REPRESENTATION TREES AND STRING-TREE CORRESPONDENCES

presented for COLING-88
Budapest, 22-27 August 1988
by

Ch.BOITET & Y.ZAHARIN
GETA, BP 68
Université de Grenoble et CNRS
38402 Saint-Martin-d'Hères, FRANCE

PTMK, Universiti Sains Malaysia
11800 Penang, MALAYSIA

ABSTRACT

The correspondence between *a* string of a language and its abstract representation, usually a (decorated) tree, is not straightforward. However, it is desirable to maintain it, for example to build structured editors for texts written in natural language. As such correspondences must be compositional, we call them "Structured String-Tree Correspondences" (SSTC).

We argue that a SSTC is in fact composed of two interrelated correspondences, one between nodes and substrings, and the other between subtrees and substrings, the substrings being possibly discontinuous in both cases. We then proceed to show how to define a SSTC with a Structural Correspondence Static Grammar (SCSG), and which constraints to put on the rules of the SCSG to get a "natural" SSTC.

Keywords: linguistic descriptors, discontinuous constituents, discontinuous phrase structure grammars, structured string-tree correspondences, structural correspondence static grammars

Abbreviations: DPSG, MT, NL, SSTC, STCG.

INTRODUCTION

Ordered trees, annotated with simple labels or complex "decorations" (property lists), are widely used for representing natural language (NL) utterances. This corresponds to a hierarchical view: the utterance is decomposed into groups and subgroups. When the depth of linguistic analysis is such that a representation in terms of graphs, networks or sets of formulas would be more direct, one often still prefers to use tree structures, at the price of encoding the desired information in the decorations (e.g., by "coindexing" two or more nodes). This is because trees are conceptually and algorithmically easier to manipulate, and also because all usual interpretations based on the linguistic structure are more or less "compositional" in nature.

If a language is described by a classical Phrase Structure Grammar, or by a (projective) Dependency Grammar, the tree structure "contains" the associated string in some easily defined sense. In particular, the surface order of the string is derived from some ordered traversal of the tree (left-to-right order of the leaves of a constituent tree, or infix order for a dependency tree).

However, if one wants to associate "natural" structures to strings, for example abstract trees for programs or predicate-argument structures for NL utterances, this is no longer true. Elements of the string may have been erased, or duplicated, some "discontinuous" groups may have been put together, and the surface order may not be reflected In the tree (e.g., for a normalized representation). Such correspondences must be compositional: the complete tree corresponds to the complete string, then subtrees correspond to substrings, etc. Hence, we call them "Structured String-Tree Correspondences" (SSTC).

For some applications, like classical (batch) Machine Translation (MT), it is not necessary to keep the correspondence explicit: for revising a translation, it is enough to show the correspondence between two sentences or two paragraphs. However, if one wants to build structured editors for texts written 1n natural language, thereby using at the same time a string (the text) and a tree (its representation), it seems necessary to represent explicitly the associated SSTC.

In the first part, we briefly review the types of string-tree correspondences which are implied by the most usual types of tree representations of NL utterances. We argue that a SSTC should in fact be composed of two interrelated correspondences, one between nodes and substrings, and the other between subtrees and substrings, the substrings being possibly discontinuous in both cases. This is presented in more detail in the second part. In the last part, we show how to define a SSTC with a Structural Correspondence Static Grammar (SCSG), and which constraints to put on the rules of the SCSG to get a "natural" SSTC.

I. WHAT IS A CORRESPONDENCE BETWEEN A STRING AND A TREE?

I. PHRASE STRUCTURE TREES (C-STRUCTURES)

Classical Phrase Structure trees give rise to a very simple kind of SSTC. To each string w = al...an, let us associate the set of intervals i_j, 0<i<j<n. w(i_j) denotes the substring ai...aj of w if i<j, 6 otherwise.

The root, or equivalently the whole tree, corresponds to w = w(0_n). Each leaf corresponds to some substring w(i_j), of length 0 or 1 (we may extend this to any length if terminals are allowed to be themselves strings Then, the correspondence is such that any internal node of the tree, or equivalently each tree "complete" in breadth and depth, corresponds to w(i_j), iff its m daughters (or its m immediate subtrees), in order, correspond to a sequence w(i1_j1),...,w(im_jm), such that il = i, jm=j, and jk = ik+l for 0<k<m.

This type of correspondence is "projective". It has however been argued that classical phrase structure trees are inadequate for characterising syntactic representations in general, especially in the case of so-called "discontinuous"constituents. Here are some examples.

- (1) John talked, of course, about politics.
- (2) He picked the ball up.
- (3) Je ne le lui ai pas donné.
  (I did not give it to him)

According to (McCawley 82), sentence (1) contains a verb phrase "talked about politics", which is divided by the adverbial phrase "of course", which modifies the whole sentence, and not only the verbal kernel (or the verbal phrase, in Chomsky's terminology). Sentence (2) contains the particle "up", which is separated from its verb "picked" by "the ball". In sentence (3), the discontinuous negation "ne...pas" overlaps with the composed form of the verb "ai...donné". Moreover, if a sentence in active voice is to be represented in a standard order (subject verb object complement), this sentence contains two displaced elements, namely the object "le" and the complement "lui".

(McCawley 82) and later (Bunt & al 87) have argued that "meaningful" representations of sentences (2) and (3) should be the following phrase structure trees, (4) and (5), respectively.

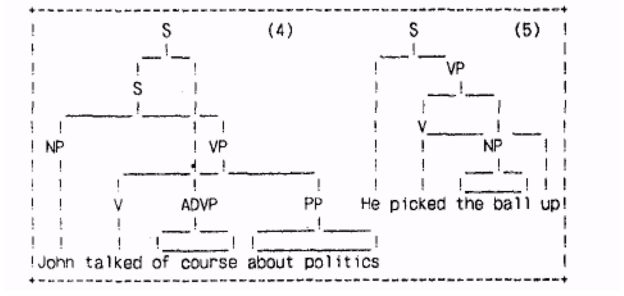
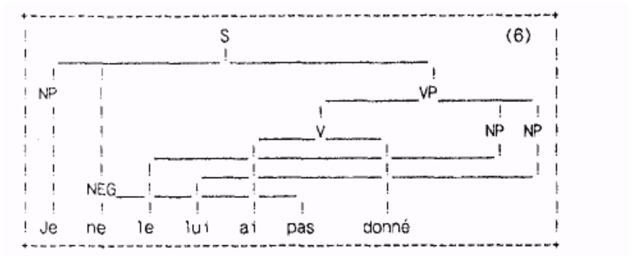


Figure I: Examples of discontinuous phrase structure trees

Along the same line, and taking into consideration the displaced elements, a "meaningful" representation for

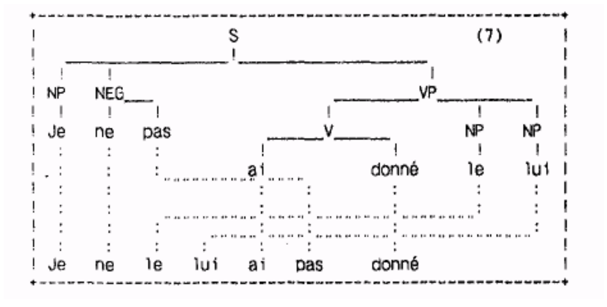


sentence (3) would be tree (6).

Figure 2: Example of discontinuity and displacement

Here, the correspondence is established between a node (or equivalently the complete subtree rooted at *a* node) and a sequence of intervals. If a displacement arises, as in (3), the left-to-right order of nodes in the tree may be incompatible with the order of the corresponding sequences of intervals in the string (the considered ordering is the natural lexicographic extension).

Rather than to introduce the awkward notion of "discontinuous" tree, as above, with intersecting branches, we suggest to keep the tree diagrams 1n their usual form and to show the string separately. For



sentence (3), then, we get the following diagram.

Figure 3: Separation of a string and its "discontinuous" PS tree

Now, as before, the root of the tree still corresponds to w=w(0_n), and a leaf corresponds to an interval of length 0 or 1 (or more, see above). But an Internal node with m daughters corresponds to a sequence of Intervals, which is the "union" of the m sequences corresponding to its daughters.

More precisely, a "sequence" of intervals 1s a list of the form S = w(i1_j1),...,w(ip_jp), in order (1k<1k+1 for 0<k<p) and without overlapping (jk<ik+1 for 0<k<p). Its union (denoted by "+") with an interval I = w(i_j) is the smallest list containing all elements of S and of I. For example, S+I is:

- S itself, if there is a k such that ik≤i and j≤jk;
- S, augmented with w(i_j) inserted in the proper place, if j<il or jp<i or there is a k<p such that jk<i and j<ik+l;
- w(i1_j1),...,w(iq_jq),w(i_jr),...,w{ip_jp), if there are q and r such that jq<i<iq+1 and ir≤j≤jr (other cases are analogous).

2. DEPENDENCY TREES (F-STRUCTURES)

In classical dependency trees, elements of the represented string appear on the nodes of the tree, with no auxiliary symbols, except a "dummy node", often indicated by "+", which serves to separate the left daughters from the right daughters.

There are two aspects 1n the correspondence. First, a node corresponds to an element of the string, usually an Interval of length l. Second, the complete subtree rooted at a node corresponds to the Interval union of the intervals corresponding to the node and to its subtree. These Intervals may not overlap.

The string can be produced from the tree by an in order traversal (one starts from the root, and, at any node, one traverses first the trees rooted at the left daughters, then the node, then the trees rooted at the right daughters, recursively).

Sentences (1) and (2) might be represented by trees (8) and (9) below.

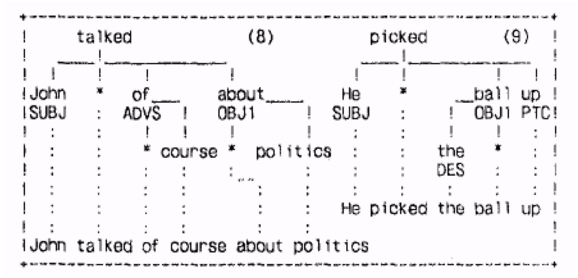


Figure 4: Examples of classical dependency trees

In those trees, the discontinuities shown 1n the PS trees (4) and (5) have disappeared, we have shown on some nodes the syntactic functions usually attached to the edges.

There may be some discussion on the structures produced. For example, some linguists would rather see "politics" dominating "about". This is not our topic here, but we will use this other possibility in a later diagram. For the moment, note that discontinuity does not always disappear in dependency trees. Here 1s an example corresponding to sentence (3).

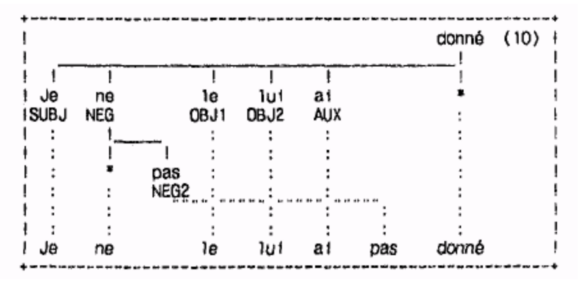


Figure 5: Example of a "discontinuous" dependency tree

Let us now take a simple example from the area of programming languages, which shows an abstract tree associated to an assignment, where some elements of the string are "missing" in the tree, and where a node corresponds to a "discontinuous" substring (a sequence of intervals).

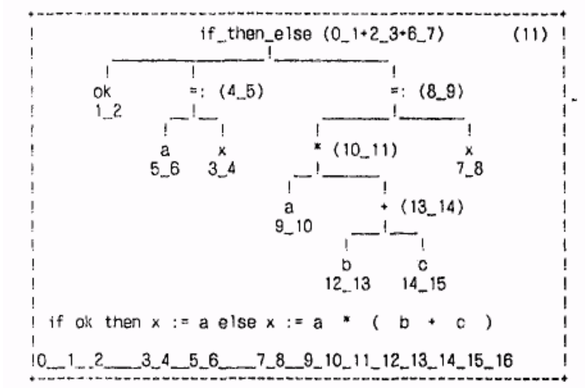


Figure 6: Example of "abstract" tree for a formal language expression

Here, we have shown the correspondence between nodes
and sequences. The parentheses are missing in the tree, which means that the sequence corresponding to the subtree rooted at node "+" is more than the union of the sequences corresponding to its subtrees. However, there is no overlapping between sequences corresponding to independent nodes or subtrees.

Another remark is that the elements appearing on the nodes are not always identical with elements of the represented string. For example, we have replaced ":=" by "=:" and the (discontinuous) substring "if then else" by " if_then_else", in a usual fashion.

3. PREDICATE-ARGUMENT TREES (P-STRUCTURES)

In "predicate-argument structures", it is usual to construct a unique node for a compound predicate, in the same spirit as the "if_then_else" operator above. With sentences (1) and (2), for example, we could get trees (12) and (13) below. Beside the logical relation (argument place) or the semantic relation, the nodes must also contain some other information, like tense, person, etc., which is not shown here.

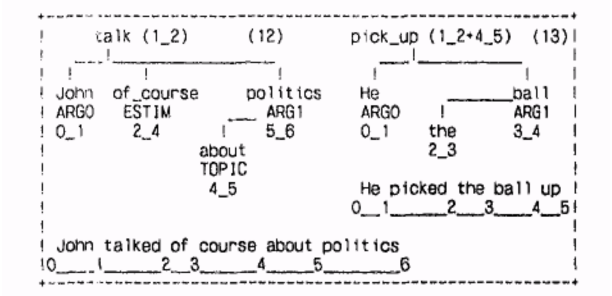


Figure 7; Examples of predicate-argument trees

We now come to situations where overlapping occurs, and where it is natural to consider "Incomplete" subtrees corresponding to "discontinuous" groups.

This occurs frequently 1n cases of coordination with elision, as in:

"John and Mary give Paul and Ann trousers and dresses."

In order to simplify the trees, we abstract this by the formal language {an v bn cn 1 n>0}, and propose the two trees (14) and (15) below for the string "a a v b b c c" (also written a.l a.2 v b.1 b.2 c.1 c..2 to show the positions) as more "natural" representations than the syntactic tree derived from a context-sensitive grammar in normal form for this language (all rules are of the form "1 A r --> 1 u r", 1 and r being the left and right context, respectively).

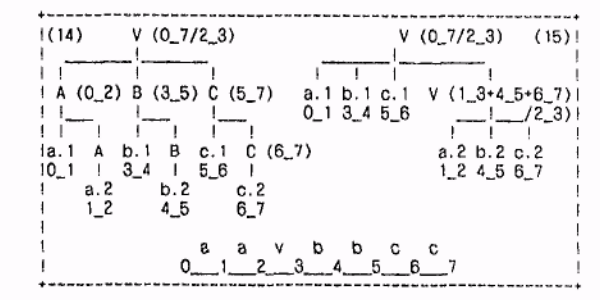


Figure 8: Examples of p-structures for a1 a2 v bl b2 c1 c2

On certain nodes, we have represented the sequence corresponding to the complete subtree rooted at the node, followed by the sequence corresponding to the node itself. For nodes A, B, C in tree (14), this "local" sequence 1s empty.

In both trees, it is clear that the sequence a1 V b1 c1 corresponds to an "incomplete" subtree, namely V(A(a1),B(b1),C(c1)) in (14) and V(al,b1,c1) in (15).

In tree (14), the coordination is shown directly on the graph, and the verb (V) is not shown as elided. It is a matter of further analysis to accept or not the distributive interpretation ("respectively" may hold between the three groups, the last two ones, or nones).

On the contrary, tree (15), in a sense, is a more "abstract" representation. It shows directly the interpretation as a coordination of two sentences, and "restores" the elided V.

4. MULTILEVEL TREES (M-STRUCTURES)

Multilevel tree structures, or m-structures for short, have been introduced by B.VAUQUOIS in 1974 (see (Vauquois 78)) for the purposes of Machine Translation. On the same graph, three "levels of interpretation" are described (constituents, syntactic dependencies, logical and semantic relations). As seen in other examples above, the nodes which refer directly to the string do not contain elements of the string, but rather representatives of (sequences of) elements of the string, called "lexical units" (LU), like "repair" for "reparation", plus some Information about the derivation used.

The graph is deduced by simple rules from a dependency tree: each internal node is "lowered" in the "*" position and its syntactic function becomes "GOV" (for "governor", or head in some other terminology), discontinuous lexical elements (like "ne...pas" or "ai...donné" are represented by one node, coordination 1s represented by "vertical lists" as 1n tree (14), lexical units of referred elements are put 1n the nodes corresponding to the pronouns, an approximation of coindexing, etc..

From the point of view of the associated correspondence between representation trees and represented strings, nothing new has to be mentioned.

II. A PROPOSAL: STRUCTURED STRING-TREE CORRESPONDENCES

Our proposal is now almost complete.

1. DEFINITIONS

a) The correspondence between a string and its representation tree is made of two interrelated correspondences:

- between nodes and (possibly discontinuous) substrings;
- between (possibly incomplete) subtrees and (possibly discontinuous) substrings.

b) It can be encoded on the tree by attaching to each node N two sequences of intervals, called SNODE(N) and STREE(N), such that:

1. SNODE(N) < STREE(N), which means that SNODE(N) is "contained" in STREE(N) with respect to its basic elements (the w(i_j)), that is, that STREE(N) = STREE(N) U SNODE(N). Note that equality can not be required, even on the leaves, because the string "(b)" may well have a representation tree with the unique node b.
2. if N has m daughters Nl...Nm, then STREE(N) > STREE(N1)+...+STREE<Nm) + SNODE(N). In case of strict containment, the difference correspond to the elements of the string which are represented by the subtree but which are not explicitly represented, like "(" and ")" in "(b)".

c) The sequence SSUBT(X,N) corresponding to a given incomplete subtree X rooted at node N of the whole tree T is defined recursively by:

- SSUBT(X,N) = STREE(X) if X = N, that is, if X is reduced to one node, not necessarily a leaf of T;
- SSUBT(X,N) = SSUBT(X1)+...+SSUBT(Xp) U SNODE (N), if N, the root of X, has p subtrees X1...Xp in T.

In other words, one takes the smallest sequence containing the biggest sequence corresponding to the leaves of X(STREE on the leaves) and compatible with the monotony rules above.

2. PROPERTIES

Here are some interesting properties of SSTCs which may help to classify them.

A SSTC is non-overlapping if

- STREE(N1) and STREE(N2) have an empty intersection if Nl and N2 are independent;
- SNODE(N1) and STREE(N2) have an empty intersection if N2 is a daughter of Nl.

A SSTC is projective if

it is non-overlapping;

- for any two sister nodes N1 and N2, Nl to the left of N2, STREE(Nl) is completely to the left of STREE(N2). This means that,
  if STREE(N1) = w(i1_j1)...w(ip_jp) or 0
  and STREE(N2) = w(k1_l1)...w(Kq_lq) or 0,
  then jp<k1.

A SSTC is direct if, for each elementary interval w(i_j+l), there is a node N such that SNODE(N) = w(i_j+1).

A SSTC is complete if each elementary interval 1s contained in SNODE(N) for some nods N.

A SSTC is of the constituent_type if SNODE(N) is empty for each non terminal node N,

3. QUESTIONS OF REPRESENTATION

In the examples above, we have encoded the correspondence in the tree. However, this is in practice not always necessary, or even practical.

In the case of explicit and projective SSTCs, for instance, the string can be obtained directly from the tree, and there is no need to show the intervals.

Note that, in the process of generating a string from a tree, one naturally starts from the top, not knowing the final length of the string, and goes down recursively, dividing this interval into smaller intervals. Rather than to introduce variables representing the extremities of the created intervals, 1t may be more practical to start from a fixed interval, say 0_i or 0_100. Then, the positions between the elements of the string will be denoted by an increasing sequence of rational numbers (0, 1/3, 1/2, 5/7), etc.

In the case of "local" non-projectivity, we have tried some devices using two relative integers (POS,LEV) associated with each node N. POS(N) shows the relative order in the subtree rooted at mother(N), if LEV{N)=0, or more generally at its LEV(N+1) ancestor, if LEV(N)>0. Unfortunately, all these schemes seem to work only for particular situations.

Also, if the SSTC is overlapping, or not complete, it may be computationally costly to find the (smallest) subtree associated with a given (possibly discontinuous) substring. But this operation would be essential in a "structural" editor of NL texts. A possibility is then to encode the correspondence both in the tree and in the string.

Finally, take the example of tree (15) above. Suppose that the user of a NL editor wants to change b1 (Paul, in the corresponding NL example) in a way which may contradict some agreement constraint between a1, V, b1 and c1. One should be able to find the smallest SSTC containing a1 and other elements, that is, the subtree V(a1.b1.c1) and the discontinuous substring a1 v b1 c1 (the notation a._.v.b._.c._. might be suitable, if one wants to avoid indices).

For these reasons, it may be worthwhile to consider the possibility of representing the SSTC independently of both the tree and the string. This is actually the idea behind the formalism of STCG (String-Tree Correspondence Grammar).

III. EXTENDING.THE STCG FORMALISM TO DEFINE A SSTC

1. BASIC NOTIONS ABOUT STCG

The static grammars of (Vauquois & Chappuy 85) are devices to define string-tree correspondences. They have been formalized by the STCGs of (Zaharin 86).

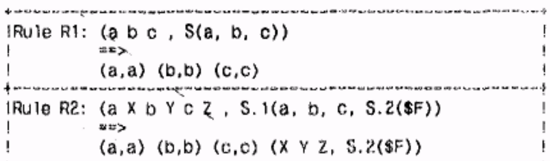
Here, a context-free like apparatus of rules (also called "boards", for "planches" in French, because they are usually written with two-dimensional tree diagrams) is used to construct the set of "legal" SSTCs.

The axioms are all pairs (X,Y($F)), where X is an unbounded string variable, Y a starting node (standing for SENTENCE, or TITLE, for example), and $F is an unbounded forest variable.

The terminals are all pairs (X,X'), where x is an element of a string and x' a one-node tree which represents it.

The rules show how a SSTC 1s made up of smaller ones. The generated language is the set of all variable-free (<str1ng>,<tree>) pairs derivable from an axiom by the grammar rules.

In order to avoid undue formalism, let us give an example for the formal language {an bn cn 1 n>0}.



```
+------------------------------------------------------------+
!Rule R1: (a b c , S(a, b, c))                               !
!         ==>                                                !
!         (a,a) (b,b) (c,c)                                  !
+------------------------------------------------------------+
!Rule R2: (a X b Y c Z , S.1(a, b, c, S.2($F))               !
!         ==>                                                !
!         (a,a) (b,b) (c,c) (X Y Z, S.2($F))                 !
+------------------------------------------------------------+
```

Figure 9: A simple SCSG for an bn cn

X, Y and Z are string variables, $F a forest variable, and the Indices are Just there to distinguish elements with the same label.

Actually, the formalism 1s a bit more precise and powerful, because it is possible to sxpress that a correspondence 1n the r.h.s. (right hand side) is obtained only by certain rules, and to restrict the possible unifications (rather, a special kind called "identifications" in (Zaharin 86)). To illustrate this, we may rewrite the last element of the r.h.s. as:

```
+----------------------------------------------------------+
!                         (X Y Z, S.2($F))                 !
!   with_ref                                               !
!    (R1: X/a, Y/b, Z/c, S.2/S                             !
!    !R2: X/aX, Y/bY, Z/cZ, $F/(a,b,c,S.2($F)))            !
+----------------------------------------------------------+
```

Figure 10 Example of with_ref part in a r.h.s.

R2: X/aX,... means that the subcorrespondence (XYZ,S.2($F)) may be generated by rule R2, thereby identifying X in XYZ with aX in aXbYcZ (in the l.h.s.).

In the version of (Zaharin 86), the correspondence is always of constituent type, because the only applications considered had been to m-structures used for MT, where non-terminal nodes do not directly correspond to substrings.

But this is by no means necessary, as the next example illustrates, with the language {an v bn cn 1 n>0}.

```
+----------------------------------------------------------+
!Rule R1: (a b c , V(a, b, c))                             !
!         ==>                                              !
!         (a,a) (v,V) (b,b) (c,c)                          !
+----------------------------------------------------------+
!Rule R2: (a X v b Y c Z , V.1(a, b, c, V.2($F))           !
!         ==>                                              !
!         (a,a) (v,V.1) (b,b) (c,c) (X v Y Z, V.2($F))     !
! with_ref                                                 !
!   (R1: X/a, Y/b, Z/c, V.2/V, $F/(a,b,c)                  !
!   !R2: X/aX, Y/bY, Z/cZ, V.2/V.1, $F/(a,b,c,V.2($F)))    !
+----------------------------------------------------------+
```

Figure 11: STCG for an v bn cn giving tree (15)

This STCG generates correspondences such as (aavbbcc, tree (15)). But something has to be added to distinguish the STREE and SNODE parts.

2. AN EXTENSION

We simply associate to each constant or variable appearing in a STCG rule one or two expressions representing the STREE and SNODE sequences, separated by a "/" if necessary, with basic elements of the form "p_q", where p and q are constant or variable indices.

In any given (<string>,<tree>) pair, we associate one such expression to each element of <string>, and two to each node of <tree>, the first for STREE and the second for SNODE. The second may be omitted: by default, SNODE is taken to be empty on internal nodes and equal to STREE on leaves.

Our last example may now be rewritten as follows.

```
+--------------------------------------------------------------+
!Rule R1:V                                                     !
!                                 (i0_i2+i3_i4+i5_i6/i1_i2)    !
!  (                                  _______!_______      )   !
!                                    !       !       !         !
!    a     v     b     c   ,      a        b        c          !
!  i0_i1 i1_i2 i3_i4 i5_i6      i0_i1    i3_i4    i5_i6        !
!  ==>                                                         !
!  (a,a), (b,b), (c,c)                                         !
+--------------------------------------------------------------+
!Rule R2:                                                      !
! (  a      X      v      b      Y      c      Z               !
!  i0_i1  i1_i2  i2_i3  i4_i5  i5_i6  i7_i8  i8_i9             !
!                                                              !
!          V.1 (i0_i3+i4_i6+i7_i9/i2_i3)                       !
!     ________!______________________________                  !
!    !        !        !                     !          )      !
!    a        b        c                    V.2                !
!  i0_i1    i4_i5    i7_i8   (i1_i3+i5_i6+i7_i9/i2_i3)         !
!                                            !                 !
!                                            $F                !
!  ==>                                                         !
!  (a,a) (b,b) (v,V.1) (c,c) (X v Y Z , V.2)                   !
!                                        !                     !
!                                        $F                    !
! with_ref                                                     !
!  (R1: X/a, Y/b, Z/c, V.2/V, $F/(a,b,c)                       !
!  !R2: X/aX, Y/bY, Z/cZ, V.2/V.1, $F/(a,b,c,V.2($F)))         !
+--------------------------------------------------------------+
```

Figure 12: Extended STCG for an v bn cn

3. CONSTRAINTS ON STCG

We will now give examples of STCGs which give rise to unnatural correspondences and try to derive some constraints on the rules. Let us first silently modify our first STCG for an bn cn.

```
+----------------------------------------------------------+
!Rule R1: (a b c , S(a, b, c))                             !
!         ==>                                              !
!         (a,a) (b,b) (c,c)                                !
+----------------------------------------------------------+
!Rule R2: (a Z b Y c X , S.1(a, b, c, S.2($F))             !
!         ==>                                              !
!         (a,a) (b,b) (c,c) (X Y Z, S.2($F))               !
+----------------------------------------------------------+
```

Figure 13: Example of "unordered" STCG

In the first element of R2, XYZ has been replaced by ZYX. The following representation tree (16) would have been naturally associated with the string a1.a2.a3.b1.b2.b3.c1.c2.c3 by our first STCG. With this modification, it becomes associated with al.c2.a3.b1.b2.b3.c1.a2.c3, as shown in the next diagram.

```
+--------------------------------------------------------------+
!        S.1 (0_9)                                     (16)    !
!     ____!_____________                                       !
!   !     !     !       !                                      !
! a.1   b.1   c.1      S.2 (1_3+4_6+7_9)                       !
! 0_1   3_4   6_7       !                                      !
!                  _____!______________                        !
!                 !     !      !       !                       !
!                a.2   b.2    c.2     S.3 (2_3+5_6+8_9)        !
!                7_8   4_5    1_2      !                       !
!                                  ____!____                   !
!                                 !    !    !                  !
!                                a.3  b.3  c.3                 !
!                                2_3  5_6  8_9                 !
!                                                              !
!   a1   c2   a3   b1   b2   b3   c1   a2   c3                 !
! 0____1____2____3____4____5____6____7____8____9               !
!                                                              !
+--------------------------------------------------------------+
```

Figure 14: Example of STCG "unordered" w.r.t. the strings

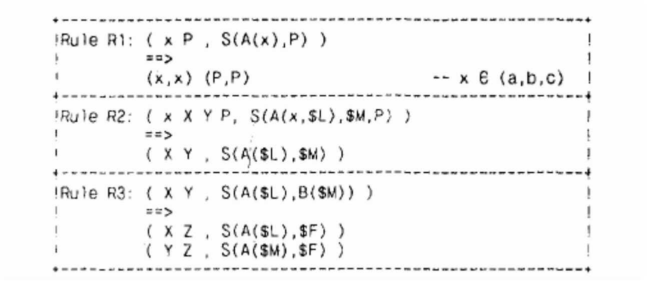
The problem here is that the subtree rooted at S.2, considered as a whole tree, should correspond to the string a2.c3.b2.b3.cZ.a3, and that it corresponds to O2.a3.b2.b3.a2.c3 when embedded in the whole tree rooted at S. 1.

The STREE correspondences are not properly defined, because one should be able to distinguish between different permutations of the intervals, which is clearly impossible with our previous definitions and representations of SSTCs.

This is because the order of the elements of the strings is not compatible in the l.h.s. and in the r.h.s,: our first constraint will be to forbid this in STCG rules.

Our second constraint will be to forbid the use of auxiliary variables which do not correspond to substrings (subtrees) of the terminal (variable-free) pairs produced by the STCG.

Let us illustrate this with the following STCG, which constructs the representation tree S(A(u),B(v)) for each word w on (a,b,c) of even length such that w=uv and Hu=Hv.



```
+-------------------------------------------------------------+
!Rule R1: ( x P , S(A(x),P) )                                  !
!         ==>                                                  !
!         (x,x) (P,P)                          -- x ∈ (a,b,c)  !
+-------------------------------------------------------------+
!Rule R2: ( x X Y P, S(A(x,$L),$M,P) )                         !
!         ==>                                                  !
!         ( X Y , S(A($L),$M) )                                !
+-------------------------------------------------------------+
!Rule R3: ( X Y , S(A($L),B($M)) )                             !
!         ==>                                                  !
!         ( X Z , S(A($L),$F) )                                !
!         ( Y Z , S(A($M),$F) )                                !
+-------------------------------------------------------------+
```

Figure 15: Example of STCG with auxiliary variables

There is a natural SSTC between the representation tree and the string. For example, we get S(A(a,b,c),B(b,a,c)) for w=abcbac. But the construction of this final correspondence involves the construction of pairs such as (abcPPP,S(A(a,b,c),P,P,P)), which are just used for counting.

If we try to put sequence expressions on the P nodes and string elements, we notice that it would be necessary to extend the intervals of w, rather than to divide them. Otherwise, we would make the first P of abcPPP correspond to the second b of w=abcbac, which is quite natural, but what would we associate to the first P of bacPPP ?

If we represent explicitly (and separately) the structure of a given (<string>,<tree>) element of the SSTC by its derivation tree in the STCG, the second constraint will allow us to instantiate all variables by substrings or subtrees of <string> and <tree>, without having to construct other auxiliary strings and trees. This, of course, would permit a more economical implementation, in terms of space.

Finally, note that the interesting properties of SSTCs mentioned in III.1 above have simple expressions as constraints on the rules of our extended STCG formalism.

CONCLUDING REMARKS

Trees have been widely used for the representation of natural language utterances. However, there have been arguments saying that they are not adequate for representing the so-called 'discontinuous' structures. This has led to various solutions, relying, for instance, on encoding the desired information in the nodes (e.g. 'coindexing'), or on defining trees with "discontinuous" constituents.

we have presented here a proposal for representing discontinuous constituents, and, more generally, non-projective and uncomplete SSTCs with overlapping.

The proposal uses the ordinary definition of ordered trees. This is made possible by separating the representation tree from the surface utterance (which the tree is a representation of). The correspondence between the two may be represented explicitly by means of sequences of intervals attached to the nodes.

This opens up a discussion on (and definitions of) structured string-tree correspondences in general. This representation might also be used in syntactic editors for programs or in syntactico-semantic editors for NL texts.

Finally, the formalism of the String-Tree Correspondence Grammar has been extended to give the means of representing the said structured correspondences.

An analogous problem is to define structured correspondences between representation trees, for Instance between source and target Interface structures in transfer-based MT systems. We do not yet know of any satisfactory proposal.

A solution to this problem would give two very Interesting results:

- first, a way to specify structural transfers in a reasoned manner, Just as STCGs are used to specify structural analysers or generators,
- second, a way to put a text and its translation in a very fine-grained correspondence. This is quite easy with word-for-word approaches, of course, and also for approaches using classical (projective) PS trees or dependency trees, but has become quite difficult with more sophisticated approaches using p-structures or m-structures.

REFERENCES

(Bunt & al 87) H.BUNT, J.THESINGH & K. VAN DER SLOOT (1987)
Discontinuous constituents in trees, rules and parsing
Proc. 3rd Conf. ACL European Chapter, Copenhagen, April 1987.

(McCawley 82) J.D. MCCAWLEY (1982)
Parenthetical and discontinuous constituent structure
Linguistic Inquiry 13 (1), 91-106, 1982.

(Vauquois 78) B.VAUQUOIS (1978)
Description de la structure intermédiaire
Communication présentée au colloque de Luxembourg, April 1978, GETA document, Grenoble.

(Vauquois & Boitet 85) B.VAUQUOIS & CH.BOITET (1985)
Automated translation at GETA (Grenoble University)
Computational Linguistics, 11:1, 28-36, January 1985.

(Vauquois & Chappuy 85) B.VAUQUOIS & S.CHAPPUY (1985)
Static Grammars
Proc. Conf. on theoretical & methodological issues in MT, Colgate Univ., Hamilton, N.Y., August 1985.

(Zaharin 86) Y.ZAHARIN (1986)
Strategies and heuristics in the analysis of natural language in Machine Translation
Ph.D. Thesis, Universiti Sains Malaysia, March 1986 (Research conducted under GETA-USM cooperation GETA document, Grenoble.

(Zaharin 87a) Y.ZAHARIN (1987)
String-Tree Correspondence Grammar: a declarative formalism for defining the correspondence between strings of terms and tree structures"
Proc. 3rd Conf. ACL European Chapter, Copenhagen, April 1987.

(Zaharin 87b) Y.ZAHARIN (1987)
The linguistic approach at GETA: a synopsis
the journal TECHNOLOGOS (LISH-CNRS), printemps, 1987, Paris.

(Zajac 86) R.ZAJAC (1986)
SCSL: a linguistic specification language for MT
Proc. of COLING-86, IKS, 393-398, Bonn, August
25-29, 1986.