Metagrammar Redux

Benoit Crabbé and Denys Duchier

LORIA, Nancy, France

Abstract. In this paper we introduce a general framework for describing the lexicon of a lexicalised grammar by means of elementary descriptive fragments. The system described hereafter consists of two main components: a control device aimed at controlling how fragments are to be combined in order to describe meaningful lexical descriptions and a composition system aimed at resolving how elementary descriptions are to be combined.

1 Introduction

This paper is concerned with the design of large-scale grammars for natural language. It presents an alternative to the classical languages used for grammatical representation such as PATRII.

The need for a new language is motivated by the development of strongly lexicalised grammars based on tree structures rather than feature structures, and by the observation that, for tree based formalisms, lexical management with lexical rules raises non trivial practical issues [1].

In this paper we revisit a framework – the metagrammar – designed in particular for the lexical representation of tree based syntactic systems. It is organised around two central ideas: (1) a *core* grammar is described by elementary tree fragments and (2) these fragments are combined by means of a control language to produce an *expanded* grammar. Throughout the paper, we illustrate the features of the framework using Tree Adjoining Grammar (TAG) [2] as a target formalism.

The paper is structured as follows. First (Section 2) we introduce the key ideas underlying grammatical representation taking PATRII as an illustration. We then provide the motivations underlying the design of our grammatical representation framework. The core metagrammatical intuition: *lexical representation by manipulating fragments made of tree descriptions* is provided in (Section 3). The motivations concerning the set up of an appropriate tree representation language are provided in Section 4. The fragment manipulation language is then developed in section 5. And finally, the computational treatment of our description language is detailed in Section 6.

2 Lexical organisation

In this section we introduce the issue and the main ideas concerning lexical organisation of tree based syntactic systems. We begin by investigating the core ideas developed in PATRII then we highlight inadequacies of PATRII for representing the lexicon of tree based syntactic systems such as Tree Adjoining Grammar.

An historical overview In PATRII, one of the first proposal for grammatical representation [3], lexical description roughly consists of specifying lexical entries together with a subcategorisation frame such as in PATRII:

```
love :
   <cat> = v
   <arg0 cat> = np
   <arg1 cat> = np
```

where we specify that the verb *love* takes two arguments: a *subject noun phrase* and an *object noun phrase*. This lexical entry, together with an appropriate grammar, is used to constrain the set of sentences in which *love* may be inserted. For instance this lexical entry is meant to express that *love* is used transitively as in *John loves Mary* but not intransitively such as in *John loves* or *John loves to Mary*.

PATRII offers two devices to facilitate lexical description: templates and lexical rules. Templates are described by [3] as macros, and permit us to easily state that that *love* and *write* are transitive verbs by writing:

```
love :
     transitiveVerb
write :
     transitiveVerb
transitiveVerb :
   <cat> = v
   <arg0 cat> = np
   <arg1 cat> = np
```

where `transitiveVerb` is a macro called in the descriptions of *love* and *write*. On the other hand, lexical rules are used to describe multiple variants of verbs. For instance, to express that a transitive verb such as *love* may be used in its active or passive variant we may add the following lexical rule to our lexicon:

```
passive :
   <out cat> = <in cat>
   <out arg1 cat> = <in arg0 cat>
   <out arg0 cat> = pp
```

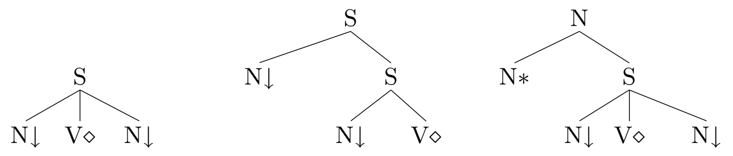
This rule says that a new lexical entry `out` is to be built from an initial lexical entry `in` where the category of `out` is identical to the category of `in`, the category of the object becomes the category of the subject and that the subject category now becomes prepositional phrase.

Lexical rules are meant to allow a dynamic expansion of related lexical variants. So for the verb *love* the application of the `passive` lexical rule to its base entry generates a new, derived, passive lexical entry meaning that both active and passive variants are licensed by the lexical entries.

Variants and improvements of this classical system have been (and are still) used for describing the lexicon in other syntactic frameworks, e.g. HPSG[4]. Whatever the differences, two leading ideas remain nowadays: lexical description aims both at factoring out information (templates) and at expressing relationships between variants of a single lexical unit (lexical rules).

Tree Adjoining Grammar: a case study Tree adjoining grammar (TAG)[1] is a tree composition system aimed at describing natural language syntax [2] which is strongly lexicalised. In other words, a tree adjoining grammar consists of a lexicon, the elementary trees, each associated to a lexical unit, and two operations used for combining the lexical units: adjunction and substitution.

Following the conventions used in TAG implementations such as XTAG [5], we work with tree schemata (or templates) such as these[2]:



Jean voit Marie	Quelle fille Jean voit	(Jean) qui voit Marie
John sees Mary	Which girl John sees	(John) who sees Mary

(1)

where the appropriate lexical word is inserted dynamically by the parser as a child of the anchor (marked ◇). The nodes depicted with an arrow (↓) are the substitution nodes and those depicted with a star (⋆) are the foot nodes.

Strikingly, lexical organisation of strongly lexicalised syntactic systems often try to provide alternative solutions to that of Shieber. The main reason is that the amount and the variety of lexical units is much greater, therefore the number of templates and lexical rules to be used is strongly increased. In the context of the development of large grammars, this situation requires the grammar writer to design complicated ordering schemes as illustrated by [1].

To overcome this, we take up an idea first introduced in [7] for Construction Grammar. Roughly speaking they describe the lexicon using a dynamic process: given a *core* lexicon, described manually, they build up an *expanded* lexicon by combining elementary *fragments* of information.

Besides strong lexicalisation, setting up a system representing a TAG lexicon raises another problem, that of the structures used. In Construction Grammar, [7] combine elementary fragments of information via feature structure unification. When working with TAG, however, one works with trees.

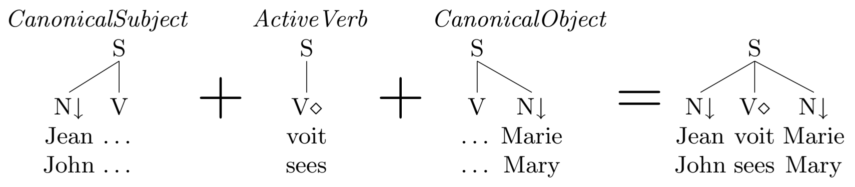
[1] Strictly speaking, we mean here Lexicalised Tree Adjoining Grammar (LTAG). Indeed, the system is usually used in its lexicalised version[5].

[2] The trees depicted in this paper are motivated by the French grammar of [6] who provides linguistic justifications in particular for not using the VP category and for using the category N at multiple bar levels instead of introducing the category NP in French.

3 Introduction to the framework

In this section we sketch the idea of describing the lexicon by controlling combinations of elementary fragment descriptions.

This idea stems from the following observation: the design of a TAG grammar consists of describing trees made of elementary pieces of information (hereafter: fragments). For instance the following tree is defined by combining a subtree representing a subject another subtree representing an object and finally a subtree representing the spine of the verbal tree:



Of course, we will also want a convenient means of expressing variants of the above tree; for example, where the subject instead of being realised in canonical position is realized as a questioned subject (*wh*) or a relative subject.

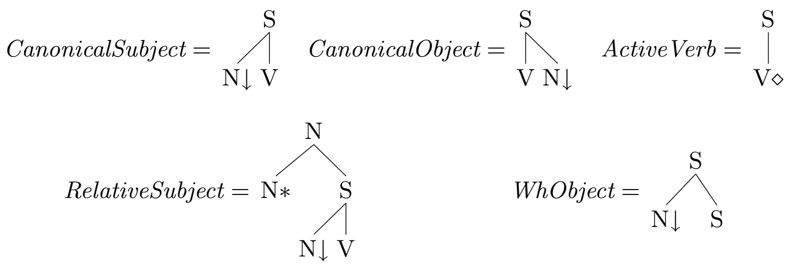
More generally while designing a grammar one wants to express general statements for describing sets of trees: for instance, a transitive verb is made of a subject, an object and a verbal active spine. In short we would like to write something like:

$$TransitiveVerb \quad = \quad Subject \ \wedge \ ActiveVerb \ \wedge \ Object$$

where *Subject* and *Object* are shortcuts for describing sets of variants:

$$\begin{aligned} Subject \quad &= \quad CanonicalSubject \ \vee \ RelativeSubject \\ Object \quad &= \quad CanonicalObject \ \vee \ WhObject \end{aligned}$$

and where *CanonicalSubject*, *WhSubject*... are defined as the actual fragments of the grammar:



Given the above definitions, a description such as *TransitiveVerb* is intended to describe the following tree schemata depicted in (1)[3]. That is, each variant

[3] The combination of relativised subject and a questioned object is rejected by the principle of extraction uniqueness (See section 4).

description of the subject embedded in the *Subject* clause is combined with each variant description of the object in the *Object* clause and the description in the *ActiveVerb* clause.

As it stands, the representation system we have introduced so far requires us to set up two components: first we investigate which language to use for describing *tree fragments* and combining them (Section 4). Second we detail the language which controls how fragments are to be combined (Section 5).

4 A language for describing tree fragments

In this section, we consider two questions: (1) how to conveniently describe tree fragments, (2) how to flexibly constrain how such tree fragments may be combined to form larger syntactic units. We first introduce a language of tree descriptions, and then show how it can be generalized to a family of formal languages parametrized by an arbitrary constraining decoration system that further limits how elements can be combined.

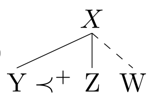
The base language L. Let $x, y, z \dots$ be node variables. We write $\lhd$ for immediate dominance, $\lhd^*$ for its reflexive transitive closure (dominance), $\prec$ for immediate precedence (or adjacency) and $\prec^+$ for its transitive closure (strict precedence). We let ℓ range over a set of node labels generally intended to capture the notion of categories. A tree description ϕ has the following abstract syntax:

$$\phi \quad ::= \quad x \lhd y \ \mid \ x \lhd^* y \ \mid \ x \prec y \ \mid \ x \prec^+ y \ \mid \ x : \ell \ \mid \ \phi \wedge \phi \tag{2}$$

L-descriptions are, as expected, interpreted over first-order structures of finite, ordered, constructor trees. As usual, we limit our attention to minimal models.

Throughout the paper we use an intuitive graphical notation for representing tree descriptions. Though this notation is not sufficient to represent every expression of the language, it nonetheless generally suffices for the kind of trees typically used in natural language syntax. Thus, the description (D_0) on the left is graphically represented by the tree notation on the right:

$$D_0 = \begin{array}{l} x \lhd^* w \wedge x \lhd y \wedge x \lhd z \\ \wedge\ y \prec^+ z \wedge z \prec w \\ \wedge\ x : X \wedge y : Y \wedge z : Z \wedge w : W \end{array}$$

(D_0) 

(3)

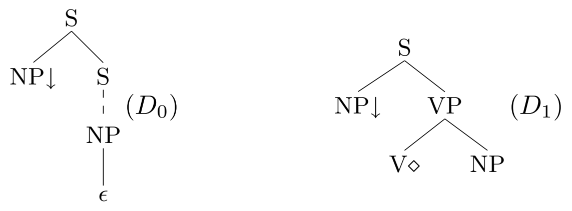
where immediate dominance is represented by a solid line, dominance by a dashed line, precedence by the symbol $\prec^+$ and adjacency is left unmarked.

A parametric family of languages. It is possible to more flexibly control how tree fragments may be combined by adding annotations to nodes together with stipulations, called principles, for how these annotations restrict admissible models and interpretations. In this manner, we arrive at the idea of a family of languages $L(C)$ parametrized by a set C of *combination schemata.*

In the remainder of this section we introduce three such principles: the coloration principle, a clitic ordering principle, and an extraction principle. These principles provide an instantiation of $L(C)$ that has been used for describing the lexicon of a large Tree Adjoining Grammar for French.

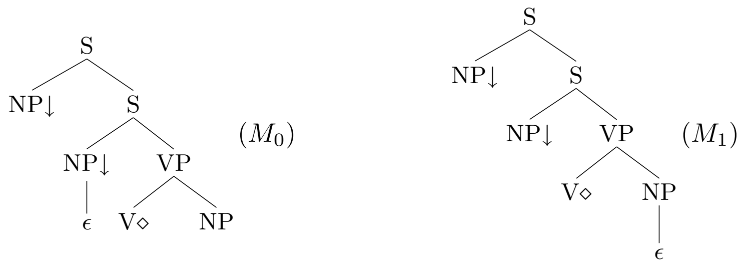
We begin by discussing the coloration principle. It provides a convenient way to constrain fragment combinations. Two alternatives have already been used in the literature: the first one, $L(\emptyset)$ is used by Xia [8], the second one $L(\textit{names})$ is used by Candito [9]. We show that neither $L(\emptyset)$ nor $L(\textit{names})$ is appropriate for describing the lexicon of a French TAG Grammar. We then introduce $L(\textit{colors})$ which we have used successfully for that purpose.

Language $L(\emptyset)$. This first instantiation of $L(C)$ is used by [8]. This language does not use any combination constraint. The combination schema C is thus empty. Equipped with such a language we can independently describe fragments such as these[4]:



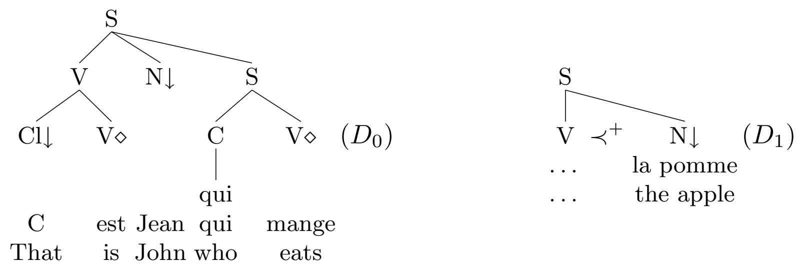
(4)

where D_0 describes a relative NP and D_1 a transitive construction. Their combination leads to the following two models:



(5)

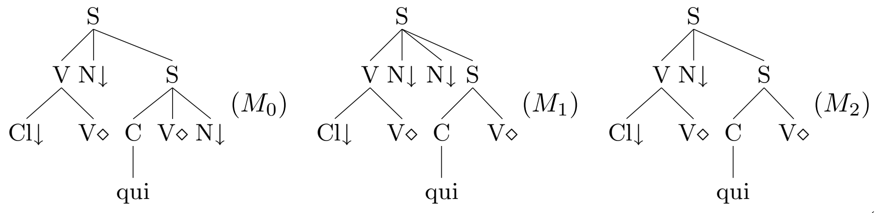
However this language faces an expressivity limit since, for the purpose of lexical organisation, linguists want to constrain combinations more precisely. For instance, in the French Grammar the following fragment composition is badly handled since:



(6)

[4] These fragments and the related models are those used by F. Xia in the context of the XTAG English Grammar.

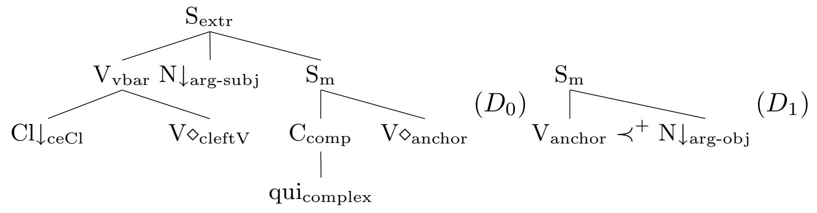
yields, among others, the following results:



(7)

where (D_0) represents a cleft construction and (D_1) a canonical object construction. In such a case, only (M_0) is normally deemed linguistically valid.(M_1) and (M_2) represent cases where the cleft construction and the canonical object construction have been mixed.

Language $L(names)$. In her thesis, M.-H. Candito [9] introduces an instance of $L(C)$ that constrains combinations to avoid cases such as the one outlined above. The combination schema C is as follows: (1) a finite set of names where each node of a tree description is associated with one name and (2) Two nodes sharing the same name are to be interpreted as denoting the same entity, hence when merging descriptions, only the nodes with the same names are merged. In other words, a model is valid if (1) every node has exactly one name and (2) there is at most one node with a given name[5].



(8)

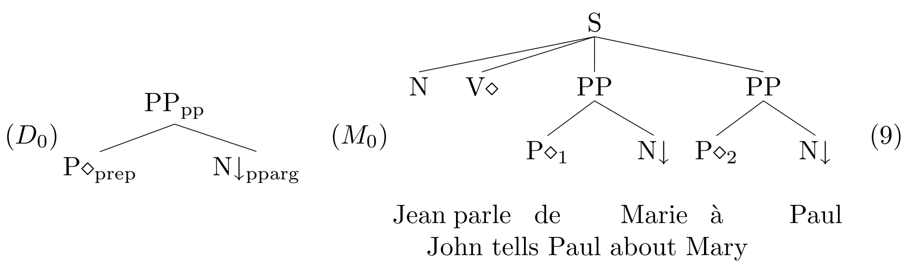
The only model resulting from merging D_0 with D_1 in (8) is only M_0 depicted in (7). In such a case, $L(names)$ corrects the shortcomings of $L(\emptyset)$: here, naming ensures that the canonical argument (D_1) cannot be realised within the cleft argument. However, during the development of a non-trivial grammar using this language, it turned out that $L(names)$ was eventually unsatisfactory for two main reasons:

The first is practical and rather obvious: the grammar writer has to manage naming by hand, and must handle the issues arising from name conflicts.

The second is more subtle: the grammar writer may need to use the same tree fragment more than once in the same description. For example, such an

[5] To be complete, M.-H. Candito uses additional operations to map multiple names onto a single node. However this does not change the content of our present discussion.

occasion arises in the case of a double PP complementation:



where one cannot use the fragment (D_0) more than once to yield M_0 since identical names must denote identically the same nodes.

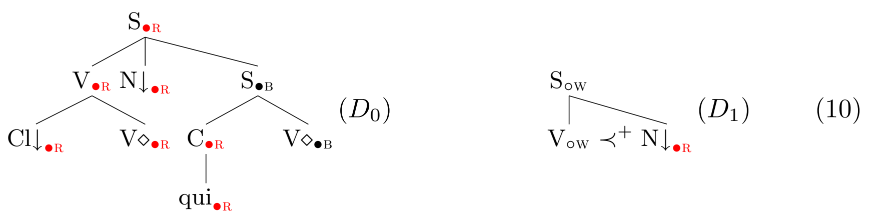
A language with coloured nodes $L(colours)$. We used this language in the development of a large scale French TAG patterned after the analysis of [6].

$L(colours)$ was designed to overcome the shortcomings of languages $L(\emptyset)$ and $L(names)$. We want (1) to be able to constrain the way fragments combine more precisely than with language $L(\emptyset)$ and (2) we want to eschew the explicit naming management of language $L(names)$.

To do this, the combination schema C used in $L(colours)$ decorates all nodes with colours: black ($\bullet_B$), white ($\circ_W$), red ($\bullet_R$) or failure ($\perp$). The additional condition on model admissibility is that each node must be either red or black.

When combining tree descriptions, nodes are merged and their colours combined. The table to the right specifies the result of combining two colours. For instance, combining a white node with a black node yields a black node; combining a white node with a red node is illegal and produces a failure. As a matter of illustration, the following colour enriched descriptions yield only the desired model (M_0) for example number (7)[6]

	$\bullet_B$	$\bullet_R$	$\circ_W$	$\perp$
$\bullet_B$	$\perp$	$\perp$	$\bullet_B$	$\perp$
$\bullet_R$	$\perp$	$\perp$	$\perp$	$\perp$
$\circ_W$	$\bullet_B$	$\perp$	$\circ_W$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$



Intuitively the colours have a semantic similar to that of ressources and requirements systems such as Interaction Grammars [10]. A tree is well formed if it is saturated. The colours representing saturation are red or black the colour representing non saturation is white and we have a colour representing failure.

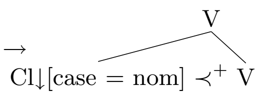
[6] We let the reader figure out how to express double PP complementation (9). It requires the use of a description similar to (D_1) depicted here but patterned for describing a prepositional phrase.

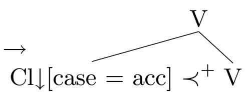
Though $L(colours)$ turns out to be satisfactory for designing a large scale French TAG, it might not be adequate for other frameworks or languages.[7] However, alternative instances of $L(C)$ might be suitable. For example a combination schema based on polarities seems a very reasonable foundation for interaction grammars [10] and even for polarity based unification grammars [11].

Besides node-colouring we introduce a clitic ordering principle. So far, the current system assumes that one can describe grammatical information by combining fragments of local information. There are however cases where the local fragments interact when realised together. Following the intuition given in section 3 we express the fact that a transitive verb is made of a subject, an object and a verb in the active form:

$$\begin{aligned} \mathit{TransitiveVerb} &= \mathit{Subject} \wedge \mathit{ActiveVerb} \wedge \mathit{Object} & (11)\\ \mathit{Subject} &= \mathit{CanonicalSubject} \vee \mathit{CliticSubject} & (12)\\ \mathit{Object} &= \mathit{CanonicalObject} \vee \mathit{CliticObject} & (13) \end{aligned}$$

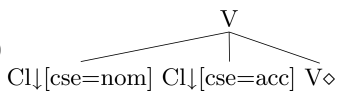
Clitic ordering According to the subject and object clauses, it is the case that among others, a description of a transitive verb is made of the composition of a clitic subject and a clitic object[8] whose definitions are as follows:

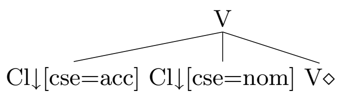
CliticSubject →


CliticObject →


(14)

When realized together, none of the clitic descriptions say how these clitics are ordered relative to each other; therefore a merge of these two descriptions yields the following two models:

(M_0)


(M_1)


where M_1 is an undesirable solution in French.

French clitic ordering is thus handled by a principle of tree wellformedness: *sibling nodes of category Clitic have to be ordered according to the respective order of their ranking property.* So, if we take the case feature of descriptions (14) to be the ranking property, and that the order defined over the property constrains (inter alia) nominative to precede accusative then in every tree where both a nominative and an accusative clitic are realised, the principle ensures that only M_0 is a valid model.

[7] The current framework is not restricted to the specific case of Tree Adjoining Grammars. It should be straightforward to adapt it to other cases of tree based syntactic systems such as Interaction Grammars.

[8] In French, clitics are small unstressed pronominal particles realised next to the verb which are ordered according to a fixed order. The problem of clitic ordering is a well known case of such an interaction. It was already described as problematic in the generative literature in the early 70's [12].

Extraction uniqueness Another principle presented hereafter (Section 6) is that of extraction uniqueness. We assume that, in French, only one argument of a given predicate may be extracted[9]. Following this, the extraction principle is responsible for ruling out tree models where more than one node would be associated with the property of extraction.

Two other principles have actually been used in the implementation of the French Grammar: a principle for ensuring clitic uniqueness and a principle for expressing island constraints[10]. The expression of an additional principle of functional uniqueness is currently under investigation.

Like the node-colouring principle, these principles are related to a specific language, French, or to a specific target formalism, TAG. Therefore, principles are additional parametric constraints that can be used or not for constraining the generated models. For dealing with other languages or other target formalisms, one could extend the library of principles presented here to fit the particular needs.

5 Controlling fragment combinations

In Section 3 we identified a number of desirable requirements for a metagrammar language: (1) it should support disjunctions to make it easy to express diathesis (such as active, passive), (2) it should support conjunction so that complex descriptions can be assembled by combining several simpler ones, (3) it should support abstraction so that expressions can be named to facilitate reuse and avoid redundancy.

In this section, we introduce the language $\mathcal{L}_C$ to control how fragments can be combined in our proposed lexical representation framework, and show how $\mathcal{L}_C$ satisfies all the requirements above.

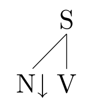
$$\text{Clause} ::= \text{Name} \rightarrow \text{Goal} \tag{15}$$

$$\text{Goal} ::= \text{Goal} \wedge \text{Goal} \quad | \quad \text{Goal} \vee \text{Goal} \quad | \quad \phi \quad | \quad \text{Name} \tag{16}$$

This language allows us to manipulate fragment descriptions (ϕ), to express the composition of statements (Goal $\wedge$ Goal), to express nondeterministic choices (Goal$\vee$Goal), and finally to name complex statements for reuse (Name $\rightarrow$ Goal).

The main motivation for the control language is to support the combination and reuse of tree fragments. Instead of directly manipulating tree descriptions, the language allows us to define abstractions over (possibly complex) statements. Thus, the clause:

$$\textit{CanonicalSubject} \quad \rightarrow \tag{17}$$



[9] Actually, cases of double extraction have been discovered in French but they are so rare and so unnatural that they are generally ruled out of grammatical implementations.

[10] This principle is related to the way one formalises island constraints in TAG [13].

defines the abstraction *CanonicalSubject* to stand for a tree description which can be subsequently reused via this new name, while the clause:

$$\mathit{TransitiveVerbActive} \quad \rightarrow \quad \mathit{Subject} \wedge \mathit{ActiveVerb} \wedge \mathit{Object} \tag{18}$$

states that a lexical tree for a transitive verb is formed from the composition of the descriptions of a subject, of an object and of an active verb.

Disjunction is interpreted as an nondeterministic choice: each of the alternatives describes one of the ways in which the abstraction can be realized. As illustrated by lexical rules as used e.g. in PATRII [3], a system of lexical representation needs to be equipped with a way to express relationships between lexical items along the lines of a passive lexical rule relating an active and a passive lexical entry. In our approach, similar relations are expressed with disjunctions. Thus the following statement expresses the fact that various realisations of the subject are equivalent:

$$\begin{aligned} \mathit{Subject} \quad \rightarrow \quad & \ \mathit{CanonicalSubject} \\ & \vee\ \mathit{WhSubject} \\ & \vee\ \mathit{RelativeSubject} \\ & \vee\ \mathit{CliticSubject} \end{aligned} \tag{19}$$

As surely has become evident, the language presented in this section has very much the flavor of a logic programming language. More precisely, it can be understood as an instance of the *Definite Clause Grammar* (DCG) paradigm. DCGs were originally conceived to express the production rules of context free grammars: they characterised the sentences of a language, i.e., all the possible ways words could be combined into grammatical sequences by concatenation. Here, instead of words, we have tree fragments, and instead of concatenation we have a composition operation. In other words, $\mathcal{L}_C$ allows us to write the grammar of a tree grammar, which surely justifies the name *metagrammar*.

6 A constraint satisfaction approach

As mentioned earlier, the control language $\mathcal{L}_C$ of Section 5 can be regarded as an instance of the *Definite Clause Grammar* (DCG) paradigm. While DCGs are most often used to describe sentences, i.e. sequences of words, here, we apply them to the description of formulae in language $L(\mathit{colors})$, i.e. conjunctions of colored tree fragments.

A consequence of regarding a metagrammar, i.e. a program expressed in language $\mathcal{L}_C$, as a DCG is that it can be reduced to a logic program and executed as such using well-known techniques. What remains to be explained is how, from a conjunction of colored tree fragments, we derive all complete trees that can be formed by combining these fragments.

For this task, we propose a constraint-based approach that builds upon and extends the treatment of dominance constraints of Duchier and Niehren [14]. We begin by slightly generalizing the language introduced in Section 4 to make it more directly amenable to the treatment described in [14], then we show how we can enumerate the minimal models of a description in that language by translating this description into a system of constraints involving set variables, and solving that instead.

Tree description language. In order to account for the idea that each node of a description is colored either red, black or white, we let x, y, z range over 3 disjoint sets of node variables: $V_{\mathrm{R}}, V_{\mathrm{B}}, V_{\mathrm{W}}$. We write $\lhd$ for immediate dominance, $\lhd^+$ for its transitive closure, i.e. strict dominance, $\prec$ for immediate precedence, and $\prec^+$ for its transitive closure, i.e. strict precedence. We let ℓ range over a set of node labels. A description ϕ has the following abstract syntax:

$$\phi \quad ::= \quad x\,R\,y \quad | \quad x \lhd y \quad | \quad x \prec y \quad | \quad x : \ell \quad | \quad \phi \wedge \phi \tag{20}$$

where $R \subseteq \{=, \lhd^+, \rhd^+, \prec^+, \succ^+\}$ is a set of relation symbols whose intended interpretation is disjunctive; thus $x\ \{=, \lhd^+\}\ y$ is more conventionally written $x \lhd^*\, y$.

In [14], the abstract syntax permitted a literal of the form $x : \ell(x_1, \ldots, x_n)$ that combined (1) an assignment of the label ℓ to x, (2) immediate dominance literals $x \lhd x_i$, (3) immediate precedence literals $x_i \prec x_{i+1}$, (4) an arity constraint stipulating that x has exactly n children. Here we prefer a finer granularity and admit literals for immediate dominance and immediate precedence. For simplicity of presentation we omit an arity constraint literal.

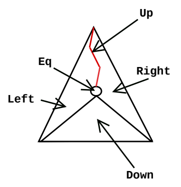
Enumerating minimal models. We now describe how to convert a description into a constraint system that uses set constraints and such that the solutions of the latter are in bijection with the minimal models of the former. Such a constraint system can be realized and solved efficiently using the constraint programming support of Mozart/Oz. Our conversion follows the presentation of [14] very closely.

The general intuition is that a literal $x\ R\ y$ should be represented by a membership constraint $y \in R(x)$ where $R(x)$ is a set variable denoting all the nodes that stand in R relationship with x. We write V^ϕ for the set of variables occurring in ϕ. Our encoding consists of 3 parts:

$$[\![\phi]\!] \quad = \quad \bigwedge_{x \in V^\phi} \mathsf{A}_1(x) \bigwedge_{x,y \in V^\phi} \mathsf{A}_2(x,y) \quad \wedge \quad \mathsf{B}[\![\phi]\!] \tag{21}$$

$\mathsf{A}_1(\cdot)$ introduces a node representation per variable, $\mathsf{A}_2(\cdot,\cdot)$ axiomatises the treeness of the relations between these nodes, and $\mathsf{B}(\cdot)$ encodes the problem-specific restrictions imposed by ϕ.

6.1 Representation



When observed from a specific node x, the nodes of a solution tree (a model), and hence the variables which they interpret, are partitioned into 5 regions: the node denoted by x itself, all nodes below, all nodes above, all nodes to the left, and all nodes to the right. The main idea is to introduce corresponding set variables Eq_x, Up_x, Down_x, Left_x, Right_x to encode the sets of variables that are interpreted by nodes in the model which are respectively equal, above, below, left, and right of the node interpreting x. First, we state that x is one of the variables interpreted by the corresponding node in the model:

$$x \in \mathit{Eq}_x \tag{22}$$

Then, as explained above, we have the following fundamental partition equation:

$$V^\phi = \mathit{Eq}_x \uplus \mathit{Up}_x \uplus \mathit{Down}_x \uplus \mathit{Left}_x \uplus \mathit{Right}_x \tag{23}$$

where $\uplus$ denotes *disjoint union*. We can (and in fact *must*, as proven in [14]) improve propagation by introducing shared intermediate results *Side*, *Eqdown*, *Equp*, *Eqdownleft*, *Eqdownright*.

$$\mathit{Side}_x = \mathit{Left}_x \uplus \mathit{Right}_x \qquad \mathit{Eqdownleft}_x = \mathit{Eqdown}_x \uplus \mathit{Left}_x \tag{24}$$
$$\mathit{Eqdown}_x = \mathit{Eq}_x \uplus \mathit{Down}_x \qquad \mathit{Eqdownright}_x = \mathit{Eqdown}_x \uplus \mathit{Right}_x \tag{25}$$
$$\mathit{Equp}_x = \mathit{Eq}_x \uplus \mathit{Up}_x \tag{26}$$

which must all be related to V^ϕ:

$$V^\phi = \mathit{Eqdown}_x \uplus \mathit{Up}_x \uplus \mathit{Side}_x \qquad V^\phi = \mathit{Eqdownleft}_x \uplus \mathit{Up}_x \uplus \mathit{Right}_x \tag{27}$$
$$V^\phi = \mathit{Equp}_x \uplus \mathit{Down}_x \uplus \mathit{Side}_x \qquad V^\phi = \mathit{Eqdownright}_x \uplus \mathit{Down}_x \uplus \mathit{Left}_x \tag{28}$$

We define $\mathsf{A}_1(x)$ as the conjunction of the constraints introduced above.

6.2 Wellformedness

Posing $\mathbf{Rel} = \{=, \lhd^+, \rhd^+, \prec^+, \succ^+\}$, in a tree, the relationship that obtains between the nodes denoted by x and y must be one in $\mathbf{Rel}$: the options are mutually exclusive. We introduce a variable C_{xy}, called a *choice variable*, to explicitly represent it and contribute a well-formedness clause $\mathsf{A}_3[\![x\ r\ y]\!]$ for each $r \in \mathbf{Rel}$.

$$\mathsf{A}_2(x,y) \quad = \quad C_{xy} \in \mathbf{Rel} \ \wedge\ \wedge\{\mathsf{A}_3[\![x\ r\ y]\!] \mid r \in \mathbf{Rel}\} \tag{29}$$
$$\mathsf{A}_3[\![x\ r\ y]\!] \quad \equiv \quad \mathsf{D}[\![x\ r\ y]\!] \wedge C_{xy} = r \quad \vee \quad C_{xy} \neq r \wedge \mathsf{D}[\![x\ \neg r\ y]\!] \tag{30}$$

For each $r \in \mathbf{Rel}$, it remains to define $\mathsf{D}[\![x\ r\ y]\!]$ and $\mathsf{D}[\![x\ \neg r\ y]\!]$ encoding respectively the relationships $x\,r\,y$ and $x\,\neg r\,y$ by set constraints on the representations of x and y.

$$\mathsf{D}[\![x = y]\!] = \mathit{Eq}_x = \mathit{Eq}_y \wedge \mathit{Up}_x = \mathit{Up}_y \wedge \ldots \tag{31}$$
$$\mathsf{D}[\![x \neg\!= y]\!] = \mathit{Eq}_x \parallel \mathit{Eq}_y \tag{32}$$
$$\mathsf{D}[\![x \lhd^+ y]\!] = \mathit{Eqdown}_y \subseteq \mathit{Down}_x \ \wedge\ \mathit{Equp}_x \subseteq \mathit{Up}_y \ \wedge \ \mathit{Left}_x \subseteq \mathit{Left}_y \ \wedge\ \mathit{Right}_x \subseteq \mathit{Right}_y \tag{33}$$
$$\mathsf{D}[\![x \neg\lhd^+ y]\!] = \mathit{Eq}_x \parallel \mathit{Up}_y \ \wedge\ \mathit{Down}_x \parallel \mathit{Eq}_y \tag{34}$$
$$\mathsf{D}[\![x \prec^+ y]\!] = \mathit{Eqdownleft}_x \subseteq \mathit{Left}_y \ \wedge\ \mathit{Eqdownright}_y \subseteq \mathit{Right}_x \tag{35}$$
$$\mathsf{D}[\![x \neg\prec^+ y]\!] = \mathit{Eq}_x \parallel \mathit{Left}_y \ \wedge\ \mathit{Right}_x \parallel \mathit{Eq}_y \tag{36}$$

where $\parallel$ represents disjointness.

6.3 Problem-specific constraints

The third part $\mathsf{B}[\![\phi]\!]$ of the translation forms the problem-specific constraints that further restrict the admissibility of well-formed solutions and only accepts those which are models of ϕ. The translation is given by case analysis following the abstract syntax of ϕ:

$$\mathsf{B}[\![\phi \wedge \phi']\!] \quad = \quad \mathsf{B}[\![\phi]\!] \wedge \mathsf{B}[\![\phi']\!] \tag{37}$$

A rather nice consequence of introducing choice variables C_{xy} is that any dominance constraint $x\ R\ y$ can be translated as a restriction on the possible values of C_{xy}. For example $x \lhd^* y$ can be encoded as $C_{xy} \in \{=, \lhd^+\}$. More generally:

$$\mathsf{B}[\![x\ R\ y]\!] \quad = \quad C_{xy} \in R \tag{38}$$

A labelling literal $x : \ell$ simply restricts the label associated with variable x:

$$\mathsf{B}[\![x : \ell]\!] \quad = \quad \mathit{Label}_x = \ell \tag{39}$$

An immediate dominance literal $x \lhd y$ not only states that $x \lhd^+ y$ but also that there are no intervening nodes on the spine that connects the two nodes:

$$\mathsf{B}[\![x \lhd y]\!] \quad = \quad C_{xy} = \lhd^+ \ \wedge\ \mathit{Up}_y = \mathit{Equp}_x \tag{40}$$

An immediate precedence literal $x \prec y$ not only states that $x \prec^+ y$ but also that there are no intervening nodes horizontally between them:

$$\mathsf{B}[\![x \prec y]\!] = C_{xy} = \prec^+ \ \wedge\ \mathit{Eqdownleft}_x = \mathit{Left}_y \ \wedge\ \mathit{Right}_x = \mathit{Eqdownright}_y \tag{41}$$

6.4 Coloring

While distinguishing left and right is a small incremental improvement over [14], the treatment of colors is a rather more interesting extension. The main question is: which nodes can or must be identified with which other nodes? Red nodes cannot be identified with any other nodes. Black nodes may be identified with white nodes. Each white node must be identified with a black node. As a consequence, for every node, there is a unique red or black node with which it is identified. We introduce the (integer) variable RB_x to denote the red or black node with which x is identified.

For a red node, x is identified only with itself:

$$x \in V_{\mathrm{R}} \qquad \Rightarrow \qquad \mathit{RB}_x = x \ \wedge\ \mathit{Eq}_x = \{x\} \tag{42}$$

For a black node, the constraint is a little relaxed (it may also be indentified with white nodes):

$$x \in V_{\mathrm{B}} \qquad \Rightarrow \qquad \mathit{RB}_x = x \tag{43}$$

Posing $V_{\mathrm{B}}^{\phi} = V^{\phi} \cap V_{\mathrm{B}}$, each white node must be identified with a black node:

$$x \in V_{\mathrm{W}} \quad \Rightarrow \quad RB_x \in V_{\mathrm{B}}^{\phi} \tag{44}$$

Additionally, it is necessary to ensure that $RB_x = RB_y$ iff x and y have been identified. We can achieve this simply by modifiying the definition (32) of $\mathsf{D}[\![x \neg\!= y]\!]$ as follows:

$$\mathsf{D}[\![x \neg\!= y]\!] \quad = \quad Eq_x \parallel Eq_y \ \wedge \ RB_x \neq RB_y \tag{45}$$

6.5 Extraction Principle

As an illustration of how the framework presented so far can be extended with linguistically motivated principles to further constrain the admissible models, we describe now what we have dubbed the *extraction principle*.

The description language is (somehow) extended to make it possible to mark certain nodes of a description as representing an *extraction*. The extraction principle then makes the additional stipulation that, to be admissible, a model must contain at most one node marked as extracted.

Let V_{XTR}^{ϕ} be the subset of V^{ϕ} of those node variables marked as extracted. We introduce the new boolean variable $\mathit{Extracted}_x$ to indicate whether the node denoted by x is extracted:

$$\mathit{Extracted}_x \quad = \quad Eq_x \cap V_{\mathrm{XTR}}^{\phi} \neq \emptyset \tag{46}$$

Posing $V_{\mathrm{RB}}^{\phi} = V^{\phi} \cap (V_{\mathrm{R}} \cup V_{\mathrm{B}})$, and freely identifying the boolean values false and true respectively with the integers 0 and 1, the extraction principle can be enforced with the following constraint:

$$\sum_{x \in V_{\mathrm{RB}}^{\phi}} \mathit{Extracted}_x \quad < \quad 2 \tag{47}$$

7 Conclusion

This paper introduces a core abstract framework for representing grammatical information of tree based syntactic systems. Grammatical representation is organised around two central ideas: (1) the lexicon is described by means of elementary tree fragments that can be combined. (2) Fragment combinations are handled by a control language, which turns out to be an instance of a DCG.

The framework described here, generalises the TAG specific approaches of [8, 9]. We have provided a parametric family of languages for tree composition as well as constraints on tree wellformedness.

Besides the non TAG specific tree composition language, it mostly differs from the TAG instantiations in that it introduces a control language allowing the explicit formulation of fragment composition and of variants of related lexical entries. The two existing systems of [8] and [9] rely mostly on an algorithmic device for expressing variants, namely a crossing algorithm for [9], and an external module of lexical rules for [8].

The introduction of the control language (1) avoids the need for different modules as in [8] and (2) introduces more flexibility in expressing variants thus eliminating the "shadow" classes that turn out to be needed in [9].

The framework presented here has been fully implemented [15]. It has led to the actual development of a large French TAG based on [6] which has been used to test its adequacy with a task of real scale grammatical development. The resulting grammar covers most of the phenomena related to the syntax of French verbs.

References

1. Prolo, C.: Generating the XTAG English grammar using metarules. In: Proc. COLING 2002, Taiwan (2002)
2. Joshi, A.K., Schabès, Y.: Tree adjoining grammars. In Rozenberg, G., Salomaa, A., eds.: Handbook of Formal Languages. Springer Verlag, Berlin (1997)
3. Shieber, S.M.: The design of a computer language for linguistic information. In: Proceedings of the Tenth International Conference on Computational Linguistics, Stanford University, Stanford, California (1984) 362–366
4. Meurers, W.D.: On implementing an HPSG theory – aspects of the logical architecture, the formalization, and the implementation of Head-Driven Phrase Structure Grammars. In: Erhard W. Hinrichs, W. Detmar Meurers, and Tsuneko Nakazawa: *Partial-VP and Split-NP Topicalization in German – An HPSG Analysis and its Implementation*. Arbeitspapiere des SFB 340 Nr. 58, Universität Tübingen (1994)
5. XTAG Research Group: A lexicalized tree adjoining grammar for English. Technical Report IRCS-01-03, IRCS, University of Pennsylvania (2001)
6. Abeillé, A.: Une grammaire d'arbres adjoints pour le français. Editions du CNRS, Paris (2002)
7. Koenig, J.P., Jurafsky, D.: Type underspecification and on-line type construction in the lexicon. In: Proceedings of WCCFL94. (1995)
8. Xia, F.: Automatic Grammar Generation from two Different Perspectives. PhD thesis, University of Pennsylvania (2001)
9. Candito, M.H.: Organisation Modulaire et Paramétrable de Grammaires Electroniques Lexicalisées. PhD thesis, Université de Paris 7 (1999)
10. Perrier, G.: Les grammaires d'interaction. Université Nancy 2 (2003) Habilitation à diriger des recherches.
11. Kahane, S.: Grammaires d'unification polarisées. In: Proc. TALN 2004, Fès (2004)
12. Perlmutter, D.: Surface structure constraints in syntax. Linguistic Inquiry **1** (1970) 187–255
13. Kroch, A., Joshi, A.K.: The Linguistic Relevance of Tree Adjoining Grammar. Technical report, IRCS, Philadelphia (1985)
14. Duchier, D., Niehren, J.: Dominance constraints with set operators. In: Proceedings of the First International Conference on Computational Logic (CL2000). Volume 1861 of Lecture Notes in Computer Science., Springer (2000) 326–341
15. Duchier, D., Leroux, J., Parmentier, Y.: The metagrammar compiler: an NLP application with a multi-paradigm architecture. In: Proceedings of MOZ'2004, Charleroi (2004)