# Some Properties of the Attribute Value Trees Used for Linguistic Knowledge Representation

Stefan Diaconescu

SOFTWIN Str. Fabrica de Glucoza Nr. 5, Sect.2, 020331 Bucharest, ROMANIA
sdiaconescu@softwin.ro

**Abstract.** AVT (Attribute Value Tree) can be considered a generalization of AVM (Attribute Value Matrix) used in many Feature Structured Unification grammars (FSUG). In this paper we give some properties of the AVT like: paths in an AVT, exclusive combinations in an AVT, logical interpretation of an AVT, well formed AVT, the equivalence of the AVTs, the ordered AVTs, the AVTs intersection, the AVTs difference, the AVT normalization, the AVTs unfiability and the AVTs unification. Using these properties, the AVTs can be used more easily in the morphological description of a language, the morphological analysis of a text, the text annotation, the grammar analysis, the generation process and even in an automatic translation process.

## 1 Introduction

A grammatical analysis of a text usually involve a part of speech (POS) tagging (or text annotation) [9], i.e. the association to each POS of an interpretation that contains a set of morphological categories (MC) with their morphological categories values (MCV). We will consider the morphological class as a special MC. In order to do the annotation we must dispose among other things of a morphological description of the language (that we will name *morphological configurator*). The morphological configurator must contain all the MC/MVC of a language with the links (dependencies) between them. These dependencies can be represented as a tree. On the other hand, when syntax of a language is described, some relations between terminals (POS) and non-terminals NT (syntactic construction) must be detailed. The terminals must have associated some MC/MCV and the NTs must have associated some syntactic categories (SC) with some syntactic category values (SCV). These MC/MCV and SC/SCV can also be represented as a tree. We will give to MC and SC the generic name of attributes (A) and to the MCV and SCV the generic name of attribute value (AV). The trees formed with A/AV will be named Attribute Value Tree (AVT). In order to work with the AVTs in natural language processing we must study the properties of these trees and define some operations that will allow us to use more easily the AVT in the morphological description of a language, the morphological analysis of a text, the text annotation, the grammar analysis, the generation process and even in an automatic translation process. AVTs can be considered a sort of generalization of attribute value matrix (AVM) used in different feature structure unification grammars (FSUG) [6] like Head-driven Phrase Structure Grammar (HPSG) [1], Lexical Functional Grammar (LFG) [7], Categorial Unification Grammar (CUG) [8], etc. The AVT defined in the paper (section 1) is based on AVT formalism used in Generative Dependency Grammars with Features (GDGF) [4]. The sections 2-6 introduce some basic notions and the sections 7-12 describe some basic operations with AVTs. In the section 13 the unsolved problems mentioned in the paper are summarized as items for other developments.

## 2 AVT definition

We will start with an AVT example (see Figure 1). This example is a simplified fragment taken from a grammar description of the Romanian language. An AVT can be used in the morphological description of a language and in the grammar description. In the last case, the AVT can be associated to a terminal/non-terminal (T/NT) and means what combination of attributes and attribute values are acceptable for this T/NT. The combination attributes-values can have quite complicate forms. In a real application, for example in the description of a grammar rule, we can use also the notions of indexed and not indexed attributes. If a T/NT has associated an attribute and this attribute must have the same value in all its apparitions beside the others T/NTs in a certain context (for example in a rule of grammar description) then this attribute will be named indexed attribute. If a T/NT has associated an attribute and this attribute can have any value (from its possible values) in all its apparitions besides the others T/NT in a certain context (for example in a rule of grammar description) then this attribute will be named not indexed attribute. The rules that we can use to build an AVT (in fact a multi attribute value tree or a forrest but we will keep the name of AVT) can be described syntactically as follows (slightly different from [4]):

1.<AVT>->{<label>::<attribute list>}|{<label>}|{<attribute list>}|<attribute list>

2.<attribute list>-><attribute><attribute list>|<attribute>

3.<attribute>-><indexed attribute>|<not indexed attribute>

4.<indexed attribute>->[<attribute content>]

5.<not indexed attribute>->(<attribute content>)

6.<attribute content>-><label>::<feature content>|<feature content>|<label>

7.<feature content>-><attribute name>:<attribute value list>

8.<attribute value list>-><attribute value element >,<attribute value list>|<attribute value element>

9.<attribute value element>-><attribute value name><AVT>|<attribute value name>

The <attribute name> and <attribute value name> and <label> are character stringsdifferent from ":]),". An AVT description can define a label (rule 1, alternant 1) named attribute value sub tree label. This label can be used later in the AVT description (rule 1, alternant 2) and that means that the label can be substituted by its definition (the definition must appear before it is used). An attribute with its list of values can define also a label (rule 6 alternant 1) named attribute label. This label can be used later in the AVT description (rule 6, alternant 3) and that means that the label can be substituted by its definition. The labels defined here are a sort of generalization of reentrancy from HPSG. The loops are not accepted here. These labels can allow describing in a more compact form an AVT. Actually, using this type of description, we can build not only trees but also a forest. Let us have an example of such an AVT description associated to a <subordinated complex group> in a description for a Romanian language grammar (see Figure 1):

{[type : correlative, distributive, logical]

[complex group role :

nominal-attributive { NominalAttributivSubjective Predicative:: [case : nominative, genitive, dative, accusative, vocative] [sequence type : governor { ArticulatePosition:: [articulate : not articulate, definite] [Position:: position against the accordant : left, right]}, subordinate {ArticulatePosition}, governor subordinate governor, governor subordinate, subordinate governor subordinate, subordinate governor]},

subjective-predicative {NominalAttributiveSubjectivePredicative},

verbal-completive [sequence type : governor [Position], subordinate [Position], governor subordinate governor, governor subordinate, subordinate governor subordinate, subordinate governor]]

[animation : not animated, animated]

[gender : masculine, feminine, neuter]

[number : singular, plural]

[person : I, II, III]}

An AVT can also be defined in XML [3] but the above description is more human readable. A software tool can convert an AVT from the above representation to XML in order to be used later by other software tools.

In order to obtain a representation of the AVT that can be used to make operations with these AVTs under the form of the expressions we will give a second definition of the AVT.

Let us have the sets: AS - the attribute set and AVS - the attribute value set. We will note (for $A \in$ AS and $a \in$ AVS):

[A/a] = The fact that the attribute $A$ is indexed (see above).

(A/a) = The fact that the attribute $A$ is not indexed.

$[A/a]^{-1}$ = The fact that $A$ has not associated the value $a$ and any apparition of $A$ will have a value different from $a$ in a certain context. It is named the inverse of [A/a] and is noted also ˜[A/a].

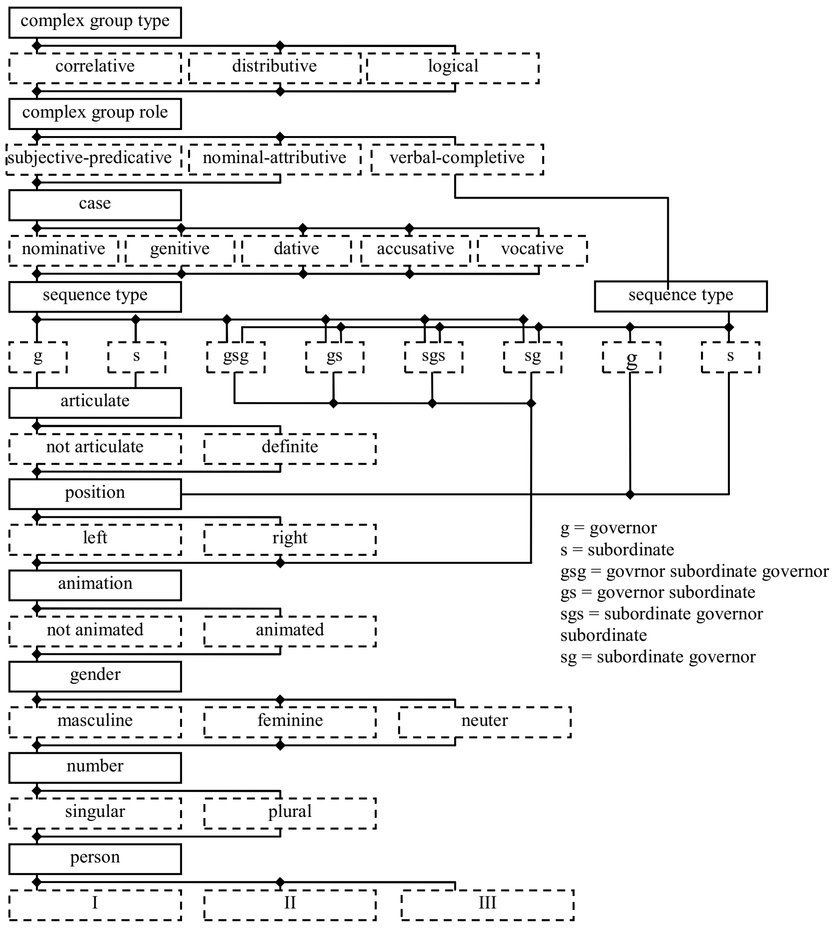

**Fig. 1.** Example of a real compact AVT

$(A/a)^{-1}$ = The fact that $A$ has not associated the value $a$ but in a certain context $A$ can have also others values. It is named the inverse of (A/a) and is noted also ˜(A/a).

Let us have M the set of the elements [A/a] and (A/a) with $A \in$ AS and $a \in$ AVS. Let us have $P$(M) the set of parts of M and two elements noted *false* and *true*.

a) We will define on the set W = $P$(M)∪ {*false*, *true*} an operation noted "*", named concatenation, under the form X * Y = Z where X∈W, Y∈W, Z∈W and if X and Y contains a common attribute with different indexing type, than in Z the not indexed attribute will be indexed. The operation "*" has the properties: commutativity, associativity, idem potency, unitary element (that we note *true*), null element (that we note *false*) and inverse element that we note ˜[A/a$_i$]. A special property is the contradiction: [A/a$_i$]*[A/a$_j$] = *false*, where A∈AS, a$_i$∈AVS, a$_j$∈AVS, a$_i$#a$_j$. The meaning of this property is the following: an attribute can not have in the same time two different values.

b) We will define on the set W = $P$(M)∪ {*false*, *true*} another operation noted "+", named alternative, under the form X + Y = Z where X∈W, Y∈W, Z∈W and if X and Y contains a common attribute with different indexing type, than in Z the not indexed attribute will be indexed. The operation "+" has the properties: commutativity, associativity, idem potency, unitary element (that we note *false*), null element (that we note *true*), inverse element. If if X and Y do not belong to {*true*, *false*} then X and Y has at least one common attribute.

c) We will define now some properties that invlve both "*" and "+":

- The distributivity of "*" against "+": X * (Y + Z) = X * Y + X * Z, where X∈W, Y∈W, Z∈W and Y and Z have at least one common attribute.

- The distributivity of "+" against "*": X + (Y * Z) = (X + Y) * (X + Z), where X∈W, Y∈W, Z∈W and X and Y have at least one common attribute and X and Z have at least a common attribute.

Using this definition we can now work with AVT like with ordinary expressions. We can see that the operations "*" and "+" are very close to logical operations AND and OR. There are only some supplementary restrictions:
i - the contradiction (for "*"); ii - the condition that the two operands of "+" has at least one common attribute; iii - the priority of indexed type over the not indexed type.

## 3 Paths in an AVT

*a) Defining paths in an AVT.* The attribute lists and the attribute value lists are points of branching in an AVT. A sequence of pairs "attribute = attribute value" selected from such branching points will constitute a path in the AVT. The number of this paths can be computed using the recursive formulas from the table 1 where LPN = List Path Number, APN = Attribute Path Number, VPN = Value Path Number.

**Table 1.** Paths in the AVT

| Paths number | Path enumeration |
|---|---|
| $LPN(T) = \sum_k APN(A_k)$ | $LPS(T) = \sum_k APS(A_k)$ |
| $APN(A) = \sum_i VPN(A_i)$ | $APS(A) = \sum_i VPS(A/a_k)$ |
| $VPN(a) = \sum_j APN(A_j)$ or $VPN(a) = 1$ | $VPS(A/a) = \sum_j A/a * APS(A_j)$ or $VPS(A/a) = A/a$ |

*Example 1:* Let us have the tree (see Figure 2):

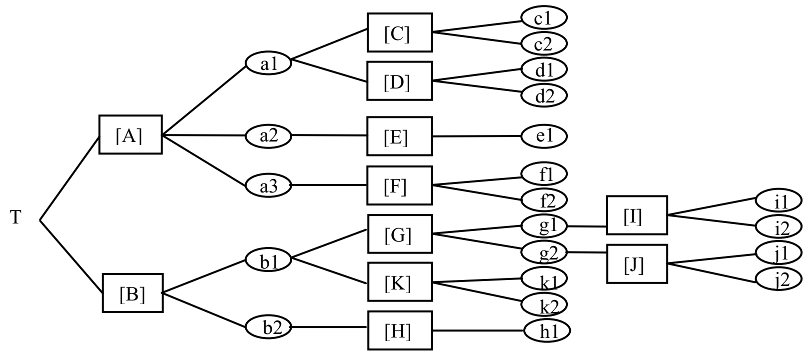

**Fig. 2.** Example of an AVT

T = [A: $a_1$(C: $c_1$, $c_2$) (D: $d_1$, $d_2$), $a_2$[E: $e_1$], $a_3$[F: $f_1$, $f_2$]] [B: $b_1$[G: $g_1$(I: $i_1$, $i_2$), $g_2$(J: $j_1$, $j_2$)] [K: $k_1$, $k_2$], $b_2$[H: $h_1$]]

The number of paths in T will be:

LPN(T) = APN(A) + APN(B) = VPN($a_1$) + VPN($a_2$) + VPN($a_3$) + VPN($b_1$) + VPN($b_2$) = ... = 14

*b) Path enumeration in an AVT.* In order to enumerate (generate) the paths in an AVT we will use the formulas from the table 1 where LPS = List Path Set, APS = Attribute Path Set, VPS = Value Path Set and where "+" and "*" are analogue with the operators defined in the section 2.

Observation: LPS(T), APS(T) and VPS(T) are sets but with a special representation.

*Example 2:* We will consider the tree from the example 1. The paths in T will be:

LPS(T) = APS([A]) + APS([B])= VPS([A/$a_1$]) + VPS([A/$a_2$]) + VPS ([A/$a_3$]) + VPS([B/$b_1$]) + VPS([B/$b_2$]) = [A/$a_1$] * APS([C]) + [A/$a_1$] * APS([D]) + [A/$a_2$] * APS([E]) + [A/$a_3$] * APS([F]) + [B/$b_1$] * APS([G]) + [B/$b_1$] * APS([K]) + [B/$b_2$] * APS([H]) = ... = [A/$a_1$] * [C/$c_1$] + [A/$a_1$] * [C/$c_2$] + [A/$a_1$] * [D/$d_1$] + [A/$a_1$] * [D/$d_2$] + [A/$a_2$] * [E/$e_1$] + [A/$a_3$] * [F/$f_1$] + [A/$a_3$] * [F/$f_2$] + [B/$b_1$] * [G/$g_1$] * [I/$i_1$] +[B/$b_1$] * [G/$g_1$] * [I/$i_2$] + [B/$b_1$] * [G/$g_2$] * [J/$j_1$] + [B/$b_1$] * [G/$g_2$] * [J/$j_2$] + [B/$b_1$] * [K/$k_1$] + [B/$b_1$] * [K/$k_2$] + [B/$b_2$] * [H/$h_1$]

## 4 Exclusive combinations in an AVT

*a) Defining the exclusive combinations.* In a grammar description, an AVT can be associated for example with an NT. A list of attribute values reflects the fact that a terminal (or a sequence of terminals) from the text this NT refer to can have one or another of these values. How many terminals (sequences of terminals) can reflect an AVT? Or, i.e., how many combinations can be built with the attributes and their values taken from N AVT but respecting the AVT structure? Such combinations are named ”exclusive combinations” (EC) because a terminal (sequence of terminals) recognized in the source text will have for each attribute only one value. We will give here a method to count and a method to enumerate these ECs.

The number of ECs in an AVT can be computed with the recursive formulas from the table 2 (see [4]) where LEN = List Exclusive Number, AEN = Attribute Exclusive Number, VEN = Value Exclusive Number.

**Table 2.** Exclusive combinations in the AVT

| Paths number | Path enumeration |
|---|---|
| $LEN(T) = \prod_k AEN(A_k)$ | $LES(T) = \prod_k AES(A_k)$ |
| $AEN(A) = \sum_i VEN(A_i)$ | $AES(A) = \sum_i VES(A/a_k)$ |
| $VEN(a) = \prod_j AEN(A_j)$<br>or<br>$VEN(a) = 1$ | $VES(A/a) = A/a * \prod_j ES(A_j)$<br>or<br>$VES(A/a) = A/a$ |

*Example 1:* We will consider the tree from the section 3, example 1. The number of ECs from T will be: LEN(T) = AEN(A) * AEN(B) = (VEN($a_1$) + VEN($a_2$) + VEN($a_3$)) * (VEN($b_1$) + VEN($b_2$)) = ... = 63

b) ECs enumeration in an AVT. In order to enumerate (generate) the ECs in an AVT we will use the formulas from the table 2 where LES = List Exclusive Set, AES = Attribute Exclusive Set, VES = Value Exclusive Set and where ”+” and ”*” are defined in the section 2.

Observation: LES(T), AES(T) and VES(T) can be considered as sets but with a special representation.

This form of the AVT can be considered a logical representation of the AVT. On this form we can make different transformations, simplifications that are requested by the treatment of a grammar that contains the AVT. We will present these operations in the following sections. Finally, the AVT can be brought to the initial representation using the method presented in [4].

*Example 2:* Let us have the tree:

$T_1$ = [a: $a_1$[b: $b_1$], $a_2$[b: $b_1$] , $a_1$[d: $d_1$]][c: $c_1$[b: $b_2$]]

Because the attribute *b* appears many times we will mark differently each apparition. The value *a1* of the attribute *a* from the list of the higher level has two apparition so we will mark differently its apparitions.

$T_1$ = [a: $a_1$[b: $b_1$], $a_2$[b’: $b_1$] , $a_1$’[d: $d_1$]][c: $c_1$[b”: $b_2$]]

The EC set will be:

LES($T_1$) = AES([a]) * AES([c]) = (VES([a/$a_1$]) + VES([a/$a_2$]) + VES ([a/$a_1$']) * VES([c/$c_1$]) = ... = [a/$a_1$]* [b/$b_1$] * [c/$c_1$] * [b"/$b_2$] + [a/$a_2$] * [b'/$b_1$] * [c/$c_1$] * [b"/$b_2$] + a/$a_1$'] * [d/$d_1$] * [c/$c_1$] * [b"/$b_2$]

We delete the markers:

LES(T1) = [a/$a_1$] * [b/$b_1$] * [c/$c_1$] * [b/$b_2$] + [a/$a_2$] * [b/$b_1$] * [c/$c_1$] * [b/$b_2$] + [a/$a_1$] * [d/$d_1$] * [c/$c_1$] * [b/$b_2$]

But [b/$b_1$] * [b/$b_2$] = *false* so:

LES(T1) = [a/$a_1$] * [b/$b_2$] * [c/$c_1$] * [d/$d_1$]

## 5 The equivalence of the AVTs

Let us have T1 and T2 two AVTs. Let LES(S) and LES(T) be the corresponding minimal EC sets. If LES(S) and LES(T) contain the same ECs (eventually in different order) then they are said to be equivalent and we note LES(S) = LES(T). The equivalence is transitive.

*Example:* We will show that the following AVTs are equivalent.

$T_1$ = [a: $a_1$[b: $b_1$, $b_2$], $a_2$[b: $b_1$, $b_2$]]

$T_2$ = [b: $b_1$[a: $a_1$, $a_2$], $b_2$[a: $a_1$, $a_2$]]

We transform $T_1$. We mark differently the identical attributes:

$T_1$ = [a: $a_1$[b: $b_1$, $b_2$], $a_2$[b': $b_1$, $b_2$]]

The EC of $T_1$ will be:

LES($T_1$) = AES([a]) = VES([a/$a_1$]) + VES([a/$a_2$]) = [a/$a_1$] * AES([b]) + [a/$a_2$] * AES([b']) = [a/$a_1$] * (VES([b/$b_1$]) + VES([b/$b_2$])) + [a/$a_2$] * (VES([b'/$b_1$]) + VES([b'/$b_2$])) = [a/$a_1$] * ([b/$b_1$] + [b/$b_2$]) + [a/$a_2$] * ([b'/$b_1$] + [b'/$b_2$])

We delete the markers:

LES($T_1$) = [a/$a_1$] * ([b/$b_1$] + [b/$b_2$]) + [a/$a_2$] * ([b/$b_1$] + [b/$b_2$]) = [a/$a_1$] * [b/$b_1$] + [a/$a_1$] * [b/$b_2$] + [a/$a_2$] * [b/$b_1$] + [a/$a_2$] * [b/$b_2$]

We transform now $T_2$. We mark differently the identical attributes:

$T_2$ = [b: $b_1$[a: $a_1$, $a_2$], $b_2$[a': $a_1$, $a_2$]]

The EC of $T_2$ will be:

LES($T_2$) = AES(b) = VES(b/$b_1$) + VES(b/$b_2$) = [b/$b_1$] * AES(a) + [b/$b_2$] * AES(a') = [b/$b_1$] * (VES(a/$a_1$) + VES(a/$a_2$)) + [b/$b_2$] * (VES(a'/$a_1$) + VES(a'/$a_2$)) = [b/$b_1$] * ([a/$a_1$] + [a/$a_2$]) + [b/$b_2$] * ([a'/$a_1$] + [a'/$a_2$])

We delete the markers:

LES($T_2$) = [b/$b_1$] * ([a/$a_1$] + [a/$a_2$]) + [b/$b_2$] * ([a/$a_1$] + [a/$a_2$]) = [a/$a_1$] * [b/$b_1$] + [a/$a_1$] * [b/$b_2$] + [a/$a_2$] * [b/$b_1$] + [a/$a_2$] * [b/$b_2$]

We can see now that LES($T_1$) = LES($T_2$)

## 6 Well formed AVT

In a real grammar description we will use generally only well formed AVT. A well formed AVT is an AVT that respects the following rules:

*Rule (a):* An attribute can not appear twice in the same EC.

*Rule (b):* An attribute has the same indexing type in all its apparitions in different ECs.

*Rule (c):* There are not two ECs that have the same elements.

*Rule (d):* An attribute value can not appear twice in the same attribute value list.

We say that the attribute lists of a well formed AVT are well formed too.

We will give here some of the properties of a minimal well formed AVT.

*Property (a):* An attribute list does not contain twice the same attribute.

This can be demonstrated observing that if an attribute list contains twice an attribute than these apparitions will generate some exclusive combinations that will contain twice the same attribute therefore the rule (a) will be infringed.

*Property (b):* A path does not contain twice the same attribute.

This can be demonstrated observing that if the (b) property is infringed than the rule (a) is infringed.

*Property (c):* Let us have an attribute $A$ and a value of $A$ in a certain point of a well formed AVT. There is a set of pats that pass by $A$. These paths pass also through different attribute lists. All these attribute lists do not contain others apparitions of $A$.

This can be demonstrated observing that if the property (c) is infringed, the rule (a) is infringed.

For a grammar description or for morphological descriptions only the well formed AVT are useful so, if we do not make special indications, than the used AVT are only well formed AVTs.

If an AVT is not well formed that there some possibility to bring them to a well formed AVT but we do not insist on this topic.

## 7 Ordering the AVTs

We introduce a total order relation on the set of attributes that belong to an AVT, for example the alphabetical order of the attributes names. If we have an AVT T, we want to build an equivalent T' where each path is a sequence of ordered attributes and all the paths are alphabetical ordered between themselves. We will name T' an ordered AVT.

There is an algorithm that can obtain T' from T (but we have not enough space here to present this algorithm). To order an AVT and to transform this AVT in a well formed AVT can take some time. But these operations are made only once (at the grammar "compilation", the grammar having AVT associated with each NT for example). Later, when the grammar is used for example to make the syntactic analysis of a text, the work operations with well formed and ordered AVT will be much faster.

Observations: The ordering operation refers to paths in the AVT and well formed form refers to ECs. An ordered AVT will allow the generation of the

ECs that will be also ordered (i.e. the attributes followed by their values in an EC will be ordered and the EC will be ordered between themselves).

*Example 1:* Let us have the following AVT T:

T = [K: $k_1$[G: $g_1$[H: $h_1$, $h_2$]][H: $h_1$[C: $c_1$[H: h1]][D: $d_1$[K: $k_1$]]], $k_2$[E: $e_1$[I: $i_1$, $i_2$][J: $j_1$, $j_2$], $e_2$][B: $b_1$[A: $a_1$, $a_2$], $b_2$[A: $a_1$, $a_2$]], $k_3$[H: $h_1$, $h_2$][A: $a_1$, $a_2$]]

The equivalent ordered AVT T' will be:

T' = [A: $a_1$[B: $b_1$[K: $k_2$], $b_2$[K: $k_2$]][K: $k_3$], $a_2$[B: $b_1$[K: $k_2$], $b_2$[K: $k_2$]], $a_3$[K: $k_3$]] [C: $c_1$[H: $h_1$[K: $k_1$]]][D: $d_1$[H: $h_1$[K: $k_1$]]] [E: $e_1$[I: $i_1$[K: $k_2$], $i_2$[K: $k_2$]][J: $j_1$[K: $k_2$], $j_2$[K: $k_2$]], $e_2$[K: $k_2$]] [G: $g_1$[H: $h_1$[K: $k_1$], $h_2$[K: $k_1$]]][H: $h_1$[K: $k_3$], $h_2$[K: $k_3$]]

## 8 The equivalence of well formed and ordered AVT

For well formed and ordered AVTs we can give a recursive definition of the equivalence.

Let us have an attribute list S that contains the attributes $s_1$, $s_2$, $s_3$, ... $s_i$, ..., $s_n$, each $s_i$ (i = 1, 2, 3, ..., n) having the values $s_{i,k}$ (k = 1, 2, 3, ..., $n_i$). Let us have too an attribute list T that contains the elements $t_1$, $t_2$, $t_3$, ..., $t_j$, ..., $t_m$, each $t_j$ (j = 1, 2, 3, ..., m), having the values $t_{j,r}$ (r = 1, 2, 3, ..., $m_j$). S and T are equivalent iff:

(for any $s_i$ (i = 1, 2, 3, ..., n) and any $s_{i,k}$ (k = 1, 2, 3, ..., $n_i$)

there is at least a $t_j = s_i$ (j = 1, 2, 3, ..., m) that has at least one value $t_{j,r} = s_{i,k}$ (r = 1, 2, 3, ..., $m_j$)) so that

($s_{i,k}$ and $t_{j,r}$ has not associated attribute lists) or ($s_{i,k}$ and $t_{j,r}$, has associated equivalent attribute lists))

and (for any $t_j$ (j = 1, 2, 3, ..., m) and any $t_{j,r}$ (r = 1, 2, 3, ..., $m_j$) there is at least an $s_i = t_j$ (i = 1, 2, 3, ..., n) having at least one value $s_{i,k} = t_{j,r}$ (k = 1, 2, 3, ..., $n_i$)) so that

($t_{j,r}$ and $s_{i,k}$ have not associated attribute lists) or ($t_{j,r}$ and $s_{i,k}$ have associated equivalent attribute lists)).

The fact that S and T are equivalent in these conditions should be demonstrated.

This property (of well formed and ordered AVT) will allow a much more faster checking of the equivalence because it not entail the generation of all the EC like in the definition from the section 6.

## 9 AVT intersection, difference and union

Let us have two AVTs, S and T with the corresponding exclusive combinations LES(S) and LES(T).

The intersection S∩T, the difference S\I and union S∪T have the definition and the properties from the set theory applied on the EC sets from LES(S) and LES(T) (with the restriction that the union can be applied only if S and T have a common attribute).

If S and T are well formed then their intersection, difference and union is well formed too. This should be demonstrated

Algorithms to compute S∩T, S\T and S∪T without enumerating the ECs should be found. Anyway, these algorithms must act on ordered trees.

*Example:* Let us have two AVTs, S and T:

S = [a: $a_1$, $a_2$, $a_3$][b: $b_1$, $b_2$[c: $c_1$, $c_2$], $b_3$[d: $d_1$, $d_2$[e: $e_1$, $e_2$], $d_3$[f: $f_1$]], $b_4$][f: $f_1$, $f_2$[g: $g_1$, $g_2$], $f_3$][x: $x_1$, $x_2$[y: $y_1$, $y_2$], $x_3$, $x_4$[y: $y_3$, $y_4$], $x_5$]

T = [b: $b_2$ [c: $c_2$, $c_3$], $b_3$ [d: $d_2$ [e: $e_1$, $e_2$, $e_3$], $d_3$[f: $f_2$], $d_4$], $b_5$][c: $c_1$[d: $d_1$, $d_2$], $c_2$, $c_3$][f: $f_1$, $f_2$[g: $g_1$, $g_2$], $f_3$][x: $x_1$, $x_3$ [y: $y_1$, $y_2$], $x_4$, $x_5$[y: $y_3$, $y_4$], $x_6$]

We will have:

I = S∩T = [b: $b_2$[c: $c_2$], $b_3$[d: $d_2$[e: $e_1$, $e_2$]]][f: $f_1$, $f_2$[g: $g_1$, $g_2$], $f_3$][x: $x_1$]

D1 = S\I = [a: $a_1$, $a_2$, $a_3$][b: $b_1$, $b_2$[c: $c_1$], $b_3$[d: $d_1$, $d_3$[f: $f_1$]], $b_4$][x: $x_2$[y: $y_1$, $y_2$], $x_3$, $x_4$[y: $y_3$, $y_4$], $x_5$]

D2 = T\I = [b: $b_2$[c: $c_3$], $b_3$[d: $d_2$[e: $e_3$], $d_3$[f: $f_2$], $d_4$], $b_5$][c: $c_1$[d: $d_1$, $d_2$], $c_2$, $c_3$][x: $x_3$[y: $y_1$, $y_2$], $x_4$, $x_5$[y: $y_3$, $y_4$], $x_6$]

U = S∪T = [a: $a_1$, $a_2$, $a_3$][b: $b_1$, $b_2$[c: $c_1$, $c_2$, $c_3$], $b_3$[d: $d_1$, $d_2$[e: $e_1$, $e_2$, $e_3$], $d_3$[f: $f_1$, $f_2$], $d_4$], $b_4$, $b_5$][c: $c_1$[d: $d_1$, $d_2$], $c_2$, $c_3$][f: $f_1$, $f_2$[g: $g_1$, $g_2$], $f_3$][x: $x_1$, $x_2$[y: $y_1$, $y_2$], $x_3$[y: $y_1$, $y_2$], $x_4$[y: $y_3$, $y_4$], $x_5$[y: $y_3$, $y_4$], $x_6$]

## 10 AVT normalization

We will present now a very important point, the normalization of the AVT that allow the simplification of the AVT.

*a) Attribute factoring.* We will define first of all the attribute factoring. Let us have a well formed and ordered AVT L that has at the first level an attribute list. Let us have an attribute *A* of this list. The attribute *A* has the values $v_i$ followed each one by an attribute list $S_i$:

L = { ... [A: $v_1S_1$, $v_2S_2$, $v_3S_3$, ..., $v_iS_i$, ..., $v_nS_n$] ... }

We note L' the attribute list formed by L but without the attribute *A*:

L' = L\[A: $v_1S_1$, $v_2S_2$, $v_3S_3$, , $v_iS_i$, , $v_nS_n$]

We note S the intersection of all the lists $S_i$ (i = 1, 2, 3, , n):

S = $S_1$∩$S_2$∩$S_3$∩ ... ∩$S_i$ ... ∩$S_n$

The intersection is realized step by step from left to right on $S_1$, $S_2$, $S_3$, ..., $S_i$, ..., $S_n$, so finally S will be ordered too.

We make the differences:

$S'_i$ = $S_i$\S (i = 1, 2, 3, , n).

During these operations we take care not to change the order of the elements from $S_i$ that remain in $S'_i$. We build a new list $L_1$:

$L_1$ = L'∪S∪[A: $v_1S'_1$, $v_2S'_2$, $v_3S'_3$, ..., $v_iS'_i$, ..., $v_nS'_n$]

$L_1$ is formed by an attribute list and each attribute list is ordered. We order the attributes from the list $L_1$ on their names at the highest level. We obtain an ordered $L_1$ too. If in $L_1$ appear two or many equivalent attributes, then we keep only one sample from each equivalent attribute group. L1 will be well formed too.

We will say that $L_1$ is the factoring of L. $L_1$ is equivalent with L (this should be demonstrated). We see that a well formed and ordered AVT remain well formed and ordered after the factoring.

The factoring has the important property that by factoring we obtain an AVT with a path number less then or equal with the initial path number (this should be demonstrated).

*Example 1:* Let us have the AVT L:

L = [a: $a_1$[c: $c_1$, $c_2$][d: $d_1$, $d_2$][e: $e_1$, $e_2$][f: $f_1$], $a_2$[d: $d_1$, $d_2$][c: $c_1$, $c_2$][f: $f_1$], $a_3$[c: $c_1$, $c_2$][f: $f_1$]][b: $b_1$[c: $c_1$, $c_2$][f: $f_1$], $b_2$[c: $c_1$, $c_2$]]

We analyze for example the attribute a.

$S_1$ = [c: $c_1$, $c_2$][d: $d_1$, $d_2$][e: $e_1$, $e_2$][f: $f_1$]

$S_2$ = [d: $d_1$, $d_1$][c: $c_1$, $c_2$][f: $f_1$]

$S_3$ = [c: $c_1$, $c_2$][f: $f_1$]

L' = L\[a: $a_1S_1$, $a_2S_2$, $a_3S_3$] = [b: $b_1$[c: $c_1$, $c_2$][f: $f_1$], $b_2$[c: $c_1$, $c_2$]]

S = $S_1 \cap S_2 \cap S_3$ = [c: $c_1$, $c_2$][f: $f_1$]

S'$_1$= $S_1$\S = [d: $d_1$, $d_2$][e: $e_1$, $e_2$]

S'$_2$ = $S_2$\S = [d: $d_1$, $d_2$]

S'$_3$ = $S_3$\S = *false* (i.e. this list does not exists)

$L_1$ = L'∪S∪[a: $a_1$S'$_1$, $a_2$S'$_2$, $a_3$S'$_3$] = [a: $a_1$[d: $d_1$, $d_2$][e: $e_1$, $e_2$], $a_2$[d: $d_1$, $d_2$], $a_3$][b: $b_1$[c: $c_1$, $c_2$][f: $f_1$], $b_2$[c: $c_1$, $c_2$]][c: $c_1$, $c_2$][f: $f_1$]

We see that L has 20 paths and L1 has 16 paths.

*b) Normalized AVT.* We say that a well formed and ordered AVT is normalized if it can not support any more a new factoring.

Only one normalized AVT correspond to a well formed and ordered tree (this should be demonstrated).

*Example 2:* We continue the factoring of L1 from the above example 1. We analyze now the attribute b.

$S_1$ = [c: $c_1$, $c_2$][f: $f_1$]

$S_2$ = [c: $c_1$, $c_2$]]

$L_1$' = $L_1$\[b: $b_1S_1$, $b_2S_2$] = [a: $a_1$[d: $d_1$, $d_2$][e: $e_1$, $e_2$], $a_2$[d: $d_1$, $d_2$], $a_3$][c: $c_1$, $c_2$][f: $f_1$]

S = $S_1 \cap S_2$ = [c: $c_1$, $c_2$]

S'$_1$ = $S_1$\S = [f: $f_1$]

S'$_2$ = $S_2$\S = *false*

$L_2$ = $L_1$'∪S∪[b: $b_1$S'$_1$, $b_2$S'$_2$] = [a: $a_1$[d: $d_1$, $d_2$][e: $e_1$, $e_2$], $a_2$[d: $d_1$, $d_2$], $a_3$][b: $b_1$[f: $f_1$], $b_2$][c: $c_1$, $c_2$][f: $f_1$]

The path number is now 12.

*c) AVT with attributes that have only one value.* The normalized form of such an AVT is formed by a single list of attributes with single values (this should be demonstrated). This property is useful, for example, when we apply a rule in a GDGF grammar.

*Example 3:* If we make the label substitutions and the factoring on the example from the section 2, we will obtain the following normalized form:

[type = correlative, distributive, logical]

[animation = not animated, animated]
[gender = masculine, feminine, neuter]
[number = singular, plural]
[person = I, II, III]
[complex group role=

nominal-attributive [case: nominative, genitive, dative, accusative, vocative] [sequence type: governor [articulate: not articulate, definite] [position against the accordant: left, right], subordinate [articulate: not articulate, definite] [position against the accordant: left, right], governor subordinate governor, governor subordinate, subordinate governor subordinate, subordinate governor],

subjective-predicative [case: nominative, genitive, dative, accusative, vocative] [sequence type: governor [articulate: not articulate, definite] [position against the accordant: left, right], subordinate [articulate: not articulate, definite] [position against the accordant: left, right], governor subordinate governor, governor subordinate, subordinate governor subordinate, subordinate governor],

verbal-completive [sequence type: governor [position against the accordant: left, right], subordinate [position against the accordant: left, right], governor subordinate governor, governor subordinate, subordinate governor subordinate, subordinate governor] ]

The path number decreased from 13824 to 52.

## 11 AVT Unfiability

There are many situations when we must check if two AVTs are unifiable, for example, in GDGF, when we want to see if an NT from the right side of a rule $R_1$ can be substituted with the right side of a rule $R_2$. In this situation we must check if the NT is unifiable with the NT from the left side of $R_2$.

Let us have two well formed, ordered and normalized AVTs, S and T. We will consider S and T as two attribute lists. The recursive definition of the unifiability of S and T is the following:

*a) Attribute list unification.* A well formed, ordered and normalized attribute list S : $s_1$, $s_2$, $s_3$, ..., $s_n$ is unifiable with a well formed, ordered and normalized attribute list T: $t_1$, $t_2$, $t_3$, ..., $t_m$ iff:

- All the attributes $s_i$ (i = 1, 2, 3, ..., n) that are found among the attributes $t_j$ (j = 1, 2, 3, ..., m) are unifiable with the corresponding attributes $t_j$ (when we say that "the attribute $s_i$ (i = 1, 2, 3,..., n) is found among the attributes $t_j$" we mean that for $s_i$ there is a $t_j$ so $s_i$ and $t_j$ have the same name and we are not interested if these attributes has the same values or not and if these values are followed by attribute lists or not).

- The attributes from the first level list of S (i.e. $s_1$, $s_2$, $s_3$, ..., $s_n$) are found at most among the attributes from the first level attribute list of T (i.e. they are found among $t_1$, $t_2$, $t_3$, ..., $t_m$ ) and they are not found on the paths that starts from $t_1$, $t_2$, $t_3$, ..., $t_m$). If for a $s_i$ (i = 1, 2, 3, ..., n) we find a $t_j$ (j = 1, 2, 3, ..., m) so $s_i = t_j$ then $s_i$ and $t_j$ have the same indexed type.

- The attributes from the first level list of T (i.e. $t_1$, $t_2$, $t_3$, ..., $t_n$) are found at most among the attributes from the first level attribute list of S (i.e. they are found among $s_1$, $s_2$, $s_3$, ..., $s_m$) and they are not found on the paths that starts from $s_1$, $s_2$, $s_3$, ..., $s_m$). If for a $t_j$ (j = 1, 2, 3, ..., n) we find a $s_i$ (i = 1, 2, 3, ..., m) so $t_j = s_i$ then $t_j$ and $s_i$ have the same indexed type.

*b) Attribute unification.* An attribute $A$ having the name $a$ and the values $a_1$, $a_2$, $a_3$, ..., $a_p$ and that belong to a well formed ordered and normalized list is unifiable with an attribute $B$ having the name $b$ and the values $b_1$, $b_2$, $b_3$, ..., $b_q$ if the following condition is satisfied:

($A$ and $B$ have the same indexed type)

and (all the values ai (i = 1, 2, 3, ..., p) are found among the values $b_j$ (j = 1, 2, 3, ..., q); we will note $x_k$ (k = 1, 2, 3, ..., r) these $a_i$ values and $y_k$ (k = 1, 2, 3, ..., r) the corresponding $b_j$ values)

and ($A$ is not found among the attributes of the paths from $B$ that start from $y_k$)

and ($B$ is not found among the attributes of the paths from $A$ that start from $x_k$)

and ((at least one of $a_i$ and $b_j$ are not followed by attribute lists) or(both $a_i$ and $b_j$ are followed by unifiable lists of attributes respective $S_i$ and $T_j$ ))

We can see that the unifiability is commutative but not transitive property.

*Example 1:* The following S and T AVTs are unifiable:

S = [a: $a_1$, $a_2$, $a_3$, $a_4$][b: $b_1$, $b_2$, $b_3$][c: $c_1$, $c_2$[f: $f_1$][g: $g_1$, $g_2$]][d: $d_1$, $d_2$[f: $f_1$, $f_2$][g: $g_3$, $g_4$]]

T = [c: $c_1$[h: $h_1$], $c_2$, $c_3$][d: $d_1$, $d_2$[g: $g_3$, $g_4$, $g_5$]][e: $e_1$, $e_2$, $e_3$, $e_4$]

*Example 2:* The following S and T AVTs are not unifiable:

S = [a: $a_1$, $a_2$, $a_3$, $a_4$][b: $b_1$, $b_2$, $b_3$][c: $c_1$, $c_2$[f: $f_1$][g: $g_1$, $g_2$]][d: $d_1$, $d_2$[f: $f_1$, $f_2$][g: $g_1$, $g_3$, $g_4$]]

T = [c: $c_1$[h: $h_1$], $c_2$, $c_3$][d: $d_1$, $d_2$[g: $g_4$, $g_5$]][e: $e_1$, $e_2$, $e_3$, $e_4$]

S is not unifiable with T because the attribute [d: $d_1$, $d_2$[f: $f_1$, $f_2$][g: $g_3$, $g_4$]] is not unifiable with [d: $d_1$, $d_2$[g: $g_4$, $g_5$]]. These two attributes are not unifiable because the lists [f: $f_1$, $f_2$][g: $g_3$, $g_4$] and [g: $g_4$, $g_5$] are not unifiable. These two lists are not unifiable because the attributes [g: $g_3$, $g_4$] and [g: $g_4$, $g_5$] are not unifiable. And these two attributes are not unifiable at least because the value $g_3$ from the first value list is not found among the values of $g$ from the second attribute.

## 12 Unification

Let us have two AVTs, S and T with the corresponding exclusive combinations LES(S) and LES(T). To unify S and T means to find the unifier of S and T. The unifier of S and T is (according to [4]) the AVT obtained from LES(S)

* LES(T). To compute the unifier we need to enumerate all the ECs of S and T and this can be a hard-working task. We will give a more complicated but faster method to build the unifier:

*a) The unifier of two attributes lists.* The unifier of two unifiable, well formed, ordered and normalized attribute lists S: $s_1$, $s_2$, $s_3$, ..., $s_n$ and T: $t_1$, $t_2$, $t_3$, ..., $t_m$ , is a list R: $r_1$, $r_2$, $r_3$, ..., $r_v$ obtained as follows:

- We insert in R all the attributes from S (first level) that are not found among the attributes from T (first level) and all the attributes from T (first level) that are not found among the attributes from S (first level). The insertion is made respecting the order relation for the attribute names.

- For each attribute $s_i$ from S that is common with an attribute $t_j$ from T we insert in R the unifier of the attributes $s_i$ and $t_j$;

*b) The unifier of two attributes.* Let us have an attribute $A$ with the name $a$ and the values $a_1$, $a_2$, $a_3$, ..., $a_p$ and an attribute $B$ with the name $b$ and the values $b_1$, $b_2$, $b_3$, ..., $b_q$. We suppose that $A$ and $B$ belong to some well formed, ordered and normalized attribute lists. We suppose too that $A$ and $B$ are unifiable. The unifier of $A$ with $B$ is an attribute $C$ with the name $c$ and the values $c_1$, $c_2$, $c_3$, ..., $c_w$ that are obtained doing for each $a_i$ that has a $b_j$ with the same name (for each $a_i$ must exist a $b_j$ with the same name because $A$ and $B$ are unifiable) the following operations:

- If neither $a_i$ nor $b_j$ are followed by an attribute list, then $a_i$ is inserted in $C$.

- If $a_i$ is not followed by an attribute list but $b_j$ is followed by the attribute list $T_j$, then $b_j$ followed by the list $T_j$ is inserted in $C$.

- If $a_i$ is followed by an attribute list $S_i$ but $b_j$ is not followed by an attribute list, then $a_i$ followed by the list $S_i$ is inserted in $C$.

- If $a_i$ is followed by the attribute list $S_i$ and $b_j$ is followed by the attribute list $T_j$, then $a_i$ (that has the same name with $b_j$) is inserted in $C$ followed by the unifier of the list $S_i$ with the list $T_j$.

(All the above insertions are made respecting the order relation for the attribute value names.)

The unifier will be also a well formed AVT (this should be demonstrated). We will note the unifier of S with T by S∘T.

*Example:* The unifier of the AVTs S and T from the example 1 of the previous section is:

R = S∘T = [a: $a_1$, $a_2$, $a_3$, $a_4$][b: $b_1$, $b_2$, $b_3$][c: $c_1$[h: $h_1$], $c_2$[f: $f_1$][g: $g_1$, $g_2$]][d: $d_1$, $d_2$[f: $f_2$, $f_3$][g: $g_3$, $g_4$]][e: $e_1$, $e_2$, $e_3$, $e_4$]

## 13 Conclusions

We presented the most important properties of the AVTs: paths, ECs, equivalence, well formed AVT, ordering, well formed and ordered AVT equivalence, intersection, difference, union, normalization, unfiability and AVT unification. Some of these properties can be viewed as operations acting on the ECs but usually the EC number of an AVT can be too great. We presented some much faster methods that avoid the EC enumeration: factoring, normalization, unfiability and unification. Other methods to avoid the EC enumeration should be found, for example for S∩T, S\T, S∪T. There are a set of statement that we presented without demonstration like: the fact that S∩T, S\T, S∪T and S∘T are well formed if S and T are well formed, the fact that if two AVT satisfy the recursive definition of the equivalence, they are equivalent. The demonstrations are not difficult so the reader is invited to exercise these demonstrations. Some other development should be presented (perhaps in other future papers): how to bring not well formed AVTs to well formed AVT, how to order an AVT.

All these operations can involve much computing time but with an appropriate software tool it should be no problem to implement them in a natural language processing system.

The presented elements of the AVT theory are implemented in a dedicated language: Grammar Abstract Language (GRAALAN). The using of this language in Romanian grammar description is now in the process.

## References

1. Pollard, C. Sag, I. A. Head-driven Phrase Structure grammar, University of Chicago Press and Standford CSLI Publications, 1994.
2. Shieber, Stuart M. An Introduction to Unification Based Approaches to Grammar, CLSII Lecture Notes Series, Number 4, Center for the study of Language and Information, Stanford University, 1986.
3. Diaconescu, Stefan (2003) Morphological Categorization Attribute Value Tree and XML, in Mieczyslav A. Klopotek, Slawomir T. Wierzchon, Krzysztof Trojanowski (Eds), Proceedings of the International IIS: IIPWM'03 Conference, Zakopane, Poland, June 2-5, Springer 131-138.
4. Diaconescu, Stefan (2002) Natural Language Understanding Using Generative Dependency Grammar, in Max Bramer, Alun Preece and Frans Coenen (Eds), in Proceedings of ES2002, the 21-nd SGAI International Conference on Knowledge Based Systems and Applied Artificial Intelligence, Cambridge UK, Springer, 439-452.
5. Diaconescu, Stefan (2003) Natural Language Agreement Description for Reversible Grammars, in Proceedings of 16th Australian Joint Conference on Artificial Intelligence University of Western Australia, Perth 2-5 December
6. Blackburn, Patrick and Striegnitz, Kristina (2002) NLP Techniques in Prolog, Vers. 1.2.4, http://www.coli.uni-sb.de/ kris/nlp-with-prolog/html/index.html
7. Yehuda N. Falk (2001) LEXICAL-FUNCTIONAL GRAMMAR An Introduction to Parallel Constraint-Based Syntax, CSLI Publications
8. Uszkoreit, H. (1986) Categorial Unification Grammars, Proceedings of COLING 1985, Bonn, 187-194
9. Tufis, Dan (1998), Tiered Tagging, in International Journal on Information Science and Technology, vol. 1, no. 2, Editura Academiei, Bucharest, 1998