Chart-Parsing Techniques and the Prediction of Valid Editing Moves in Structured Document Authoring

Marc Dymetman
Xerox Research Centre Europe
Grenoble

marc.dymetman@xrce.xerox.com

ABSTRACT

We present an approach to controlled document authoring that significantly extends the functionality of existing methods by allowing bottom-up and top-down specifications to be freely mixed. A finite-state automaton is used to represent the partial, evolving, description of the document during authoring. Using a generalization of chart-parsing techniques to FSAs rather than fixed input strings, we show how the authoring system is able to automatically detect the consequences of the choices already made by the author so as to only propose for the next authoring steps choices which may provably lead to a globally valid document.

We start by considering the case of authoring purely textual documents controlled by a context-free grammar, then show a generalization of this approach to structured documents controlled by a specification whose formal expressive power is at least that of Regular Hedge Grammars (closely related to RELAX NG Schemas) and therefore greater than that of DTDs.

Categories and Subject Descriptors

I.7.1 [**Computing Methodologies**]: Document and Text Processing – *languages and systems, markup languages.*

General Terms

Algorithms, Documentation, Languages.

Keywords

Document authoring tools and systems, XML, computational linguistics, parsing.

DocEng'04, October 28–30, 2004, Milwaukee, Wisconsin, USA.

1. INTRODUCTION

Increasing demands for structured information exchange over the Internet have stimulated the development of such standards as the XML markup language (Bray et al, 2001) and associated document specification languages such as DTDs and Schemas (Thompson et al, 2001). These languages can be used not only for checking the validity of the structure of a document (document validation), but also for guiding an author during the production of this structure (controlled authoring). While DTDs and Schemas, by and large, only focus on the structure of a document as revealed by its tags, some researchers (Ranta 2002; Dymetman et al, 2000) have developed related grammatical formalisms allowing for the controlled authoring, not only of the structure, but also of the syntactical and semantical aspects of the *text* of the document. These "document grammars" can be very precise and prototypes have been developed for authoring detailed documents such as drug leaflets or biological experiment reports (Brun et al, 2000; Brun et al. 2003).

In contrast to validation (Lee et al, 2000), the theory of authoring is under-developed. It presents some specific challenges:

- The author has to interact with an underspecified document description, which "subsumes" the complete document to be produced; each authoring choice should act as an additional constraint that restricts the set of subsumed documents, up to the point where this set is reduced to a singleton;
- At any stage, the underspecified description should remain compatible with the document grammar, in the sense that at least one complete document, valid relative to the grammar, is subsumed by the description;
- Consequences of choices already made should be precomputed so as to guide the author efficiently: (i) the author should only be proposed authoring choices which may lead to a valid document, (ii) choices which cannot be avoided without violating compatibility should be performed automatically.

To our knowledge, all existing controlled authoring tools (e.g. (XMLSPY, Kim 2002)) or models (e.g. (Kuo et al. 2003)) address a limited version of these questions by taking a strictly top-down approach to the authoring problem; the underspecified description is in effect a partial derivation tree in the grammar, and the author is required to specify the upper levels of the derivation before

being allowed to state any knowledge she may have about the lower levels (such as lower-level tags, or words in models allowing for textual specifications). This is a severe practical limitation as it is often the case (e.g. with document grammars for specific technical domains) that a few lower level elements taken together will be more discriminative (and intuitive) than upper-level tags, and may conspire to force much of the whole structure into place.[1] On a more theoretical side, and given the increasing recognition of authoring as a specific modality of grammar use (along with the standard parsing and generation), developing general techniques for representing underspecified documents and interacting with them, under grammatical constraints, may be of some independent interest.

The aim of this paper is to describe a novel, flexible, approach to document authoring, in which top-down and bottom-up specifications can be freely mixed. The approach relies on using a finite state automaton for describing the underspecified document, and on intersecting this automaton with a CFG (context-free grammar) representing the underlying document model. This intersection is done through a chart-based process that is also able to predict, at each authoring step, which choices to present to the author in order to guarantee the global compatibility with choices already made.

The paper is organized as follows. In section 2, we review the basics of chart parsing, and the connections to the problem of intersecting a CFG with a regular language. In section 3, we show how a sequence of more and more specific regular languages can be used for representing the evolving underspecified document description and we use chart parsing for checking compatibility with the underlying CFG; In effect the chart performs the task of tracking the effects (local or non-local) of the author's editing actions; In order to efficiently compute these effects, we introduce a carefully chosen automaton (β-automaton), which, once intersected with the underlying CFG, is able to completely predict which choices should be proposed to the author for the next step. In section 4, we show how allowing in the CFG terminals in the CFG that encode opening and closing tags permits to apply the approach to structured documents, and not only to linear text. We introduce a type of document specification (tag-grammar) that is more expressive than regular hedge grammars (Murata 1999), the formalism on which RELAX NG schemas are based. Finally, in section 5, we explore some remaining problems and promising research directions.

2. CHARTS

2.1 Charts and parsing

Chart-parsing (Kaplan, 1973) is a well-known technique for constructing in polynomial time (in the size of the input string) a compact representation of all the possible context-free analyses of a word string. Let's explain the basic idea through the example of a chart given in Fig. 1.

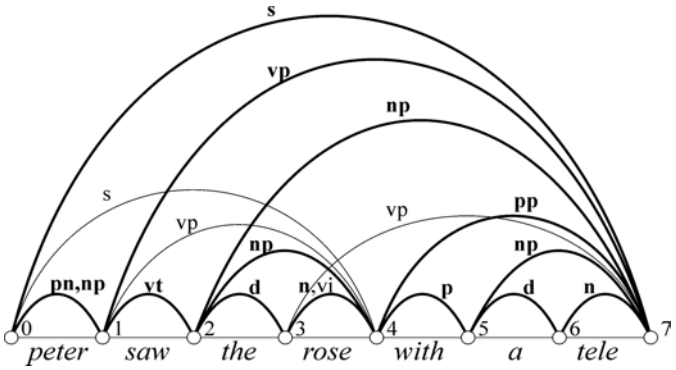

Figure 1: A chart. Reachable symbols are in bold (explained later in the text).

This chart corresponds to parsing the sentence *peter saw the rose with a telescope* relative to the following context-free grammar *CFG0* (terminals in italics):

s → np vp	d → *the*
np → pn	d → *a*
np → d n	n → *rose*
np → d n pp	n → *telescope*
vp → vi	pn → *peter*
vp → vi pp	vi → *rose*
vp → vt np	vt → *saw*
vp → vt np pp	p → *with*
pp → p np	

The chart is constructed according to the following process. We start by introducing nodes 0, ..., 7 for the seven terminals in the input. Then we associate an arc (i-1,i) to the i'th terminal, obtaining the linear graph at the bottom of the figure, consisting of labelled arcs that we will notate `peter(0,1)`, `saw(1,2)`, etc. Then, we progressively build new labelled arcs over the nodes, in the way that we now explain; each time we have a rule in the grammar of the form:

$A \rightarrow B_1\ B_2 \ldots\ B_k$,

and also labelled arcs in the current chart:

$B_1(i_0,i_1)\ B_2(i_1,i_2)\ \ldots\ B_k(i_{k-1},i_k\}$,

then we add to the chart a new labelled arc $A(i_0,i_k)$, unless such an arc already exists, in which case we add nothing. At some point, because there are only finitely many possible labelled arcs to be considered, we reach a situation in which we cannot add anything to the chart.

[1] To take a simple example in the domain of drug leaflets, even a single mention of the word *intravenous* may entail the presence in a valid document of an <injection> structure inside a <drug administration> structure, along with several additional tags.

At the end of this bottom-up process we can decide whether the input string is in the language generated by the grammar by simply checking whether we have produced a symbol `s` spanning the whole input, as we have here with `s(0,7)`. More generally, it can be proven that there is a constituent of category A spanning the input between i and j if and only if a labelled arc A(i,j) has been constructed in the chart.

We may wish to extract a bit more information from the bottom-up process than just recognizing which constituents are associated with which parts of the input. We may want to be able to recover all the different *parses* for the sentence we have just recognized. In order to do that, every time we identify, as previously explained, a sequence of arcs in the chart of the form:

$B_1(i_0,i_1)\ B_2(i_1,i_2)\ \ldots\ B_k(i_{k-1},i_k)$,

we will write down a context-free rule of the form:

$A(i_0,i_k) \rightarrow B_1(i_0,i_1)\ B_2(i_1,i_2)\ \ldots\ B_k(i_{k-1},i_k)$,

where the terminal and nonterminal symbols are now seen as indexed by a pair of nodes `i,j`. We take care of never introducing such a rule if we have already constructed the same rule in a previous step, and it is then easy to see that the process must stop after a finite number of steps. The collection of rules thus constructed constitute a new context-free grammar, called the *specialized CFG associated with the input string* relatively to the original CFG. Fig. 2 shows the rules constructed for the example, organized in stages according to the order in which they are constructed. By convention, the initial nonterminal of the specialized CFG is s07, representing a sentence covering the whole input string.

pn01→peter01; vt12→saw12; d23→the23; n34→rose34; vi34→rose34; **p45→with45; d56→a56; n67→telescope67**
np01→pn01; np24→d23 n34; vp34→vi34; **np57→d56 n67**
vp14→vt12 np24; **pp47→p45 np57**
s04→np01 vp14; **vp17→vt12 np24 pp47; np27→d23 n34 pp47;** vp37→vi34 pp47
vp17→vt12 np27; s07→np01 vp17

Figure 2: Rules associated with the chart. The reachable rules are in bold. The notation pn01 is used as an abbreviation for pn(0,1).

The process for recovering the different parses of the input string is now simple: construct all derivations of s07 relative to the specialized grammar. In our example, there are two such derivations:

```
s07(np01(pn01(peter01)),
   vp17(vt12(saw12),
      np27(d23(the23),
         n34(rose34),
         pp47(p45(with45),np57(d56(a56),n67(telescope67))))))

s07(np01(pn01(peter01)),
   vp17(vt12(saw12),
      np24(d23(the23),n34(rose34)),
      pp47(p45(with45),np57(d56(a56),n67(telescope67)))))
```

which correspond to the ambiguous attachment of the prepositional phrase to a noun phrase or to a verb phrase.[2]

The process of enumerating all the derivations associated with the specialized grammar is a top-down process starting from s07. In this process, certain nonterminals and rules can never be reached, because there do not lie on a path to s07. Such unreachable nonterminals and rules can be eliminated from the specialized grammar without affecting its generated language.[3] The other nonterminals and rules are called *reachable* and are indicated in bold font in Fig. 2; the same convention was applied to the chart in Fig. 1: reachable nonterminals are in bold and thicker lines are for arcs associated with at least one reachable nonterminal.

For instance, we see that the 0-4 arc in Fig. 1 is decorated with a non-reachable s. This means that while "*peter saw the rose*" has the potential of being a s, in the context of the remaining words of the input, it cannot play such a role. Thus the top-down process performs a kind of "contextual inference" which is going to play a central part in the remainder of this paper.

2.2 Charts and automata

While the procedure that we just described was applied to an input consisting of a completely specified string of words, some researchers (Billot and Lang, 1989) have observed that the chart parsing algorithm *only depends on its input being represented as a finite-state automaton*, the case of a string of words being a limit case of such a situation. In fact, looking back at the description we gave of the procedure in the previous section, we see that the indices that we used to characterize the extremities of arcs did not mention *numerical positions* of words in the string (as is usually done in expositions of chart parsing), but instead mentioned *nodes* in an implicit "flat" automaton. Everything we said immediately generalizes to any finite-state automaton labelled with the terminal symbols of the CFG.[4] For instance,

[2] For complex sentences, there may be many more ambiguities, and the number of derivations can be exponential in the length of the input string; however, and crucially for parsing purposes, the specialized grammar can only contain a polynomial number of rules: it is a *compact* representation of all the ambiguities in the input string (Billot and Lang, 1989).

[3] We then obtain what is called in formal language theory a *reduced* grammar, that is, a cfg where all nonterminals can participate in a derivation of a string from s (Harrison, 1978).

[4] In particular, there is no condition that the nodes involved in a chart construction step be consecutive or even different. We may have at some point the two labelled arcs d(0,0) and n(0,0),

even if we consider an automaton with cycles, the bottom-up procedure for constructing arcs must stop because it can never twice introduce the same arc with the same label, and such edges having extremities in the finite set of nodes of the automaton, are themselves in finite number. A similar reasoning applies to the construction of the rules, and the resulting grammar is called the *specialized CFG associated with the automaton*. It can be shown that the language generated by the specialized CFG is actually the intersection between the language generated by the original CFG and the language generated by the automaton.[5] In particular, it can be shown that *this intersection is not empty exactly in the case that there is an s-labelled arc in the chart connecting an initial and a final state of the automaton.*

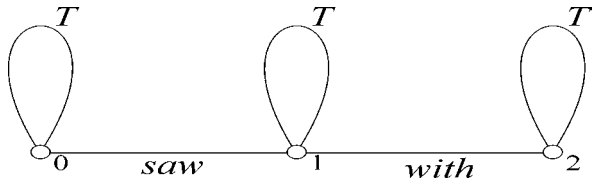

Figure 3: An automaton describing an infinite language. T is the set of terminal symbols in the CFG. By convention we assume that the leftmost and righmost nodes represent the initial and final state of the automaton, and that the arcs are oriented from left to right.

Fig. 3 gives an example of an automaton with cycles, representing an infinite language containing strings of the form σ_1 *saw* σ_2 *with* σ_3, with σ_i in the language T* of all strings constructed over T.

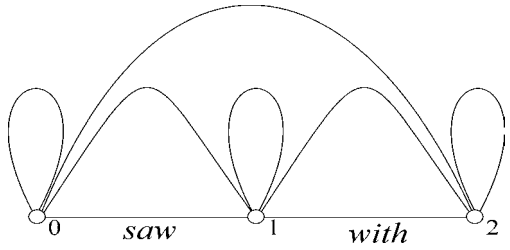

Figure 4: Chart constructed over the previous automaton.

from which we will build the labelled arc np(0,0), along with adding the rule np00→d00 n00 in the specialized CFG.

[5] Bernard Lang has remarked (personal communication) that the chart-parsing procedure actually is a graphical way of reconstructing the original procedure for intersecting context-free grammars and regular languages of (Bar-Hillel, Perles and Shamir, 1961).

0-0	**a, the**, **peter, rose, tele, with,** saw	**d, n, np, p, pn**, **pp**, s, vi, vp, vt
0-1	**saw**	s, vp, **vt**
1-1	**a, the, peter, rose, tele, with,** saw	**d, n, np, p, pn**, **pp**, s, vi, vp, vt
1-2	**with**	**p, pp, np,** vp**,** s
2-2	**a, the, peter, rose, tele, with,** saw	**d, n, np, p, pn**, **pp**, s, vi, vp, vt
0-2		**vp, s**

Figure 5: Labels for the chart. The second column lists the terminals, the third the nonterminals. Reachable terminals and nonterminals are indicated in bold.

In Fig. 4, we show graphically the chart constructed over this automaton, the labels for the arcs being given in Fig. 5 to avoid cluttering the graph. The list of rules that were constructed by the chart procedure are given in Fig. 6. The collection of reachable rules in this table (in bold) constitutes the specialized CFG associated with the automaton of Fig. 3, or in other words, the intersection of *CFG0* with the automaton.

Reading the chart, we see that there is a s-labelled arc connecting the initial node 0 and the final node 2 of the automaton, and therefore we can conclude that the language of the automaton has a non-empty intersection with *CFG0*. We will say that the automaton is *compatible* with *CFG0*.

The top-down filtering mechanism works again exactly as in the string-input case. The only difference is that, now, not only certain nonterminals can be seen as unreachable, but also some terminals, such as saw11 or saw22.[6] These terminals can never participate in a complete derivation from s of a string belonging to the language associated with the automaton of Fig. 3.

d00→the00; **d00→a00**; **n00→rose00**; **n00→telescope00**; **pn00→peter00**; vi00→rose00; vt00→saw00; **p00→with00**; **vt01→saw01**; **d11→the11**; **d11→a11**; **n11→rose11**; **n11→telescope11**; **pn11→peter11**; vi11→rose11; vt11→saw11; **p11→with11**; **p12→with12**; **d23→a23**; **d22→the22**; **d22→a22**; **n22→rose22**; **n22→telescope22**; **pn22→peter22**; vi22→rose22; vt22→saw22; **p22→with22**
np00→pn00; **np00→d00 n00**; vp00→vi00; **np11→pn11**; **np11→d11 n11**; vp11→vi11; **np22→pn22**; **np22→d22 n22**; vp22→vi22
s00→np00 vp00; vp00→vt00 np00; vp01→vt01 np11; **pp00→p00 np00;** s11→np11 vp11; vp11→vt11 np11; **pp11→p11 np11; pp12→p12 np22;** s22→np22 vp22; vp22→vt22 np22; **pp22→p22 np22**

[6] In the input-string case, such a situation never happens when the input string is in the language of *CFG0*, because then any terminal in the string must belong to a derivation from s.

s11→np00 vp01; **np00→d00 n00 pp00;** vp00→vi00 pp00;vp00→vt00 np00 pp00; vp01→vt01 np11 pp11; **vp02→vt01 np11 pp12; np11→d11 n11 p11; np12→d11 n11 pp12;** vp11→vi11 pp11; vp12→vi11 pp12; vp11→vt11 np11 pp11; vp12→vt11 np11 pp12; **np22→d22 n22 pp22;** vp22→vi22 pp22; vp22→vt22 np22 pp22
s02→np00 vp02; vp02→vt01 np12; vp02→vt01 np12 pp22; s12→np12 vp22; s12→np11 vp12; vp12→vt11 np12; vp12→vt11 np12 pp12; **pp12→p11 np12**

Figure 6: Rules associated with the chart, separated according to different construction stages. The reachable rules are in bold.

Reciprocally, any reachable terminal listed in Fig. 5 can participate in such a derivation.

To illustrate by an example, we can traverse the automaton through the steps 0-the-0-rose-0-with-0-a-0-tele-0-saw-1-peter-1-with-2-a-2-tele-2, producing a string of terminals belonging to the language of *CFG0,* therefore it should be the case that the00, rose00, with00, etc., are reachable, which is indeed what the chart asserts to be the case.

3. AUTHORING WITH CHARTS

3.1 α-automata

Let '...' (ellipsis) be a reserved symbol not in T, and let's consider a string τ over elements of T ∪ {'...'}, such that τ does not contain two consecutive occurrences of '...'. To each such τ corresponds a regular language over T, called an *α-language*, namely the language of all strings obtained by replacing each ellipsis in τ by an arbitrary string (possibly the empty string) over T. So for instance, if τ is the expression '... *saw* ... *with* ...', then the corresponding α-language is the set of all strings of the form σ_1 *saw* σ_2 *with* σ_3, with σ_i an arbitrary string in T*. The α-languages play a central role in our approach to authoring: they represent the *partial state of knowledge* that the author has about the evolving document, at any given step of the authoring process.

An α-language can be represented by an automaton such as the one of Fig. 3, called an *α-automaton*, where each ellipsis is accounted for by a circular arc labelled by all elements of T.

3.2 Authoring steps

The document is considered completely described when the α-language representing its current state contains no more ellipsis. When some ellipses remain, the author has to further refine the current specification. First she must choose one of the ellipses in the specification, then she must select one of the following four *refinement moves*:[7]

[7] Left and right refinements can be simulated by a sequence of an inside and an empty refinement, and could be dispensed with, but it is practical to allow the author to perform such moves as a single step rather than two.

[*empty refinement*] Rewrite the ellipsis ... as the empty string ε;

[*inside refinement*] Rewrite the ellipsis ... under the form ... *t* ..., that is, as a new ellipsis followed by a specific terminal symbol *t*, followed by another ellipsis;

[*left refinement*] Rewrite the ellipsis ... under the form t ..., that is, as a terminal symbol followed by an ellipsis;

[*right refinement*] Rewrite the ellipsis ... under the form ... t, that is, as an ellipsis followed by a terminal symbol.

Clearly, by making such moves, the author progresses from the current α-language to a strictly included α-language, up to the point where the last specification contains no ellipsis and defines a single text.

Life and death. We want to maintain a certain crucial invariant during the authoring process, namely, that the current α-automaton should remain *compatible* with the original CFG. If this were not the case, then at the point where we remove the last ellipsis and obtain a complete document, this document would not be in the language of the original CFG, and the author would have to backtrack and suppress some choices already made.

In this respect, let's consider a few potential authoring moves starting from ... *saw* ... *with* ... :

(a) ... *saw* ... *with* ... *saw* ...
(b) ... *saw* ... *rose* ... *with* ...
(c) ... *saw rose* ... *with* ...
(d) ... *saw with* ...

To each of these α-languages corresponds an α-automaton. It is possible to test the compatibility of the automaton with *CFG0* by constructing the chart and checking that an s arc spanning the whole automaton (that is, connecting initial and final states) can be obtained. If this test is performed (not shown), we find that (b) is compatible with *CFG0*, and is said to be *live*, while (a), (c) and (d) are not, and are said to be *dead*.[8]

While the chart construction algorithm gives us here a decision procedure for the life/death problem, it would obviously be rather inconvenient to inform the author of the life or death of a move *only* after she has proposed it. Instead, we would like to *precompute* the live moves so as to only propose those moves to the author. Of course, it would be in principle possible for the system to consider all potential moves and check their liveness before presenting them to the author, but we would like to find a more efficient way of precomputing the live moves.

[8] The possibility of deciding the life/death of an authoring step is in our opinion *the central technical problem faced by controlled authoring in general.* This issue has been previously discussed in another context in (Dymetman, 2002), which uses a more expressive authoring formalism, based on Definite Clause Grammars, and where it is shown that the life/death problem may be decidable or not, in this context, depending on certain conditions on the underlying specifications.

A partial solution to this problem is already suggested by our current considerations. Consider again the α-automaton of Fig. 3 and the chart of Figs. 4 and 5 constructed over it. Note that the chart tells us that "saw" is not reachable on arc 2-2. This implies that we now know, at the point where we have constructed the chart, that (a) is not live, for if it were, it would mean that "saw" could appear *somewhere* to the right of "with". Similarly, the chart tells us that "rose" *is* reachable on arc 1-1, and this means that "rose" can appear *somewhere* between "saw" and "with", which implies that (b) is live. So far, so good. However, (c) cannot be decided on this basis. While the chart tells us that "rose" may appear somewhere between "saw" and "with", it does *not* tell us whether it can appear directly to the right of saw! Similarly (d) cannot be decided on the basis of the chart, for while we know that some words ("a", "the", "peter", "rose", "tele", "with") may appear somewhere between "saw" and "with", and some others may not appear ("saw"), we do not know whether any word actually *has* to appear.

While we therefore do not have a complete solution to our problem, we will now see that we are close, and that, by introducing a slightly more "informative" variant of α-automata, we do get such a solution.

3.3 β-automata and precomputing of choices

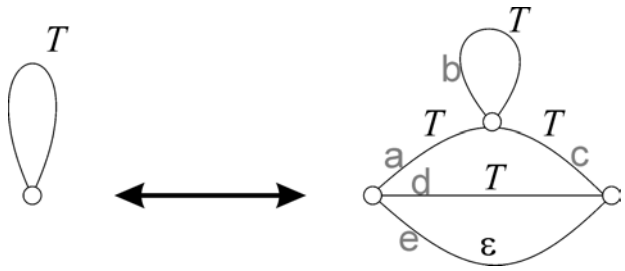

Figure 7: Equivalent automata for the language T*. Letters a, b, c, d, e are for reference to the "rattle" arcs.

Consider the two automata in Fig. 7. These automata both represent the language T* of all strings built over the vocabulary T. This is clear for the left automaton; for the right automaton, which we will refer to under the name *rattle* (by analogy to the baby toy), the language described is the union of the empty string, of the set of strings of length one over T, and of the set of strings of length at least two over T, which is to say, it is equal to T*.

Now consider the automaton, called a β-automaton, shown in Fig. 8. From what has just been said, it is clear that this automaton represents the same α-language as the α-automaton of Fig. 3, namely the language that we abbreviated as ... *saw* ... *with* ... .

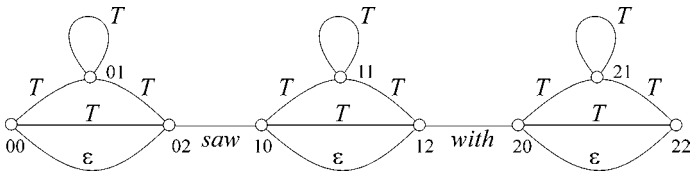

Figure 8: A β-automaton.

The chart algorithm that we have described can be applied to any automaton, and in particular to a β-automaton. *The advantage of using a β-automaton over an α-automaton for our present purpose is that it permits to elicit more informative responses from the chart algorithm regarding the position of reachable terminals*. In particular, it will allow us to decide liveness of the four potential refinements.

For instance, if we apply our chart-computing program to the β-automaton of Fig. 8, we find the following table of *reachable* terminals for the different edges of the three rattles in the automaton (we do not list the obvious reachable arcs, namely 02-10 and 12-20).

	a	b	c	d	e
0	a, the	a,the,rose,tele, with	peter,rose, tele	peter	
1	a,the, peter	a,the,peter,rose, tele,with	peter,rose, tele	peter	
2	a,the, peter	a,the,peter,rose, tele,with	peter,rose, tele	peter	

Figure 9: Reachable arcs relative to the β-automaton of Fig. 8. The numbers 0, 1 and 2 refer to each of the three rattles in Fig. 8, a, b, c, d and e to the arcs in each rattle. Empty entries correspond to unreachable arcs; reachability of the ε-arc is indicated by a '+' in the e column (not used here, but in some later figures).

From this table, we can now easily compute the live refinements of the four types. For instance, for rewriting the third ellipsis in ... *saw* ... *with* ..., an empty refinement (... → ε) is possible iff the arc 20-22 ε is reachable (not the case), a left refinement (... → t ...) iff t is reachable on one of the arcs 20-21 or 20-22 (possible with t equal to *a, the* or *peter*), a right refinement (... → ... t) iff t is reachable on one of the arcs 21-22 or 20-22 (possible with t equal to *peter, rose* or *telescope*), and an inside refinement (... → ... t ...) iff t is reachable on one of the arcs 20-21, 21-21, 21-22 or 20-22 (possible with t equal to *a, the, peter, rose, telescope* or *with*).[9] Similar computations can be done for the first and second ellipsis, and from them we can immediately answer questions about the liveness of such moves as (a), (b), (c), and (d) above, finding that (b) is alive, and (b), (c) and (d) are dead.

[9] Note that left and right refinements are special cases of inside refinements: the liveness of a left or of a right refinement with t implies the liveness of an inside refinement with t.

This procedure, while explained on the basis of an example, can be generalized without any difficulty. Thus the construction of the chart over the β-automaton provides a complete precomputation of all the live refinements, providing a full answer to the life/death problem initially posed.

4. AUTHORING TAGGED DOCUMENTS

4.1 Tag-CFGs

We will now introduce a notion that will allow us to apply the techniques we have just introduced to the world of structured documents and to the XML markup language.

A *tag-CFG* is a context-free grammar such that:

- Its terminal vocabulary contains special terminal symbols, called *opening* and *closing tags*, notated <*tag*> and </*tag*>; These symbols always come in pairs of one opening and the corresponding closing tag;

- Rules in the grammar are of two different types; *non-tag rules* do not have any tag in their rhs; *tag rules* have exactly two tags in their rhs, an opening tag appearing as the first item in the rhs, and a corresponding closing tag appearing as the last element of the rhs.

The following grammar *TCFG0* is a tag-CFG:

s → np vp	d → *the*
np → <*np*> pn </*np*>	d → *a*
np → <*np*> d n </*np*>	n → *rose*
np → <*np*> d n pp </*np*>	n → *telescope*
vp → <*vp*> vi </*vp*>	pn → *peter*
vp → <*vp*> vi pp </*vp*>	vi → *rose*
vp → <*vp*> vt np </*vp*>	vt → *saw*
vp → <*vp*> vt np pp </*vp*>	p → *with*
pp → <*pp*> p np </*pp*>	

This grammar is similar to *CFG0*, apart from the addition of some tags permitting to "materialize" part (but not all) of the constituent structure of the strings generated by the grammar. According to *TCFG0*, such strings as the following one are valid (indentation shown for readability):[10]

```
<np> peter </np>
<vp> saw
 <np> the rose
  <pp> with <np> a telescope </np> </pp>
 </np>
</vp>
```

Tag-CFGs are a simple, yet powerful, formalism, which can be used for describing linear as well as structural aspects of a language. The first point was already noted: they can generate all context-free word languages; As for the second point, let's first note that they are more powerful than XML DTDs. Consider the following tag-CFG (we do not describe the levels below 'book'):

library → <*library*> books </*library*>
books → tbook
books → tbook books
tbook → <*book*> book </*book*>
book → ...

The strings generated by this grammar (for instance <*library*> <*book*> ... </*book*> <*book*> ... </*book*> </*library*>) have well-balanced tags, and can therefore be interpreted as XML structures. The ability of *not* introducing tags in the right-hand sides has the important consequence that the generated *strings* correspond to *structures* that with DTDs would be specified with the help of regular expressions over elements:

<!ELEMENT library (book+)>

In fact, tag-grammars allow to simulate a larger class of tree grammars than allowed by DTDs, namely the class of so-called *regular hedge grammars* (Murata, 1999), on which the RELAX NG schema proposal is based (RELAX NG, 2001).[11,12]

[10] Note that a tag-grammar, being a CFG, is formally a *string grammar*, and not a *tree-grammar*. However the strings it generates do *encode* proper trees, for clearly any string in the language of a tag-grammar only contains well-balanced tags. Thus a tag-grammar also implicitely generates, through the encoding, a tree language. In substance, our chart-based procedures, because they guarantee that a valid tagged string can always be produced starting from the current authoring state, also guarantee that there exists some valid tree in this tree language that is compatible with the current authoring state.

[11] Regular hedge grammars (RHGs) are grammars for generating trees which have productions of the form N → t(RE), where N is a nonterminal, t is a tag, and RE is a regular expression over nonterminals. It is easy to see that any RE in a RHG can be simulated in a tag-grammar through the addition of auxiliary nonterminals in a way similar to the *library* example. This implies that any RHG can be simulated by a tag-grammar; however the converse is not true: some tag-grammars generate languages that go beyond the expressive power of RHGs. To see that, note that there exists a context-free language over a vocabulary of two elements that is not regular (Harrison 1978). It is possible to use such a language to build a tag-grammar that will generate documents of the form <doc> γ </doc> where γ is a sequence of "empty" tags <a/> and <b/> with γ in L, with L context-free but not regular. It is simple to see that such documents cannot be generated by a RHG: in a RHG, the labels of the nodes under a given node (here the node labelled doc) must generate a regular language.

[12] An anonymous referee has pointed us to (Berstel and Boasson, 2000) and (Berstel and Boasson, 2002), which introduce and formally study a notion of extended context-free grammars (*balanced grammars* and the slightly less general *xml grammars*) strongly related to RHGs. These grammars however, rather than being presented as *tree* grammars, as RHGs, are

We will now illustrate the flexibility of chart authoring with tag-CFGs in two domains: interactive disambiguation and structured document authoring.

4.2 Interactive disambiguation

Let's consider the expression:

...0 peter ...1 saw ...2 the ...3 rose ...4 with ...5 a ...6 telescope ...7

where the ellipses (indexed for reference) can be "filled" with an arbitrary sequence of *tag terminals* of *TCFG0*.[13] This expression defines a regular language containing such strings as the example just given in the previous subsection. To the expression we can associate in an obvious way a β-automaton similar to the one of Fig. 8, only with *T* interpreted this time as the set {<np>, </np>, <vp>, </vp>, <pp>, </pp>}, and not as the set of all terminals.

If we compute the table of reachable terminals for the rattles representing the eight ellipses, we find the following table.

	a	b	c	d	e
0				<np>	
1	</np>		<vp>		
2				<np>	
3					+
4	</np>		<pp>	<pp>	
5				<np>	
6					+
7	</np>	</np>, </pp>	</vp>		

Forced moves. We observe here for the first time an interesting phenomenon: there are some *forced moves* for the next authoring steps. The only way to traverse the rattle representing the first ellipsis (*...0*) is through the d arc, with an attached <np> label, and similar considerations apply to *...2* and *...5*, as well as to the ε-arcs in *...3* and *...6* ; In the same way, in *...1* , we have first to traverse the a arc with </np>, immediately followed by a traversal of the b arc with <vp>; in *...7* we have to traverse arc a with with </np>, then *may* traverse arc b with a certain number of occurrences of </np> and </pp>, then must traverse arc c with </vp>; finally in *...4* we have the choice of two traversals of the rattle, but these two traversals end with the same terminal <pp>.

The forced moves just discussed are instances either of forced empty, left, or right refinements (sometimes, as in the move refining the first ellipsis, a forced move is both a left and a right refinement). In general, all forced moves that are either empty, left, or right refinements can be directly determined through inspection of the reachability table for the rattles representing the different ellispses in the current expression (the straighforward algorithm is not detailed).

Taking now the forced moves into account, the system can automatically rewrite the expression as:

<np> peter </np> <vp> saw <np> the rose ...9 <pp> with <np> a telescope </np> ...10 </vp>.

The procedure can be iterated until there are no more forced moves; in the present case, one more iteration is necessary (not shown) and we then find the expression:

<np> peter </np> <vp> saw <np> the rose ...11 <pp> with <np> a telescope </np> </pp> ...12 </vp>

for which we construct the table:

	a	b	c	d	e
11				</np>	+
12				</np>	+

There are now no more forced moves, and control is handed back to the author, who is given the choice either to refine or into </np> or to refine *...12* into ε or into </np>. Refining *...11* into </np> forces a rewriting of *...12* into ε and is tantamount to choosing attachment of the pp to the vp; refining *...11* into ε forces a rewriting of *...12* into </np> and is tantamount to choosing attachment of the pp to the np. Dual considerations apply to refining *...12* first. What we are seeing here is the chart-authoring mechanism making systematic conclusions when it can and giving the hand to the user at points where real structural ambiguities appear.

4.3 Structured document authoring

Let's now go back to the α-language *...0 saw ...1 with ...2* , assuming this time that the relevant terminal vocabulary is the set of all terminals in *TCFG0.* The β-automaton associated with this language is identical with the automaton of Fig. 8, but *T* should now be understood as referring to this new terminal vocabulary.

If we now construct a chart over this automaton, we find the following table of reachable terminals:

presented as *string* grammars, as are our tag-grammars. While RHGs as well as balanced grammars allow regular (rather than finite) RHS's, they require *each* production to have a RHS starting and ending with an opening and a closing tag, contrarily to ours. We have not attempted a precise comparison between tag-grammars and balanced grammars, but we believe that balanced grammars can be represented as tag-grammars, but that the converse is false, as was the case with regular hedge grammars.

[13] We are thus considering here a variant of our previous α-languages, in which only a subset of all terminals is used for the ellipses.

	a	b	c	d	e
0	<np>	<np>, <pp>, </np>, </pp>, a, the, peter, rose, tele, with	<vp>		
1	<np>	<np>, <pp>, </np>, </pp>, a, the, peter, rose, tele, with	<pp>		
2	<np>	<np>,< pp>, </np>, </pp>, a, the, peter, rose, tele, with	</vp>		

from which a certain number of forced choices follow, leading us to the α-language:

<np> ... <vp> saw <np> ... <pp> with <np> ... </vp>.

Iterating the process (not detailed) leads us to one more forced choice, giving the α-language (α-L1):

<np> ...$_3$ </np> <vp> saw <np> ...$_4$ <pp> with <np> ...$_5$ </vp>,

at which point we have the table:

	a	b	c	d	e
3	a, the	<np>, <pp>, </np>, </pp>, a, the, peter, rose, tele, with	</pp>, rose, tele	peter	
4	a, the, peter	<np>, <pp>, </np>, </pp>, a, the, peter, rose, tele, with	</np>, rose, tele		
5	a, the, peter	<np>, <pp>, </np>, </pp>, a, the, peter, rose, tele, with	</np>, </pp>		

At this point no more forced choices exist for refining an ellipsis by an empty, left, or right refinement. For instance, for refining '*...$_3$*' as 't ...', the author could choose t = *a*, *the* or *peter* and for refining '*...$_3$*' as '... t', she could choose t = *</pp>*, *rose*, *telescope* or *peter* (and similarly for the other ellipses). Control is thus given back to the author, who may choose any of the refinements licensed by the table.

Let's note that this approach allows a *free combination of top-down and bottom-up instructions* to the authoring system. In order to emulate a standard top-down system, the author should left-refine an expression of the form '<otag1> ...' into '<otag1> <otag2> ...' (where <otag1> and <otag2> are different opening tags) in order to create a direct child of '<otag1>', or an expression of the form '<ctag1> ...' (where <ctag1> is a closing tag) into '<ctag1> <otag2> ...' in order to create a direct sibling of <ctag1>; on the other hand, by inside-refining an ellipsis into a word or a low-level tag, the author is in effect giving a bottom-up instruction to the system.

5. SOME RESEARCH PERSPECTIVES

5.1 Balanced tag authoring

While the method we have just presented achieves the central goal of always presenting to the author exactly the set of live refinements, it has some deficiencies in regard to informing the author early about *all* the necessary consequences of the current choices.

In particular, in the case of tag-CFGs, it is clear that any valid string must have balanced opening and closing tags, and this may imply that certain ellipses must contain certain inside tags, possibly not found immediately to the right or to the left of a terminal already in place.

To give an example, it can be proven that a necessary consequence of the expression (α-L1) is the following refined α-language (α-L2):

<np>$_1$ ... </np>$_1$ <vp>$_1$ saw <np>$_2$ ... <pp>$_1$ with <np>$_3$ ... </np>$_3$ ... </pp>$_1$ ... </vp>$_1$;

In this expression, two new "inside" tags </np> and </pp> have appeared to the right of with, but in addition, we have indicated through indices the pairing of corresponding opening and closing tags. Such expressions as (α-L2) can be useful during authoring for giving more structured information to the author about the status of the evolving document.

In substance, indicating such pairings renders explicit the *tree* structure of the evolving tags, which is only implicitely represented by their *linear* structure. Despite the fact that the chart-parsing techniques we have described until now are based on this linear structure, and not on an *explicit* representation of a tree structure, we believe that some simple extensions of these techniques exist, and permit to infer that (α-L2) is a necessary consequence of (α-L1).

5.2 Optimization

A naive implementation of the algorithms described so far consists in constructing a new chart from scratch after each refinement step.[14] However, it is clear that, after each such step, there are large portions of the previous chart that are still valid in the new chart, and this fact could be used to avoid many recomputations. One promising way of doing that would be to exploit the view of a chart as a specialized CFG briefly noted in section 2.2: seen in this light each new refinement step consists in specializing the previous CFG one step more through intersection with a new regular constraint.

[14] This is what the current prototype does.

6. CONCLUSION

We have presented a new approach to controlled document authoring which significantly extends the functionality of current systems. The author can state knowledge about the document in a flexible way and concentrate on the most informative pieces first rather than follow a rigid top-down specification order. The system automatically detects the consequences of the choices already made in order to only propose provably live choices for the next authoring steps. The approach is based on a solid theoretical foundation—intersection of a regular language with a CFG, implemented through a chart process—and may provide the basis for further developments in the area of controlled authoring tools.

7. REFERENCES

Bar-Hillel, Y., M. Perles, and E. Shamir. 1961. *On formal properties of simple phrase structure grammars.* Zeitschrift f˙ur Phonetik, SprachWissenschaft und Kommunicationsforschung, 14:143--172. Reprinted in Bar-Hillel's Language and Information -- Selected Essays on their Theory and Application, Addison Wesley series in Logic, 1964, pp. 116-150.

J. Berstel and L. Boasson. *XML grammars*. In N. Nielsen and B. Rovan, editors, Mathematical Foundations of Computer Science (MFCS'2000), volume 1893 of LNCS, pages 182-191. Springer, 2000.

J. Berstel and L. Boasson. *Balanced grammars and their languages*. In Formal and Natural Computing: Essays Dedicated to Grzegorz Rozenberg, LNCS 2300, pages 3-25. Springer, 2002.

S. Billot and B. Lang. *The structure of shared forests in ambiguous parsing.* In 27th Meeting of the ACL, 1989.

T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. 2001. *Extensible Markup Language (XML) 1.0* (Second Edition). Technical report, W3C.

Caroline Brun, Marc Dymetman, and Veronika Lux. Document structure and multilingual text authoring. In *Proceedings of the International Natural Language Generation Conference (INLG-2000)*, Mitzpe Ramon, Israel, 2000

Caroline Brun, Marc Dymetman, Eric Fanchon, Stanislas Lhomme, Sylvain Pogodalla: *Semantically-based text authoring and the concurrent documentation of experimental protocols*. ACM Symposium on Document Engineering 2003: 193-202

Marc Dymetman, Veronika Lux, Aarne Ranta: *XML and Multilingual Document Authoring: Convergent Trends.* COLING 2000: 243-249

Marc Dymetman: *Text Authoring, Knowledge Acquisition and Description Logics*. COLING 2002

M.A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, Mass., 1978.

Kaplan, R.M. 1973. *A general linguistic processor*. In *Natural Language Processing.* Ed. Rustin, R, pages 193-241. Algorithmics Press, New York.

Larry Kim. *The official XMLSPY Handbook*. Wiley, 2002.

Kuo Y. S., Wang. J and Shih N.C.. *Handling syntactic constraints in a DTD compliant XML editor*. In Proceedings of the 2003 ACM symposium on Document engineering. Grenoble, France, 2003.

Guy Lapalme, Caroline Brun et Marc Dymetman. *XML Based Multilingual Authoring.* Dans Vlado Keselj et Tsutomu Endo. (éditeurs), Proceedings of PACLING'03, p. 191-199, Halifax, August 2003.

D. Lee, M. Mani, and M. Murata. 2000. *Reasoning about XML Schema Languages using Formal Language Theory*. Technical Report, IBM Almaden Research Center.

Murata, M.: *Hedge Automata: A Formal Model for XML Schemata*. Technical Report, Fuji Xerox Information Systems (1999)

A. Ranta. 2002. *Grammatical Framework: a type theoretical grammar formalism*. To appear in J. of Functional Programming.

RELAX NG Specification, 2001. http://www.oasis-open.org/committees/relax-ng/spec-20011203.html

Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. 2001. X*ML Schema Part 1: Structures.* Technical report, W3C.