BIJLAGE O

WG10: Imposing structure upon TEX

—SGML flavored—

Kees van der Laan

March 1991

Introduction

After my SGML-TEX presentation at the fall 90 NTG meeting some e-mail discussion arose.
Ton's reaction to Sake's afterthoughts contains a.o. the statement

> ' . . . LATEX (and TEX) lack the semantic properties to enforce compliance to a structure . . . '

Well, that is not true. Simple counterexamples are: LATEX's title environment, which is rather restricted and another is the shielding of TEX commands from LATEX, but those are elementary redeclarations.

A conclusion of my work was that this area needs more development/research, especially how to add to a format/style in a simple and robust way, compliance enforcing code. From the discussion I understand that that point has been missed completely. I don't know yet whether TEX is capable of providing full generality, whatever that may be.

Imposing structure

In order to make it more clear let us concentrate on some simple examples. Suppose a DTD consists of

- **an element, S(equence),**
  which consists of two elements which have to be used in the given sequence order
  <!ELEMENT S - - (a, b)>
- **an element, O(r),**
  containing a and b, which can be used in any order
  <!ELEMENT O - - (a| b)>

Furthermore, a and b may contain character data <!ELEMENT (a, b) - - CDATA>.
The above means that correctly marked up copy complies with.

```
SGML marked up    TeX marked up
<S><a>a text</a>  \bS\ba a text\ea
   <b>b text</b>     \bb b text\eb
</S>              \eS
and
<O><a>a text</a>  \bO\ba a text\ea
</O>              \eO
as well
<O><b>b text</b>  \bO\bb b text\eb
</O>              \eO
```

Below the Sequence and Or structures are separately elaborated.

Sequence

The idea is to have counters and whenever a or b are entered to increment the `ca`, respectively `cb` counter. Furthermore when b is entered `aprecb` := (ca=1). At the end of S $ca = 1 \wedge cb = 1 \wedge aprecb$ is checked. It is not elegant to have b spoiled with code in order to verify whether a preceded, especially when b is to be used in other structures as well.[1] .
The sequence structure can be imposed as shown by the following TEXing.

```
%<!ELEMENT S -- (a,b)>
%<!ELEMENT (a|b) -- CDATA>
%implemented in TeX via
\newif\ifSenv  \Senvfalse  %S environment
\newif\ifaprecb\aprecbfalse%a precedes b?
\newcount\ca
\newcount\cb
\def\bS{\bgroup\ca0\cb0
              \Senvtrue}%end bS
\def\eS{%Check
%ca=1 and cb =1 and \aprecbtrue,
\ifnum\ca=1
   \ifnum\cb=1
       \ifaprecb %Ok
       \else\errmessage{At end
         Seq, a did not precede b}
       \fi
   \else\errmessage{At end Seq,
              ca=1 and cb<>1}
   \fi
\else
  \errmessage{At end Seq ca<>1}
\fi\egroup}%end eS
\def\ba{\advance\ca1}
\def\ea{}
\def\bb{\ifnum\ca=1 \aprecbtrue\fi
        \advance\cb1}
\def\eb{}
```

[1] Andrew suggested either to use the `\let` mechanism, or even better to take advantage of TEX's stacking. See the appendix.

Or

Again counters are used. At the end of Or $(ca = 1 \wedge cb = 0) \vee (ca = 0 \wedge cb = 1)$ is checked. The TEXing reads.

```
%<!ELEMENT O -- (a|b)>
%<!ELEMENT (a|b) -- CDATA>
%implemented in TeX via
\newcount\ca
\newcount\cb
\def\bO{\bgroup\ca0\cb0}%local cond
\def\eO{%check:
%(ca=1 and cb=0) xor (ca=0 and cb=1)
\ifnum\ca=1
   \ifnum\cb=0 %Ok
   \else\errmessage{Or element
     ended with ca=1 and cb<>0}
   \fi
\else
 \ifnum\ca=0
   \ifnum\cb=1 %Ok
   \else\errmessage{Or element
     ended with ca=0 and cb<>1}
   \fi
 \else\errmessage{Or element
     ended with ca<>0, 1 }
 \fi
\fi
\egroup}%end eO
%
\def\ba{\advance\ca1}
\def\ea{}
\def\bb{\advance\cb1}
\def\eb{}
```

Combinations

Generalization and combination of the above could yield a document structure which might have repeated Or-paths or Sequence-paths: for example Or-Or-Seq-Or-. . . .
Three possibities (a| b| c) can be composed from ((a| b)| c) and similarly three in a row (a, b, c) from ((a, b), c), both are repetitions. The optional occurrence of an element, say d, is just an Or of d with empty, (| d). Check for repetitions ask for appropriate counter checking.

I believe that a format can be build from these elements conforming to an SGML DTD, because a DTD prescribes essentially a tree, with repetitions. Insertions and the like are not considered for the moment.

I don't consider this difficult to elaborate in principle, but I can imagine of already available tools supporting the coding. Furthermore what to do when the copy does not adhere to the syntax is left open. I welcome any suggestions: does regular grammars come in? Is YACC handy?

But, would I really like it? Honestly speaking, NO! Not at all. I just love so much the freedom TEX provides. So why should I continue working in a direction that inhibits that? By the above I can protect myself, if needed, against my own weaknesses. And that is that. However, the idea circulated around and out of the blue Andrew Dobrowolski from ArborTEXt commented the approach as practical unattainable. His idea for using TEX's stack mechanism looks promising. As an appendix I'm happy to add his opinions[2] with respect to the discussion.

Appendix (Andrew's comments)

From: arbortext!aed@sharkey.cc.umich.edu (Andrew Dobrowolski)
Answered 03/11 17:33 by JPC @ADMIN (see #10)

When discussions of using TEX to enforce SGML like content models arise I think it is best to make the following clear right from the beginning:

> Full SGML context checking by TEX is a practical impossibility because TEX lacks the necessary data structures to keep track of arbitrarily complex contexts.

This is not to say that some ingenious TEXperts cannot emulate these structures using macros, but that the resulting code would run slower than my grandmother.

To begin with, we must admit the concession that TEX is not going to read the SGML dtd, that is the SGML declarations which define the allowable document structures and their content models. Any change to the dtd would mean custom work by the TEX programmer.

For a practical solution we will need to restrict the allowable content models that we are trying to enforce. At bare minimum we would have to allow the following:

- Comma groups of unrestricted length. That is a content model of the form (a, b, c, . . .), meaning that this element must contain the element a, followed by the element b, and so on.
- Or groups of unrestricted length. That is a content model of the form (a| b| c| . . .), meaning that this element must contain one and only one of the elements a or b or c or . . . .
- The optional modifier "?", meaning that the preceeding element need not appear. A paper may have an optional sub-title for example.
- The zero or more modifier "*" meaning that the preceeding element may appear zero or more times. An example would be the paragraphs following a chapter title and preceeding the first section.

To begin with we can exclude the less frequently used "&" groups, the one or more modifier "+" and the esoteric inclusions and exclusions. We can even restrict ourselves from mixing comma and or groups within one element's content model.

With these restrictions it would not be so difficult to get some limited context checking to work, assuming good

[2] LATEX edited, adapted to two column format and some use of quoting for emphasizing important ideas.

faith on the part of the user. But I think there is more work to it than first meets the eye. To begin with, if we build context checking into the elements themselves, then we run into the problem that an element may appear in any number of structures (such as a title that may appear in the structures chapter, section, figure, table, extract, etc.) Context checking code for each of these situations would be different, and so the definitions of the most commonly used elements would be the most complex. This is a definite burden on performance. To avoid the burden the TEX programmer could redefine the title (using `\let`) for every context in which it is allowed.

But this is not yet enough to solve the content checking problem. An element may appear two or more times within the same model. For example in following model for the content of the element A, B appears twice: `<!ELEMENT A - - (B, C, B, D)>`. It needs to check that it is followed by a C in the first instance and by a D in the second. Once again, either B gets more complicated or C redefines B using `\let`. This gets very tricky if C is optional: `<!ELEMENT A - - (B, C?, B, D)>`. Which element redefines B if C is missing? If the first B then what if it is also optional?
Further complications appear when an element is allowed to contain itself such as when an emphasis tag is allowed to contain another emphasis tag. In the model `<!ELEMENT B - - (A | B)>` the end of the B element must perform different functions depending on whether it is the inner B or the outer B element. So even assuming simplified unmodifiable sgml content models, I do not think that having context built in to the elements themselves is a viable approach.

I would propose a different attack, one which takes advantage of TEX's stacking. I apologize if this gets too technical.

Define a macro data structure called a model. A model begins with a group type character (one of "," or "|" or "?" or "*") followed by one or more brace delimited groups. Each of these groups contains either a string representing an element name or another model. The topmost model ends with a period. The model would allow for comma groups, 'or' groups, the optional modifiers "?" and the zero or more modifiers "*". For example the model for "A" above may be defined:
`\def\modelA{,{B}{?{C}}{B}{D}.}`
where the definition begins with a (possibly active) character giving the group type (a comma group here) followed by the members of that group. The two group types are "," and "|". While the modifiers are "?" and "*". The final period is used to indicate the end of the model.

The current model is maintained in a local macro called `\cmodel`. As we work through the structures contained in A, the current model is modified to reflect what is still to be expected. `\cmodel` will take on values equivalent to the following definitions:

```
\def\cmodel{,{B}{?{C}}{B}{D}.}
\def\cmodel{,{?{C}}{B}{D}.}
\def\cmodel{,{B}{D}.}
\def\cmodel{,{D}.}
\def\cmodel{,.}
```

Each of the structures within the "A" structure would first check themselves for context and report any context error as required. They would do this by calling a macro `\ccheck` that takes one argument, the name of the calling structure. So "B" would call `\ccheck{B}`, which in turn would examine `\cmodel`. It is possible that `\cmodel` will be modified by this call:
Body (replacement text) `\cmodel`

```
BEFORE \ccheck{B}    AFTER \ccheck{B}
,{B}{?{C}}{B}{D}.    ,{?{C}}{B}{D}.
,{?{C}}{B}{D}.       ,{D}.
,{B}{D}.             ,{D}.
,{D}.                ,{D}.
,.                   ,.
```

The last two calls would also cause error messages along the lines:
Element B is out of context here, expecting D. I will treat the B as an inclusion.

After B's context check operation is over, it starts a new TEX group and redefines `\cmodel` to be `\modelB`. The old `\cmodel` is still on TEX's save stack and will pop back when element B is finished.

It is easy to see how this can be extended to allow for more complex models, including the nesting of comma and or groups and the addition of and groups.

> The beauty of the idea is that the SGML dtd no longer becomes part of the definitons of the macros that make up the document.

Its effect is only seen in the beginning of the macro file in the definitions of `\modelA`, `\modelB`, etc. I will not go into any more details here, since other projects are pressing for my attention, except to say that although I have not implemented such macros, they should not take more than a few hard days of work. If someone attempts to do this, I would not mind hearing about it at TUG (or sooner).

Andrew Dobrowolski
ArborTeXt
aed@arbortext.com