexpressions in the following expected way.

Lemma 4.1 *A content model is unambiguous if and only if it has no ambiguous subexpressions.*

Proof. Straightforward induction on the structure of a content model. □

5 Inclusion exceptions and content models

We begin the discussion of exceptions in SGML DTDs by considering how to eliminate inclusion exceptions from SGML content models. We demonstrate, in Section 5.1, how inclusion exceptions can be compiled into content models, giving content models that (locally) realize the effect of inclusions.

The ability to remove exceptions from SGML DTDs is a much more subtle issue than we first thought [15]. The reasons are two-fold. First, any transformation of a DTD to remove exceptions must preserve the set of document instances in a way we make precise. Second, the resulting DTD without exceptions must also be unambiguous in the SGML sense [13] as characterized by Brüggemann-Klein and Wood [8]. Third, the transformation must take into account instances of #PCDATA appearing in content models [14]. In the preliminary version of this paper [15] we ignored this issue completely.

In the case of SGML the elimination of exceptions may also lengthen content models by an exponential factor. Since such pathological cases are almost certainly rare, we believe that the methods we have developed are of practical value. One application that requires the elimination of exceptions from content models is the translation of SGML DTDs into static database schemas. This method of integrating textual documents into an object-oriented database has been suggested by Christofides *et al.* [10]. A second application is the conversion of legacy SGML DTDs into XML DTDs [14] since XML does not allow exceptions.

The SGML Standard requires that content models must be **unambiguous,** meaning that each nonempty prefix of a string uniquely determines which symbols of the content model match the symbols of the prefix. Our methods of eliminating exceptions preserve the unambiguity of the original content models. In this respect our work extends the work of Brüggemann-Klein and Wood [4, 5, 6, 7, 8].

5.1 Compilation of inclusion exceptions

We consider the elimination of inclusion exceptions from SGML content models. We begin by formalizing the meaning of inclusions according to the SGML Standard. We then show how inclusion exceptions can be eliminated from content models to get inclusion-free content models that describe the same language. The methods are based on the insertion of repetitions of included symbols as new subexpressions in the content model as we did when eliminating inclusion exceptions from extended context-free grammars in Section 3. The & operator complicates inclusion elimination in an essential way. Also content models with #PCDATA cause problems. Indeed some content models with #PCDATA do not allow the elimination of inclusions. Therefore, we first present inclusion elimination for content models without #PCDATA and without the & operator. Second, we discuss inclusion elimination from content models that contain #PCDATA. Third, we present the more involved methods needed to eliminate inclusion exceptions from content models with the & operator but without #PCDATA. Lastly, we show the correctness of the elimination methods; that is, that they produce content models which describe the same language as the original content model with inclusions without introducing ambiguity.

5.2 Local semantics of inclusion exceptions

The SGML Standard describes the basic meaning of inclusions as follows: "Elements named in an inclusion can occur anywhere within the content of the element being defined, including anywhere in the content of its subelements." The description is refined by the rule specifying that "...an element that can satisfy an element token in the content model is considered to do so, even if the element is also an inclusion." This refinement means, for example, that a content model $(a \mid b)$ with inclusion a describes ba but does not describe ab, whose first symbol "satisfies the content model" and is therefore not treated as an inclusion. The Standard recommends that inclusions "...should be used only for elements that are not logically part of the content"; for example, neither for a nor for b in the preceding example. Since the difficulty of understanding inclusions is caused, however, by the inclusion of elements that appear in the content model, we have to consider them specifically.

Let L be a language over an alphabet V. We need to refer to symbols that can start some string in the language L and to strings that are suffixes of some string in L whose prefix has been given. We define the sets

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

and

$$\mathrm{tail}(L, w) \quad = \quad \{u \in V^* \mid wu \in L\},$$

for every $w \in V^*$. Observe that $\text{tail}(L, \lambda) = L$.

Let $I = \{i_1, \ldots, i_k\}$, a subset of V, be a set of inclusion symbols. We define the **SGML effect of inclusions** I **on language** L as the language

$$\begin{aligned} L_{\oplus I} = \{w_0 a_1 \cdots w_{n-1} a_n w_n \quad | \quad & a_1 \cdots a_n \in L, n \geq 0, \\ & w_i \in (I - \text{first}(\text{tail}(L, a_1 \cdots a_i)))^*, i = 0, \ldots, n\}. \end{aligned}$$

Intuitively, to obtain $L_{\oplus I}$, we may add, for any prefix v of any string w in L, a string u of included symbols from I such that vu is not a prefix of any string in L. Notice that $L_{\oplus I} \subseteq L_{+I}$. For example, $\{a, b\}_{+\{a\}} = \{a^k \mid k > 0\} \cup \{a^k b a^l \mid k, l \geq 0\}$ and the language $\{a, b\}_{\oplus\{a\}}$ consists of all strings of the forms aa^k and ba^k, where $k \geq 0$.

5.3 Element content models without & groups

The basic idea of compiling the inclusion of a set $I = \{i_1, \ldots, i_k\}$ of symbols in a content model E is to insert new subexpressions of the form $(i_1 \mid \cdots \mid i_k)^*$ in E. Preserving the unambiguity of the content model requires extra care, since inserting new symbols in content models easily introduces ambiguity. As a simple example, consider capturing the effect of inclusions $I = \{b, c\}$ on the content model $E = ab$. The straightforward insertion of inclusion symbols would yield content model

$$F = (b \mid c)^* a (b \mid c)^* b (b \mid c)^*,$$

which is ambiguous since the symbol b of the string ab can be matched by either the second or the third occurrence of b in F. On the other hand, the method that we present later gives the content model

$$(b \mid c)^* a c^* b (b \mid c)^*,$$

which is unambiguous.

We need to consider the relationships of positions in a content model to define an appropriate transformation. These relationships are captured for content models E and their positions x by the sets $\text{first}(E)$, $\text{last}(E)$ and $\text{follow}^-(E, x)$. The set $\text{first}(E)$ contains the positions that can begin a string described by a marked content model E. It is defined inductively as follows:

$$\text{first}(x) \quad = \quad x \text{ when } x \in \Pi;$$

$$
\begin{aligned}
\mathrm{first}(F \mid G) &= \mathrm{first}(F) \cup \mathrm{first}(G);\\
\mathrm{first}(FG) &= \begin{cases} \mathrm{first}(F), & \text{if } \lambda \notin L(F),\\ \mathrm{first}(F) \cup \mathrm{first}(G), & \text{if } \lambda \in L(F); \end{cases}\\
\mathrm{first}(F_1 \& \cdots \& F_n) &= \bigcup_{1 \le i \le n} \mathrm{first}(F_i);\\
\mathrm{first}(F?) &= \mathrm{first}(F^*) = \mathrm{first}(F^+)\\
&= \mathrm{first}(F).
\end{aligned}
$$

The inductive definition of the set $\mathrm{last}(E)$ that consists of the positions of a content model E that can end a string described by a marking of E, is similar to the definition of $\mathrm{first}(E)$, except for the case of catenation, which is defined as follows:

$$
\mathrm{last}(FG) = \begin{cases} \mathrm{last}(G), & \text{if } \lambda \notin L(G),\\ \mathrm{last}(F) \cup \mathrm{last}(G), & \text{if } \lambda \in L(G). \end{cases}
$$

Third, for a marked content model E and its positions $x \in \mathrm{pos}(E)$, we need to consider the set of positions that can follow x in strings described by E. Moreover, for content models with & groups we have to consider a slightly restricted subset of following positions denoted by $\mathrm{follow}^-(E, x)$, which was first suggested by Clark [11]. The inductive definition, adapted from Brüggemann-Klein's exposition [5], is:

$$
\begin{aligned}
\mathrm{follow}^-(x, x) &= \emptyset;\\
\mathrm{follow}^-(F \mid G, x) &= \begin{cases} \mathrm{follow}^-(F, x), & \text{if } x \in \mathrm{pos}(F),\\ \mathrm{follow}^-(G, x), & \text{if } x \in \mathrm{pos}(G); \end{cases}\\
\mathrm{follow}^-(FG, x) &= \begin{cases} \mathrm{follow}^-(F, x), & \text{if } x \in \mathrm{pos}(F), x \notin \mathrm{last}(F),\\ \mathrm{follow}^-(F, x) \cup \mathrm{first}(G), & \text{if } x \in \mathrm{last}(F),\\ \mathrm{follow}^-(G, x), & \text{if } x \in \mathrm{pos}(G); \end{cases}\\
\mathrm{follow}^-(F_1 \& \cdots \& F_n, x) &= \begin{cases} \mathrm{follow}^-(F_i, x), & \text{if } x \in \mathrm{pos}(F_i), x \notin \mathrm{last}(F_i),\\ \mathrm{follow}^-(F_i, x) \cup & \\ \quad \bigcup_{j \ne i, \lambda \in L(F_j)} \mathrm{first}(F_j), & \text{if } x \in \mathrm{last}(F_i); \end{cases}\\
\mathrm{follow}^-(F?, x) &= \mathrm{follow}^-(F, x);\\
\mathrm{follow}^-(F^*, x) &= \mathrm{follow}^-(F^+, x)\\
&= \begin{cases} \mathrm{follow}^-(F, x), & \text{if } x \in \mathrm{pos}(F), x \notin \mathrm{last}(F),\\ \mathrm{follow}^-(F, x) \cup \mathrm{first}(F), & \text{if } x \in \mathrm{last}(F). \end{cases}
\end{aligned}
$$

The next result gives the relationship of the unambiguity of E and the $\mathrm{follow}^-(F,x)$ sets for subexpressions F of E. It forms the basis of linear-time unambiguity testing of content models [5, 6].

Lemma 5.1 *[5] A marked content model E is unambiguous if and only if it satisfies the following three conditions:*

1. *If $x, y \in \mathrm{first}(L(E))$ and $x \neq y$, then $\chi(x) \neq \chi(y)$.*
2. *If $x, y \in \mathrm{follow}^-(E,z)$ for some $z \in \mathrm{pos}(E)$ and $x \neq y$, then $\chi(x) \neq \chi(y)$.*
3. *If $E_1\&\cdots\&E_n$ is a subexpression of E and there is some $z \in \mathrm{last}(E_i)$, for some $1 \leq i \leq n$, such that $x \in \mathrm{follow}^-(E_i,z)$ and $y \in \mathrm{first}(E_j)$, for some $j \neq i$, then $\chi(x) \neq \chi(y)$.*

We modify a context model E with inclusion exceptions I in two steps to obtain a new content model $E_{\oplus I}$ that captures the effect of I on E. We first insert the appropriate repetitive component after each position in a content model E to give a new content model $E_{\bullet I}$. We use A^* as a shorthand notation for the expression $(a_1 \mid \cdots \mid a_k)^*$ when $A = \{a_1, \ldots, a_k\}$ is a set of symbols. We define $E_{\bullet I}$ inductively as follows:

$$\begin{array}{rcl}
(F \mid G)_{\bullet I} & = & F_{\bullet I} \mid G_{\bullet I}, \\
(FG)_{\bullet I} & = & F_{\bullet I} G_{\bullet I}, \\
(F_1\&\cdots\&F_n)_{\bullet I} & = & F_{1\bullet I}\&\cdots\&F_{n\bullet I}, \\
(F^*)_{\bullet I} & = & (F_{\bullet I})^*, \\
(F^+)_{\bullet I} & = & (F_{\bullet I})^+, \\
(F?)_{\bullet I} & = & (F_{\bullet I})?.
\end{array}$$

The base case which inserts new symbols into the content model for a position x with $\chi(x) = a \in V$ is

$$x_{\bullet I} = a(I - \chi(\mathrm{follow}^-(E,x)))^*.$$

Notice that the definition of $x_{\bullet I}$ avoids the introduction of new positions, which would compete with existing ones having the same underlying character, by insertion of, immediately after position x, only such new symbols that do not collide with the underlying characters of the set $\mathrm{follow}^-(E,x)$.

Second, we construct a new content model $E_{\oplus I}$ that is defined as

$$(I - \mathrm{first}(L(E)))^* E_{\bullet I}.$$

Example 5.2 *Consider the content model* $E = (a^+ \mid b^+)c$ *and a set* $I = \{b, c\}$ *of inclusions. Now,* $\mathrm{follow}^-(E, a) = \{a, c\}$, $\mathrm{follow}^-(E, b) = \{b, c\}$ *and* $\mathrm{follow}^-(E, c) = \emptyset$*; therefore,*

$$E_{\bullet I} = ((ab^*)^+ \mid b^+)c(b \mid c)^*.$$

Since $\mathrm{first}(L(E)) = \{a, b\}$,

$$E_{\oplus I} = c^*((ab^*)^+ \mid b^+)c(b \mid c)^*.$$

□

We will present a complete method for eliminating inclusion exceptions, in Section 5.5, from element content models that possibly contain & operators. The correctness of the method is shown in Section 5.6.

5.4 Mixed content models

The SGML Standard defines an element to have **mixed content** if it can contain data characters intermixed with subelements. Otherwise, it defines an element to have **element content.**

Notice that since $L(\#\mathrm{PCDATA}) = \Sigma^*$, for any $a \in V$,

$$L(\#\mathrm{PCDATA})_{\oplus\{a\}} = \{u_0 v_1 u_1 \cdots v_n u_n \mid u_i \in \{a\}^*, \text{ for } i = 0, \ldots, n, \text{ and } v_j \in \Sigma^* \text{ for } j = 1, \ldots, n\}.$$

This semantics is difficult or even impossible to capture for unrestricted content models with #PCDATA. For example, consider content model $E = a \mid \#\mathrm{PCDATA}$. We could try to capture the language $L(E)_{\oplus\{b\}}$ with a content model of the form $F \mid G$, where F describes the language $L(b^*ab^*)$ and G describes any sequence of strings $v_1, \ldots, v_n \in \Sigma^*$ intermixed with sequences of b. Such a content model is ambiguous, since an initial symbol b of a string could be matched by the occurrence of b either in F or in G. Alternatively, we could try to express the effect of the inclusion by a content model of the form $F(G \mid H)J$, where J is either a model group or it is omitted, and $L(F) = L(b^*)$, $L(GJ) = L(ab^*)$ and $L(HJ)$ consists of strings beginning with a string of data characters followed by sequences of b intermixed with strings of data characters. The only construct available in SGML to define strings over Σ in a content model is #PCDATA, which also describes the null string. Therefore, the second candidate content model is also unambiguous, since any initial occurrence of symbol b in a string could be matched by either a position of F or by a position in $\mathrm{pos}(H) \cup \mathrm{pos}(J)$.

Clause 11.2.4 of the SGML standard recommends that #PCDATA should be used only when strings of Σ^* are permitted to appear anywhere in the content of the element; that is, either when the content model is only #PCDATA or "when **or** is the only connector used in any model group". The reason for this recommendation is to avoid problems caused by interpreting separator characters in a mixed element instance as data characters. Goldfarb's annotations of the standard [12, p. 411], as well as those of some other authors [17, 16], interpret these recommendations to mean that a mixed content model should always be "a repeatable OR group".

We say that a content model E is a **mixed content model** if both data characters and element symbols can occur in the strings of $L(E)$; otherwise, we say that E is an **element content model.** We can trivially recognize a mixed content model: *A content model is a mixed content model if and only if it contains an occurrence of the keyword #PCDATA.* We say that a mixed content model E is **simple** if it has the form

$$(\#\mathrm{PCDATA} \mid A_1 \mid \cdots \mid A_n)\,op,$$

where $A_1, \ldots, A_n$, $n \geq 0$, are symbols and op is either * or $^+$. Notice that when $n = 0$, a simple mixed content model is equivalent to (#PCDATA). XML restricts mixed content models to be simple [2].

It is easy to capture the effect of inclusion exceptions I on a simple mixed content model E with the content model

$$F = (\#\mathrm{PCDATA} \mid A_1 \mid \cdots \mid A_n \mid B_1 \mid \cdots \mid B_k)\,op,$$

where $\{B_1, \ldots, B_k\} = I - \{A_1, \ldots, A_n\}$. It is not difficult to show that $L(F) = L(E)_{\oplus I}$. Also, F is unambiguous if and only if E is unambiguous. To see this, observe that F contains two competing positions with the same underlying symbol if and only if E does.

5.5 Element content models with & groups

We demonstrate the difficulties that are caused by the & operator with the following example. Consider the content model $E = a?\&b?$, which is unambiguous. A content model that captures the inclusion of symbol a in E

should describe strings of the form $ba \cdots a$. A straightforward transformation would produce a content model of the form $E_1 = F\&((ba^*)?)$ or of the form $E_2 = (F\&b?)a^*$, where $a \in L(F)$ and $\lambda \in L(F)$. It is easy to see that these content models are ambiguous. In the case of E_1, the second symbol of the string ba could be matched by either F or by a^*. In the case of E_2, any string that begins with an a can be matched by both F and a^*. Our strategy to handle such problematic subexpressions $F\&G$ is first to replace them by the equivalent subexpression $(FG \mid GF)$. Notice that this substitution may not suffice, since the replacement can be ambiguous even if $F\&G$ is unambiguous. For example, the content model $(a?b? \mid b?a?)$ is ambiguous, whereas the content model $a?\&b?$ is unambiguous. Also, substituting other expressions for & groups that occur as a subexpression of an iterative expression F^* or F^+ has to be done with care to avoid introducing ambiguity. As a matter of fact, we know that the language of some unambiguous content models with & groups cannot be expressed by unambiguous content models without & groups [3, 7].

We introduce three mutually recursive transformations $\mathcal{E}$, $\mathcal{F}$ and $\mathcal{G}$ to first eliminate problematic & groups as a preliminary step of inclusion elimination. Under certain conditions, transformations $\mathcal{E}$, $\mathcal{F}$ and $\mathcal{G}$ preserve the unambiguity of the original content models. To specify these conditions we introduce the notions of *sublanguage expressions* and *iterative & groups*. Let E be a model group and F be a subexpression of E. We say that F is a **sublanguage expression** of E, if $L(F) \subseteq L(E)$. We say that F is a **sublanguage expression of an iteration**, or simply **iterative,** (in E) if it is a sublanguage expression of a subexpression G^* or G^+ of E. If subexpression F is not iterative (in E), we say that it is **noniterative** (in E). As an example, consider the content model $E = ((a \mid b^*)(c?\&d)e)^+$. Its subexpression $F = (c?\&d)$ is not a sublanguage expression of E, since each string in $L(E)$ ends with symbol e, which does not occur in F. On the other hand, F is a sublanguage expression of $G = ((a \mid b^*)(c?\&d)e^*)^+$, and it is also iterative in G.

The purpose of transformation $\mathcal{G}$ is to modify a given content model E to describe the same language as E except possibly for the null string λ. The value of $\mathcal{G}(E)$, for a content model E over V or Π, is defined as follows:

$$\begin{aligned}
\mathcal{G}(x) &= x \text{ when } x \in V \text{ or } x \in \Pi; \\
\mathcal{G}(F \mid G) &= \mathcal{G}(F) \mid \mathcal{G}(G);
\end{aligned}$$

$$\mathcal{G}(FG) = \begin{cases} FG, & \text{if } \lambda \notin L(FG), \\ (\mathcal{G}(F) \mid \mathcal{G}(G)), & \text{if } \lambda \in L(FG) \text{ and } FG \text{ is iterative}, \\ ((\mathcal{G}(F)G) \mid \mathcal{G}(G)), & \text{if } \lambda \in L(FG) \text{ and } FG \text{ is noniterative}; \end{cases}$$
$$\mathcal{G}(E_1\&\cdots\&E_n) = \begin{cases} E_1\&\cdots\&E_n, & \text{if } \lambda \notin L(E_1\&\cdots\&E_n), \\ \mathcal{F}(E_1\&\cdots\&E_n), & \text{if } \lambda \in L(E_1\&\cdots\&E_n); \end{cases}$$
$$\mathcal{G}(F?) = \mathcal{G}(F);$$
$$\mathcal{G}(F^*) = \mathcal{G}(F^+) = (\mathcal{G}(F))^+.$$

Notice that transformation $\mathcal{G}$ does not work for #PCDATA since neither SGML nor XML provides any means of expressing the language Σ^+ of nonempty unstructured strings.

Transformation $\mathcal{F}$ is applied to & groups. The value of $\mathcal{F}(E)$ for an & group E is a corresponding | group that describes the same language as E apart from the null string. It is defined for an & group $E = E_1\&\cdots\&E_n$ as follows:

$$\begin{aligned} \mathcal{F}(E) &= \mathcal{G}(E), \text{ if } n = 1; \\ \mathcal{F}(E) &= (\mathcal{G}(E_1)\mathcal{E}(E_2\&E_3\&\cdots\&E_n)) \\ &\quad \mid (\mathcal{G}(E_2)\mathcal{E}(E_1\&E_3\&E_4\&\cdots\&E_n)) \\ &\quad \vdots \\ &\quad \mid (\mathcal{G}(E_n)\mathcal{E}(E_1\&E_2\&\cdots\&E_{n-1})), \text{ if } n > 1. \end{aligned}$$

That is, $\mathcal{F}(E_1\&\cdots\&E_n)$ consists of an | group of n subexpressions, each of which begins with a copy of a different E_i modified by transformation $\mathcal{G}$. When $n = 1$, the & group consists of a single subexpression; therefore, we apply $\mathcal{G}$ directly to it.

Finally, transformation $\mathcal{E}$ is just a "wrapper" for transformation $\mathcal{F}$. Its purpose is to eliminate the & operators from a given & group and to preserve its language including the null string.

$$\mathcal{E}(E) = \begin{cases} (\mathcal{F}(E))?, & \text{if } \lambda \in L(E); \\ \mathcal{F}(E), & \text{if } \lambda \notin L(E). \end{cases}$$

For example,

$$\begin{aligned} \mathcal{E}(a\&b?\&c^*) = &\quad (a((b((c)^+)?) \mid ((c)^+(b)?))?) \\ &\mid (b((a((c)^+)?) \mid ((c)^+a))) \\ &\mid ((c)^+((a(b)?) \mid (ba))) \end{aligned}$$

Kilpeläinen [14] proved the following result, which states that the preceding transformations achieve their goal in terms of the languages described by the expressions.

Lemma 5.3 *Let E be a model group over V and let F be an & group. Then,*

1. $L(\mathcal{G}(E)) = L(E) - \{\lambda\}$.
2. $L(\mathcal{E}(F)) = L(F)$.

$\square$

An expression F is **nullable** if $\lambda \in L(F)$. To compute $\mathcal{G}(E)$ we need to determine the nullable subexpressions of E. They can be computed in linear time using a postorder traversal of the corresponding expression tree for E [1, 14]. Given this information, we can compute $\mathcal{E}(E)$ in one pass; however, the expansion of subexpressions of the form $E_1 \& \cdots \& E_n$ causes a superexponential increase in the size of the resulting expressions. Specifically, when $a_1, \ldots, a_n$ are distinct symbols, expression $\mathcal{E}(a_1 \& \cdots \& a_n)$ contains at most $en!$ occurrences of symbols, where $e = 2.71828\ldots$ is the base of the natural logarithm and this bound is tight [14].

The following results about the preservation of unambiguity by transformations $\mathcal{E}$ and $\mathcal{G}$ were derived by Kilpeläinen [14].

Lemma 5.4 *Let E be an unambiguous model group which contains no nullable and iterative & groups. Then, $\mathcal{G}(E)$ is unambiguous.* $\square$

Lemma 5.5 *Let $E = E_1 \& \cdots \& E_n$ be an unambiguous & group such that none of E_i, $i = 1, \ldots, n$, contains either #PCDATA or a nullable and iterative & group as its sublanguage expression. Then, $\mathcal{E}(E)$ is unambiguous.* $\square$

The third result concerns the unambiguity of content models whose & subexpressions are replaced by the $\mathcal{E}$ transformation. To discuss these, we consider markings of content models. Note that each subexpression F of a marked expression E is unique. Therefore, we use the notation $E[F/\mathcal{E}(F)]$ to unambiguously denote the expression that results by replacing subexpression F of a marked content model E by $\mathcal{E}(F)$.

Lemma 5.6 *Let E' be a marking of an unambiguous element content model and let $F' = F'_1 \& \cdots \& F'_n$ be a noniterative subexpression of E' which contains no iterative & group as a subexpression. Then, $\chi(E'[F'/\mathcal{E}(F')])$ is unambiguous.* $\square$

Brüggemann-Klein [3, 7] has demonstrated that some iterative & groups cannot be eliminated from content models while simultaneously preserving unambiguity. For the purpose of eliminating inclusion exceptions this does not matter, however. The following property is sufficient for the correctness of our method: For any unambiguous content model E without noniterative & groups, the sets $\text{follow}^-(E', x)$ of positions x in E's marking E' capture the information of the **tail set** for strings accepted by a path leading to x. This property is formulated as follows.

Lemma 5.7 *Let E be an unambiguous content model that has no noniterative & groups as a subexpression. Let E' be a marking of E with $x \in sym(E')$, $w \in V^*$ and $w' \in \Pi^*$ such that, for some $v \in V^*$ and $v' \in \Pi^*$, we have*

$$w\chi(x)v \in L(E),\ w'xv' \in L(E') \text{ and } w = \chi(w').$$

Then,

$$\text{first}(\text{tail}(L(E), w\chi(x))) = \chi(\text{follow}^-(E', x)).$$

Sketch of the proof. The unambiguity of E implies that position x is determined by the string $w\chi(x)$ unambiguously. By structural inspection of E' it is easy to see that $\chi(\text{follow}^-(E', x)) \subseteq \text{first}(\text{tail}(L(E), w\chi(x)))$. On the other hand, let F' be a subexpression of E' such that $x \in sym(F')$ and let u', t' be in Π^* such that $u'xt' \in L(F')$. The only case in which $y \in \text{first}(\text{tail}(L(F'), u'x))$ but $y \notin \text{follow}^-(F', x)$ is when $x \in \text{last}(F_i)$ and $y \in \text{first}(F_j)$ for some subexpression $F_1 \& \cdots \& F_n$ of F'. Now, by the assumptions of the lemma, $F_1 \& \cdots \& F_n$ is a sublanguage expression of some subexpression G^* or G^+ of E', which implies that $y \in \text{first}(G)$, $x \in \text{last}(G)$, and $y \in \text{follow}^-(E', x)$. $\square$

Now we are ready to provide a method of computing, for a given content model E with element content and a set I of inclusions, a new content model $E_{\oplus I}$ such that $L(E_{\oplus I}) = L(E)_{\oplus I}$. The algorithm is given in Figure 4.

Example 5.8 *Let $E = (a?\&b?)c$ and $I = \{a, c\}$. We first obtain the content model*

$$F = \mathcal{E}(a?\&b?)c = (ab? \mid ba?)?c$$

	Algorithm II
1.	Replace any noniterative subexpressions $F = F_1 \& \cdots \& F_n$ of E by $\mathcal{E}(F)$.
2.	$E_{\oplus I} := (I - \mathrm{first}(L(E)))^* E_{\bullet I}$.

Figure 4: Computation of the content model $E_{\oplus I}$ from a content model E.

and then the content model

$$F_{\bullet I} = (aa^*(ba^*)? \mid b(aa^*)?)?c(a \mid c)^*.$$

Since $I \subseteq \mathrm{first}(L(E))$, *we have* $E_{\oplus I} = F_{\bullet I}$. □

5.6 Correctness of inclusion elimination

We sketch the proof of correctness of Algorithm II that is given in Fig. 4. Let E be an unambiguous element content model. Let G denote expression E as modified by Step 1 of Algorithm II. By Lemma 5.6, G is also unambiguous and Lemma 5.3 implies that $L(G) = L(E)$.

Lemma 5.9 *If G is an unambiguous content model, then*

$$\begin{aligned} L(G_{\bullet I}) \quad = \quad & \{a_1 w_1 \cdots a_n w_n \mid a_1 \cdots a_n \in L(G), n \geq 0, \\ & w_i \in (I - \mathrm{first}(\mathrm{tail}(L(G), a_1 \cdots a_i)))^*, i = 1, \ldots, n\}. \end{aligned}$$

Proof. Let G' be a marking of G. The claim follows from Lemma 5.7, which implies that

$$I \cap \mathrm{first}(\mathrm{tail}(L(G), a_1 \cdots a_n)) = I \cap \chi(\mathrm{follow}^-(G', x_n)),$$

for any $a_1 \cdots a_n v \in L(G)$ and $x_1 \cdots x_n w \in L(G')$, such that $\chi(x_1 \cdots x_n w) = a_1 \cdots a_n v$. □

We can apply Lemma 5.1 to show that the $()_{\bullet I}$ transformation does not introduce ambiguity in G.

Lemma 5.10 *If G is unambiguous, then $(G)_{\bullet I}$ is unambiguous.*

Proof. Assume that G is unambiguous. Let G' be a marking of G and let H' be a marking of $H = (G)_{\bullet I}$ such that $L(G') \subseteq L(H')$. Such markings clearly exist. Then, the first condition of Lemma 5.1 for the unambiguity of H' is satisfied since $\mathrm{first}(L(H')) = \mathrm{first}(L(G'))$ and G is unambiguous.

For the second condition, consider a symbol z in G' and the corresponding subexpression $z(y_1 \mid \cdots \mid y_k)^*$ in H'. Now,

$$\begin{aligned} \text{follow}^-(H', z) &= \text{follow}^-(H', y_i), i = 1, \ldots, k, \\ &= \text{follow}^-(G', z) \cup \{y_1, \ldots, y_k\}, \end{aligned}$$

which by the construction of $(G)_{\bullet I}$ and the unambiguity of G cannot contain two positions with the same underlying symbol. Finally, consider the case $z \in \text{last}(H_i)$ for a subexpression $H_1 \& \cdots \& H_n$ of H', $x \in \text{follow}^-(H_i, z)$, and $y \in \text{first}(H_j)$ for some $j \neq i$. Then z is either some symbol $x' \in \mathit{sym}(G')$ or some symbol $x_i \in \mathit{sym}(H') - \mathit{sym}(G')$ in a subexpression $x'(x_1 \mid \cdots \mid x_k)^*$. Let $G_1 \& \cdots \& G_n$ be the subexpression of G' for which $H_1 \& \cdots \& H_n = (G_1 \& \cdots \& G_n)_{\bullet I}$. Now

$$\text{follow}^-(H_i, z) = \text{follow}^-(G_i, x') \cup \{x_1, \ldots, x_k\}$$

and $y \in \text{first}(H_j)$ implies that $y \in \text{follow}^-(G', x')$, since $\text{first}(H_j) = \text{first}(G_j)$ and $G_1 \& \cdots \& G_n$ is iterative in G'. Therefore, $\chi(x) \neq \chi(y)$, for $x \in \{x_1, \ldots, x_k\}$, by the construction of $(G)_{\bullet I}$, and $\chi(x) \neq \chi(y)$, for $x \in \text{follow}^-(G_i, x')$, by the unambiguity of G and Lemma 5.1. □

The preceding properties of $(G)_{\bullet I}$ imply the correctness of the construction of $E_{\oplus I}$, which we summarize as follows.

Theorem 5.11 *Let E be an unambiguous element content model. Then, the content model $E_{\oplus I}$ satisfies the following two conditions:*

1. *$L(E_{\oplus I}) = L(E)_{\oplus I}$.*
2. *$E_{\oplus I}$ is unambiguous.*

In addition, $E_{\oplus I}$ has size $O(|E|!)$ and it can be computed from E in $O(|E|!)$ time, in the worst case.

Corollary 5.12 *For each unambiguous element content model with inclusions there is an equivalent unambiguous content model without inclusions.*

6 Exclusion exceptions and content models

Exclusion exceptions modify the meaning of content models by precluding optional elements from their content. The precise meaning of exclusion exceptions is not quite clear from the SGML standard. The standard gives rather vague restrictions on the applicability of exclusion exceptions. We propose a simple and rigorous definition of the meaning and of the applicability of exclusion exceptions. We also present an optimal algorithm that modifies a given content model to capture the (local) effect of exclusions, and simultaneously checks their applicability to the content model.

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 the phrase "precluding options" is not clear from the Standard. Our first task is, therefore, to formalize the intuitive notion of exclusion. As a motivating example consider excluding the symbol b from the content model $E = a(b \mid c)c$, which defines the language $L(E) = \{abc, acc\}$. The element b is clearly an alternative to the first occurrence of c, and we can realize its exclusion by modifying E to give $E' = acc$. Now, consider excluding b from the content model $F = a(bc \mid cc)$. This case is not as clear since b appears in a **seq** subexpression. On the other hand, both E and F define the same language; therefore, we define the effect of exclusions on languages of content models rather than on the content models themselves.

Let $L \subseteq V^*$ be a language and let $X \subseteq V$. Motivated by the preceding examples, we define the effect of excluding X from L, which we denote by L_{-X}, to be the set of all strings in L that do not contain any symbol of X. As an example, the effect of excluding $\{b\}$ from the language of the preceding content models E and F is

$$L(E)_{-\{b\}} = L(F)_{-\{b\}} = \{acc\}.$$

Notice that an exclusion always specifies a subset of the original language.

We next show how we can compute a content model $E_{\ominus X}$ such that $L(E_{\ominus X}) = L(E)_{-X}$ from a given content model E and a given set X of excluded symbols. The modified content model $E_{\ominus X}$ is unambiguous if the original content model E is unambiguous. The computation of $E_{\ominus X}$ takes time linear in the size of E.

For the transformation, we extend content models to allow the empty-set symbol $\emptyset$ and the null-string symbol λ, 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:

$$\begin{aligned} L(\emptyset) &= \emptyset, \\ L(\lambda) &= \{\lambda\}. \end{aligned}$$

Let E be a content model with exclusions X. We define the corresponding extended content model $E_{\ominus X}$ inductively as follows:

$$a_{\ominus X} = \begin{cases} \emptyset, & \text{if } a \in X, \\ a, & \text{otherwise;} \end{cases}$$

$$(FG)_{\ominus X} = \begin{cases} \emptyset, & \text{if } F_{\ominus X} = \emptyset \text{ or } G_{\ominus X} = \emptyset, \\ F_{\ominus X}, & \text{if } G_{\ominus X} = \lambda, \\ G_{\ominus X}, & \text{if } F_{\ominus X} = \lambda, \\ F_{\ominus X} G_{\ominus X}, & \text{otherwise;} \end{cases}$$

$$(F \mid G)_{\ominus X} = \begin{cases} F_{\ominus X}, & \text{if } G_{\ominus X} = \emptyset, \\ G_{\ominus X}, & \text{if } F_{\ominus X} = \emptyset, \\ \lambda, & \text{if } F_{\ominus X} = \lambda \text{ and } G_{\ominus X} = \lambda, \\ F_{\ominus X}?, & \text{if } F_{\ominus X} \neq \lambda \text{ and } G_{\ominus X} = \lambda, \\ G_{\ominus X}?, & \text{if } F_{\ominus X} = \lambda \text{ and } G_{\ominus X} \neq \lambda, \\ F_{\ominus X} \mid G_{\ominus X}, & \text{otherwise;} \end{cases}$$

$$(F_1 \& \cdots \& F_n)_{\ominus X} = \begin{cases} \emptyset, \quad \text{if } F_{i\ominus X} = \emptyset \text{ for some } i = 1, \ldots, n, \\ \lambda, \quad \text{if } F_{i\ominus X} = \lambda \text{ for all } i = 1, \ldots, n, \\ F_{i_1 \ominus X} \& \cdots \& F_{i_k \ominus X}, \text{ otherwise, where} \\ \qquad 1 \leq i_1 < \cdots < i_k \leq n \text{ are such that} \\ \qquad F_{i_j \ominus X} \neq \lambda, \text{ for each } j = 1, \ldots, k, \text{ and} \\ \qquad F_{j \ominus X} = \lambda \text{ for each } j \in \{1, \ldots, n\} \setminus \{i_1, \ldots, i_k\}; \end{cases}$$

$$(F?)_{\ominus X} = \begin{cases} \lambda, & \text{if } F_{\ominus X} = \emptyset \text{ or } F_{\ominus X} = \lambda, \\ F_{\ominus X}?, & \text{otherwise;} \end{cases}$$

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

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

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

Lemma 6.1 *(Basic properties of $E_{\ominus X}$.)*
If E is a content model over V and $X \subseteq V$, then the following five properties hold for the content model $E_{\ominus X}$:

1. *$E_{\ominus X}$ is a content model ($E_{\ominus X}$ contains neither $\emptyset$ nor λ) if and only if $E_{\ominus X} \notin \{\emptyset, \lambda\}$.*

2. *$L(E_{\ominus X}) = \emptyset$ if and only if $E_{\ominus X} = \emptyset$.*

3. *$L(E_{\ominus X}) = \{\lambda\}$ if and only if $E_{\ominus X} = \lambda$.*

4. *$L(E_{\ominus X}) = L(E)_{-X}$.*

5. *$E_{\ominus X}$ can be computed from E in $O(\mid E \mid)$ time, in the worst case.*

Proof. Properties 1 through 4 can be shown to hold by structural induction on E. (Note that Property 1 implies Properties 2 and 3.) Property 5 can be seen to hold by considering a recursive algorithm based directly on the definition of $E_{\ominus X}$.

To give a flavor of the proofs, we show how to handle the case $E = F \mid G$ in the proof that $L(E_{\ominus X}) = L(E)_{-X}$. In this case,

$$\begin{aligned} L(E)_{-X} &= L(F \mid G)_{-X} \\ &= (L(F) \cup L(G))_{-X} \\ &= L(F)_{-X} \cup L(G)_{-X}. \end{aligned}$$

If $F_{\ominus X} = \emptyset$, then $L(E_{\ominus X}) = L(G_{\ominus X})$. The inductive assumptions $L(F_{\ominus X}) = L(F)_{-X} = \emptyset$ and $L(G_{\ominus X}) = L(G)_{-X}$ imply that $L(E_{\ominus X}) = L(E)_{-X}$. The case $G_{\ominus X} = \emptyset$ is symmetric. If $F_{\ominus X} = G_{\ominus X} = \lambda$, then $L(E_{\ominus X}) = \{\lambda\}$ and $L(F)_{-X} \cup L(G)_{-X} = \{\lambda\}$ by the induction hypothesis. If $F_{\ominus X} = \lambda$ and $G_{\ominus X} \notin \{\emptyset, \lambda\}$, then

$$\begin{aligned} L(E_{\ominus X}) &= L((G_{\ominus X})?) \\ &= L(G_{\ominus X}) \cup \{\lambda\} \end{aligned}$$

and, by induction, $L(E_{\ominus X}) = L(E)_{-X}$. The case $F_{\ominus X} \notin \{\emptyset, \lambda\}$ and $G_{\ominus X} = \lambda$ is symmetric. Finally, if $E_{\ominus X} = F_{\ominus X} \mid G_{\ominus X}$, then we have $L(F_{\ominus X} \mid G_{\ominus X}) = L(F_{\ominus X}) \cup L(G_{\ominus X})$, which gives, by induction, $L(F)_{-X} \cup L(G)_{-X} = L(E)_{-X}$. □

As a restriction of 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 is that $L(E)_{-X} \not\subseteq \{\lambda\}$.* Note that an ordinary content model cannot describe a language that is either $\emptyset$ or $\{\lambda\}$. Intuitively,

$E_{\ominus X} = \emptyset$ or $E_{\ominus X} = \lambda$ means that excluding X from E precludes all elements from the content of E. On the other hand, $E_{\ominus X} \notin \{\emptyset, \lambda\}$ means that X precludes only elements that are optional in $L(E)$. Thus, we propose that this requirement be the formalization of how a model group indicates that an element is required. Notice that computing $E_{\ominus X}$ is a reasonable and efficient test for the applicability of exclusions X on a content model E.

Finally, we show that capturing the effect of exclusions X on a content model E by transforming E into $E_{\ominus X}$ does not sacrifice the unambiguity of E. Let E be a content model and E' be a marking of E. We define $E'_{\ominus X}$ in a similar way to the definition of $E_{\ominus X}$ except that the base case becomes

$$x_{\ominus X} = \begin{cases} \emptyset, & \text{if } \chi(x) \in X, \\ x, & \text{otherwise.} \end{cases}$$

Lemma 6.2 *If E is unambiguous, then $E_{\ominus X}$ is unambiguous.*

Proof. Let E' be a marking of E. Assume that $E_{\ominus X}$ is ambiguous, which means that there are strings u, v and w over Π and symbols x and y in Π such that both uxv and uyw are in $L(E'_{\ominus X})$, $\chi(x) = \chi(y)$, and $x \neq y$. (Note that $E'_{\ominus X}$ is a marking of $E_{\ominus X}$.) Now, $L(E'_{\ominus X}) \subseteq L(E')$, which implies that both uxv and uyw are in $L(E')$ and E is also ambiguous. $\square$

7 SGML DTDs and exceptions

We are now in a position to consider the removal of exceptions from a DTD. Although extended context-free grammars are a reasonable model for SGML DTDs, they are not a perfect one as we have pointed out in Section 5. One important point is that content models contain only one terminal symbol, #PCDATA, that is equivalent to Σ^*; all other symbols are essentially nonterminals. #PCDATA captures textual data that has no further structure from the viewpoint of the given DTD. (It may, of course, be highly structured for some other application, or for some other DTD which uses the special features of SGML that allow some content to be interpreted as markup.)

One subtle issue is that #PCDATA always includes the null string; there is no positive #PPCDATA, say, that corresponds to Σ^+ and excludes the null string. This issue is crucial when removing inclusion exceptions as we have seen in Section 5.

The elimination of exceptions from an extended context-free grammar introduces new nonterminals. If we apply the exception removal transformation of Fig. 3 to an SGML DTD with exceptions, then we do indeed obtain a new DTD without exceptions that is structurally equivalent to the original DTD. Unfortunately, the document instances of the original DTD do not conform to the new DTD, since the new DTD has new elements and new tags corresponding to these elements that do not appear in the old DTD instances.

Therefore, a natural question is: How useful are our results? First, the results are interesting in their own right as a contribution to the theory of extended context-free grammars and SGML DTDs. We can eliminate exceptions to give structurally equivalent grammars and DTDs while preserving their SGML unambiguity.

Second, during the DTD design phase, it is perhaps convenient to use exceptions. Our results imply we can eliminate the exceptions algorithmically and produce a final DTD design *before any document instances are created.*

Third, the creation of a new DTD without exceptions is useful for producing a DTD database schema as suggested by Christofides and his coworkers [10].

Fourth, and perhaps most important, we can use the exception-freeness transformation to produce XML DTDs from SGML DTDs since XML does not allow exceptions.

Fifth, rather than producing a new DTD, we can emulate it with an extended context-free grammar and its parser. We first apply the exception-removal transformation to the extended context-free grammar with exceptions given by the original DTD with exceptions. We then modify the productions of the resulting exception-free extended context-free grammar to explicitly include old tags. For example, we transform a production of the form:

$$A_{+I-X} \to E_A$$

into a production of the form:

$$A_{+I-X} \to \text{`}< A >\text{'} E_A \text{`}< /A >\text{'},$$

where '$< A >$' and '$< /A >$' $\in \Sigma'$ are the start and end tags that the new grammar has to use as delimiters for the element A. Thus, the new productions can be applied to the old DTD instances.

Lastly, we can attack the document-instance problem head on by translating old instances into new instances. We are planning to investigate a class of DTD-based transductions which would be applicable to the DTD database schema issue raised by Christofides et al. [10] and to the conversion of instances of SGML DTDs into instances of corresponding XML DTDs.

8 Closing remarks

A major open problem is whether we can avoid the exponential worst-case blow-up in the size of a DTD. Is there an exception-removal transformation that does not have exponential blow up in the worst case? As we conjecture that there is no such transformation, a follow-up question is: What do we expect the increase in size to be? In addition, we may be able to avoid the worst-case complexity by constructing a modified DTD on the fly while parsing a document instance. The argument in favor of the on-the-fly approach is that SGML parsers already interpret a DTD on the fly; therefore, it appears that we can avoid the blow up in the size of a DTD, at least in the expected case, by implementing our algorithms locally and on the fly.

Acknowledgements

We would like to thank Anne Brüggemann-Klein and Gaston Gonnet for the discussions that encouraged us to continue our investigation of the exception problem in SGML.

References

[1] J. Albert, D. Giammarresi, and D. Wood. Extended context-free grammars and normal form transformations. In *Automata Implementation: Third International Workshop on Implementing Automata, WIA '98*, Heidelberg, Germany, 1998. Springer-Verlag.

[2] T. Bray, J. Paoli, and C.M. Sperberg-McQueen, editors. *Extensible Markup Language (XML) 1.0.* 1998. W3C Recommendation 10-February-1998. The latest version is available at `http://www.w3.org/TR/REC-xml`.

[3] A. Brüggemann-Klein. Formal models in document processing. Habilitationsscrift, Faculty of Mathematics at the University of Freiburg, 1993.

[4] A. Brüggemann-Klein. Regular expressions into finite automata. *Theoretical Computer Science*, 120:197–213, 1993.

[5] A. Brüggemann-Klein. Unambiguity of extended regular expressions in SGML document grammars. In Th. Lengauer, editor, *Algorithms — ESA 93*, pages 73–84. Springer-Verlag, 1993.

[6] A. Brüggemann-Klein. Compiler-construction tools and techniques for SGML parsers: Difficulties and solutions. Universität Freiburg, Institut für Informatik, May 1994.

[7] A. Brüggemann-Klein and D. Wood. The validation of SGML content models. *Mathematical and Computer Modelling*, 25(4):73–84, February 1997.

[8] A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 142:182–206, 1998.

[9] H.A. Cameron and D. Wood. Structural equivalence of extended context-free and extended E0L grammars. Submitted for publication, 1999.

[10] V. Christofides, S. Christofides, S. Cluet, and M. Scholl. From structured documents to novel query facilities. *SIGMOD Record*, 23(2):313–324, June 1994. (Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data).

[11] J. Clark. Source code for the SGMLS parser. Available from `ftp://ftp.jclark.com/pub/sgmls/`, 1992.

[12] C. F. Goldfarb. *The SGML Handbook*. Clarendon Press, Oxford, 1990.

[13] International Organization for Standardization. *ISO 8879: Information Processing—Text and Office Systems—Standard Generalized Markup Language (SGML)*, October 1986.

[14] P. Kilpeläinen. SGML & XML content models. *Markup Languages: Theory & Practise*, 1999, to appear. Preliminary version available as Department of Computer Science Report C-1998-12, University of Helsinki.

[15] P. Kilpeläinen and D. Wood. SGML and exceptions. In C. Nicholas and D. Wood, editors, *Principles of Document Processing, Proceedings of the Third International Workshop, PODP'96*, pages 39–49. Springer-Verlag, 1997. Preliminary version available as technical report HKUST-CS96-30 of The Hong Kong University of Science & Technology, Department of Computer Science, June 1996.

[16] E. Maler and J. El Andaloussi. *Developing SGML DTDs: From Text to Model to Markup.* Prentice Hall PTR, New Jersey, 1996.

[17] B. Travis and D. Waldt. *The SGML Implementation Guide: A Blueprint for SGML Migration.* Springer Verlag, 1995.

[18] J. Warmer and S. van Egmond. The implementation of the Amsterdam SGML parser. *Electronic Publishing*, 2(2):65–90, July 1989.

[19] D. Wood. *Theory of Computation.* John Wiley & Sons, Inc., 1987.