Polarized Unification Grammars

Sylvain Kahane

Polarized Unification Grammars

Sylvain Kahane
Modyco, Université Paris 10
sk@ccr.jussieu.fr

Abstract

This paper proposes a generic mathematical formalism for the combination of various structures: strings, trees, dags, graphs and products of them. The polarization of the objects of the elementary structures controls the saturation of the final structure. This formalism is both elementary and powerful enough to strongly simulate many grammar formalisms, such as rewriting systems, dependency grammars, TAG, HPSG and LFG.

1 Introduction

Our aim is to propose a generic formalism as simple as possible but powerful enough to write real grammars for natural language and to handle complex linguistic structures. The formalism we propose can strongly simulate most rule-based formalisms used in linguistics.[1]

Language utterances are both strongly structured and compositional and the structure of a complex utterance can be obtained by combining elementary structures associated to the elementary units of language.[2] The most simple way to combine two structures A and B is unification, that is, to build a new structure C by partially superimposing A and B and identifying a part of the objects of A with those of B. This idea recalls an old idea, used by Jespersen (1924), Tesnière (1934) or Ajduckiewicz (1935): the sentence is like a molecule whose words are atoms, each word bearing a valence (a linguistic term explicitly borrowed from chemistry) that forces or allows it to meet some other words. Nevertheless, unification grammars cannot directly take into account the fact that some linguistic units are unsaturated in a sense that they must absolutely combine with other structures to form a stable unit. Saturation is ensured by additional mechanisms, such as the distinction of terminal and non-terminal symbols in rewriting systems or by requiring some features to have an empty list as a value in HPSG.

This paper presents a new family of formalisms, *Polarized Unification Grammars* (PUGs). PUGs extend Unification Grammars with an explicit control of the saturation of structures by attributing a *polarity* to each object. Using polarities allows integrating the treatment of saturation in the formalism of the rules. Thus the processing of saturation will pilot the combination of structures during the generation processing. Some polarities are neutral, others are not, but a final structure must be completely neutral. Two non-neutral objects can unify (that is, identify) and form a neutral object (that is, neutralizing each other). Proper unification holds no equivalent.

Polarization takes its source in categorial grammar and subsequent works on resource-sensitive logic (see Lambek's, Girard's or van Benthem's works). Nasr (1995) is among the first to introduce a rule-based formalism using an explicit polarization of structures. Duchier & Thater (1999) propose a formalism for tree description where they put forward the notion of polarity (and they uses the terms of *polarity* and *neutralization*). Perrier (2000) is probably the

[1] A formalism A *strongly simulates* a formalism B if A has a better strong generative capacity than B, that is, if A can generate the languages generated by B with the same structures associated to the utterances of these languages.

[2] Whether a natural language utterance contains one or several structures depends on our point of view. On the one hand it is clear that a sentence can receive various structures according to the semantic, syntactic, morphological or phonological point of view. On the other hand these different structures are not independent from each other and even if they are not structures on the same objects (for instance the semantic units do not correspond one to one to the syntactic units, that is the words) there are links between the different objects of these structures. In other words, considering separately the different simple structures of the sentence does not take into account the whole structure of the sentence, because we lost the interrelation between structures of different levels.

first to develop a linguistic formalism entirely based on these ideas, the *Interaction Grammar*.

PUG is both an elementary formalism (structures simply combine by identifying objects) and a powerful formalism, equivalent to Turing machines and capable of handling strings, trees, dags, *n*-graphs and products of such structures (such as ordered trees).[3] But, above all, PUG is a well-adapted formalism for writing grammars and it is capable of strongly simulating many classic formalisms.

Part 2 presents the general framework of PUG and its system of polarities. Part 3 proposes several examples of PUG and the translation in PUG of rewriting grammars, TAG, HPSG and LFG. We hope that these translations shed light on some common features of these formalisms.

2 Polarities and unification

2.1 Polarized Unification Grammars

Polarized Unification Grammars generate sets of finite structures. A structure is based on *objects*. For instance, for a (directed) graph, objects are *nodes* and *edges*. These two *types* of objects are linked, giving us the proper structure: if X is the set of nodes and U, the set of edges, the graph is defined by two *maps* π_1 and π_2 from U into X which associate an edge with its *source* and its *target*.

Our structures are *polarized*, that is, objects are associated to polarities. The set P of polarities is provided with an operation noted "**.**" and called *product*. The product is commutative and generally associative; (P, **.**) is called the *system of polarities*. A non-empty strict subset N of P contains the *neutral* polarities. A polarized structure is *neutral* if all its polarities are neutral.

Structures are defined on a collection of objects of various types (syntactic nodes, semantic nodes, syntactic edges …) and a collection of maps*: structural maps* linking objects to objects (such as *source* and *target* for edges), *label maps* linking objects to labels and *polarity maps* linking objects to polarities.

Structures combine by *unification*. The unification of two structures A and B gives a new structure A⊕B obtained by "pasting" together these structures and identifying a part of the objects of the first structure with objects of the second structure. When two polarized structures A and B are unified, the polarity of an object of A⊕B obtained by identifying two objects of A and B is the product of their polarities; if the two objects bear the same map, these maps must be identified and their values, unified. (For instance identifying two edges forces us to identify their sources and targets.)

A Polarized Unification Grammar (PUG) is defined by a finite family T of types of objects, a set of maps attached to the objects of each type, a system (P,**.**) of polarities, a subset N of P of neutral polarities, and a finite subset of elementary polarized structures, whose objects are described by T; one elementary structure is marked as the initial one and must be used exactly once. The structures *generated* by the grammar are the neutral structures obtained by combining the initial structure and a finite number of elementary structures. In the derivation process, elementary structures combine successively, each new elementary structure combining with at least one object of the previous result; this ensures that the derived structure is continuous. Polarities are only necessary to control the saturation and are not considered when the strong generative capacity of the grammar is estimated. Polarities belong to the declarative part of the grammar, but they rather play a role in the processing of the grammar.

2.2 The system of polarities

In this paper we will use the system of polarities P = {▪,□,–,+,■} (which are called like this: ■ = *black* = *saturated*, + = *positive*, – = *negative*, □ = *white* = *obligatory context* and ▪ = *grey* = *absolutely neutral*), with N = {▪,■}, and a product defined by the following array (where ⊥ represents the impossibility to combine). Note that ▪ is the neutral element of the product. The symbol – can be interpreted as a *need* and + as the corresponding *resource*.

.	▪	□	–	+	■
▪	▪	□	–	+	■
□	□	□	–	+	■
–	–	–	⊥	■	⊥
+	+	+	■	⊥	⊥
■	■	■	⊥	⊥	⊥

The system {□,■} is used by Nasr (1995), while the system {▪,■,–,+}, noted {=,↔,←,→}, is considered by Bonfante *et al*. (2004), who show advantages of negative and positive polarities for prefiltration in parsing (a set of structures bearing negative and positive polarities can only

[3] A *dag* is a directed acyclic graph. An *n*-graph is a graph whose nodes are edges of a (*n*-1)-graph and a 1-graph is a standard graph.

be reduced into a neutral structure if the sum of negative polarities of each object type is equal the sum of positive polarities).

The system (P, **.**) we have presented is commutative and associative. Commutativity implies that the combination of two structures is not procedurally oriented (and we can begin a derivation by any elementary structure, provided we use only once the initial structure). Associativity implies that the combination of structures is unordered: if an object B must combine with A and C, there is no precedence order between the combination of A and B and the one of B and C, that is, A⊕(B⊕C) = (A⊕B)⊕C.

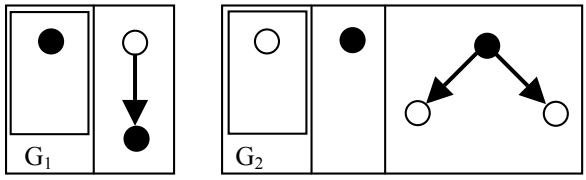
If we leave polarities aside, our formalism is trivially monotonic: the combination of two structures A and B by a PUG gives us a structure A⊕B that contains A and B as substructures. We can add a (partial) order on P in order to make the formalism monotonic.[4] Let ≤ be this order. In order to give us a monotonic formalism, ≤ must verify the following *monotonicity property*: $\forall x,y \in P\ x.y \geq x$. This provides us with the following order: ■ < □ < +/– < ■. A PUG built with an ordered system of polarities (P, **.** ,≤) verifying the monotonicity property is monotonic. Monotonicity implies good computational properties; for instance it allows translating the parsing with PUG into a problem of constraint resolution (Duchier & Thater, 1999).

3 Examples of PUGs

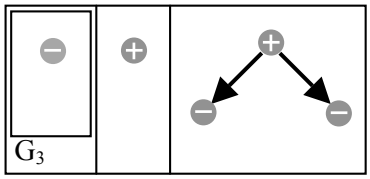
3.1 Tree grammars

The first tree grammars belonging to the paradigm of PUGs was proposed by Nasr 1995. The following grammar G_1 allows generating all finite *trees* (a tree is a connected directed graph such that every node except one is the target of at most one edge); objects are nodes and edges; the initial structure (the box on the left) is reduced to a black node; the grammar has only one other elementary structure, which is composed of a black edge linking a white node to a black node. Each white node must unify with a black node in order to be neutralized and each black node can unify with whatever number of white nodes. It can easily be verified that the structures generated by the grammar are trees, because every node has one and only one governor, except the node introduced by the initial structure, which is the root of the tree.

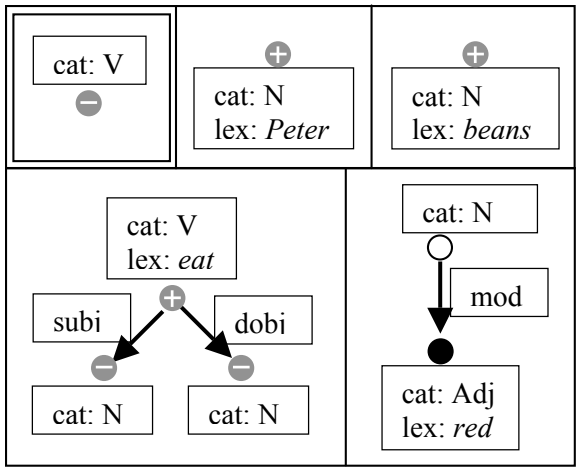
[4] I was suggested this idea by Guy Perrier.



The grammar G_1 does not control the number of dependents of nodes. A grammar like G_2 allows controlling the valence of each node, but it does not ensure that generated structures are trees, because two white nodes can unify and a node can have more than one governor.[5] The grammar G_3 solves the problem. In fact, G_3 can be viewed as the superimposition of G_1 and G_2. Indeed, if P_0 = {□,■}, $P_1 = P_0 \times P_0$ = {(□,□),(□,■),(■,□),(■,■)} is equivalent to {□,+,–,■}. The first polarity controls the tree structure as G_1 does, while the second polarity controls the valence as G_2 does.



With the same principles, one can build a *dependency grammar* generating the syntactic dependency trees of a fragment of natural language. Grammar G_4, directly inspired from Nasr 1995, proposes a fragment of grammar for English generating the syntactic tree of *Peter eats red beans*. Nodes of this grammar are labeled by two label maps, /cat/ and /lex/. Note that the root of



G_4 (Dependency grammar for English)

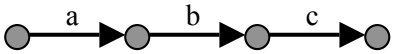
[5] Nasr 1995 proposes such a grammar in order to generate trees. He uses an external requirement, which forces, when two structures are combined, the root of one to combine with a node of the other one.

the elementary structure of an adjective is a white node, allowing an unlimited number of such structures to adjoin to a noun.

3.2 Rewriting systems and ordered trees

PUG can simulate any rewriting system and have the weak generative capacity of Turing machines. We follow ideas developed by Burroni 1993 or Dymetman 1999, themselves following van Kampen 1933's ideas.

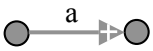
A sequence *abc* is represented by a string of labeled edges *a, b* and *c*:



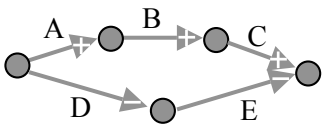
Intuitively, edges are intervals and nodes model their extremities. This is the simplest way to model linear order and precedence rules: X precedes Y iff the end of X is the beginning of Y.

The initial category S of the grammar gives us the initial structure:

A terminal symbol *a* corresponds to a positive edge:

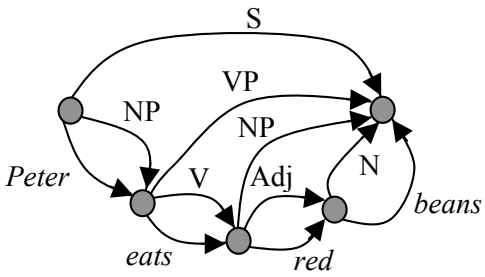


A rewriting rule ABC → DE gives us the elementary structure:

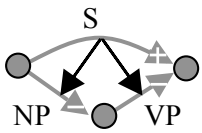


This elementary structure is a "cell" whose upper frontier is a string of positive edges corresponding to the left part of the rule, while the lower frontier is a string of negative edges corresponding to the right part of the rule. Each positive edge must unify with a negative edge and vice versa, in order to give a black edge. Nodes are grey (= absolutely neutral) and their unification is entirely driven by the unification of edges.

Cells will unify with each other to give a final structure representing the derivation structure of a sequence, which is the lower edge of this structure. The next figure shows the derivation structure of the sequence *Peter eats red beans* with a standard phrase structure grammar, which can be reconstructed by the reader. In such a representation, edges represent phrases and correspond to intervals in the cutting of the sequence, while nodes are bounds of these intervals.



For a *context-free rewriting system*, the grammar generates the derivation tree, which can be represented in a more traditional way by adding the branches of the tree (giving us a 2-graph).



Let us recall that a derivation tree for a context-free grammar is an ordered tree. An *ordered tree* combines two structures on the same set of nodes: a structure of tree and a precedence relation on the node of the tree. Here the precedence relation is explicitly represented (a "node" of the tree precedes another "node" if the target of the first one is the source of the second one). It is not possible, with a PUG, to generate the derivation tree, including the precedence relation, in a simpler way.[6] Note that the usual representation of ordered trees (where the precedence relation is not explicit, but only deductible from the planarity of the representation) is very misleading from the computational viewpoint. When they calculate the precedence relation, parsers (of the CKY type for instance) in fact calculate a data structure like the one we present here, where beginnings and ends of phrase are explicitly considered as objects.

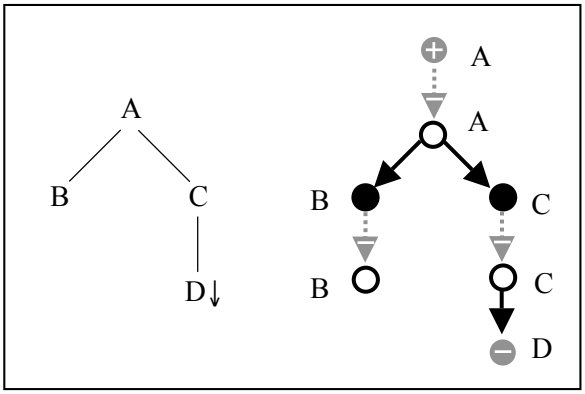
3.3 TAG (Tree Adjoining Grammar)

PUG has a clear kinship with TAG, which is the first formalism based on combination of structures to be studied at length. TAGs are generally presented as grammars combining (ordered) trees. In fact, as a tree grammar, TAG is not

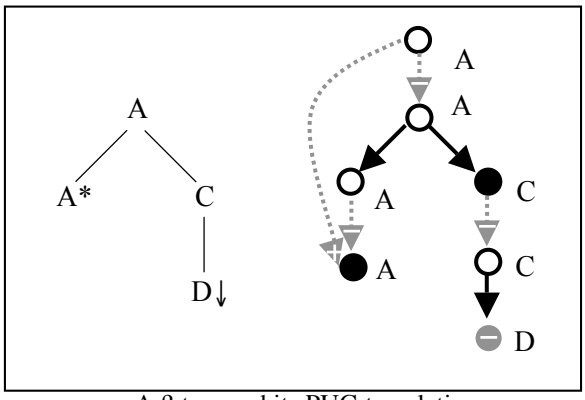
[6] The most natural idea would be to encode a rewriting rule with a tree of depth 1 and the precedence relation with edges from a node to its successor. The difficulty is then to propagate the order relation to the descendants of two sister nodes when we apply a rewriting rule by substituting a tree of depth 1. The simplest solution is undeniably the one presented here, consisting to introduce objects representing the beginning and the end of phrases (our grey nodes) and to indicate the relation between a phrase, its beginning and its end by representing the phrase with an edge from the beginning to the end.

monotonic and cannot be simulated with PUG. As shown by Vijay-Shanker 1992, to obtain a monotonic formalism, TAG must be viewed as a grammar combining quasi-trees. Intuitively, a *quasi-tree* is a tree whose nodes has been split in two parts and have each one an upper part and a lower part, between which another quasi-tree can be inserted (this is the famous adjoining operation of TAG). Formally, a quasi-tree is a tree whose branches have two types: dependency relations and dominance relations (respectively noted by plain lines and dotted lines). Two nodes linked by a negative dominance relation are potentially the two parts of a same node; only the lower part can have dependents.

The next figures give an α-tree (= to be substituted) and a β-tree (= to be adjoined) with the corresponding PUG structures.[7] A substitution node (like D↓) gives a negative node, which will unify with the root of an α tree. A β-tree gives a white root node and a black foot node, which will unify with the upper and the lower part of a split node. This is why the root and the foot node are linked by a positive dominance link, which will unify with a negative dominance link connecting the two parts of a split node.



An α tree and its PUG translation



A β tree and its PUG translation

At the end of the derivation, the structure must be a tree and all nodes must be reconstructed: this is why we introduce the next rule, which presents a positive dominance link linking a node to itself and which will force two semi-nodes to unify by neutralizing the dominance link between them.

This last rule again shows the advantage of PUG: the reunification of nodes, which is procedurally ensured in Vijay-Shanker 1992 is given here as a declarative rule.

3.4 HPSG (Head-driven Phrase Structure Grammar)

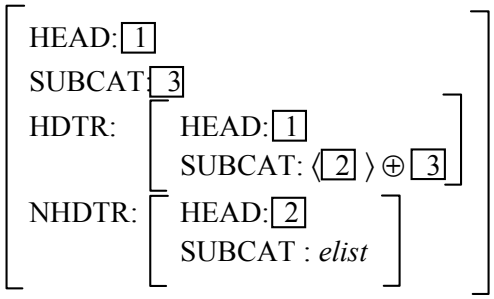
There are two ways to translate feature structures (FSs) into PUG. Clearly atomic values must be labels and (embedded) feature structures must be nodes, but features can be translated by maps or by edges, that is, objects. Encoding features by maps ensures to identify them in PUG. Encoding them by edges allows us to polarize them and control the number of identifications.[8]

For the sake of clarification of HSPG structures, we choose to translate structural features such as HDTR and NHDTR, which give the phrase structure and which never unify with other "features", by edges and other features by maps (which will be represented by hashed arrows). In any case, the result looks like a dag whose "edges" (true edges and maps) represent features and whose nodes represent values (e.g. Kesper & Mönnich 2003). We exemplify the translation of HPSG in PUG with the schema of combination

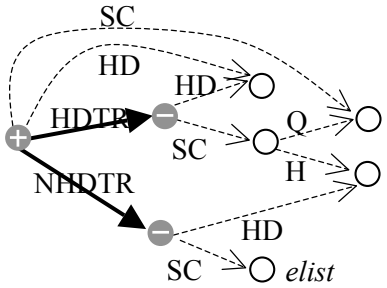
[7] For sake of simplicity, we leave aside the precedence relation on sister nodes. It might be encoded in the same way as context-free rewriting systems, by modeling semi-nodes of TAG trees by edges. It does not pose any problem but would make the figures difficult to read.

[8] Perrier 2000 uses a feature-structure based formalism where only features are polarized. Although more or less equivalent we prefer to polarize the FS themselves, i.e. the nodes.

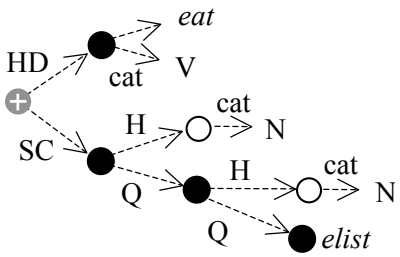
of head phrase with a subcategorized sister phrase, namely the *head-daughter-phrase*:[9]



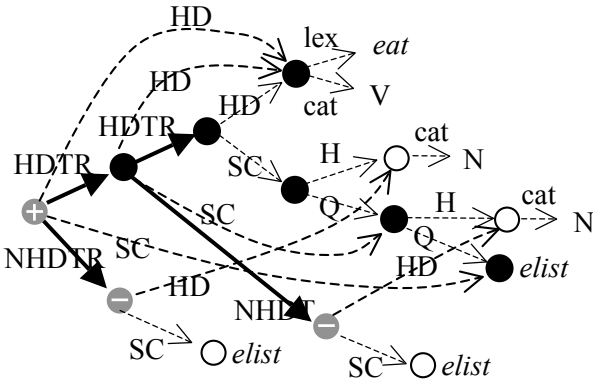
This FS gives the following structure, where a list is represented recursively in two pieces: its head (value of H) and its queue (value of Q).



A negative node of this FS can be neutralized by the combination with a similar FS representing a phrase or with a lexical entry. The next figure proposes a lexical entry for *eat*, indicating that *eat* is a V whose SUBCAT list contains two phrases headed by an N (for sake of simplicity we deal with the subject as a subcategorized phrase).



The combination of two *head-daughter-phrases* with the lexical entry of *eat* gives us the previous lexicalized rule, equivalent to the rule for *eat* of the dependency grammar G_4 (/subj/ is the NHDTR of the maximal projection and /obj/ the NHDTR of the intermediate projection of *eat*).

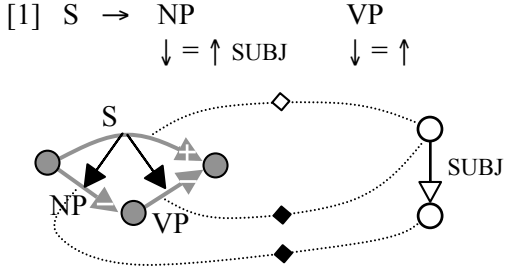


Polarization of objects shows exactly what is constructed by each rule and what are the requests filled by other rules. Moreover it allows us to force SUBCAT lists to be instantiated (and therefore allows us to control the saturation of the valence), which is ensured in the usual formalism of HPSG by a bottom-up procedural presentation.

3.5 LFG (Lexical Functional Grammar) and synchronous grammars

We propose a translation of LFG into PUG that makes LFG appear as a synchronous grammar approach (see Shieber & Schabes 1990). LFG synchronizes two structures (a phrase structure or c-structure and a dependency/functional structure or f-structure) and it can be viewed as the synchronization of a phrase structure grammar and a dependency grammar.

Let us consider a first LFG rule and its translation in PUG:



Equations under phrases (in the right side of [1]) ensure the synchronization between the objects of the c-structure and the f-structure: each phrase is synchronized with a "functional" node. Symbols ↓ and ↑ respectively designate the functional node synchronized with the current phrase and the one synchronized with the mother phrase (here S). Thus the equation ↓=↑ means that the current phrase (VP) and its mother (S) are synchronized with the same functional node. The

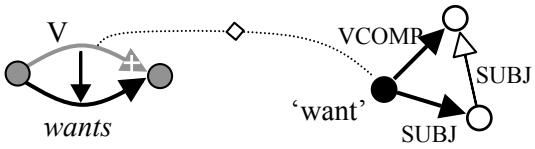
[9] Numbers in boxes are values shared by several features. The value of SUBCAT (= SC) is a list (the list of subcategorized phrases). The non-head daughter phrase (NHDTR) has a saturated valence and so needs an empty SUBCAT list (*elist*). The subcat list of the head daughter phrase (HDTR) is the concatenation, noted ⊕, of two lists: a list with one element that is the description of the non-head daughter phrase and the SUBCAT list of the whole phrase. The rest of the description of this phrase (value of HEAD) is equal to the one of the head daughter phrase.

expression ↑ SUBJ designates the functional node depending on ↑ by the relation SUBJ.

In PUG we model the synchronization of the phrases and the functional nodes by *synchronization links* (represented by dotted lines with diamond-shaped polarities) (see Bresnan 2000 for non-formalized similar representations). The two synchronizations ensured by the two constraints ↓=↑ SUBJ and ↓=↑ of [1], and therefore built by this rule, are polarized in black.

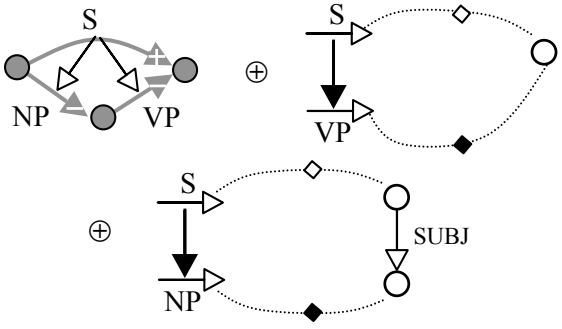
A phrasal rule such as [1] introduces an f-structure with a totally white polarization. It will be neutralized by lexical rules such as [2]:

[2] V → *wants*
↑ PRED = 'want ⟨SUBJ,VCOMP⟩'
↑ SUBJ = ↑ VCOMP SUBJ



The feature Pred is interpreted as the labeling of the functional node, while the valence ⟨SUBJ,VCOMP⟩ gives us two black edges and two white nodes. The functional equation ↑SUBJ = ↑ VCOMP SUBJ introduces a white edge SUBJ between the nodes ↑ SUBJ and ↑VCOMP (and is therefore to be interpreted very differently from the constraints of [1], which introduce black synchronization links.)

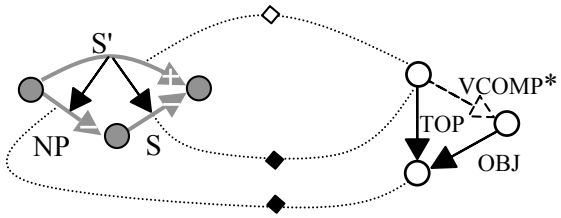
PUG allows to easily split up a rule into more elementary rules. For instance, the rule [1] can be split up into three rules: a phrase structure rules linearizing the daughter phrases and two rules of synchronization indicating the functional link between a phrase and one of its daughter phrases.



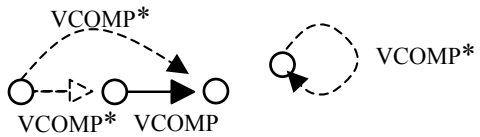
Our decomposition shows that LFG articulated two different grammars: a classical phrase structure generating the c-structure and an interface grammar between c- and f-structures (and even a third grammar because the f-structure is really generated only by the lexical rules). With PUG it is easy to join two (or more) grammars: it suffices to add on the objects by both grammars a white polarity that will be saturated in the other grammar (and vice versa) (Kahane & Lareau 2005).

Let us consider another problem, illustrated here by the rule for the topicalization of an object. The unbounded dependency of the object with its functional governor is an undetermined path expressed by a regular expression (here VCOMP* OBJ; functional uncertainty, Kaplan & Zaenen 1989).

[3] S' → NP S
↓ = ↑ VCOMP* OBJ ↓ = ↑
↓ = ↑ TOP



The path VCOMP* (represented by a dashed arrow) is expanded by the following regular grammar, with two rules, one for the propagation and one for the ending.



Again the translation into PUG brings to the fore some fundamental components of the formalism (like synchronization links) and some non-explicit mechanisms such as the fact that the lexical equation ↑ PRED = 'want ⟨SUBJ,VCOMP⟩' introduces both resources (a node 'want') and needs (its valence).

4 Conclusion

The PUG formalism is extremely simple: it only imposes that combining two structures involves at least the unification of two objects. Forcing or forbidding more objects to combine is then entirely controlled by polarization of objects. Polarization will thus guide the process of combination of elementary structures. In spite of its simplicity, the PUG formalism is powerful enough to elegantly simulate most of the rule-based formalisms used in formal linguistics and NLP. This sheds new light on these formalisms and allows us to bring to the fore the exact nature

of the structures they handle and to extract some procedural mechanisms hidden by the formalism. But above all, the PUG formalism allows us to write separately several modules of the grammar handling various structures and to put them together in a same formalism by synchronization of the grammars, as we show with our translation of LFG. Thus PUGs extend unification grammars based on feature structures by allowing a greatest diversity of geometric structures and a best control of resources. Further investigations must concern the computational properties of PUGs, notably restrictions allowing polynomial time parsing.

Acknowledgements

I thank Benoît Crabbé, Denys Duchier, Kim Gerdes, François Lareau, François Métayer, Piet Mertens, Guy Perrier, Alain Polguère and Benoît Sagot for their numerous remarks and enlightening commentaries.

References

Ajdukiewicz K. 1935. Die syntaktische Konnexität. *Studia Philosophica,* 1:1-27.

Bonfante G., Guillaume B., Perrier G. 2004. Polarization and abstraction of grammatical formalisms as methods for lexical disambiguation. *Proceedings of CoLing*, Genève, Switzerland, 303-309.

Bresnan J. 2001. *Lexical-Functional Syntax*. Blackwell.

Burroni A. 1993. Higher-dimensional word problems with applications to equational logic. *Theoretical Computer Sciences*, 115:43-62.

Duchier D., Thater S. 1999. Parsing with tree descriptions: a constraint-based approach. *NLULP 1999 (Natural Language Understanding and Logic Programming)*, Las Cruces, NM.

Dymetman M. 1999. Some Remarks on the Geometry of Grammar. *Proceedings of MOL'6 (6th Meeting on the Mathematics of Language)*, Orlando, FA.

Girard J.-Y. 1987. Linear logic. *Theoretical Computer Sciences,* 50(1):1-102.

Kahane S., Lareau F. 2005. Meaning-Text Unification Grammar: modularity and polarization, *Second International Conference on Meaning-Text Theory*, Moscow, 197-206.

Kaplan R.M., Zaenen A. 1989. Long-distance dependencies, constituent structure, and functional uncertainty. In *Alternative Conceptions of Phrase Structure*, M. Baltin & A. Kroch (eds), 17-42, Chicago Univ. Press.

Kepser S., Mönnich U. 2003. Graph properties of HPSG feature structures. *Proceedings of Formal Grammar*, Vienna, Austria, 115-124.

Nasr A. 1995. A formalism and a parser for lexicalised dependency grammars. *Proceedings of the 4th Int. Workshop on Parsing Tecnologies*, State Univ. of NY Press.

Perrier G. 2000. Interaction Grammars. *Proceedings of CoLing*, Sarrebrücken, 7 p.

Shieber S. M. & Schabes Y. 1990. Synchronous tree-adjoining grammars, *Proceedings of CoLing*, vol. 3, 253-258, Helsinki, Finland.

Tesnière L. 1934. Comment construire une syntaxe. *Bulletin de la Faculté des Lettres de Strasbourg,* 7: 219-229.

van Kampen E. 1933. On some lemmas in the theory of groups. *American Journal of Mathematics,* 55:268–73.

Vijay-Shanker K. 1992. Using description of trees in a Tree Adjoining Grammar. *Computational Linguistics*, 18(4):481-517.