Using inheritance in Object-Oriented Programming to combine syntactic rules and lexical idiosyncrasies

Benoît HABERT
Ecole Normale Supérieure de Fontenay Saint Cloud
31 avenue Lombart
F-92260 FONTENAY-AUX-ROSES FRANCE
internet: bh@litp.ibp.fr
bitnet: bh@frunip61.bitnet
Phone: (33) 1-47-02-60-50 Ext 415
Fax: (33) 1-47-02-34-32

ABSTRACT

In parsing idioms and frozen expressions in French, one needs to combine general syntactic rules and idiosyncratic constraints. The inheritance structure provided by Object-Oriented Programming languages, and more specifically the combination of methods present in CLOS, Common Lisp Object System, appears as an elegant and efficient approach to deal with such a complex interaction.

In parsing idioms and frozen expressions in French, one needs to combine general syntactic rules and idiosyncratic constraints. As a matter of fact, representing such an interaction via an inheritance lattice appears as an elegant and efficient approach. For the sake of explanation, English idioms will be used as examples. However this combination of syntactic rules and idiosyncratic behaviour via manipulations of the inheritance structure and the methods attached to it, has been designed for French compound adverbials. More than 6,000 compound adverbials have been listed and studied at LADL 1 (Gross, 1990). A lexicon-grammar of 2,525 compound adverbials coming from the LADL files has been used in parsing a test corpus of 72,000 words.

IDIOMS: A PECULIAR COMBINATION OF REGULARITIES AND IDIOSYNCRASIES

The semantics of idioms will not be accounted for here, since it is a controversial problem. LFG, GPSG and TAGs made quite different claims on this topic [2]. Within a syntactic category, it has been shown for French, at LADL, that frozen expressions are generally more numerous than 'free' ones: 20, 000 frozen verbs (12,000 free), 6,000 adverbials (1,500 free). More than 25, 000 compound nouns have been studied so far, but their number is far greater, as they constitute the major part of new terms in sublanguages (Grishman & Kittredge, 1986). A small proportion of frozen expressions have constituents existing only in such contexts (such as "umbrage" in "to take umbrage at NP"), or are taken from foreign languages ("a priori"), or follow forgotten rules. Apart from these marginal cases, idioms consist of the same words as the free phrases, and they follow the same syntactic rules: "in contrast", "by the way", for instance, are just ordinary PP. Furthermore, as shown for English by Wasow et al., 1982, and for French by Gross, 1988, 1990, the syntactic behaviour of idioms is much more systematic than is usually thought: 'transformations' apply to them. Some kind of 'meta-rules' must be then used to account for these related structures.

While following to a large extent the general syntactic rules, frozen expressions present idiosyncrasies. At a syntactic level, an idiom can accept a modifier ("in (loving)

1 Laboratoire d'Automatique Documentaire et Linguistique: Université Paris 7 and CNRS.

2 For Bresnan, 1982b, constituants of an idiom very often have a regular syntactic behaviour without contributing at all to the meaning of the whole expression. According to Gazdar et al.,, 1985, p 236-242, the semantic behaviour of idioms is more often compositional than has generally been assumed,. The approach of (Abeillé & Schabes, 1989) characterizes idioms by the combination of syntactic regularity and semantic non compositionality.

memory of"), or not (#"by the new way") [3]. It can require certain syntactic features for some of its constituents. For instance, it may need a certain type of determiner: "for the sake of" versus "#for a sake of". Lastly, an idiom is associated with fixed lexical items. Usually it is not possible to replace them by synonyms: #"by the road" versus "by the way". Since most of frozen expressions follow general syntactic rules, and since 'transformations' apply to them, it is not reasonable to try and process them in a first lexical step. Recognizing idioms belongs therefore to the whole syntactic analysis. Nevertheless their idiosyncratic features must be taken into account in rules.

STATING THE GENERAL BEHAVIOUR OF A FAMILY OF IDIOMS

OLMES [4] is a general parser written in CLOS [5] (Keene, 1989; Steele, 1990, p 770-864), and tested with the Victoria Day implementation of PCL [6] (provided by Xerox Laboratories), using Lucid Common Lisp 3.0.1, on a Sun 3 workstation, at LITP 7. OLMES belongs to the active chart parser family. The input text can be parsed from left to right, or the other way round, or even both ways at the same time (around pivots). Top-down, bottom-up or bottom-up then top-down strategies are available. The rules used by OLMES follow the formalism created for PATR-II (Shieber, 1986), because it is a kind of "lingua franca" for unification-based grammars. Additional constraints can be associated with ordinary context-free rules so as to analyse mildly context-sensitive languages (Gazdar, 1988). Each symbol in the rule is the root of a Directed Acyclic Graph (DAG). In such category structures, each edge is labelled, and leads either to an atom or to another complex category structure (Gazdar et al., 1988).

[3] We use the same convention as (Gazdar et al., 1985): '#' indicates that a structure is acceptable, but with a literal meaning.

[4] Objects, Language, Means for Exploring and Structuring (Texts).

[5] Common Lisp Object System.

[6] Portable Common Loops

7 Laboratoire d'Informatique Théorique et de Programmation: Université Paris 6, Université Paris 7 and CNRS.

For instance, a lot of adverbials in English use the following rule, in PATR-II form:

```
        LHS -> RHS1 RHS2 RHS3
        <LHS cat> = adv
        <RHS1 cat> = prep
        <RHS2 cat> = det
        <RHS3 cat> = noun
        <RHS2  agreement>  =  <RHS3
agreement>
```

The sequence of a first right-hand side symbol dominating a DAG with an edge "cat(egory)" having "prep(osition)" as its value, a second symbol with "cat" "det(erminer)", and a third symbol with "noun" as "cat" makes an "adv(erbial)". Additionally the second and the third symbol must share the same value for the feature "agreement".

A graphical equivalent could be:

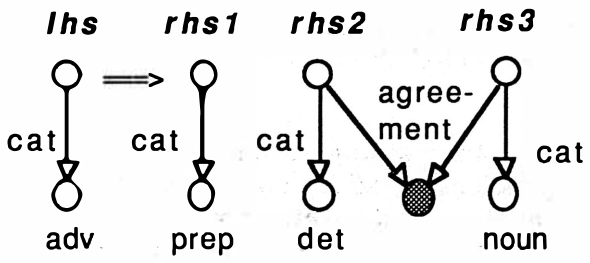


Figure 1: sequence of DAGs defining an adverbial

In the lexicon, one can find entries 8 such as:

```
a
   cat det
   cat-precisions
      determiner-type article
      article-type indefinite
at
   cat prep
by
   cat prep
in
   cat prep
end
   cat noun
   agreement
      number singular
```

8 The features not relevant for the rule are not mentioned.

```
moment
   cat noun
   agreement
      number singular
my
   cat det
   cat-precisions
   determiner-type possessive
the
   cat det
   cat-precisions
      determiner-type article
      article-type definite
this
   cat det
   cat precisions
      determiner-type demonstrative
those
   cat det
   cat-precisions
      determiner-type demonstrative
   agreement
      number plural
way
   cat noun
   agreement
      number singular
```

The rule above would recognize as idioms "at the moment", "in a way", "in the end", using this toy lexicon. Note that the completed rule is more restrictive than the context-free part of it. The latter would accept "*by those way", the former would not, because "those" and "way" do not agree.

THE GRAMMAR: A NETWORK OF ACTIVE AGENTS ENCAPSULATING CONSTRAINTS

The context-free rules of the grammar are represented by a network of classes. Each class in the network corresponds to an occurrence of a symbol, whether terminal or not, appearing in the grammar. The topology of the network mirrors exactly the strategy (top-down versus bottom-up) and the direction of exploration (left-right, right-left or bi-directional) chosen by the user when compiling the grammar. This approach extends the work done within the actor paradigm by Yonesawa & Ohsawa, 1990.

There are two main classes: active and inactive. An inactive agent corresponds to a (possibly partial) constituent which has been found. For instance, for each left-hand side symbol in the grammar, a class is created inheriting from the inactive agent class. The active agents correspond to the right-hand side symbols of the grammar. Each of them is searching for a constituent meeting certain constraints, as defined in the corresponding DAG in the rule. If it finds such a constituent, it then creates an instance of the class corresponding to the following symbol in the right-hand side part of the rule. When the last active agent of the rule "succeeds", it creates an instance of the class corresponding to the left-hand side of the rule. The pivot of the rule is the symbol starting the whole analysis. It need not be the left-most one.

For the rule above, in bottom-up parsing, four classes are defined: `LHS-1`, `RHS1-2`, `RHS2-3`, `RHS3-4`, respectively (figure 2). `RHS1-2`, `RHS2-3` and `RHS3-4` are subclasses of `LHS-1`, their instances will be active agents examining the text from right to left. The pivot of the rule is the class `RHS3-4` (in bold font), corresponding to a noun.

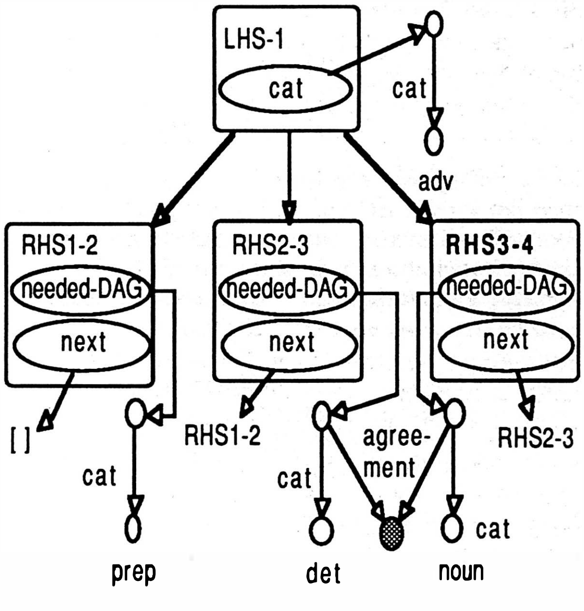


Figure 2: classes resulting from the compilation

To indicate that a word can belong to the type of idiom described in the rule, the lexicon associates the class-name `RHS3-4` with this word. It could be the case for the word "moment". In a bottom-up analysis, for each occurrence of "moment" in the input text, OLMES creates an instance of RHS3-4. This instance searches for a noun, and finds it: "moment". It creates an instance of RHS2-3 which examines the word on the left of "moment", and which stores a partial parse tree. If this word is a determiner, and has a feature "agreement" matching with the corresponding feature of "moment", the new partial parse tree is transmitted to the instance of RHS1-2 which is then created and whose constraints are matched against the word on the left of the determiner found by the instance of RHS2-3. In the case that the instance of RHS1-2 finds a preposition, it then creates an instance of LHS-1 storing the complete parse tree and the additional information gathered from the unification on the rest of the DAGs.

Changing the grammar rules from sequences of 'passive' labels to a network of active classes makes it possible to increase as necessary the knowledge the instances of these classes can utilise, and to use inheritance not only in the lexicon (Shieber, 1986), but in the grammar rules as well.

USING THE INHERITANCE STRUCTURE TO TAKE IDIOSYNCRASIES INTO ACCOUNT

The rule stated above is not restrictive enough. For instance, it would parse as an idiom "by a way" in the sentence: "he arrived by a way new to me". It would be rather an unsatisfactory approach to create as many rules as combinations found between the preposition and the type of determiner used in such idioms. What we need instead is a means to adjoin new constraints to the set of conditions defined in the rule, in a modular way, that is, using inheritance. In the CLOS philosophy, it means that some 'mixin' classes are created. Such classes are not intended to have instances on their own. On the contrary, they are only used as constituents (super-classes) in defining more specialized classes.

For instance, one can define the following 'mixin' classes (see figure 3). Each 'mixin' class used to specialize the rule has a method constraints which states particular constraints on the determiner. The content of this method (in PATR form) follows the class name, below.

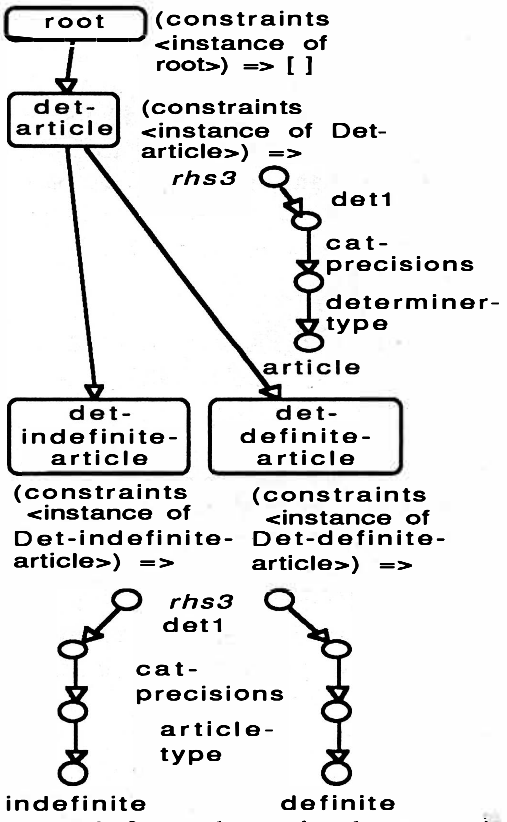
```
det-article
   <RHS3 det1 cat-precisions
determiner-type> = article

det-definite-article (subclass of
det-article)
   <RHS3 det1 cat-precisions
article-type> = definite

det-indefinite-article (subclass of
det-article)
   <RHS3 det1 cat-precisions
article-type> = indefinite

det-possessive
   <RHS3 det1 cat-precisions
determiner-type> = possessive

det-demonstrative
   <RHS3 det1 cat-precisions
determiner-type> = demonstrative
```


root
(constraints <instance of root>) => [ ]
det-article
(constraints <instance of Det-article>) =>
rhs3
det1
cat-precisions
determiner-type
article
det-indefinite-article
det-definite-article
(constraints <instance of Det-indefinite-article>) =>
(constraints <instance of Det-definite-article>) =>
rhs3
det1
cat-precisions
article-type
indefinite
definite
Figure 3: Some classes for the constraints on determiners

The rule given above (figure 1) is slightly redefined : from now on, the pivot transmits to the RHS1 the form of preposition, and to the RHS2 precisions on the type of determiner which is needed (the dark nodes indicate this sharing of values in figure 4).

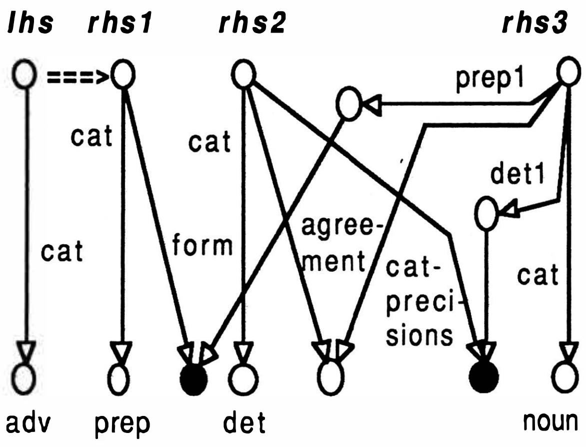


Figure 4: Redefined rule for adverbials

It is now possible to create final classes for the pivots of the idioms:

- `adv=prep_det-definite-article_noun`, subclass of `RHS3-4` and `det-definite-article`. E.g.: by the way.

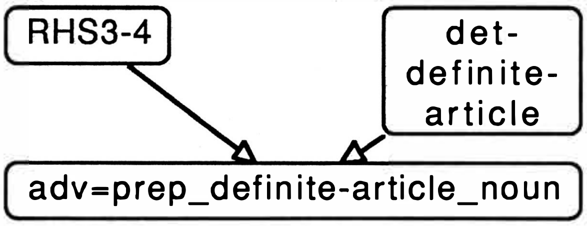


Figure 5: An example of final class

- `adv=prep_det-indefinite-article_noun`, subclass of `RHS3-4` and `det-indefinite-article`. E.g.: in a way.
- `adv=prep_det-demonstrative-noun`, subclass of `RHS3-4` and `det-demonstrative`. E.g.: in this respect.
- `adv=prep_det-possessive_noun`, subclass of `RHS3-4` and `det-possessive`. E.g.: in my opinion.

Of course, it could have been possible to define mixin classes to deal with constraints on the preposition. Such classes would have looked like:

```
prep-in
```

```
   <RHS1 prepl form> = in
prep-by
   <RHS1 prepl form> = by
```
and so on.

It should be noted that the constraints on the preposition and the conditions on the determiner are not on the same level. The latter are in a way more syntactic: some syntactic properties of the idiom as a whole depend on the nature of the determiner. The form of the initial preposition is purely idiosyncratic. It does not even always contribute to the meaning of the expression. For that reason, the way to specify the initial preposition does not use inheritance. A list of associations *{<parameter><value>}* is being used at the initialization of the instance of a given pivot class to deal with such litteral constraints. For instance, the list "RHS1 by" will trigger the adjunction of:

```
   <RHS1 prepl form> = by
```

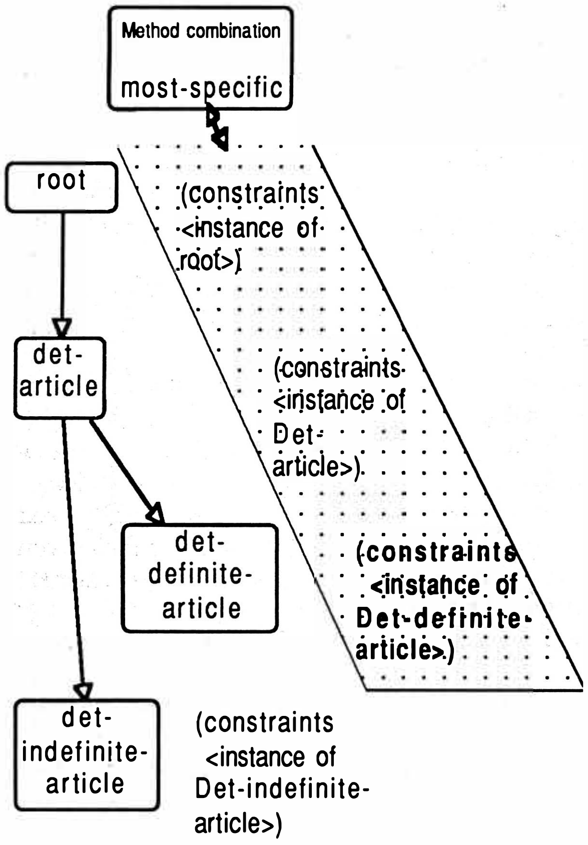
to the conditions specified for the idiomatic rule.

Usually, in an Object-Oriented programming language, when a method is called for an instance of a class, and there are different methods of the same name linked with the ancestors of this class, the most specific method is actually used, overriding the other ones. For instance, calling `(constraints det-definite-article-1)`, `det-definite-article-1` being an instance of `det-definite-article`, would yield 9:

```
   <RHS3 det1 cat-precisions
article-type> = definite
```

As shown in figure 6, three methods are applicable (inside the grey frame): `constraints` of `Det-definite-article` (`det-definite-article-1` is an instance of this class), `constraints` of `Det-article` (an instance of `Det-definite-article is a Det-article`) and `constraints` of `root` (for the same reason). The last one (in bold font) shadows the others.

9 We do not use the actual Lisp syntax for the result, as it is not relevant.



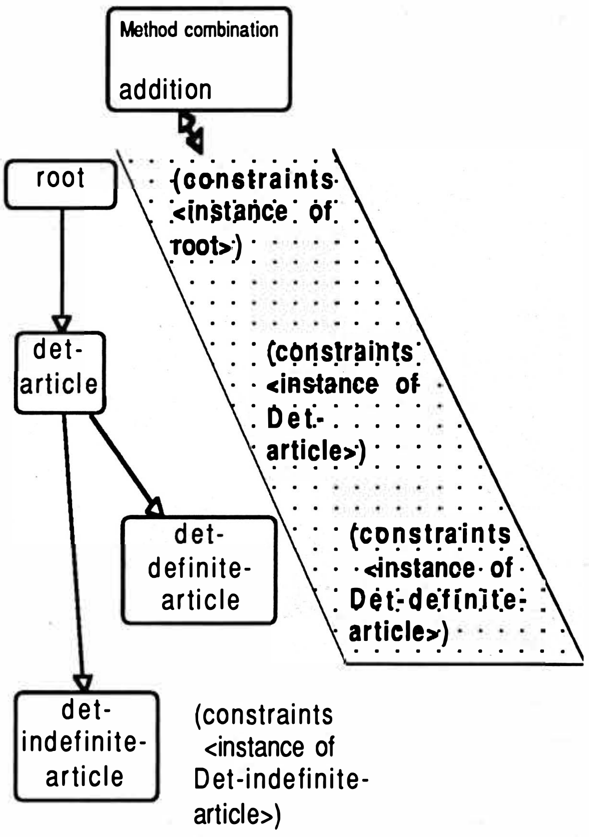
Method combination
most-specific
root
det-article
det-definite-article
det-indefinite-article
(constraints <instance of root>)
(constraints <instance of Det-article>)
(constraints <instance of Det-definite-article>)
(constraints <instance of Det-indefinite-article>)

Figure 6: Standard method combination

One of the salient characteristics of CLOS, inherited from its ancestors, COMMON LOOPS (Bobrow et al., 1986) and NEW FLAVORS, is the control given over the combination of methods having the same name and present in the super-classes of a given class (Keene, 1989) [10]. In this case, it is possible to specify that all the methods `constraints` accessible from a given class should be called in turn, and the final result should be the addition of all the returned values. With this combination of methods, the result of the function call `(constraints det-definite-article-1)` would be (figure 7):

```
   <RHS3 det1 cat-precisions
determiner-type> = article
```

```
   <RHS3 det1 cat-precisions
article-type> = definite
```



Method combination
addition
root
det-article
det-definite-article
det-indefinite-article
(constraints <instance of root>)
(constraints <instance of Det-article>)
(constraints <instance of Det-definite-article>)
(constraints <instance of Det-indefinite-article>)

Figure 7: Special method combination

This method combination provides a means to add constraints present in the inheritance lattice, and only the relevant ones.
In the lexicon, the entries for pivots of adverbials could mention (among other information):

moment
```
   adv=prep_definite-article_noun
RHS1 at
```

opinion
```
   adv=prep_possessive_noun RHS1 in
```

way
```
   adv=prep_definite-article_noun
RSH1 by
   adv=prep_indefinite-article_noun
RHS1 in
```

When coming across "way" in the input text, OLMES would therefore create one instance of each class. For example, in the case of the instance of `adv=prep_definite-`

[10] When the most specific method represents nothing but an addition to the action of one super-method, it is generally possible in an Object-Oriented Programming Language to combine it with this super-method, so as to share common behaviours.

article_noun, because this class is a subclass of det-definite-article, and because the method combination for constraints is redefined, the constraints inherited via det-definite-article and det-article are added to the general constraints defined in the rule and inherited through RHS3-4. The arguments following the name of the class are used as well. In the end, the parser will actually try the following rule (figure 8):

```
   LHS -> RHS1 RHS2 RHS3
   <LHS cat> = adv
   <RHS1 cat> = prep
   <RHS1 form> = <RHS3 prep1 form>
   <RHS2 cat> = det
   <RHS3 cat> = noun
   <RHS3 prep1 form> = by
   <RHS2 agreement> = <RHS3
agreement>
   <RHS2 cat-precisions> =
<RHS3 det1 cat-precisions>
   <RHS3 det1 cat-precisions
determiner-type> = article
   <RHS3 det1 cat-precisions
article-type> = definite
```

(The basic constraints are in normal font, the inherited ones in bold font, and the parametrized ones are underlined.)

This rule will accept "by the way", but will reject "by a way", "in the way" ... The rule and the parameters for "moment" would allow the parsing of "at the moment", and those for "opinion" the acceptance of "in my opinion"...

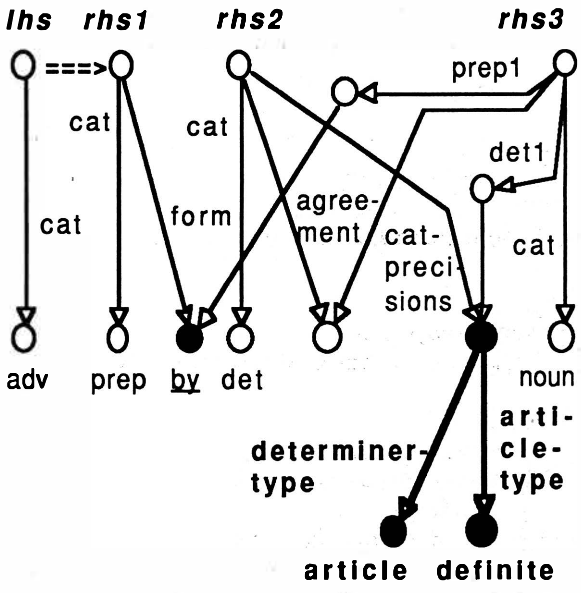


Figure 8: Sequence of DAGs for the final rule

This very simple example does not do justice to the complexity of syntactic and lexical properties of adverbial idioms. However it stresses the hierarchy of these features, and the way in which the inheritance graph can, at the same time, mirror this structure and take advantage of it. Note that the 'mixin' classes defined above are useful as such. They constitute the primitives to complete the basic syntactic rules. They correspond to organized constraints which are interesting on their own, as they can be reused in different contexts. For instance, another rule for adverbials is:

```
   LHS -> RHS1 RHS2 RHS3 RHS4
   <LHS cat> = adv
   <RHS1 cat> = prep
   <RHS1 form> = <RHS3 prep1 form>
   <RHS2 cat> = det
   <RHS3 cat> = noun
   <RHS2 agreement> = <RHS3
agreement>
   <RHS2 cat-precisions> = <RHS2
det1 cat-precisions>
   <RHS4 cat> = prep
   <RHS4 form> = <RHS3 prep2 form>
```

A class would be created for each symbol of the rule. For example, the class corresponding to the pivot (the noun) is RHS4-9. New specialized classes are then defined:

- adv=prep_definite-article_noun_prep (super-classes: RHS4-9, det-definite-article)
- adv=prep_indefinite-article_noun_prep (super-classes: RHS4-9, det-indefinite-article)...

And the lexicon now has entries like:

```
eye
   adv=prep_indefinite-
article_noun_prep RHS1 with RHS4 to

form
   adv=prep_definite-
article_noun_prep RHS1 in RHS4 of
```

which could recognize "with an eye to" and "in the form of", respectively. It is possible in OLMES to express that a certain word can enter different linked syntactic structures at the same time, thus providing 'meta-rules'. One of theses families of rules is the class: [adv = prep definite-article noun prep + adv = prep possessive-determiner noun] gathering the following rules

```
   adv=prep_definite-
article_noun_prep
   adv=prep_det-possessive_noun
```

And, in the lexicon, the entry time mentions:

```
    [adv = prep definite-article
noun prep + adv = prep possessive-
determiner noun] RHS1 for RHS4 of
```

Therefore, the parser, when finding time in the input text, creates an instance of the class [adv = prep definite-article noun prep + adv = prep possessive-determiner noun] , which in turn creates an instance of adv=prep_definite-article_noun_prep and an instance of adv=prep_det-possessive_noun. This instance of [adv = prep definite-article noun prep + adv = prep possessive-determiner noun] also transmits to them the correct values for the parameters RHS1 and RHS4, possibly leading to the parsing of for the sake of or for its sake.

RELATED WORK: PARSING IDIOMS IN TREE ADJOINING GRAMMARS (TAGS)

Recent work, within the TAG formalism (Abeillé, 1990; Abeillé & Schabes, 1989), claimed that idioms should be parsed during the whole syntactic analysis, using the same formal devices as for parsing non idiomatic expressions. This approach uses a slightly modified version of TAGs, namely lexicalized TAGs, in which each 'rule', i.e. each tree, is anchored with a lexical item. In fact, there are no separate phrase structure rules any more: they are collapsed into the lexicon. Only the relevant rules are used while parsing, as they are triggered by lexical items. Meta-rules are provided by means of families of trees. The trees corresponding to idioms include several lexical items: take and bucket in the case of to take the bucket . As an additional filtering, the search of an idiom is triggered only if all the lexical heads are actually present in the sentence, in the right order.

As a matter of fact, giving the rules a pivot and associating the words in the lexicon with these pivots, as shown above, is a first step in lexicalizing a grammar. On the other hand, Lexicalized TAGs do not use phrase structure rules any more, but trees directly stating to any depth the constituents needed, their structure, and possibly their lexical heads. For this very reason, the TAG formalism deals with idioms in a more natural and powerful way. For the sake of explanation, the rules given in this paper are flattening the structure of the phrases. In order to give to the relevant idioms the same structure as the corresponding free phrases, one would need some complex transmission of features among related rules (Habert, 1991).

STRUCTURING GRAMMARS VIA INHERITANCE

Relying to such an extent on the inheritance structure partly breaks the decentralization rule which is central to object oriented programming 11. When slightly modifying a class, here is a risk of triggering a

11 (Meyer, 1988, p 251) In most cases, clients of a class should not need to know the inheritance structure that led to its implementation.

chain reaction of changes. As Sakkinen, 1989, states:

> Features aiming at "exploratory programming" need not necessarily make the programmer into a Vasco de Gama or an Amundsen; (s)he may well become Alice in Wonderland, never knowing what metamorphoses some seemingly innocent act may cause.

The danger is a real one. Nevertheless, so far, it has been most beneficial to take advantage of the inheritance structure to portray the linguistic knowledge we are dealing with. In doing so, we stress the classification tools present in Objet-Oriented Programming Languages: the inheritance lattice is used to progressively constrain the class of the solution (Wegner, 1987). This approach uses a unification-based formalism with a clear-cut distinction between phrase structure rules and subcategorization frames. In spite of this, it combines properly the generalizations stated by the syntactic rules and additionnal constraints necessary to account for the idiosyncrasies that the idioms show. This solution is by no means limited to frozen expressions. It contributes to a clear expression of the complex interactions found in the grammar between syntactic and lexical rules (Abeillé 90). It is thus worth investigating the ways in which inheritance can help in structuring not only the lexicon but also the grammar.

AKNOWLEDGMENTS

I greatly benefited from discussions with F.-X. Testard-Vaillant (LITP), Pierre Fiala (ENS de Fontenay Saint Cloud), and Anne Abeillé (LADL) on Object-Oriented Programming, Idioms and TAGs respectively.

REFERENCES

Abeillé Anne
1990 "Lexical and syntactic rules in a tree adjoining grammar", ACL'90

Abeillé Anne, Yves Schabes
1989 "Parsing Idioms in Lexicalized TAGs", EACL'89.

Bresnan Joan
1982a (editor) The mental representation of grammatical representation, The MIT Press.
1982b "The passive in Lexical Theory", (Bresnan, 1982a, p 2-86).

Bobrow Daniel G., Kenneth Kahn, Gregor Kickzales, Larry Masinter, Mark Stefik and Frank Zdybel
1986 "CommonLoops: merging Lisp and Object-Oriented Programming, OOPSLA'86.

Gazdar Gerald
1988 "Applicability of Indexed Grammars to natural languages", in Natural language parsing and linguistic theories, Reyle and Rohrer editors, D. Reidel Publishing Company.

Gazdar Gerald, Ewan Klein, Geoffrey Pullum, Ivan Sag
1985 Generalized Phrase Structure Grammar, Harvard University Press.

Gazdar Gerald, Mellish Chris
1989 Natural Language Processing in Lisp, Addison-Wesley.

Gazdar Gerald, Geoffrey K. Pullum, Robert Carpenter, Ewan Klein, Thomas E. Hukari, Robert D. Levine
1988 "Category structures", Computational Linguistics, Vol. 14, #1.

Grishman Ralph, Richard Kittredge (editors)
1986 Analyzing language in restricted domains: sublanguage description and processing, Lawrence Erlbaum Associates.

Gross Maurice
1988 "Sur les phrases figées complexes du français", Langue Française 77.
1990 Grammaire transformationnelle du français: 3 - syntaxe de l'adverbe, ASTRIL.

Habert Benoît
1990 "Controlling the generic dispatch to represent domain knowledge", Proceedings of the third CLOS users and implementors workshop, OOPSLA'90.
1991 Langages à objets et analyse linguistique, Doctoral thesis.

Keene Sonya E.
1989 Object-Oriented Programming in Common Lisp, Addison Wesley.

Kay Martin
1985 "Parsing in Functional Unification Grammar", in Natural language parsing, D. Dowty, L. Karttunen and A. Zwicky editors, Cambridge University Press.

Kickzales Gregor, Luis Rodriguez
1990 "Efficient method dispatch in PCL", Proceedings of the 1990 conference on Lisp and Functional Programming.

Meyer Bertrand
1988 Object-Oriented Software Construction, Prentice Hall.

Pollard Carl, Ivan A. Sag
1987 Information-based syntax and semantics, Vol 1: Fundamentals, CSLI.

Sakkinen Markku
1989 "Disciplined Inheritance", ECOOP'89

Shieber Stuart
1986 An introduction to unification-based approaches to grammar, CSLI.

Steele Guy L.
1990 Common Lisp: The Language, 2nd edition, Digital Press.

Wegner Peter
1987 "The Object-Oriented Classification Paradigm", in Research Directions in Object-Oriented Programming, The MIT Press.

Wasow Thomas, Ivan A. Sag, Geoffrey Nunberg
1982 "Idioms: an interim report", Proceedings of the 13th International Congress of Linguists,

Yonesawa Akinori, Ichiro Ohsawa
1990 "Object-Oriented Parallel Parsing for Context-Free Grammars", in ABCL: an Object-Oriented Concurrent System, Akinori Yonezawa editor, The MIT Press.