Triangle: Language, Literature, Computation, n. 8, 2012
Publicacions Universitat Rovira i Virgili · ISSN: 2013-939X
https://revistes.urv.cat/index.php/triangle

GNS: Abstract Syntax for Natural Languages

Vincenzo Manca[1] and M. Dolores Jiménez-López[2]

[1] Department of Computer Science
University of Verona
Verona, Italy
E-mail: vincenzo.manca@univr.it

[2] Research Group on Mathematical Linguistics
Universitat Rovira i Vigili
Tarragona, Spain
E-mail: mariadolores.jimenez@urv.cat

Summary. This paper presents an overview of General Natural Syntax (GNS), a formal theory of general explicative power that generalizes and formalizes syntactic concepts in order to offer a general notion of syntax that is independent of any particular language.

1 General Natural Syntax: Basic Idea

General Natural Syntax (GNS) is a formal theory of general explicative power that intends to formalize a general notion of syntax that is independent of any particular language. The idea behind this formalism is that it reduces syntactic constructions to a few principles related to their semantic functions, but which can be defined independently of semantics.

GNS generalizes and formalizes syntactic concepts so that they can be incorporated into a formal theory of general explicative power. As we have said, the general principles we proposed as elements of GNS are related to their semantic function but definable independently of semantics. GNS representations stand between the concrete syntax of specific natural languages and the semantic function of syntactic phenomena. In this regard, GNS can be related to the so-called *abstract syntax* of programming languages [4]. Abstract syntax is an important tool in the translation process between programming languages: it provides a deep description of syntactic constructs and is independent of any particular syntactic encoding. The starting point for this approach was an algebraic notion of syntax, more abstract than a concrete syntax, placed at an intermediate level as a bridge between syntax and semantics. Symbols of an *abstract syntax expression* denote semantic functions, but at the same time, all the information is included so that the concrete syntax of the expression can be generated.

GNS is also related to the notion of tagged text, which is the basis of many markup languages (TeX, HTML, XML). Also here, abstract expressions are used that mix the pure textual information with information about text format and visualization (fonts, dimensions, paragraphs, etc.). So, the structure of a text is represented in a way that in many aspects is independent from the way a text is realized by a compiler or a browser. Marked texts *abstractly* express properties and relations of the textual units at a logical level.

Considering all the above ideas and taking into account that the use of algebraic formalization has a strong and deep tradition in natural language analysis within the fields of semantics [15, 14] and morphology [13], what we propose here is to develop an analogous logic-algebraic approach at an intermediate level between semantics and morphology: the level of the *abstract or general syntax.*

2 Formal Prerequisites

In this section we provide the formal prerequisites that are needed to understand the formalization presented in this paper. For further information on the theory of formal languages and mathematical logic see [17, 16, 2].

A nonempty set V of *symbols* or *letters* is called an *alphabet.* A *word* or a string over an alphabet V is an element of the free monoid V^* generated by the symbols of V under a binary associative operation of *concatenation* (denoted by juxtaposition). The *empty word* λ is the neuter element with respect to concatenation ($x\lambda = \lambda x = x$). The *length* of a string $x \in V^*$ (the number of symbol occurrences in x) is denoted by $|x|$.

We use symbols $\rightarrow$, $\neg$, $\wedge$, $\vee$, $\leftrightarrow$, $\forall$, $\exists$, $\equiv$, $=$ with the standard syntactic and semantic first order logic meaning. We assume that the reader is familiar with logical notions such as variables, constants, predicates, functors, first order formulae and terms, and free and bound variables. A set of formulae, called *axioms*, and all their *logical consequences* constitute a *theory* (with respect to some notion of logical consequence or of logical deduction).

A *model* $\mathcal{M}$, or a (relational) structure, is given by: i) a set D, called *domain* of $\mathcal{M}$, ii) some elements $a, b, \ldots \in D$, called *individual constants* of $\mathcal{M}$, and iii) some operations $f, g, \ldots$ and relations $R, Q, \ldots$ over D (an *arity* is associated with each operation and relation and it specifies its number of arguments). Usually $\mathcal{M}$ is indicated by:

$$\mathcal{M} = (D, a, b, \ldots, f, g, \ldots, R, Q, \ldots)$$

The set $Term(\mathcal{M})$ of the terms over $\mathcal{M}$ is given by all the expressions that can be constructed, in the usual algebraic sense, by applying the operations of $\mathcal{M}$ to the individual constants of $\mathcal{M}$. For example, if f has arity 1 and g has arity 2, then the following are terms over the model given above:

$$f(a), \quad g(a,b), \quad f(g(a,b)), \quad g(a,f(a)), \quad f(g(a,f(a))), \quad \ldots$$

An equation such as $g(a, f(a)) = b$ means that by applying the operation f to the constant a we get an element of D, say c, and by applying g to the pair a and c we get b. This means that $g(a, f(a))$ is considered as the *denotation* of the element of D obtained by applying the operations following the way algebraic expressions are usually evaluated (in the order specified by parentheses). However, we can consider the term as denoting itself, i.e. a sequence of symbols for individual constants, operations, commas, and parentheses, disregarding any meaning of symbols. It is important to distinguish between these two aspects. If we want to be precise we write $g(a, f(a))$ to refer to the element of D (if it exists) denoted by the term, while in the other case we write $\lceil g(a, f(a)) \rceil$. However, in practice the context will indicate in which sense a term is used.

A *string model* or a *monoidal model* $\mathcal{M}$ is a model that has: i) a domain that includes a free monoid V^* over an alphabet V, ii) constants that include a constant λ for an empty string and for symbols of V, iii) operations that include a binary operation that on V^* coincides with the string concatenation on V^* (indicated by juxtaposition).

A signature Σ is essentially a set of symbols for denoting objects (individual constants), operations and relations of a relational structure (relations can be identified as operations that provide Boolean values as results). In Σ, any symbol of operation or relation is equipped with an *arity* that specifies the number of arguments of the corresponding operation or relation.

A *string theory* or a *monoidal theory* $\mathcal{T}$, over an alphabet A and a signature Σ, is a first order logic theory of axioms Φ over the signature Σ such that all the symbols of A and the empty string λ are individual constants of Σ. The concatenation operation of monoids is denoted in Σ, and Φ also includes the monoid axiom: $\forall x, y, z(x(yz) = (xy)z \wedge x\lambda = x \wedge \lambda x = x)$. In other words, a monoidal theory over an alphabet A and a signature Σ is a theory where terms include the strings over A. This means that concatenation is applied not only to the symbols of A, but to the terms that can be built on the signature Σ. This possibility provides syntactic constructs where not only symbols of an alphabet, but even terms, are concatenated.

A *monoidal system* or simply a *monoidal* indicates, generically, a string model or a string theory. Monoidals are good environments for defining the syntax of formal languages [11].

Let us indicate by $GNS(L)$ the terms, with their syntactic category, that can be constructed starting from the lexical items of L in a suitable string theory that will be defined in the course of the paper. The alphabet of this theory is the set of usual Latin letters plus other special symbols that will appear in the axioms given in the next sections. Variables are indicated by letters $x, y, z, u, v, \ldots$ which may have apices or subscripts. Individual constants are indicated by special strings of capital letters. Operations and predicates are indicated by strings starting with a backslash (\). For a better reading, given a predicate P we write $t : P$ (t is of type P), instead of $P(t)$, in fact, $t : P$ intuitively means that term t belongs to the syntactic category P.

3 General Natural Syntax: the Formalism

GNS has been defined as a general formal framework given by some *axioms* according to which some *operations* are applied to *strings* of some categories and get other strings (of some specified categories). The formalism consists of:

- Eight basic categories;
- Thirty syntactic operations;

- Forty axioms;
- A few hundred grammemes;
- A few thousand lexical items (basic lexicon).

In what follows we will present the above elements in order to provide an overview of GNS.

3.1 Three basic operations

In order to construct GNS, three basic operations have been assumed as the preliminary data of the analysis:

1. *Conjugation* ($\backslash conjug$): adds temporal and dynamic parameters to syntactic elements that can play verbal roles.
2. *Determination* ($\backslash determ$): adds spatial and contextual information to elements that can play nominal roles.
3. *Predication* ($\backslash pred$): is the basic sentence building construction.

We assume that we know when these operations can be applied to some arguments.

3.2 Eight basic categories

Taking into account the three basic operations introduced in the above section we consider the following eight basic categories:

1. **Verb** ($Verb$): We define the class $Verb$ of elements x such that, for some conjugative y, $\backslash conjug(x,y)$ provides a result. Formally:

$$Verb = \{x \mid \exists\, y\; z\; \backslash conjug(x,y) = z\;,\; y \in Conjugative\}$$

2. **Noun** ($Noun$): We define the class $Noun$ of elements x, such that, for some determinative y, $\backslash determ(x,y)$ provides a result. Formally:

$$Noun = \{x \mid \exists\, y\; z\; \backslash determ(x,y) = z\;,\; y \in Determinative\}$$

3. **Substantive** ($Subst$): A substantive is the result of the determination operation. Formally:

$$\forall xyz(\backslash determ(x,y) = z \;\wedge\; y : Determinative \;\rightarrow\; z : Subst)$$

4. **Verbative** ($Verbt$): A verbative is the result of the conjugation operation. Formally:

$$\forall xyz(\backslash conjug(x,y) = z \ \wedge \ y : Conjugative \ \rightarrow \ z : Verbt)$$

5. **Proposition** ($Prop$). A proposition is the result of a $\backslash pred$ operation that takes as arguments a substantive ($Subst$) and a verbative ($Verbt$) and provides a proposition as a result. Formally:

$$Prop = \{z \mid \exists \ x \ y \ \backslash pred(x,y) = z \ , \ x \in Subst \ , \ y \in Verbt\}$$

We write $x : Cat$ to state that x is an expression of category Cat. We use Cat to indicate any of the previous categories. To those five basic categories we need to add the following three *Ad categories* ($AdCat$ indicates any of them):

6. **AdProp**
7. **AdNoun**
8. **AdVerb**

It is important to note here that although the terms *proposition*, *noun*, *verb*, and *substantive* are taken from traditional grammatical, logical, and semantical analysis, the definition we provide of our basic types is completely formal, based on the assumption of some initial operations.

3.3 Grammemes

The eight basic categories above are those of *full* linguistic elements. To them we add a category of *empty* linguistic elements that we have called *grammemes*[3]. Grammemes determine the surface syntactic realization of the abstract syntax operations considered in GNS.

Syntactic operations defined in GNS allow us to construct complex linguistic expressions with one of the eithg basic categories described above. However, many syntactic operations need some additional parameters, which we have called *grammemes*. Grammemes are defined as elements that individuate the features required to evaluate terms of GNS(L) into expressions of L: that is, kinds of information that the grammar of L requires to provide the

[3] Note that we are not using 'grammeme' here with its usual meaning in linguistic morphology.

morphological realization corresponding to the abstract syntactic representation.

It is important to distinguish *grammemes* from *particles*. Particles are strings that are provided in the surface syntactic realization of the abstract syntax operations. Particles are strings that are inserted in a text as a consequence of applying a syntactic operation with some grammeme. Examples of grammemes are: singular (`SING`), plural (`PLUR`), present (`PRES`), present-progressive (`PPG`), past (`PAST`), disjunctive (`OR`), definite (`DEF`), agent (`AGENT`), and first, second, third (`I, II, III`). The difference between grammemes and particles can be observed in the following English example: \conjug(*go*, `III, SING, PPG, PAST`)= *was going*.

Grammemes are classified into several types. Although these types depend on the particular language we believe that they vary very little even within wide classes of languages. What is essential in the notion of grammemes is that they are a "closed set", which is a fixed and small number of elements not exceeding a few hundred units (apart from numeral and ordinal elements). Classes of grammemes in English, Italian, Spanish, and many other languages are listed in Table 1.

Note that some of the above classes consist of a *sequence of grammemes*, which depends on the language we are considering. For example, the class *conjugative* consists of a sequence which, depending on the language, may include types such as *personal, number, gender, tense, modal-attitudinal, aspect....*

3.4 Operations: Generalities

The syntactic operations we defined in GNS are related to fundamental semantic roles, but only the categories of their arguments and the categories of their results are essential in their determination. In this regard, a general syntactic construction is a kind of bridge between syntax and semantics. It does not deal with the particular morphological features of the final linguistic form of the expression that is the result of applying the syntactic operation, but at the same time, it does not concern the meaning of the final expressions and the way this meaning is related to those of the arguments of the operation. Therefore, a syntactic operation is defined by:

i) its name,
ii) the categories of its arguments, and
iii) the category of the resulting expression.

CLASSES OF GRAMMEMES
1. Conjugative:
- Personal
- Number
- Tense
- Mood
- Gender
- Aspect
- Modal-Attitudinal
2. Determinative
3. Copulative
4. Circumstantial
5. Complementative
6. Locative
7. Cardinal
8. Ordinal
9. Deictic
10. Quantifier
11. Intensive
12. Comparative
13. Coordinative
14. Subordinative

Table 1. Classes of Grammemes

The syntactic operations defined can be classified as constructs of one of three general schemata:

1. *Combination Schema* puts together strings of different categories and provides a string of a category other than the categories of its components.
2. *Expansion Schema* takes as its argument a string of a given category and provides a bigger string of the same category.
3. *Transcategorization Schema* transforms a string of a given category into a string of another category.

Syntactic operations defined in GNS are listed in Table 2.

3.5 Axioms

GNS is given by some *axioms* according to which operations are applied to strings of some categories and produce other strings (of some specified cat-

N.	OPERATION	NOTATION
1	Conjugation	\ conjug
2	Determination	\ determ
3	Predication	\ pred
4	Copulation	\ copul
5	Modification	\ modif
6	Apposition	\ appos
7	Complementation	\ complem
8	Localization	\ loc
9	Coordination	\ coord
10	Subordination	\ subord
11	Negation	\ neg
12	Passivation	\ passiv
13	Adjectivation	\ adj
14	Adverbialization	\ adverb
15	Adproposition	\ adprop
16	Relativization	\ relat
17	Anacoluthon	\ anacoluth
18	Numeralization	\ num
19	Ordinalization	\ ord
20	Deictification	\ deixis
21	Anaphorization	\ anaphor
22	Quantification	\ quant
23	Intensification	\ intens
24	Comparison	\ compar
25	Correlation	\ correl
26	Substantivation	\ substantiv
27	Nominalization	\ nomin
28	Quotation	\ quot
29	Interrogation	\ interr
30	Imperativization	\ imper

Table 2. GNS Operations

egories). In this section we introduce the forty axioms related to the above thirty syntactic operations. Some examples of their functioning are also provided.

We start by providing the first five axioms of our model according to which the operations of *conjugation*, *determination* and *predication* are applied.

Axiom 1: Conjugation

$$\forall xyz(\backslash conjug(x,y) = z \ \wedge \ y : Conjugative \ \rightarrow \ z : Verbt)$$

The type *Conjugative* refers to the conjugation parameters that in English, Spanish and Italian are given by *Tense, Mood, Person, Number*. Examples of conjugative grammemes are: `PRES` (present), `PAST` (past), `FUT` (future), `PPG` (present-progressive), `IND` (indicative), `SUB` (subjunctive), `PART` (participle), `I` (first), `II` (second), `III` (third), `SING` (singular), `PLUR` (plural), `NEC` (necessity), `PERF` (perfect), `IMP` (imperfect), `MASC` (masculine), `FEM` (feminine), `NEUT` (neuter).

Axiom 2: Determination

$$\forall xyz(\backslash determ(x,y) = z \ \wedge \ y : Determinative \ \rightarrow \ z : Subst).$$

The basic *determinative* grammemes used in determination `DEF` (definite/near) and `INDEF` (indefinite/far) in many languages need to be accompanied with grammemes of *number* (`SING` (singular), `PLUR` (plural)) and of *gender* (`MASC` (masculine), `FEM` (feminine), `NEUT` (neuter)). In other languages the set of determinatives could be richer and related to an intrinsic mechanism of noun classification.

Predication is the basis of the grammatical schema that provides a sentence by combining a 'subject' and a 'predicate'. The following axioms show how to apply this operation.

Axiom 3: Predication

$$\forall x(x : Prop \rightarrow \exists u, v(u : Subst \wedge v : Verbt \wedge x = \backslash pred(u,v)))$$

Axiom 4: Predication

$$\forall uvwz((u : Subst \wedge v : Verbt \wedge w : Verb \wedge v \neq \lambda \ \wedge z : Conjugative \ \wedge v = \backslash conjug(w,z) \wedge Agree(u,v)) \rightarrow \backslash pred(u,v) : Prop)$$

Axiom 5: Predication

$$\forall xuv(x : Prop \wedge x = \backslash pred(u,v) \rightarrow v = PREDICATE(x) \wedge u = SUBJ(x)$$

Knowing how conjugation, determination and predication work, we can consider the sentence: *'The dog barks'*. It can be formalized as follows (by *DEF* we mean the definite determiner):

$$\backslash pred(\backslash determ(\text{dog}, \texttt{DEF}) , \backslash conjug(\text{bark}, \texttt{III, SING, PRES}))$$

The deep linguistic level of abstract syntax representations is easily understood if we consider the Italian and Spanish translations of the above sentence: *'Il cane abbaia'* and *'El perro ladra'*, respectively, whose abstract syntax representations are the following:

$$\backslash pred(\backslash determ(\text{cane}, \texttt{DEF}), \backslash conjug(\text{abbaiare}, \texttt{III, SING, PRES}))$$
$$\backslash pred(\backslash determ(\text{perro}, \texttt{DEF}), \backslash conjug\ (\text{ladrar}, \texttt{III, SING, PRES}))$$

where we can see that these two representations can be obtained from the English one by replacing the lexical items *dog* and *bark* with the corresponding lexical items in Italian and Spanish.

Copulation is an operation that transforms a $Noun$ and $AdNoun$, or a $Subst$ into a $Verb$. Axiom 6 provides the requirements to apply this operation.

Axiom 6: Copulation

$$\forall xy((x : Noun \vee x : AdNoun \vee x : Subst) \wedge y : Copulative \rightarrow \backslash copul(x, y) : Verb)$$

Copulative grammemes are `BE, BECOME, SEEM...` Examples of copulative constructions are the following:

- *'was a lawyer'*: $\backslash conj$(`III, SING, PAST` ($\backslash copul$(lawyer, `BE`))
- *'became president'*: $\backslash conj$(`III, SING, PAST` ($\backslash copul$(president, `BECOME`))

Modification ($\backslash modif$) is the combination of a phrase of category $AdCat$ with a phrase of a category Cat. Modification introduces the notions of *kernel* and *modifier*. Axioms 7 and 8 formalize this operation:

Axiom 7: Modification

$$\forall xy(x : Cat \wedge y : AdCat \wedge Agree(x, y) \rightarrow \backslash modif(x, y) : Cat)$$

Axiom 8: Modification

$$\forall x(\exists uv : u : Cat \wedge v : AdCat \wedge x = \backslash modif(u, v) \rightarrow KER(x) = u \wedge MODIFIER(x) = v)$$

The following examples show how modification works in GNS:

- *'young artist'*: $\backslash modif$(artist, young)
- *'to walk slowly'*: $\backslash modif$(walk, $\backslash adverb$(slow))

Apposition ($\backslash appos$) is an expansion of a $Subst$ with a $Noun$ or an $AdNoun$. The noun or $AdNoun$ does not add any further element into the determination of the substantive, but it gives only additional descriptive aspects of what is identified by the substantive. Axiom 9 accounts for this operation.

Axiom 9: Apposition

$$\forall xy(x : Subst \wedge y : Noun \vee y : AdNoun \rightarrow \backslash appos(x, y) : Subst)$$

An example of apposition is the following:

- *'Rome, the capital of Italy'*: $\backslash appos$(Rome, capital of Italy)

Two basic axioms that refer to *agreement* features are the following. Note that axiom 10 axiomatizes the requirements for the relation $Agree$ that guarantees the presence of some common features between the subject and the predicate of a predication. Axiom 11 concerns some commutativity requirements between the kernel of a modification and the agreement features of a verb or a substantive. We indicate by $Feature$ a typical feature (e.g. gender, number, person, ...)

Axiom 10: Agreement

$$\forall xyz(\backslash pred(x, y) = z \rightarrow (x : Subst \wedge y : Verb \wedge Agree(x, y)))$$

Axiom 11: Agreement

$$\forall xy(x : Cat \wedge y : AdCat \rightarrow Feature(x) = Feature(\backslash modif(x, y)))$$

Axioms 12, 13, 14 and 15 show how complementation is applied. *Complementation* ($\backslash complem$) is an operation in which a $Noun$ or a $Verb$ is expanded either with a $Subst$ or with a $Noun$. This means that we can identify four possible types of complementation:

1. $\backslash complem_{VS}$ (a verb with a substantive);
2. $\backslash complem_{NS}$ (a noun with a substantive);
3. $\backslash complem_{VN}$ (a verb with a noun);
4. $\backslash complem_{NN}$ (a noun with a noun).

These four types of complementation are formalized in the following axioms:

Axiom 12: Complementation

$$\forall xyz(x : Verb \wedge y : Subst \wedge z : Complementative \rightarrow \backslash complem_{VS}(x, z, y) : Verb)$$

Axiom 13: Complementation

$$\forall xyz(x : Noun \wedge y : Subst \wedge z : Complementative \rightarrow \backslash complem_{NS}(x, z, y) : Noun)$$

Axiom 14: Complementation

$$\forall xyz(x : Verb \wedge y : Noun \wedge z : Complementative \rightarrow \backslash complem_{VN}(x, z, y) : Verb)$$

Axiom 15: Complementation

$$\forall xyz(x : Noun \wedge y : Noun \wedge z : Complementative \rightarrow \backslash complem_{NN}(x, z, y) : Noun)$$

Complementative are grammemes that identify the *roles* of complementations. They correspond to the *cases* typical of many languages (e.g. Latin, Greek, Russian, ...) and can be represented with specific grammemes related to a few main functionalities: `AGT` (agent), `INST` (instrument); `GOAL` (goal), `MANN` (manner), `MATT` (matter), `OWN` (owner), `REF` (referent), `REC` (receiver), `UNI` (union), `CONT` (content), `PURP` (purpose), and `CAUS` (cause).

Examples of complementation are the following:

- 'Jane *eats the cracker*': $\backslash complem_{VS}$(eats, the cracker).
- *'cup of tea'*: $\backslash complem_{NN}$(cup, `CONT`, tea).

Locative phrases are a special type of modifiers used to indicate a spatio-temporal localization. Axiom 16 accounts for this syntactic phenomenon which we call *localization* ($\backslash loc$):

Axiom 16: Localization

$$\forall xy(x : Subst \wedge y : Locative \rightarrow \backslash loc(x, y) : AdCat)$$

Locative grammemes can be `IN, OUT, TO, FROM, AROUND, ACROSS, BETWEEN, NEAR, OVER, TOWARDS, BEFORE, AFTER,` etc.
An example of a locative phrase is:

- *'(Jane) found the turtle under the table'*: $\backslash modif(\backslash complem_{VS}$(found, the turtle), $\backslash loc$(the table, `UNDER`))

Coordination ($\backslash coord$) allows us to connect two (or more) categories of the same type by providing a category of that type. If two elements are of the same category, then they can be joined together and the resulting unit is of the same type. Any category can be coordinated (by means of a coordinative grammeme: `AND` (addition), `OR` (alternation), `BUT` (variation)) with a category of the same type and the result will be an item of the same category. Axiom 17 formalizes this idea.

Axiom 17: Coordination

$$\forall xyz(x : Cat \wedge y : Cat \wedge z : Coordinative \rightarrow \backslash coord(xzy) : Cat)$$

Consider the sentence 'We saw many *students of chemistry and doctors of medicine'*, where the elements in italics are coordinated. In terms of GNS, coordinated categories can be formalized as follows:

$\backslash coord((\backslash complem_{NN}$(students, chemistry)), `AND`, ($\backslash complem_{NN}$(doctors, medicine)))

Subordination connects elements that have a different grammatical 'status', one of which is subordinate to or dependent on the other. *Subordinative* grammemes are: `BECAUSE` (reason), `IF` (condition), `WHERE` (place), `WHEN, AFTER, BEFORE` (time), `WHILE, UNTIL` (duration), `THOUGH` (concession), `FOR, TO` (purpose). Axioms 18 and 19 give a formal account in GNS of this operation.

Axiom 18: Subordination

$$\forall xyz(x : Cat \wedge y : Cat \wedge z : Subordinative \rightarrow \backslash subord(xzy) : Cat)$$

Axiom 19: Subordination

$$\forall xyz(x : Cat \wedge y : Cat \wedge z : Subordinative \rightarrow \backslash subord(xzy) : Cat \ \wedge x = Ker(\backslash subord(xzy)))$$

Examples of the subordination of various classes of categories are: *'poor but happy'*, *'late though not too late'*, *'enemy for joke'*, *'going to see'*... The reader can easily check how to formalize these examples by using the above axiom.

Negation ($\backslash neg$) is a syntactic operation that can be applied to any Cat and the result will be an item of the same Cat. Axiom 20 formalizes this operation.

Axiom 20: Negation

$$\forall x(x : Cat \rightarrow \backslash neg(x) : Cat)$$

Examples of the application of this operation in the GNS framework are:

- '(I do) not agree': $\backslash neg$(agree)
- '(It is) not soon': $\backslash neg$(soon)

Passivation ($\backslash passiv$) is an operation that like negation does not change the syntactic category of the element it is applied to. It is based on the fact that the order of the subject and object can be reversed in a predication with transitive verbs. In this case, if P is a proposition to which passivation can be applied, then $\backslash passiv$(P) is its passive form. The actual means to calculate the value of $\backslash passiv$(P) is a matter of concrete syntax. Formally:

Axiom 21: Passivation

$$\forall x(x : Prop \rightarrow \backslash passiv(x) : Prop)$$

Axioms 22, 23 and 24 formalize cases of *transcategorization. Adcategorization* stands for the operations of *adjectivation, adverbialization* and *adproposition* considered in the following axioms.

Adjectivation ($\backslash adj$) takes as its input a $Noun$, a $Subst$, a $Verb$ or a $Verbt$ and provides an $AdNoun$ as shown in axiom 22.

Axiom 22: Adjectivation

$$\forall xy((x : Noun \vee x : Subst \vee x : Verb \vee x : Verbt) \wedge \backslash adj(x) : AdNoun)$$

Adverbialization ($\backslash adverb$) takes as an input an $AdNoun$ and provides an $AdVerb$.

Axiom 23: Adverbialization

$$\forall x(x : AdNoun \rightarrow \backslash adverb(x) : AdVerb)$$

And finally, *adproposition* ($\backslash adprop$) yields an *adproposition* element from either an $AdNoun$ or a $Subst$.

Axiom 24: Adproposition

$$\forall x(x : AdNoun \vee x : Subst \rightarrow \backslash adprop(x) : AdProp)$$

Examples of the above operations are the following:

- *'milk cup'*: $\backslash modif$(cup, $\backslash adj$(milk))
- '*slowly*': $\backslash adverb$(slow)
- *'This way*, (Doris feeds her guppies)': $\backslash adprop$(this way)

Relativization ($\backslash relat$) is the syntactic operation that transforms a $Prop$ into a $Noun$. In order to describe this construction, we assume a special grammeme REL that has the category of $Subst$ (Actually, a finite number of different relative grammemes REL1, REL2, REL3, ..., could be necessary in certain cases). If proposition P is 'REL *was on the table'*. In this case, $\backslash relat$(P) is a noun, so the following syntactic term:

'I am looking for the \modif (pen, \adj (\relat (P))'

represents the statement: *'I am looking for the pen that was on the table.'*

Axiom 25 formalizes relativization, where Prop(REL) are the propositions constructed by the special substantive REL.

Axiom 25: Relativization

$$\forall x(x : Prop(\mathtt{REL}) \rightarrow \backslash relat(x) : Noun)$$

A syntactic operation formally described by the element of Prop (REL) is the one that underlies the so-called phenomenon of *anacoluthon*, a construction present in ancient languages and in colloquial forms of modern languages. We introduce the operation of *anacoluthon* ($\backslash anacoluth$) defined in axiom 26.

Axiom 26: Anacoluthon

$$\forall x(x : Prop(\mathtt{REL}) \rightarrow \backslash anacoluth(x) : Verb)$$

An example of anacoluthon formalized in GNS terms is the following:

'John, I told him to go away.': \pred(John, \conjug(\anacoluth(I told `REL` to go away), `III, SING`))

Axioms 27 and 28 formalize the operations of *numeralization* and *ordinalization*, respectively. *Numeralization* (\num) takes as its arguments a *cardinal number* (two, three, ten, two hundred, three thousand, a dozen, hundreds) and a *noun* and provides a *noun*. However, this operation is only defined for some nouns, which are called *countable* nouns. *Ordinalization* (\ord) takes as its arguments an *ordinal number* (first, second, last, next...) and a *noun* and provides a *noun*.

Axiom 27: Numeralization

$$\forall xy(x : Noun \wedge y : Cardinal \rightarrow \backslash num(x, y) : Noun)$$

Axiom 28: Ordinalization

$$\forall xy(x : Noun \wedge y : Ordinal \rightarrow \backslash ord(x, y) : Noun)$$

The following two phrases are examples of these operations:

- *'Three friends'*: \num(friend, `3`))
- *'The first friend'*: \determ(\ord(friend, `1`), `DEF`)

Deictification (\deixis) is an operation that takes as its arguments some grammemes (*conjugative, determinative*: `PRES, PAST, I, II, III, DEF`/`INDEF`, `NEAR, FAR`) and provides a *Subst*. Axiom 29 formalizes this idea.

Axiom 29: Deictification

$$\forall x(x : Deictic \rightarrow \backslash deixis(x) : Subst)$$

English examples of deictification are the following:

- \deixis(`PRES`) = *'Now'*
- \deixis(`I, SING`) = *'I'*
- \deixis(`NEAR`) = *'Here'*

Anaphorization ($\backslash anaphor$) is a construct that assigns a label to syntactic terms (*pronoun*) which can be used instead of the labeled term itself. Notice that the label (pronoun or whatever) is something that concerns the concrete syntax and not the abstract syntax we are defining here. Taking this into account we define this operation in axiom 30.

Axiom 30: Anaphorization

$$\forall x(x : Cat \rightarrow \backslash anaphor(x) : Cat)$$

An example:

- *'On the table there is a pen,* **it** *is red'*: $\backslash anaphor(\text{a pen}) =$ *'it'*

Quantification ($\backslash quant$) can be considered as a special case of determination. However, it presents some subtle logical and semantic features that mean that it has to be identified separately from determination. First of all, quantification can only be applied to *countable* nouns or their plurals and provides as its result a substantive with a 'collective and distributive' nature. It is formalized in axiom 31.

Axiom 31: Quantification

$$\forall xy(x : Quantifier \wedge y : Noun - Countable \rightarrow \backslash quant(x, y) : Subst)$$

$$\forall xy(x : Quantifier \wedge y : Subst - Plural \rightarrow \backslash quant(x, y) : Subst)$$

Quantifiers are grammemes that express the functionalities of words that classical grammars classify as distributive pronouns: `EVERY`, `EACH`, `ALL`, `ANY`, `FEW`, `SOME`, `MOST`, `MANY`, etc. (Some of them are very often referred to as 'partitive' pronouns).

Intensification ($\backslash intens$) consists of the use of grammemes that express intensity or non-exact quantity with uncountable nouns. Examples of *intensive* grammemes are `MUCH`, `A LITTLE`, `FEW`, `ALMOST`, `MORE`, `LESS`, `VERY`, `WHOLE`, `PARTIAL`, `ENOUGH`, `TOO`... Axiom 32 provides a formal account of this operation.

Axiom 32: Intensification

$$\forall xy(x : Noun - Uncountable \wedge y : intensive \rightarrow \backslash intens(x, y) : Noun)$$
$$\forall xy(x : AdNoun \wedge y : intensive \rightarrow \backslash intens(x, y) : AdNoun)$$

Comparison ($\backslash compar$) is related to degree particles such as *more-than, less-than, so-as*. Comparison can be obtained in three ways: 1) with two terms of a given category Cat and a comparison grammeme (this is, $(<, >, =)$); 2) with a $Verb$ or an $AdNoun$, a comparison grammeme, and a $Subst$; 3) with a $Verb$ or an $AdNoun$, a comparison grammeme, and a $Prop$. This idea is captured in axiom 33.

Axiom 33: Comparison

$$\forall xyz(x : Cat \wedge y : Cat \wedge z : Comparative \rightarrow \backslash compar(x, y, z) : Cat) \wedge$$

$$\forall xyzT(x : T \wedge \wedge(T = Verb \vee T = AdNoun) \wedge y : Subst \wedge z : Comparative \rightarrow \backslash compar(x, y, z) : T) \wedge$$

$$\forall xyzT(x : T \wedge (T = Verb \vee T = AdNoun) \wedge y : Prop \wedge z : Comparative \rightarrow \backslash compar(x, y, z) : T)$$

Examples of comparison are the following:

- $\backslash compar$(beautiful, good, $>$): *'more beautiful than good'*
- $\backslash compar$(beautiful, Mary, $=$): *'as beautiful as Mary'*
- $\backslash compar$(good, the original, $=$): *'as good as the original'*

Axioms 34 and 35 account for the operation we have called *correlation. Correlation* ($\backslash correl$) is found in constructions where two propositions are related by means of an intensivity degree of a $Verb$ or an $AdNoun$ that occurs in the first proposition (e.g. *so-that, such-that*).

Axiom 34: Correlation

$$\forall pqxuvwy(p : Prop \wedge q : Prop \wedge p = uwv \wedge (w : Verb \vee w : AdNoun) \wedge w = \backslash intens(x, y) \rightarrow correl(p, w, q) : Prop)$$

Axiom 35: Correlation

$$\forall pqxuvwzyrst\ ((p : Prop \wedge p = uwv \wedge (w : Verb \vee w : AdNoun) \wedge\ (q : Prop \wedge q = zxy \wedge (x : Verb \vee x : AdNoun) \wedge\ t = \backslash compar(w, r, \texttt{REL}) \wedge d = \backslash compar(x, s, \texttt{REL}) \rightarrow correl(p, t, q, d) : Prop)))$$

Examples of those two types of correlation are the following:

- '(This is) *so big that you cannot carry it.*': $\backslash correl$(This is very big, $\backslash intens$(big, `VERY`), you cannot take it)
- *'The quicker you are, the more they appreciate your work'*: $\backslash correl$(you are quick, $\backslash compar$(quick, >, `REL`), they appreciate your work, $\backslash compar$(appreciate, >, `REL`).

Substantivation ($\backslash substantiv$) and *nominalization* ($\backslash nomin$) transcategorize some categories into *Subst* (*substantives*) and *Noun* (*nouns*), respectively. According to a general principle of language, any linguistic entity can become something that can be spoken about; that is, any category can be nominalized and substantivized. We formalize such general principles with axioms 36 and 37, which account for substantivation and nominalization ($\backslash nomin$), respectively.

Axiom 36: Substantivation

$$\forall x(x : Prop \vee x : Noun \vee x : AdNoun \rightarrow \backslash substantiv(x) : Subst)$$

Axiom 37: Nominalization

$$\forall x(x : Verb \vee x : AdNoun \rightarrow \backslash nomin(x) : Noun)$$

Examples of the two operations above are:

- *'eating'*: $\backslash substantiv(\backslash nomin$(to eat))
- *'emptiness'*: $\backslash nomin$(empty)

A particular case of substantivation is *quotation* ($\backslash quot$) which transforms any sequence of letters into a *Subst*. This particular case of substantivation is formalized in axiom 38.

Axiom 38: Quotation

$$\forall x(x : Cat \rightarrow \backslash quot(x) : Subst)$$

Consider the following sentence:

"He said: 'Tomorrow I will finish the paper"

Here a full proposition works as a substantive. So,

'Tomorrow I will finish the paper': \quot (Tomorrow I will finish the paper)

The last two operations in the GNS framework are *interrogation* (\interr) and *imperativation* (\imperat). Those general syntactic operations transform any descriptive proposition into the corresponding proposition with a communication modality which is interrogative, or imperative respectively. Axioms 39 and 40 account for them.

Axiom 39: Interrogation

$$\forall x(x : Prop \rightarrow \backslash interrog(x) : Prop)$$

Axiom 40: Imperativation

$$\forall x(x : Prop \rightarrow \backslash imperat(x) : Prop)$$

4 An Example

In this section we provide an example of how GNS works. First, we analyze the following English sentence, and then we show that the formal representation of this English sentence –by just translating basic words– can be used to generate equivalent Italian and Spanish sentences.

"The two children ran towards the river with their hands raised till they reached the bank"

The above English sentence is analyzed in the following way (we use a # to indicate the line number):

1. hand
2. to raise
3. child
4. river
5. to run

6. to reach
7. bank
8. \conjug(# 2, `PART, PAST`) = *raised*
9. \modif(#1, \adj(# 8)) = *hand raised*
10. \determ(# 9, `DEF, PLUR`) = *their hands raised*
11. \num(# 3, 2) = *two children*
12. \determ(# 11, `DEF`) = *the two children*
13. \conjug(# 5, `III, PLUR, PAST, IMP`) = *ran*
14. \determ(# 4, `DEF`) = *the river*
15. \loc(# 14, `TOWARDS`) = *towards the river*
16. \complem(# 13, # 15) = *ran towards the river*
17. \complem(# 16, `MANN`, #10) = *ran towards the river with their hands raised*
18. \pred(# 12, # 17) = *the two children ran towards the river with their hands raised*
19. \conjug(# 6, `III, PLUR, PAST, PERF`) = *reached*
20. \determ(# 7, `DEF`) = *the bank*
21. $\backslash complem_{VS}$(# 19, # 20) = *reached the bank*
22. \pred(\deixis(`III, PLUR`) , # 21) = *they reached the bank*
23. \subord(# 18, `UNTIL`, # 22) = *The two children ran towards the river with their hands raised till they reached the bank.*

In the end we have the basic elements and operations shown in Table 3.

BASIC ELEMENTS	**OPERATIONS**
child	\ subord
to run	\ pred
river	\ determ
hand	\ num
to raise	\ complem
to arrive	\ conjug
bank	\ deixis
	\ loc
	\ modif

Table 3: Basic Elements and Operations.

If we now take the above GNS formalization and just translate the so-called basic elements into Italian or Spanish in Table 4 below.

English	**Italian**	**Spanish**
child	bambino	niño
to run	correre	correr
river	fiume	río
hand	mano	mano
to raise	alzare	levantar
to reach	raggiungere	alcanzar
bank	riva	orilla

Table 4: Translation of Basic Elements

we obtain the following two sentences:

- *I due bambini correvano verso il fiume con le mani alzate finché non raggiunsero la riva.*
- *Los dos niños corrieron hacia el río con las manos levantadas hasta que alcanzaron la orilla.*

5 Conclusions

In this paper we have presented a formal theory of general explicative power: General Natural Syntax. The aim behind this framework is to reduce syntactic constructions to a few principles related to their semantic functions but defined independently of semantics. In this regard, it can be related to the so-called *abstract syntax* of programming languages. In fact, GNS establishes the requirements for the construction of terms and the rules for assigning them syntactic categories. It does not deal with the final surface syntactic forms of concrete syntax. It does not try to cope with the particular morphological features of the final linguistic form of the expression resulting from the application of the syntactic operation, but it is interested in a deeper syntactic level: GNS aims to define an *abstract syntax for natural languages.*

Within GNS, many classical notions can be stated formally, in terms of basic categories and basic abstract syntactic operations. GNS has the advantage of simplicity. With a very small number of simple ingredients, GNS can account for general syntactic constructions in any natural language. GNS generalizes and formalizes syntactic phenomena present in every natural language and incorporates them into a formal theory of general explicative power. Although GNS is very recent and still requires further research, if its features are taken into account we feel that it may be suitable in the fields of natural language processing, machine translation and linguistics.

Our future work will focus on two main aspects: 1) basic lexicon and grammemes; and 2) formal and computational work. Regarding the former, we are interested in establishing a basic lexicon and an exact number of grammemes. Regarding the latter, it would be very useful to define an interface to generate GNS formula and to develop a system that can generate sentences from the GNS formula. Besides those two topics, it would be very interesting to go deeper into the possible applications of the framework.

References

1. Baker, C.L. (1997). *English syntax.* Cambridge: MIT Press.
2. Bell, J.L. and M. Machover (1977). *A Course in mathematical logic.* Amsterdam: North-Holland.
3. Downing, A. and Ph. Locke (1995). *A university course in English grammar.* London: Phoenix ELT.
4. Goguen, J.A., J.W. Thatcher and E.G. Wagner (1978). An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh (ed.), *Current trends in programming methodology. IV*, pp. 80-149. Englewood Cliffs: Prentince Hall.
5. Huddleston, R. (1984). *Introduction to the grammar of English.* Cambridge University Press.
6. Manca, V. (1993). Typology and logical structure of natural languages. In K. Sikkel and A. Nijholt (eds.), *Parsing natural language. Proceedings of the 6th Twente Workshop on Language Technology*, pp. 23-36. Twente University.
7. Manca, V. (1995). A logical formalism for intergrammatical representations. In A. Nijholt, G. Scollo, and R. Steetkamp (eds.), *Algebraic methods in language processing. Proceedings of the 10th TWLT & 1st AMiLP Workshop*, pp. 247-254. Twente University.
8. Manca, V. (1996). Toward a logical universal grammar. In V. Manca, *Metagrammatical representations (VI Tarragona Seminar on Formal Syntax and Semantics)*, pp. 53-75. GRLMC Report 11/97, Tarragona.
9. Manca, V. (1998). A metagrammatical logical formalism. In C. Martín-Vide (ed.), *Mathematical and computational analysis of natural language.* Amsterdam: John Benjamins.
10. Manca, V. (1999). Logical splicing in natural languages. In C. Martín Vide, C. (ed.), *Issues in mathematical linguistics*, 131–143. Amsterdam: John Benjamins.
11. Manca, V. (2001). Logical string rewriting. *Theoretical Computer Science.*

12. Manca, V. (2004). String models and string theories. In C. Martíin-Vide, V. Mitrana and Gh. Păun (eds.), *Formal languages and applications*, pp. 439-456, Berlin: Springer,
13. Marcus, S. (1967). *Algebraic linguistics: Analytical models.* London: Academic Press.
14. Montague, R. (1974). *Formal philosophy.* In R.H. Thomason (ed.). London: Yale University Press.
15. Reichenbach, H. (1947). *Elements of symbolic logic.* New York: MacMillan Limited.
16. Rozenberg, G. and Salomaa, A. (eds.) (1997). *Handbook of formal languages.* Berlin: Springer.
17. Salomaa, A. (1973). *Formal languages.* New York: Academic Press.
18. Wardhaugh, R. (1995). *Understanding English grammar. A linguistic approach.* Oxford: Blackwell.