Natural Language Agreement Description for Reversible Grammars

Stefan Diaconescu[1]

SOFTWIN, Str. Fabrica de glucoza, Nr. 5, Sect.2, Bucharest, Romania
sdiaconescu@softwin.ro

Abstract. The paper presents a very general method to describe the agreement in the natural languages. The method can be used in automatic translation. It allows the analysis of a text and then the generation of the text in target language so it can be embeded in the reversible grammars. It allows also in the case of analysis the treatment of the agreement errors. The method is based on a set of simple expressions having logical values. We define a tetravalent logic to work with these expressions and that can be implemented in an automatic treatment. The agreement expressions are transformed in a normal form that can be used in both text analysis and text generation.

1 Introduction

It is already well known the agreement definition given by Steele: "The term agreement commonly refers to some systematic covariance between a semantic or formal property of one element and a formal property of another [20]. But the attitude concerning the agreement is quite contradictory: starting from a lack of interest [13] in the case of the very low inflected languages (like English) until a special interest [9][18][9][10][14] in the case of high inflected languages [16].

The agreement description we will present is a declarative one.The declarative grammars (like different types of constraint grammars [15][12]) are suitable to be used as reversible grammars [17]. The parsing (syntactic analysis) and the generation can be realized using the same reversible grammar [21].

The syntactic analysis allows to determine a deep structure of a surface text. This deep structure contains syntactic elements (terminals that have associated some categories with some values) and relations between these elements. In the course of the analysis, the agreement rules must indicate how the combination of the syntactic categories and values must be associated with the syntactic elements (considering the relations that exist between these syntactic elements).

The generation allows obtaining a surface text using the deep structure. The deep structure can be obtained during a process of machine translation or by the inference process based on a data base or knowledge base. During the generation, the agreement rules must force some values of the syntactic categories associated to some elements in order to agree with others elements.

There are are three types of using reversible grammars [5]: 1 - the parsing and the generation are done by the same interpreter using one grammar; 2 -

the parsing and the generation are done by different interpreters using the same grammar; 3 - the parsing and the generation are done by two different programs that use two different compiling of the same grammar.

We consider that the agrement description we will present (section 2) can be used for all three modes. This agreement description that can be used by human is handled with a special tetravalent logic (section 3) and is transformed in a normal form (section 4) that is better for automatic treatment. The agreement (under its normal form) is used in the syntactic analysis and text generation so it is particularly suitable for reversible grammars (section 5).

2 A Way to Represent the Grammatical Agreement

We will consider that the term agreement will refer not only to the relation between words but also to the relation between syntactic notions (non terminals) that are used in grammar descriptions. According to the above definition (section 1) the agreement is a sort of matching between different elements that appear in a text.

We will consider that two parts are implied in such a matching: a controller C and a target T and we note the agreement by C <- T. In our notation, the arrow look to the controller (and not to the target, like in [10][9]). The controller can be simple C = c (formed by one element) or multiple C = c_1, c_2, c_3 ... (formed by two or many elements). If the controller is a multiple one, we will consider that two consecutive elements will have an *agreement value* that can substitute in the text the two elements from the agreement point of view. We can have for example the following sequence of transformations:

c_1, c_2, c_3 <- T

$c_{1,2}$, c_3 <- T

$c_{1,2,3}$ <- T

We consider the agreement as an asymmetric relation. The target has a subordinate position because its features must agree with the controller. For example, if a subject is a multiple controller, than the predicate must have the plural number. (Observation: this is the reason for us to consider here the predicate as subordinate though usually the subject is considered as a subordinate).

Let us suppose that we have a description of the syntax of a natural language. We are not interested here what is the method used for this description but we suppose that it contains at least three elements: non-terminals, terminals (i. e. elements that belong to the natural language we describe) and rules that indicate how the non-terminals and terminals (we will note NT&T the terminals and non-terminal) are sequenced in the syntax description. Each NT&T have associated some syntactic categories with their values. Among the rules that are used in the syntactic description there are the agreement rules, very important especially for the languages with a strong inflection and quite free word order. These agreement rules contain the relations that must exist between the syntactic categories of the NT&T. The number of this agreement rules can be very

large. We will consider that somewhere in the grammar description there is a section that contains syntactic rules. A syntactic rule contains many NT&T. Each NT&T have associated a label and a list of syntactic categories, each category having one or many values. The NT&T labels will be used by the agreement rules description. We can not give here the complete syntax of the agreement description. We will present only the main features of this syntax and few examples in order to give the flavor of this description. The general form of the agreement description is as follows:

Agreement if ({condition}) true ({expression}) false ({expression}) not applicable ({expression}) undefined ({expression}).

The {condition} is a logical expression that uses the logical operators "and", "or" and negation "˜". The logical operands are {simple expressions} that have two forms:

{operand} + {operand} <- {operand}

{operand} <- operand}

The first form is used for multiple controllers and the second for simple controllers. An {operand} is a label of an NT&T and a set of categories with their values. These categories and values can be described using AVT (Attribute Value Trees - see [11]). The {expression} can contain others "if"s or some {action list}. This action list contains error messages and indicates how the analysis will be continued after an error was found.

A condition can have one of four values (in a tetravalent logic as we can see in section 3): TRUE, FALSE, NOTAPPLICABLE, UNDEFINED. Therefore, after "if({condition expression})" a list of many alternatives will follow (and not only two as in the case of "if" in a bivalent logic). If some of the four alternatives are identical, we can use "else".

Example 1: An agreement expression of the form if ($Label_1$ (person = I) -> $Label_2$ (person = I)) true (OK) else (Message = "Person agreement error", OK) will be read: "If the NT&T with the label $Label_1$ from the syntactic description has the person I and the NT&T with the label $Label_2$ from the syntactic description has the person I then continue the analysis, otherwise give the error message "Person agreement error" and continue the analysis as it was not an error.

Example 2: Let us have two non terminals that are found in a grammar rule:

$Label_1$: {non-terminal$_1$}(a = av1, av2)(b = bv_1, bv_2, bv_3)(c = cv_1, cv_2, cv_3)

$Label_2$: {non terminal$_2$}(d = dv_1, dv_2)(e = ev_1, ev_2, ev_3)(f = fv_1, fv_2, fv_3)

Let us have a simple expression of the form:

$Label_1$(a = av_1, av_2)(b = bv_2,bv_3) -> $Label_2$(e = ev_1, ev_2)(f = fv_2)

During the syntactic analysis, {non-terminal$_1$} and {non-terminal$_2$} will have some lexical or syntactic categories with some values. The operand $Label_1$(a = av_1, av_2)(b = bv_2, bv_3) will subsume the non terminal {non-terminal$_1$} when the {non-terminal$_1$} will have associated (by the syntactic analysis): the category a with the value av_1 or av_2 and the category b with the value bv_1 or bv_3. The operand $Label_2$(e = ev_1, ev_2)(f = vf_2) will subsume the non terminal {non-terminal$_2$} when {non-terminal$_2$} will have associated (by the syntactic

analysis): the category e with the value ev_1 or ev_2 and the category f with the value fv_2.

Example 3: We will refer to the general agreement rules (in Romanian language) for the number (sg = singular, pl = plural), gender (m = masculine, f = feminine, n = neuter), (p = animate, l = inanimate) and person (I, II, III) that must be observed by a predicate with verb to the passive voice when we have two subjects (i.e. a multiple subject).

Let us have some non terminals:

$Label_1$: {complex nominal group} [nominal group type = conjunctive] (position against the subject = left) (position against the predicate = left) (animation = p, l) (gender = m, f, n) (number = sg, pl) (person = I, II, III)

$Label_2$: {complex nominal group} [nominal group type = conjunctive, disjunctive] (position against the subject = right) (position against the predicate = left) (animation = p, l) (gender = m, f, n) (number = sg, pl) (person = I, II, III)

$Label_3$: {verbal group} [predicate type = verbal, nominal] [voice = active, reflexive, passive] (gender = m, f, n) (person = I, II, III) (number = sg, pl)

Here there are few rules for persons and numbers:

Agreement if (

/*1,1*/ $Label_1$(animation = p) (gender = m) (number = sg, pl) (person = II) + $Label_2$(animation = p) (gender = m, f) (number = sg, pl) (person = III) <- $Label_3$(gender = m) (number = plural) (person = II) or

/*1,2*/ $Label_1$(animation = p) (gender = m, f) (number = sg, pl) (person = II) + $Label_2$(animation = p) (gender = m, f) (number = sg, pl) (person = III) <- $Label_3$(gender = m) (number = plural) (person = II) or

/*1,3*/ $Label_1$(animation = p) (gender = m) (number = sg, pl) (person = II) + $Label_2$(animation = l) (gender = m, f, n) (number = sg, pl) (person = III) <- $Label_3$(gender = m) (number = plural) (person = II) or

/*2,1*/ $Label_1$(animation = p) (gender = m) (number = sg, pl) (person = III) + $Label_2$(animation = p) (gender = m, f) (number = sg, pl) (person = II) <- $Label_3$(gender = m) (number = plural) (person = II) or

/*2,2*/ $Label_1$(animation = p) (person = III) (gender = m, f) (number = sg, pl) (person = III) + $Label_2$(animation = p) (gender = m, f) (number = sg, pl) (person = II) <- $Label_3$(gender = m) (number = plural) (person = II) or

/*2,4*/ $Label_1$(animation = l) (person = III) (gender = m, f, n) (number = sg, pl) (person = III) + $Label_2$(animation = p) (gender = m) (number = sg, pl) (person = II) <- Label3(gender = m) (number = plural) (person = II) or

/*3,1*/ $Label_1$(animation = p) (gender = m) (number = sg, pl) (person = II) + $Label_2$(animation = p) (gender = m, f) (number = sg, pl) (person = II) <- $Label_3$(gender = m) (number = plural) (person = II) or

/*3,2*/ $Label_1$(animation = p) (gender = m, f) (number = sg, pl) (person = II) + $Label_2$(animation = p) (gender = m) (number = sg, pl) (person = II) <- $Label_3$(gender = m) (number = plural) (person = II)) true (Ok)

/* Continue for others situations*/

Using the above language we made a complete description of the accord in the Romanian language. We found that the Romanian language has 1350 agreement situations between a controller (simple or multiple) and a target. The situation's number can be even larger if we will try to capture also the error situations (that are quite frequent, and now we can see why!). Because it is a formal description, it can be used in automatic treatments.

else (/*.....*)

3 A Tetravalent Logic

In order to work with agreement rules we will define a more formal tetravalent logic

a) Truth values. We will use the following four truth values: i - "TRUE" noted with "1"; ii - "FALSE" noted with "0"; iii - "NOTAPPLICABLE" noted with "#"; iv - "UNDEFINED" noted with "*".

The variables that will take values in the this tetravalent logic are simple expressions described above. A simple expression contains operands. We will say that an operand subsumes or do not subsumes an NT&T obtained during the syntactic analysis. By "subsumption" we mean the followings:

Let us have in the agreement description an operand A_1 of the form:

$Label_1(Y_1 = y_{1,1}, y_{1,2},...)$ $(Y_2 = y_{2,1}, y_{2,2}, ...)$ $(Y_3 = y_{3,1}, y_{3,2}, ...)$ ...

Let us have an interpretation A_2 of the NT&T corresponding to the label $Label_1$ obtained by the syntactic analysis:

$(X_1 = x_{1,1}, x_{1,2},...)$ $(X_2 = x_{2,1}, x_{2,2}, ...)$ $(X_3 = x_{3,1}, x_{3,2}, ...)$ ...

The operand A_1 will subsume the NT&T A_2 if:

- A_1 and A_2 have not common lexical/syntactic categories or
- If A_1 and A_2 have some common lexical/syntactic categories (for example $Y_1 = X_1$, $Y_2 = X_2$, $X_3 = Y_3$) and the lexical/syntactic category values taken from A_1 are found among the corresponding lexical/syntactic category values taken from A_2. In the above example (where $Y_1 = X_1$, $Y_2 = X_2$, $Y_3 = Y_3$):

$y_{1,1}, y_{1,2}$, ... must be found among $x_{1,1}, x_{1,2}$, ...

$y_{2,1}, y_{2,2}$, ... must be found among $x_{2,1}, x_{2,2}$, ...

$y_{3,1}, y_{3,2}$, ... must be found among $x_{3,1}, x_{3,2}$, ...

...

We will say that an operand A_1 do not subsume an NT&T if A_1 and the NT&T have at least one common category and, for at least one common category, at least one category value from A_1 are not found among the corresponding category value from NT&T.

A simple expression is TRUE if all its operands have the property "subsumption" of some NT&T.

A simple expression is FALSE if the operands representing the controller subsume the corresponding NT&T and the operands representing the targets do not subsume the corresponding NT&T.

A simple expression is NOTAPPLICABLE if at least one of the operands representing the controller does not subsume the corresponding NT&T.

A simple expression can not be UNDEFINED. An undefined value can appear by logical operations with simple expressions as we will see below. A logical variable or a logical expression is UNDEFINED if we do not know if the corresponding value is TRUE or FALSE.

The subsumption property can be defined in a more complex (and more general) way using the AVT (Attribute Value Tree). In this case, the subsumption property must be replaced by the unification, i.e. the operand subsumes the corresponding NT&T if the operand is unifiable with the corresponding NT&T [11].

b) Basic unary logical operations. Let us have a logical variable x that can have one of the four truth values: 1, 0, #, *. We will define the following basic unary logical operations:

- "true" or "identity" noted by the variable itself x or by ^{1}x;
- "false" or "negation" noted by ˜ x or ^{0}x;
- "not applicable" noted by $^{\#}$x;
- "undefined" noted by *x.

The definitions are done in the Table 1. We can use also parentheses: (x) or x, ˜(x) or ˜x. (Observation: In a binary logic we can define $2^2 = 4$ unary operations. In a tetravalent logic we can define $4^4 = 256$ unary operations.)

Table 1. The unary operations in the tetravalent logic

x	"true" x, ^{1}x	"false" ˜x, ^{0}x	"not app." $^{\#}$x	"undefined" *x
1	1	0	*	#
0	0	1	#	*
#	#	*	1	0
*	*	#	0	1

c) Basic binary logical operations. Let us have two tetravalent logical variables x and y. We will define the basic binary logical operations "and" (noted by ".") and "or" (noted by "+") according to Table 2. (Observation: In a binary logic we can define $2^{2*2} = 16$ binary operations. In a tetravalent logic we can define $4^{4*4} = 4\ 294\ 967\ 296$ binary operations.)

d) Other properties. We can very easy verify that the usual logical properties are true: De Morgan Relations, the distributiveness of "and" against "or", the distributiveness of "or" against "and", the logical implication, the double logical implication. Using the double logical implication a <-> b = ˜a . ˜b + a . b, the fact that c is true, false, not applicable or undefined can be expressed as follows:

c <-> 1, c <-> 0, c <-> #, c <-> *

In the next section we will see how the agreement expressions can be transformed in tetravalent logical expressions.

Table 2. The definition of "and" and "or" in the tetravalent logic

x	y	x . y	x + y
1	1	1	1
1	0	0	1
1	#	#	1
1	*	*	1
0	1	0	1
0	0	0	0
0	#	0	#
0	*	0	*
#	1	#	1
#	0	0	#
#	#	#	#
#	*	0	1
*	1	*	1
*	0	0	*
*	#	0	1
*	*	*	*

4 Normalizing the Agreement Rules

The agreement rules (section 2) are written manually. We will present now a transformation of these rules to a tetravalent logical form (section 3) that can be used both in the automatic analysis process and automatic generation process. The transformation itself can be done automatically too.

Each rule from an agreement rule list can be represented intuitively as follows:

if ({condition}) true({a new rule beginning with "if" or an action list}) false({a new rule beginning with "if" or an action list}) not applicable({a new rule beginning with "if" or an action list}) undefined({a new rule beginning with "if" or an action list})

We can write more compact:

if (condition) true (a) false (b) not applicable (c) undefined (d)

By definition this expression is equivalent with a logical expression of the form of a conjunction of implications. These implications have the form:

- for true(a): (c <-> 1) -> a
- for false(b): (c <-> 0) -> b
- for not applicable(c): (c <-> #) -> c
- for undefined (d): (c <-> *) -> d

These forms can be also noted:

if(condition <-> 1) then (a)
if(condition <-> 0) then (b)
if(condition <-> #) then (c)
if(condition <-> *) then (d)

One or a group of clauses true(...), false(...), not applicable(...), undefined(...) can appear as being replaced by a single clause:

then({a new rule beginning with "if" or an action list})

To normalize an agreement rule list means to obtain a new agreement list under the form of a conjunction of rules of the form :

if({conjunctive condition})then({action list})

By {conjunctive condition} we mean a logical expression in conjunctive form (a conjunction of disjunctions).

The steps of such a transformation are as follows:

1. Transforming "else if". The form "...else if ({condition expression})..." will be replaced with "...else (if ({condition expression})...)" i.e. we insert an open parenthesis between "else" and "if" and we insert a closed parenthesis at the end of the expression.

2. Transforming "else". The transformation rule results immediately from the following example. Let us have a rule containing an "else":

if ({condition}) false(x_1) not applicable(x_2) else(x_3)

We will replace "else(x_3)" as follows:

if ({condition}) true(x_3) false(x_1) not applicable(x_2) undefined(x_3)

We replaced "else(x_3)" with the missing clauses ("true", "undefined") from the four possible clauses ("true", "false", "not applicable", "undefined"). The new clauses will have between the parentheses the content of the parentheses of "else".

After this transformation, all the expressions will have the same form: an "if" followed by one or many clauses from the four possible and the "else" will disappear".

3. Transforming an "if" that appeared in the parentheses of a "true", "false", "not applicable" or "undefined". After the above transformations there is not a clause "else". All the expression is now a sort of tree that has as nodes: "if({condition})", "true(...)", "false(...)", "not applicable(...)", "undefined(...)", "{action list}". An "if" node has at most four sons (at most one of each "true" , "false", "not applicable" or "undefined" sons) and at least one of the four types "true", "false", "not applicable" and "undefined" sons. Each node of the type "true", "false", "not applicable" and "undefined" has as sons a node of type "if" or a node of type action list. We will bring this tree to the form of a conjunction of expressions of the form:

if({condition}) then ({action list})

We can see that the root of the tree is an "if" node and the leaves are action list nodes. All the paths i that link the root with the leaves are of the form:

node($c_{i,1}$), $xx_{i,1}$ (...), nod($c_{i,2}$), $xx_{i,2}$(...), ...nod($c_{i,j}$), $xx_{i,j}$(...), ... nod($c_{i,ni}$), $xx_{i,ni}$(...), a_i

where $xx_{i,j}$ (...) are nodes of the type "true", "false" or "not applicable", "undefined" and a_i is an "action list" node. We will apply the following procedure:

a) We write all the paths of the above form.

b) For each path:

- We build a condition of the form:

$C_i = (c_{i,1}$ <-> $x_{i,1})$. $(c_{i,2}$ <-> $x_{i,2})$ . ... $(c_{i,j}$ <-> $x_{i,j})$ . ... $(c_{i,n}$ <-> $x_{i,n})$

- We build the expression:

if(C_i) then (a_i)

- Finally we put in a conjunction all the built expression:

if(C_1) then (a_1) . if(C_2) then (a_2) . ... . if(C_i) then (a_i) . ... . if(C_n) then(an)

Example 1: Let us have the expression:

if(c_1) true(if(c_2) true(a_1)) false (if(c_3) true(a_2))

After the transformation we obtain:

if((c_1 <-> 1) . (c_2 <-> 1)) then (a_1) . if((c_1 <-> 0) . (c_3 <-> 1)) then (a_2)

Example 2: Let us have the expression:

if(c_1) true(if(c_2) true(if(c_3) true(a_1)) false (if(c_4) true(a_2))) false (if(c_5) not applicable (a_3) undefined(a_4)) undefined (if(c_6) true (if(c_7) true(if(c_8) true(a_5))) undefined (if (c_9) true (a_6)))

The tree will become:

if((c_1 <-> 1) . (c_2 <-> 1) . (c_3 <-> 1)) then(a_1) . if((c_1 <-> 1) . (c_2 <-> 0) . (c_4 <-> 1)) then(a_2) . if((c_1 <-> 0) . (c_5 <-> #)) then(a_3) . if((c_1 <-> 0) . (c_5 <-> *)) then(a_4) . if((c_1 <-> *) . (c_6 <-> 1) . (c_7 <-> 1) . (c_8 <-> 1)) then(a_5) . if((c_1 <-> *) . (c_6 <-> *) . (c_9 <-> 1)) then(a_6)

The validity of these transformations can be demonstrated to the general case but here we will demonstrate only a particular case in the following example.

Example 3: Let us have the expression:

if(c) true (if(c_1) true(a_1)) false (if(c_2) true(a_2)) not applicable (if(c_3) true(a_3)) undefined (if(c_4) true(a_4))

The expression will become:

if((c <-> 1) . (c_1 <-> 1)) then(a_1) . if((c <-> 0) . (c_2 <-> 1)) then(a_2) . if((c <-> #) . (c_3 <-> 1)) then(a_3) . if((c <-> *) . (c_4 <-> 1)) then(a_4)

It is very easy to demonstrate that the two expression are equivalent.

4. Normalizing the conditions. The obtained condition expressions are logical expression that contains:

- logical operators "." and "+" (the priority of the operator "." is greater than the priority of the operator "+");
- the negation "~" (the priority of "~" is greater tan the priority of ".");
- operands that are simple expressions;
- parentheses that change the priority of the operations;
- the logical constants: 1, 0, #, * .

We do not enter here in the structure of the simple expressions. We will consider that each simple expression is noted by a letter: a, b, c, etc. To normalize the conditions means to put the conditions in the disjunctive form. There different simple algorithms to do this transformation. Such an algorithm is the following:

a) We apply the negation that is found before the parentheses using De Morgan relations until there are no more negations before the parentheses.

b) We open the parenthesis using the distributiveness of "." against "+".

5. The decomposition of the conjunctive forms. After the above transformations the agreement rules became conjunctions of expressions of the form:

if({condition}) then ({action list})

where {condition} is in the disjunctive form:

if((a1 and a2 ....) or (b1 and b2 and ...) or (c1 and c2 and ...)...) then (action list)

These forms can be rewritten as follows:

if(a1 and a2 and ...) then (action list) and if(b1 and b2 and ...) then (action list) and if(c1 and c2 and ...) then (action list) and ....

This equivalence can be demonstrated for the general case but we give here only a demonstration in a particular case, for illustration. Let us have the expression:

if((a and b) or (c and d))then(action list)

We rewrite the expression using only logical symbols and noting action list by A:

(a . b + c . d) -> A = ˜(a . b + c . d) + A = ˜(a . b) . ˜(c . d) + A = (˜(a . b) + A) . (˜(c . d) + A) = ((a . b) -> A) . ((c . d) -> A))

We pass to the initial notation and we obtain:

if(a and b) then (action list) and if(c and d) then (action list)

This normalized form will simplify not only the agreement check during the syntactic analysis but also especially the generation process where the controllers will force the attributes of the targets.

5 The agreement evaluation

As we showed in the section 3, the agreement between different NT&T is described under the form of an agreement rule list:

if({expression$_1$}) then ({action-list$_1$}) and if({expression$_2$}) then ({action-list$_2$}) and if({expression$_3$}) then ({action-list$_3$}) ... if({expression$_n$}) then ({action-list$_n$})

Each {expression$_i$} is a logical expression using as operands simple expressions.

a) During the syntactic analysis, the agreement check will be realized by the "execution" of this agreement rules. The execution consists in the following steps:

1. We evaluate all the simple expressions that appears in the expression i obtaining the corresponding truth values (see the section 3 point (a)).

2. We evaluate the truth value of each expression {expression i} using the above calculated values for the operands.

3. If the truth value of the expression i is "TRUE" (i.e. it is not "FALSE", "NOTAPPLICABLE" or "UNDEFINED") then the actions from the {action-list$_i$} will be executed. An action list can contain the actions OK, KO and error message. Executing an error message means to put this error message in a list of messages that will be eventually showed to the user. The execution of an

OK means in fact no operation. The execution of a KO will mean to note the apparition of an error.

4. When all the n expressions were evaluated:

- If no KO appeared, then the treatment is positively finished.
- If at least a KO is appeared, then treatment is negatively finished.

The syntactic analysis is accordingly continued.

b) During the text generation, the treatment concerns only the simple expressions and consists of:

1. We search a matching between the controller part of the simple expressions and the corresponding elements from the deep structure (this depends on how the deep structure is represented). Let us have the simple expression operand_1 -> operand_2 + operand_3 or operand_1 -> operand_2. We note {controller} the controller part and {target} the target part of this expressions. We consider that in the deep structure we have the corresponding elements {controller'}and {target'}. To have a matching between {controller} and {controller'} means that {controller} subsumes {controller'} (see the section 4).

2. When we find such a matching, the corresponding {target'} is forced to have such an attribute/value combinations that {target} can subsume {target'}.

We can see that the same representation of the agreement can be used in the automatic treatment of the analysis and of the generation.

6 Conclusions

The method of agreement description we presented permits a compact and systematic representation of natural language reversible grammars. The theoretical basis and the agreement expression handling do not imply excessively complicated problems of implementation in a computing system. The presented elements were embedded in a more general language GRAALAN (Grammar Abstract Language). Using this method and also others features of GRAALAN like the possibility to use macros) a complete description of the agreement in Romanian language was implemented.

References

1. Baker, M.: Complex Predicates and Agreement in Polysynthetic Languages, in A. Alsina, J. Bresnan, and P. Sells (eds.) Complex Predicates, CSLI, Stanford, (2002), 249-290.
2. Baker, M.: On Zero Agreement and Polysynthesis, Rutgers University (2002)
3. Barlow M., Agreement as a Discourse Phenomenon. In: Folia Lingusitica XXXIII/2, (1999) 187-210
4. Barlow, M. and Ferguson, C.: Agreement in Natural Language: Approaches, Theories, Descriptions, CSLI Publication (1988)
5. Block, Hans Ulrich: Compiling Trace & Unification Grammar for Parsing and Generation
6. Chung, S. : The Design of Agreement, University of Chicago Press (2000)

7. Corbett, G..: Morphology and Agreement. In: Zwicky , A. and Spencer, A. (eds) A Handbook of Morphology, Blackwell Oxford, (1998) 191-205
8. Corbett, G.: Constraints on Agreement. In: Caron, B. (ed.) Proceedings of the 16th International Congress of Linguists. Pergamon. Oxford. CD publication, (1998)
9. Corbett, G.: Agreement: Terms and boundaries. In: Griffin, W. (ed.): The Role of Agreement in Natural Language Proceedings of the Texas Linguistic Society Conference, Austin, Texas (2001)
10. Corbett, G.: Agreement: Canonical instances and the extent of the phenomenon. In: DeCesaris, J., Booij G., Ralli A. and Scalise, S. (eds.) Proceedings of the Third Mediterranean Morphology Meeting, Barcelona (2001)
11. Diaconescu, S.: Natural Language Understanding using Generative Dependency Grammar, in Proceedings of the twenty-second Annual International Conference of the British Computer Society's Specialist Group on Artificial Intelligence (SGES), (ES2002), Liverpool, UK, (2002)
12. Gertjan van Noord: Reversibility in Natural Language Processing, dissertation, 1993
13. Hudson, R: Subject-verb agreement in English, UCL (1998)
14. Katho, Andreas: Agreement and the Syntx Morphology Interface in HPSG, UC Berkley (1997)
15. Neumann, G. and Gertjan van Noord: Reversiility and Self-Monitoring in Natural Language Generation
16. Osenova, P. : On Subject-Verb Agreement in Bulgarian (An HPSG-based account)In: Proc. of the fourth Formal Description of Slavic Languages Conference, Potsdam, Germany, (2001)
17. Isabelle,Pierre: Towards Reversible MT Systems, in MT Summit II, 1989
18. Pollard, C.J., Sag, I.A.: Head-Driven Phrase Structure Grammar, University of Chicago Press and Stanford CSLI Publications (1994)
19. Reiter, E. and Dale R.: Building Natural Language Systems, Cambridge University Press, (2000)
20. Steele, Susan: Word order variation: a typological study. In Greenberg, J. H., Ferguson C. A. and Moravsik, E.A. (edc.), Universals of Human Language: IV: Syntax, Stanford University Press, (1978), 585-623
21. Strzalkowski, Tomek (ed): Reversible Grammar in Natural Language Processing, USA Kluwer Academic Publishers, Boston Hardbound, 1993