Coreference handling in XMG

Claire Gardent
CNRS/LORIA
615, rue du jardin botanique, B.P. 101
54602 Villers lès Nancy CEDEX
France
Claire.Gardent@loria.fr

Yannick Parmentier
INRIA Lorraine
615, rue du jardin botanique, B.P. 101
54602 Villers lès Nancy CEDEX
France
Yannick.Parmentier@loria.fr

Abstract

We claim that existing specification languages for tree based grammars fail to adequately support identifier managment. We then show that XMG (*eXtensible MetaGrammar*) provides a sophisticated treatment of identifiers which is effective in supporting a linguist-friendly grammar design.

1 Specifying tree-based grammars

Whilst the development of standard unification-based grammars is well supported by the design of formalisms such as PATR-II, Ale or TDL (Krieger and Schafer, 1994), the situation is less well established for Tree-Based Grammars such as Tree Adjoining Grammars (Joshi and Schabes, 1997), Tree Description Grammars (Kallmeyer, 1996) or Interaction Grammars (Perrier, 2003).

Roughly, two main types of specification formalism for Tree-Based Grammars can be distinguished: formalisms based on tree fragments and non monotonic inheritance and formalisms based on tree descriptions and monotonic inheritance.

The tree fragment approach is advocated in (Evans et al., 1995) which proposes to encode lexicalised TAGs using the DATR representation language[1]. In this approach, tree fragments are combined within a non monotonic inheritance hierarchy. Furthermore, new fragments can be derived from existing ones by means of lexical rules. This first approach suffers from the procedural character of non-monotonic inheritance. In specifying the grammar, the grammar writer must keep in mind the order in which non-monotonic statements have been made so as to be able to predict how explicit statements interact with defaults and non-monotonic inheritance in determining the final output. When developing a large coverage grammar, this rapidly become extremely cumbersome. Moreover, as (Candito, 1996) remarks, non-monotonicity may result in an information loss which makes it impossible to express the relation existing for instance between an active object and the corresponding passive subject.

The approach based on tree descriptions (often called, the *metagrammar approach*) obviates the procedural character of the non-monotonic approach by taking tree descriptions rather than trees to be the basic units (Candito, 1996; Xia et al., 1999; Vijay-Shanker and Schabes, 1992). In essence, tree fragments are described using tree descriptions and tree descriptions are combined through conjunction or inheritance. The idea is that the minimal models satisfying the resulting descriptions are TAG elementary trees. In some cases, lexical rules are also used to derive new trees from existing ones.

One main drawback with this second type of approach concerns the management of node identifiers. Either nodes are represented by nameless variables and node identification is forced by well-formedness constraints e.g., *wff*-constraints on trees and *wff*-constraints given by the input tree description (cf. e.g., (Duchier and Gardent, 1999)) or nodes are named and nodes with identical names are forced to denote the same entity. The first option is unrealistic when developing a large core grammar as it is easy to omit a necessary constraint and thereby permit overgeneration (the description will be satisfied by more trees than intended). The second option greatly degrades

[1] A tree based approach is also used in(Becker, 2000) but this time in combination with metarules. In that particular approach, procedural aspects also come into play as the order in which metarules apply affect the results.

modularity as the grammar writer must remember which names were used where and with which interpretation. As we shall see below, it also has the undesirable effect that the same tree fragment cannot be used twice in a given tree description. Nevertheless, this is the option that is adopted in most grammar formalisms and grammar compilers (Candito, 1996; Xia et al., 1999; Gaiffe et al., 2002).

In this paper, we present an approach which remedies these shortcomings by combining monotonic inheritance of tree descriptions with an *explicit management of identifier scope and identifiers equality*[2]. The proposed approach thus eschews both the inconvenients induced by a non monotonic framework (by using tree descriptions rather than trees) and those resulting from a global treatment of identifiers (by providing greater expressivity wrt identifiers).

Specifically, we show that the proposed approach supports several ways of identifying (node or feature) values, we motivate this multiplicity and we identify the linguistic and/or technical criteria for choosing among the various possibilities.

The paper starts in section 2 by introducing the syntax of the XMG formalism. In section 3, we show that XMG provides four different ways of identifying two (node or variable) identifiers. In section 4, we motivate each of these four different ways and indicate when each of them can and should be used.

2 The XMG formalism

We start by briefly introducing XMG (eXtended MetaGrammar). First, we show that it supports the description and the combination of *blocks* consisting of tree fragments and/or semantic representations. Then, we show that it supports a sophisticated treatment of identifiers.

2.1 Defining blocks

At the syntactic level, the basic units are tree descriptions which are specified using the following tree logic:

$$\begin{aligned} \textit{Description} ::= \; & x \rightarrow y \mid x \rightarrow^{+} y \mid x \rightarrow^{*} y \mid \\ & x \prec y \mid x \prec^{+} y \mid x \prec^{*} y \mid \\ & x[f{:}E] \mid x = y \mid \\ & \textit{Description} \wedge \textit{Description} \end{aligned} \tag{1}$$

where x, y represent node variables, $\rightarrow$ immediate dominance (x is directly above y), $\rightarrow^{+}$ strict dominance (x is above y), and $\rightarrow^{*}$ large dominance[3] (x is above or equal to y). Similarly $\prec$ denotes immediate precedence, $\prec^{+}$ strict precedence, and $\prec^{*}$ large precedence. Finally $x[f{:}E]$ constrains feature f with associated expression E on node x, and $x = y$ indicates node identification.

The XMG formalism also supports the association of semantic representations with elementary trees. The semantic representation language is a flat semantic representation language (Bos, 1995) with the following syntax:

$$\begin{aligned} \textit{Description} ::= \; & \ell{:}p(E_1, ..., E_n) \mid \\ & \neg\ell{:}p(E_1, ..., E_n) \mid E_i \ll E_j \\ & \textit{Description} \wedge \textit{Description} \end{aligned} \tag{2}$$

where ℓ is a label, p is a predicate and $E_1, .., E_n$ are parameters. Further, $\neg$ denotes negation and $E_i \ll E_j$ expresses a scope constraint between E_i and E_j (E_j is in the scope of E_i).

2.2 Combining blocks

As in other existing tree-based formalisms, in XMG, blocks can be combined using inheritance. However, XMG additionally supports block conjunction and block disjunction.

Specifically, a *Class* associates a name with a content:

$$\textit{Class} \quad ::= \quad \textit{Name} \rightarrow \{\, \textit{Content} \,\} \tag{3}$$

A *Content* is either a *Description* (i.e., a tree description, a semantic formula or both), a class name, a conjunction or a disjunction of class name:

$$\begin{aligned} \textit{Content} \quad ::= \quad & \textit{Description} \mid \textit{Name} \mid \\ & \textit{Name} \vee \textit{Name} \mid \textit{Name} \wedge \textit{Name} \end{aligned} \tag{4}$$

Further, XMG allows *multiple* inheritance: a given class can *import* or inherit one or more classes (written C_i here):

[2] Recently, (Villemonte de la Clergerie, 2005) has proposed a highly compact representation formalism for tree-based grammars which also features explicit identifier management. His approach differs from ours in that it includes neither a colouring mechanism (cf. section 3.4) nor interfaces (cf. section 3.3).

[3] By large, we mean the transitive reflexive closure of dominance.

$$\textit{Class} \quad ::= \quad \textit{Name} \angle C_1 \wedge \ldots \wedge C_n \rightarrow \{\, \textit{Content} \,\} \qquad (5)$$

The semantic of the *import* instruction is to include the description of the *imported* class within the current one. This makes it possible to refine a class e.g., by adding information to a node or by adding new nodes[4].

2.3 Managing identifiers

We now introduce the treatment of identifiers supported by XMG. We show in particular, that it integrates:

- a convenient way of managing identifier scope based on *import/export declarations* inspired from standard Object Oriented Programming techniques (section 2.3.1);
- an alternative means of identifying feature values based on the use of *unification*
- *polarity*- (here called *colour*-) based node identification as first proposed in (Muskens and Krahmer, 1998) and later used in (Duchier and Thater, 1999; Perrier, 2000).

The next sections will detail the linguistic and technical motivations behind this variety of identifier handling techniques.

2.3.1 Import/Export declaration

In XMG, the default scope of an identifier is the class in which it is declared. However, *export* specifications can be used to extend the scope of a given identifier outside its declaration class. The export of identifier ?X ouside class A is written :[5]

$$\mathtt{A}_{?\mathtt{X}} \quad \rightarrow \quad \{\ldots ?\mathtt{X} \ldots\}$$

Export declarations interact with inheritance, conjunction and disjunction specifications as follows (where A, B, C are classes):

Inheritance: if the class A is *imported* either directly or indirectly by a class B, then ?X is visible in B. In case of multiple inheritance e.g., if $\mathtt{B} \angle C_1 \wedge \ldots \wedge C_n$, then all identifiers exported by $C_1 \wedge \ldots \wedge C_n$ are visible from B *provided they have distinct names*. In other words, if two (or more) classes in $C_1 \wedge \ldots \wedge C_n$ export the *same* identifier ?X, then ?X is not directly visible from B. It can be accessed though using the dot operator. First A is identified with a local identifier (e.g., ?T = A), then ?T.?X can be used to refer to the identifier ?X exported by A.

Conjunction: if classes A and B are *conjoined* inside a class C, then all the identifiers exported by A or B are visible within C using the *dot* operator.

Disjunction: if classes A and B are *disjoined* inside a class C, then all the identifiers exported by A or B are visible within C using the *dot* operator. However in this case, both A and B have to be associated with the same local identifier.

In sum, export/import declarations permit extending/restricting the scope of an identifier within a branch of the inheritance hierarchy whilst the dot operator allows explicit access to an inherited identifier in case the inheriting class either displays multiple inheritance or is combined by conjunction or disjunction with other classes. More specifically, inheritance allows implicit coreference (the use of an imported name ensures coreference with the object referred to when declaring this name) and the dot operator explicit coreference (through an explicit equality statement e.g., ?A.?X = ?B.?Y).

2.3.2 Class interface

In XMG, a class can be associated with a **class interface** i.e., with a feature structure. Furthermore, when two classes are related either by inheritance or by combination (conjunction or disjunction), their interfaces are *unified*. Hence class interfaces can be used to ensure the unification of identifiers across classes.

Here is an illustrating example:

$$\begin{array}{lcl} \mathtt{A} & \rightarrow & \{\ldots ?\mathtt{X} \ldots\} * = [n1 = ?\mathtt{X}] \\ \mathtt{B} & \rightarrow & \{\ldots ?\mathtt{Y} \ldots\} * = [n1 = ?\mathtt{Y}] \end{array}$$

In A (resp. B), the *local* identifier ?X (resp. ?Y) is associated with an interface feature named $n1$. If

[4] Note that disjunctive inheritance is not supported which would allow a block to be defi ned as importing one or more classes from a given set of imported classes

[5] Similarly, *import* declaration can be used to restrict the set of accessible identifi ers to a subset of it.

these two classes are combined either by conjunction or by inheritance, their interfaces are unified and as a result, the local identifiers ?X and ?Y are unified. In the case of a disjunction, the interface of the current class (C here) is non deterministically unified with that of A or B.

In practice, interface-based identification of values is particularly useful when two distinct features need to be assigned the same value. In (Gardent, 2006) for instance, it is used to identify the semantic index associated with e.g., the subject node of a verbal tree and the corresponding semantic index in the semantic representation associated with that tree.

2.3.3 Colouring nodes

Finally, XMG provides a very economical way of identifying node variables based on the use of *colours* (also called *polarities* in the literature). The idea is that node variables are associated with a specific colour and that this colouring will either prevent or trigger node identifications based on the following identification rules:

	$\bullet_B$	$\bullet_R$	$\circ_W$	$\bot$
$\bullet_B$	$\bot$	$\bot$	$\bullet_B$	$\bot$
$\bullet_R$	$\bot$	$\bot$	$\bot$	$\bot$
$\circ_W$	$\bullet_B$	$\bot$	$\circ_W$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

and on the requirement that valid trees only have red or black nodes. In effect, node colouring enforces the following constraints : (i) a white node must be identified with a black node, (ii) a red node cannot be identified with any other node and (iii) a black node may be identified with one or more white nodes.

Contrary to other means of value identification, colours are restricted to node identifiers. Hence they are best used to induce node identification in those contexts where neither inheritance nor explicit identification are appropriate (see section 4).

3 XMG at work

Recall (section 1) that one main problem when developing a factorised specification of tree based grammars is to ensure a consistent treatment of identifiers and in particular, of identifier unification. That is, when combining two units of information, the grammar writer must ensure that her specification correctly states which objects are the same and which are distinct.

In what follows, we show that XMG supports four different ways of identifying objects. We illustrate this by demonstrating that the following tree can be obtained in four different ways:

Figure 1: A tree that can be derived in four ways

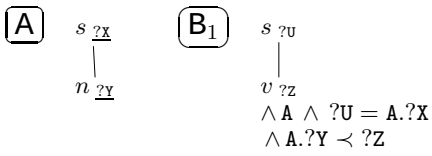
In section 4, we will show that these four ways of identifying nodes and/or features values support both explicitness and economy thereby reducing the risks of specification errors.

3.1 Using explicit identification

The most basic way to identify two identifiers is to explicitly state their identity. Thus the above tree can be produced by combining the following two classes[6] :

$$\begin{aligned} \mathtt{A}_{?X,?Y} \rightarrow \quad & \{\, ?\mathtt{X}\,[cat : s] \rightarrow ?\mathtt{Y}\,[cat : n]\,\} \\ \mathtt{B}_1 \rightarrow \quad & \{\, ?\mathtt{U}\,[cat : s] \rightarrow ?\mathtt{Z}\,[cat : v] \\ & \wedge\ \mathtt{A} \wedge\ ?\mathtt{U} = \mathtt{A}.?\mathtt{X}\ \wedge\ \mathtt{A}.?\mathtt{Y} \prec ?\mathtt{Z}\,\} \end{aligned}$$

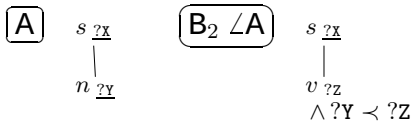
To improve readability, we use from now on a graphical representation. For instance, the classes above are represented as follows (exported identifiers are underlined and boxed letters indicate class names):



Thus, the class A describes the left branch of the tree in Figure 1 and the class $\mathtt{B}_1$ its right branch. The root of A and B are named ?X and ?U respectively. Since ?X is exported, ?X is visible in $\mathtt{B}_1$. The explicit identification ?U=A.?X then enforces that the two roots are identified thus constraining the solution to be the tree given in Figure 1.

3.2 Using inheritance

Using inheritance instead of conjunction, the same nodes identification can be obtained in a more economical way. We reuse the same class A as before, but we now define a class $\mathtt{B}_2$ as a sub-class of A:



[6]Here and in what follows, we abbreviate the conjunction of a class identifi cation ?T = A and a dot notation T.?X to A.?X. That is,

$$?\mathtt{T} = \mathtt{A} \wedge \mathtt{T}.?\mathtt{X} \rightarrow_{abbrev} \mathtt{A}.?\mathtt{X}$$

Since the identifiers ?X and ?Y are exported by A, they are visible in B_2. Thus, in the latter we only have to indicate the precedence relation between ?Y and ?Z.

In sum, the main difference between explicit identification and identification through simple exports, is that whilst inheritance of exported identifiers gives direct access to these identifiers, class combination requires the use of a prefix and dot statement. Note nevertheless that with the latter, identifiers conflicts are a lot less likely to appear.

3.3 Using interfaces

A third possibility is to use interfaces to force node identifications as illustrated in figure 2.

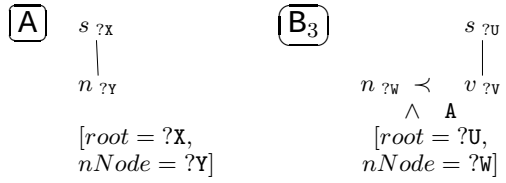
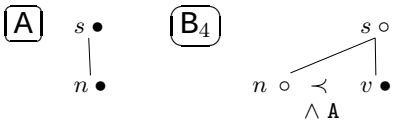


Figure 2: Structure sharing using interfaces

Class A is the same as before except that the identifiers ?X and ?Y are no longer exported. Instead they are associated with the interface features $root$ and $nNode$ respectively. Similarly, class B_3 associates the identifiers (?U and ?V) with the interface features $root$ and $nNode$. As the tree fragment of class B_3 is conjoined with A, the interface features of A and B_3 are unified so that ?X is identified with ?U and ?Y with ?V.

3.4 Using node colours

Finally, colours can be used as illustrated in the Figure below:



Now, class B_4 contains three nodes: two white ones whose categories are s and n and which must be identified with compatible black nodes in A; and a black node that may but need not be identified with a white one. To satisfy these constraints, the black s node in A must be identified with the white s node in B and similarly for the n nodes. The result is again the tree given in Figure 1.

Note that in this case, none of the identifiers need to be exported. Importantly, the use of colours supports a very economical way of forcing nodes identification. Indeed, nodes that are identified through colouration need neither be exported nor even be named.

4 Which choice when?

As shown in the previous section, XMG allows four ways of identifying values (i.e., nodes or feature values): through simple exports, through explicit identification, through colour constraints and through the interface. We now identify when each of these four possibilities is best used.

4.1 Exports

As shown in section 2.3, an identifier ?X can be explicitly exported by a class C with the effect that, within all classes that inherit from C, all occurrences of ?X denote the same object.

In essence, exports supports variable naming that is global to a branch of the inheritance hierarchy. It is possible to name an identifier within a given class C and to reuse it within any other class that inherits from C. Thus the empirical difficulty associated with the use of exported identifiers is that associated with global names. That is, the grammar writer must remember the names used and their intended interpretation. When developing a large size grammar, this rapidly makes grammar writing, maintenance and debugging an extremely difficult task. Hence global identifiers should be use sparingly.

But although non trivial (this was in fact one of the main motivations for developing XMG), this empirical limitation is not the only one. There are two additional formal restrictions which prevent a general use of exported identifiers.

First, as remarked upon in (Crabbe and Duchier, 2004), global names do not support multiple use of the same class within a class. For instance, consider the case illustrated in Figure 3.

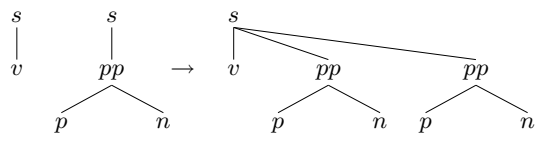


Figure 3: Case of double prepositional phrase.

In this case, the aim is to produce the elementary tree for a verb taking two prepositional arguments such as *parler à quelqu'un de quelque chose* (*to tell someone about something*). Ideally, this is done by combining the verbal fragment on the left

with two occurrences of the PP class in the middle to yield the tree on the right. However if, as is likely in a large size metagrammar, any of the pp, the p or the n node bears an exported identifier, then the two occurrences of this node will be identified so that the resulting tree will be that given in (4).

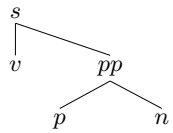


Figure 4: Double prepositional phrase with exported identifiers.

We will see below how colours permit a natural account of such cases.

Second, exported modifiers do not support identifier unification in cases of conjunction, disjunction and multiple inheritance. That is, in each of the three cases below, the various occurrences of ?X are not identified.

$$\mathsf{C}_1\ ?\mathtt{x} \wedge \mathsf{C}_2\ ?\mathtt{x}$$
$$\mathsf{C}_1\ ?\mathtt{x} \vee \mathsf{C}_2\ ?\mathtt{x}$$
$$\mathsf{C}_3\ ?\mathtt{x} \measuredangle \mathsf{C}_1\ ?\mathtt{x} \wedge \mathsf{C}_2\ ?\mathtt{x}$$

In such cases, the multiple occurrences of ?X need to be explicitly identified (see below).

In practice then, the safest use of simple exports (ie without explicit identifier equalities) consists in using them

- in combination with inheritance only and
- within a linguistically motivated subpart of the inheritance hierarchy

4.2 Colours

As discussed in section 2.3, node identifications can be based on colours. In particular, if a node is white, it *must* be identified with a black node.

The main advantage of this particular identification mechanism is that it is extremely economical. Not only is there no longer any need to remember names, there is in fact no need to chose a name. When developing a metagrammar containing several hundreds of nodes, this is a welcome feature.

This "no-name" aspect of the colour mechanism is in particular very useful when a given class needs to be combined with many other classes. For instance, in SEMFRAG (Gardent, 2006), the semantic index of a semantic functor (*i.e.*, a verb, an adjective, a preposition or a predicative noun) needs to be projected from the anchor to the root node as illustrated in Figure 5. This can be done, as shown in the figure by conjoining C_{Sem} with C_V or C_A and letting the colour unify the appropriate nodes.

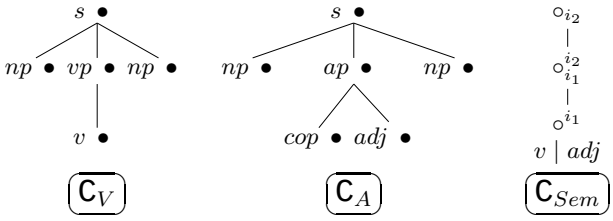


Figure 5: Case of semantic projections.

Colouring also solves the problem raised by the multiple reuse of the same class in the definition of a given class. The colouring will be as shown in Figure 6. Since the pp, p and n nodes are black, their two occurrences cannot be identified. The two white s nodes however will both be unified with the black one thus yielding the expected tree.

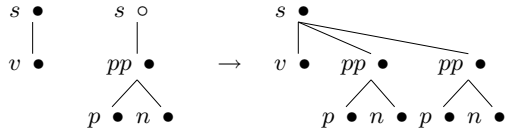


Figure 6: Case of double prepositional phrase with coloured descriptions.

As for exports however, colours cannot always be used to force identifications.

First, colours can only be used in combination with conjunction or inheritance of non exported identifiers. Indeed, inheritance does not allow the identification of two different objects. Hence if a class C containing a white node named ?X inherits from another class C' exporting a black node also named ?X, compilation will fail as a given identifier can only have one colour[7]. In contrast, when solving a description containing the conjunction of a black and a white node (where these two nodes have either no names or distinct names), the well formedness constraint on coloured tree will ensure that these two nodes are in fact the same (since a tree containing a white node is ill formed).

Second, colour based identification is non deterministic. For instance, in Figure 5, if the lowest

[7]However, different occurrences of the same unnamed node can have distinct colours.

node b of $\mathtt{C}_{Sem}$ was not labelled $cat = v \mid adj$, $\mathtt{C}_A \wedge \mathtt{C}_{Sem}$ would yield not one but two trees: one where b is identified with the cop node and the other where it is identified with the adj one. In other words, colour based unification is only possible in cases where node decorations (or explicit node identifications) are sufficiently rich to constrain the possible unifications.

To sum up, colours are useful in situations where:

- a given class needs to be combined with many other classes – in this case it is unlikely that the names used in all classes to be combined are consistent (ie that they are the same for information that must be unified and that they are different for information that must not) and
- the nodes to be identified are unambiguous (the white and the black nodes contain enough information so that it is clear which white node must be identified with which black one)

4.3 Interfaces

Interfaces provide another mechanism for global naming. They are particularly useful in cases where two fundamentally different objects contain non-node identifiers that must be unified.

Recall (cf. section 4.2) that exported identifiers are best used within restricted, linguistically well defined hierarchies. In a case where the objects containing the two identifiers to be identified are different, these will belong to distinct part of the inheritance hierarchy hence identifier export is not a good option.

Node colouring is another possibility but as the name indicates, it only works for *nodes*, not for feature values.

In such a situation then, interfaces come in handy. This is the case for instance, when combining a semantic representation with a tree. The semantic formula and the tree are distinct objects but in the approach to semantic construction described in (Gardent and Kallmeyer, 2003), they share some semantic indices. For instance, the subject node in the tree is labelled with an index feature whose value must be (in an active form tree) that of the first argument occurring in the semantic representation. The encoding of the required coreference can be sketched as follows:

$$\begin{aligned}
\texttt{Subj} &\rightarrow \{\ldots \texttt{?X} \ldots\} * = [subjectIdx = \texttt{?X}] \\
\texttt{Sem} &\rightarrow \{\ldots \texttt{?Y} \ldots\} * = [arg1 = \texttt{?Y}] \\
\texttt{Tree} &\rightarrow \texttt{Subj} * = [subjectIdx = \texttt{?Z}] \wedge \\
&\qquad \texttt{Sem} * = [arg1 = \texttt{?Z}]
\end{aligned}$$

The first two lines show the naming of the identifiers `?X` and `?Y` through the interface, the third illustrates how unification can be used to identify the values named by the interface: since the same variable `?Z` is the value of the two features $arg1$ and $subjectIdx$, the corresponding values in the `Subj` and `Sem` classes are identified.

4.4 Explicit identification of exported identifiers

The explicit identification of exported identifiers is the last resort solution. It is not subject to any of the restrictions listed above and can be combined with conjunction, disjunction and inheritance. It is however uneconomical and complexifies grammar writing (since every node identification must be explicitly declared). Hence it should be used as little as possible.

In practice, explicit identification of exported identifiers is useful :

- to further constrain colour based identification (when the feature information present in the nodes does not suffice to force identification of the appropriate nodes)
- to model general principles that apply to several subtrees in a given hierarchy

The second point is illustrated by Subject/Verb agreement. Suppose that in the metagrammar, we want to have a separate class to enforce this agreement. This class consists of a subject node `?SubjAgr` bearing agreement feature `?X` and of a verb node `?VerbAgr` bearing the same agreement feature. It must then be combined with all verbal elementary trees described by the metagrammar whereby in each such combination the nodes `?SubjAgr, ?VerbAgr` must be identified with the subject and the verb node respectively. This is a typical case of multiple inheritance because both the subject and the verb nodes are specified by inheritance and `?SubjAgr, ?VerbAgr` must be further inherited. Since nodes must be identified and multiple inheritance occur, simple identifier exports cannot be used (cf. section 2.3.1). If colours cannot be sufficiently

	Pros	Cons	Practice
Export	Economy	Name management Not with multiple inheritance Not with conjunction Not with disjunction Not with multiple reuse	Use in linguistically motivated sub-hierarchy
Colours	Economy ++ Multiple reuse OK	Non deterministic Not with inheritance and identically named identifi ers	Use when a given class combines with many classes
Interface	Global	Name management	Use for Syntax/Semantic interface
Explicit identifi cation	Usable in all cases	Uneconomical	Last Resort solution

Figure 7: Summary of the pros and cons of sharing mechanisms.

constrained by features, then the only solution left is explicit node identification.

Figure 7 summarises the pros and the cons of each approach.

5 Conclusion

In this paper, we have introduced a specification formalism for Tree-Based Grammars and shown that its expressivity helps solving specification problems which might be encountered when developping a large scale tree-based grammar.

This formalism has been implemented within the XMG system and successfully used to encode both a core TAG for French (Crabbe, 2005; Gardent, 2006) and a core Interaction Grammar (Perrier, 2003). We are currently exploring ways in which the XMG formalism could be extended to automatically enforce linguistically-based well-formedness principles such as for instance, a kind of Head Feature Principle for TAG.

References

T. Becker. 2000. Patterns in metarules. In A. Abeille and O. Rambow, editors, *Tree Adjoining Grammars: formal, computational and linguistic aspects*. CSLI publications, Stanford.

J. Bos. 1995. Predicate Logic Unplugged. In *Proceedings of the 10th Amsterdam Colloquium, Amsterdam*.

M.H. Candito. 1996. A principle-based hierarchical representation of LTAGs. In *Proceedings of COLING'96, Kopenhagen.*

B. Crabbe and D. Duchier. 2004. Metagrammar Redux. In *Proceedings of CSLP 2004, Copenhagen*.

B. Crabbe. 2005. *Représentation informatique de grammaires fortement lexicalisées : Application à la grammaire d'arbres adjoints*. Ph.D. thesis, Université Nancy 2.

D. Duchier and C. Gardent. 1999. A constraint based treatment of descriptions. In *Proceedings of the 3rd IWCS, Tilburg*.

Denys Duchier and Stefan Thater. 1999. Parsing with tree descriptions: a constraint-based approach. In *NLULP*, pages 17–32, Las Cruces, New Mexico.

R. Evans, G. Gazdar, and D. Weir. 1995. Encoding lexicalized tree adjoining grammars with a nonmonotonic inheritance hierarchy. In *Proceedings of the 33rd Annual Meeting of the ACL, 77-84*.

B. Gaiffe, B. Crabbe, and A. Roussanaly. 2002. A new metagrammar compiler. In *Proceedings of TAG+6, Venice*.

C. Gardent and L. Kallmeyer. 2003. Semantic construction in FTAG. In *Proceedings of EACL'03, Budapest*.

C. Gardent. 2006. Intégration d'une dimension sémantique dans les grammaires d'arbres adjoints. In *Actes de La 13ème édition de la conférence sur le TALN (TALN 2006)*.

A. Joshi and Y. Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69 – 124. Springer, Berlin, New York.

L. Kallmeyer. 1996. Tree description grammars. In *Results of the 3rd KONVENS Conference*, pages 330 – 341. Mouton de Gruyter ed., Hawthorne, NY, USA.

H.-U. Krieger and U. Schafer. 1994. TDL – a type description language for constraint-based grammars. In *Proceedings of COLING-94, pp. 893–899*.

R. Muskens and E. Krahmer. 1998. Description theory, ltags and underspecifi ed semantics. In *TAG'4*.

G. Perrier. 2000. Interaction grammars. In *Proceedings of 18th International Conference on Computational Linguistics (CoLing 2000)*, Sarrebrcken.

G. Perrier. 2003. Les grammaires d'interaction. HDR en informatique, Université Nancy 2.

K. Vijay-Shanker and Y. Schabes. 1992. Structure sharing in lexicalized tree adjoining grammars. In *Proceedings of COLING'92, Nantes, pp. 205 - 212*.

E. Villemonte de la Clergerie. 2005. DyALog: a tabular logic programming based environment for NLP. In *Proceedings of CSLP'05*, Barcelona.

F. Xia, M. Palmer, and K. Vijay-Shanker. 1999. Toward semi-automating grammar development. In *Proc. of NLPRS-99, Beijing, China*.