# Parsing Coordination Extragrammatically

Valmi Dufour-Lussier, Bruno Guillaume, Guy Perrier

# Parsing Coordination Extragrammatically

Valmi Dufour-Lussier, Bruno Guillaume, and Guy Perrier

LORIA (CNRS, Inria, Université de Lorraine, Nancy),
BP 239, 54506 Vandœuvre-lès-Nancy, France
valmi.dufour@loria.fr, bruno.guillaume@loria.fr, guy.perrier@loria.fr

**Abstract.** We propose to process coordination at the parsing level as a linguistic performance issue, outside the grammar, rather than as a matter of competence. We apply a specific algorithm to combine coordinated syntactic structures that were partially parsed using a coordination-less grammar, resulting in a directed acyclic parse graph in which constituent sharing appears sharply. This article presents an algorithm working within the framework of tree-adjoining grammars (although it can be adapted to other formalisms) that is able to handle many types of coordinating constructions, including left and right node raising, argument clusters, and verb gapping.

## 1. Introduction

Coordination is a frequent feature of natural language, yet it is extremely difficult to parse. One reason is that coordination of non-constituents is difficult to describe using the same formal tools as are used to model the "basic", coordination-free part of language.

Coordination gives rise to two linguistic phenomena, sharing of syntactic substructures and elision of constituents [8], which cannot be appropriately captured with the classical form of a tree, neither in a constituency nor in a dependency-based approach.

A first answer is to ignore the aspects violating treeness of structures. Most statistical parsing methods aim at building trees and thus choose to ignore complex structures going beyond treeness [6].

The main attempts to take complex coordinated structures into account are found in formal grammar-based approaches of parsing [10, 5, 9, 1, 7, 8], using one of two ways:

- Adding specific elementary constructions to the grammar. Since coordination is highly polymorphic, the number of structures added can be important, especially in the case of lexicalized grammars.
- Modifying the parsing algorithms to take the specificity of coordination into account.

In both cases, this results in a reduction of the efficiency of parsing algorithms [11]: the ambiguity in the choice of elementary structures and bounds of conjuncts increases, and the resulting structures are more complex.

All these approaches have a common feature: they integrate the treatment of coordination within a unique parsing process. Turning away from a formal grammar-based approach, we propose to move the treatment of coordination outside of the general grammatical parsing process. This fits in with a linguistic idea that coordination may be beyond the scope of competence [3].

The principle is to alternate general parsing steps with coordination processing steps: choices are postponed until enough information is available to guide the coordination processing, as is the case with some existing parsers that use several "passes" [2]. This is not simply a matter of ordering parsing steps: the method is designed to produce directed acyclic graphs (DAGs), a syntactic representation richer than the one generated by the tree-based, coordination-less grammar.

The algorithm that encodes the coordination resolving steps requires the definition of notions related to DAGs. Section 2 is dedicated to this.

The general design of this alternation between general parsing steps and coordination resolving steps is described in Section 3.

The following sections present the specialization of the algorithm in the three cases that are considered in this article: coordination of constituents without sharing (Section 4), coordination with peripheral sharing (Section 5), and coordination with head gapping (Section 6).

The algorithm is not linked to a specific grammatical formalism, but we chose one to help explain it in details. We used Tree-Adjoining Grammar (TAG), a simple and well-known formalism that has two syntactic composition operations: substitution and adjunction [4]. For the sake of simplicity, we consider only two features: the grammatical category, written *cat*, and the syntactic function, written *funct*. We do not distinguish between top and bottom features.

## 2. Preliminary definitions

By using DAGs instead of trees as representations for the syntactic structure of strings, many usual concepts such as root or leaf do not have an obvious sense. In this section, we define useful concepts applicable in directed acyclic graphs parsing, keeping as close as possible to the tree terminology.

We call the sink vertices **leaf nodes** and the source vertices **root nodes**. A DAG with a unique root is called a **rooted DAG** or an RDAG; it is worth noting that an RDAG is always connected. In the rest of the paper, all the graphs we consider are lists of RDAGs. In fact, trees are particular cases of RDAGs and all the algorithms presented in the paper preserve this graph property.

Nodes in parse graphs are labelled with morpho-syntactic features. According to the TAG formalism, leaf nodes are divided into **substitution nodes**, **foot nodes** and **anchor nodes**. Substitution nodes are to be merged with the root of an initial tree, foot nodes are to be used in an adjunction and anchor nodes are

labelled with words of the language. A DAG is **saturated** if it has no substitution nodes and no foot node[1].

We define two partial orders in DAGs. In TAG trees, all children of a given node are totally ordered. This order is maintained through derivation and thus, in DAGs, the out-edges of a given node are totally ordered. Anchor nodes are totally ordered too (by the word order of the input sentence). This order is written $\prec$.

In a tree, if $N$ is an ancestor of $M$, there is a unique path (a list of nodes such that each one is a child of the previous one) from $N$ to $M$. In a DAG, there can be several paths from $N$ to $M$. The most interesting paths are the leftmost and rightmost paths (written $\overleftarrow{\mathbf{path}}$ and $\overrightarrow{\mathbf{path}}$). They are defined recursively as follows:

- $\overleftarrow{\mathbf{path}}(N, N) = \overrightarrow{\mathbf{path}}(N, N) = \{N\}$;
- If $N$ is an ancestor of $M$, with $N \neq M$, then at least one daughter of $N$ is an ancestor of $M$. Let $N_l$ be the leftmost child of $N$ which is an ancestor of $M$. Then, define $\overleftarrow{\mathbf{path}}(N, M) = \{N\} :: \overleftarrow{\mathbf{path}}(N_l, M)$;
- If $N$ is an ancestor of $M$, with $N \neq M$, then at least one daughter of $N$ is an ancestor of $M$. Let $N_r$ be the rightmost child of $N$ which is an ancestor of $M$. Then, define $\overrightarrow{\mathbf{path}}(N, M) = \{N\} :: \overrightarrow{\mathbf{path}}(N_r, M)$;

Every node $N$ of a DAG has a **yield**, denoted $\mathbf{yield}(N)$, which is the set of all its descendants that are anchor nodes.

Let $\mathcal{T}$ be a RDAG and $w_1, \ldots, w_n = \mathbf{yield}(\mathbf{root}(\mathcal{T}))$ such that $w_1 \prec w_2 \prec \ldots \prec w_n$. The **right frontier** of $\mathcal{T}$ (written $\mathcal{T}_{\searrow}$) is defined as $\overrightarrow{\mathbf{path}}(\mathbf{root}(\mathcal{T}), w_n)$. The **left frontier** of $\mathcal{T}$ is defined as $\overleftarrow{\mathbf{path}}(\mathbf{root}(\mathcal{T}), w_1)$.

In order to compare nodes to establish their suitability for coordination, we introduce the notation $N_1 \sim N2$, which stands for the fact that $N_1$ and $N_2$ have the same value for both the *cat* and the *funct* features.

## 3. Alternation between parsing and resolving

The general idea of the algorithm is that control is successively exchanged between a *partial parser* and a *coordination resolver*. The sentence to parse is first split into segments following the punctuation and the coordination conjunctions. This does not require the bounds of the conjuncts to be determined at this step. Consider the following sentence:

(1) {*John knows Peter*}, {*whom Mary likes*} and {*Max hates*}, but {*never met him*}.

It is split into four segments delimited with curly brackets.

The parser then builds syntactic structures representing partial parses of the different segments. With the example, we obtain four syntactic structures corresponding to the four segments.

[1] In TAG, a node can also contain a mandatory adjunction, so a TAG tree is saturated only if all its mandatory adjunctions have been performed.

At this point, control is transferred to a selector, which picks two contiguous syntactic structures, $\mathcal{S}_\mathcal{L}$ and $\mathcal{S}_\mathcal{R}$, to be combined. The selector uses information coming from the different structures, for instance the category of their roots, but the purpose of this article is not to explain how the selector works.

If $\mathcal{S}_\mathcal{L}$ and $\mathcal{S}_\mathcal{R}$ are separated with a punctuation sign, control is then transferred to a punctuation resolver. If they are separated with a coordination conjunction, control is transferred to the coordination resolver. This article focuses on the coordination resolver only.

For instance, assume that in example (1), the selector has chosen the syntactic structures associated with *whom Mary likes* and *Nicolas hates* for $\mathcal{S}_\mathcal{L}$ and $\mathcal{S}_\mathcal{R}$. The syntactic structures $\mathcal{S}_\mathcal{L}$ and $\mathcal{S}_\mathcal{R}$ are lists of RDAGs.

Three types of sharing between conjuncts are distinguished and require a different treatment:

- In the coordination of constituents without sharing, the unique root of $\mathcal{L}$ (resp. $\mathcal{R}$) is coordinated with a node from the left frontier of $\mathcal{R}$ (resp. the right frontier of $\mathcal{L}$).
- In the coordination with peripheral sharing, the left (resp. right) frontier of $\mathcal{L}$ and the left (resp. right) frontier of $\mathcal{R}$ can share substructures (possibly introducing graphs as replacement for trees).
- In the coordination of argument clusters and coordination with verb gapping, a correspondence between the roots of two or three sub-RDAGs of $\mathcal{L}$ and the roots of two or three leftmost RDAGs of $\mathcal{S}_\mathcal{R}$ can be found with respect to certain conditions. A parallel structure is re-built by duplicating some parts of $\mathcal{L}$ and by combining them with the leftmost RDAGs of $\mathcal{S}_\mathcal{R}$.

The resolver uses a merging mechanism which combines $\mathcal{S}_\mathcal{L}$ and $\mathcal{S}_\mathcal{R}$ into a shorter list or RDAGs. In the two first cases, described in Sections 4 and 5, the merging implies the rightmost RDAG $\mathcal{L}$ of $\mathcal{S}_\mathcal{L}$ and the leftmost RDAG $\mathcal{R}$ of $\mathcal{S}_\mathcal{R}$. In the last case, described in Section 6, two or three RDAGs of the same side are concerned by the merging.

## 4. Constituent coordination without sharing

In the following, we use the notations $\mathcal{C}_\mathcal{L}$ and $\mathcal{C}_\mathcal{R}$ for the sub-RDAGs that are coordinated: the first step of the algorithm is to identify $\mathcal{C}_\mathcal{L}$ and $\mathcal{C}_\mathcal{R}$.

We assume that at least one the the RDAGs $\mathcal{L}$ or $\mathcal{R}$ is saturated and represents a complete constituent. The three examples below illustrate this case. The projections of $\mathcal{L}$ and $\mathcal{R}$ on the sentence are represented between square brackets. If one of them is saturated, its projection is represented in bold.

(2) [*Max introduces the son of his friend*]$_\mathcal{L}$ and [***Mary***]$_\mathcal{R}$ *to his director.*

(3) *Today* [***the engineer***]$_\mathcal{L}$ and [***the boss of the company***]$_\mathcal{R}$ *are coming.*

(4) [***John knows that Mary likes tea***]$_\mathcal{L}$ but [***she hates coffee***]$_\mathcal{R}$.

In example (2), $\mathcal{L}$ is not saturated because we consider that the verb *introduces* also requires an indirect object; in examples (3) and (4), both $\mathcal{L}$ and $\mathcal{R}$ are saturated.

To combine $\mathcal{L}$ with $\mathcal{R}$, one has first to select one of them that is saturated. Say that the selected saturated RDAG is $\mathcal{R}$, hence $\mathcal{C}_\mathcal{R} = \mathcal{R}$.

Then, in the right frontier $\mathcal{L}_\searrow$ of $\mathcal{L}$, we have to find a node representing a constituent that can be coordinated with the constituent represented by $\mathbf{root}(\mathcal{R})$. We consider that two constituents can be coordinated if they have the same grammatical category and the same syntactic function[2]. This is expressed with an equivalence relation between nodes, denoted $\sim$. Let $\mathcal{H}_\mathcal{L} = \{N \in \mathcal{L}_\searrow \mid N \sim \mathbf{root}(\mathcal{R})\}$. If $\mathcal{H}_\mathcal{L} = \emptyset$, the coordination fails. If $\mathcal{H}_\mathcal{L}$ has more than one node, there is coordination scope ambiguity, which is the case for all examples above. For instance, for sentence (2):

(2-a) *Max introduces the son of* $[$*his friend*$]_{\mathcal{C}_\mathcal{L}}$ and $[$*Mary*$]_{\mathcal{C}_\mathcal{R}}$ *to his director.*

(2-b) *Max introduces* $[$*the son of his friend*$]_{\mathcal{C}_\mathcal{L}}$ and $[$*Mary*$]_{\mathcal{C}_\mathcal{R}}$ *to his director.*

We pick a node $S$ from $\mathcal{H}_\mathcal{L}$. A new node $C$ is created and interposed between $S$ and its parent, and $\mathbf{root}(\mathcal{R})$ is made a right sister of $S$. The new node $C$ has the same category and the same syntactic function as $S$ and $\mathbf{root}(\mathcal{R})$.

In cases of extraction, the algorithm takes the barriers to extraction into account. Consider the following example:

(5) $[$***John knows Peter whom Mary likes***$]_\mathcal{L}$ and $[$***Nicolas hates him***$]_\mathcal{R}$.

The algorithm succeeds by considering the parse tree of *Nicolas hates him* as the selected saturated RDAG $\mathcal{R}$. The left RDAG $\mathcal{L}$ is the parse tree of *John knows Peter whom Mary likes*. In its right frontier, three nodes are candidate to coordination with the root $N_\mathcal{R}$ of $\mathcal{R}$, if we only consider their grammatical category S: the roots $N_{\mathcal{L}_1}$, $N_{\mathcal{L}_2}$ and $N_{\mathcal{L}_3}$ of the respective RDAGs of *John knows Peter whom Mary likes*, of *whom Mary likes*, and of *Mary likes*. The nodes $N_{\mathcal{L}_2}$ and $N_{\mathcal{L}_3}$ should be rejected, and it is done, if we consider the functions of the nodes: they have the function of noun modifier whereas $N_{\mathcal{L}_1}$ does not. $N_{\mathcal{L}_1}$ is therefore the only node that is equivalent to $N_\mathcal{R}$.

## 5. Peripheral sharing

In the previous section, the algorithm coordinates two independent RDAGs $\mathcal{C}_\mathcal{L}$ and $\mathcal{C}_\mathcal{R}$ by considering that the root of one is the conjunct of a node in the frontier for the other one. However, coordination often entails sharing between $\mathcal{C}_\mathcal{L}$ and $\mathcal{C}_\mathcal{R}$:

– $\mathcal{C}_\mathcal{L}$ and $\mathcal{C}_\mathcal{R}$ may have an identical syntactic context which is expressed with identical tree fragments over $\mathcal{C}_\mathcal{L}$ and $\mathcal{C}_\mathcal{R}$; when they are coordinated, the identical fragments are merged;

[2] We left aside the subtleties concerning the constraints in the coordination of constituents, such as coordination of unlikes.

– $\mathcal{C}_\mathcal{L}$ and $\mathcal{C}_\mathcal{R}$ may share sub-RDAGs representing common arguments or modifiers; sharing of sub-RDAGs is realized with common daughter nodes, which is possible because syntactic structures are DAGs and not trees.

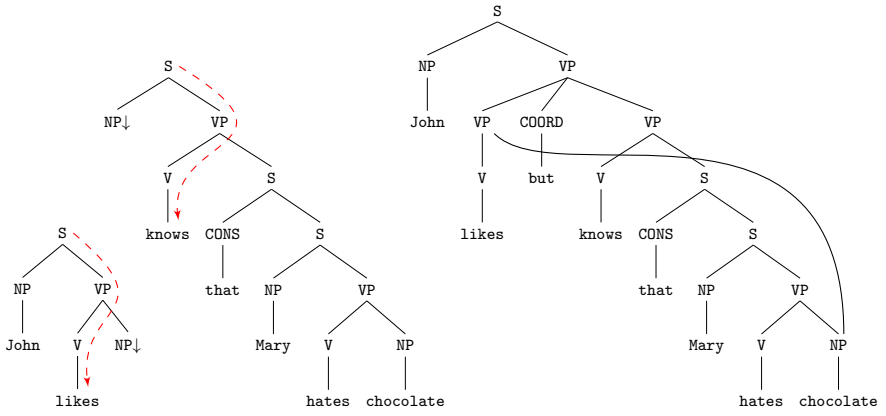
We call the first kind of sharing *context merging* and the second kind of sharing *argument sharing*—the latter includes the sharing of modifiers. The three examples below illustrate these two cases of sharing. Projections of $\mathcal{C}_\mathcal{L}$ and $\mathcal{C}_\mathcal{R}$ on the sentence appear between square brackets; projections of the shared part corresponding to argument sharing are underlined, and projections of the shared part corresponding to context sharing are overlined.

(6) $\overline{\textit{Nicolas}}$ [$\underline{\overline{\textit{often}}}$ *carries goods*]$_{\mathcal{C}_\mathcal{L}}$ and [*takes persons at the same time* $\underline{\textit{from Paris to Lyon}}$ $\underline{\textit{with his green truck}}$ ]$_{\mathcal{C}_\mathcal{R}}$.

(7) *John knows Peter* $\overline{\textit{whom}}$ [*Mary likes*]$_{\mathcal{C}_\mathcal{L}}$ and [*Nicolas hates*]$_{\mathcal{C}_\mathcal{R}}$.

(8) $\overline{\textit{John}}$ [*likes*]$_{\mathcal{C}_\mathcal{L}}$ but [*knows that Mary hates* $\underline{\textit{chocolate}}$]$_{\mathcal{C}_\mathcal{R}}$.

In examples (6) and (8), $\mathcal{C}_\mathcal{L}$ and $\mathcal{C}_\mathcal{R}$ represent the syntactic structure of two coordinated verb phrases, whereas in example (7) they represent incomplete sentences. To take peripheral sharing into account, it is necessary to extend the algorithm presented in Section 4.




(a) $\mathcal{L}$ and $\mathcal{R}$ (b) Resulting RDAG

Fig. 1: Algorithm run on sentence *John likes but knows that Mary hates chocolate.*

### 5.1. The run of the algorithm on an example

We first present the extension of the algorithm in an intuitive way on sentence (8). We travel along the right frontier of $\mathcal{L}$, that is `S`, `VP`, `V`, `likes`, and the left frontier of $\mathcal{R}$, that is `S`, `VP`, `V`, `knows`, from the top to the bottom.

At each step of the travel, we denote the current position of the examined node in the right frontier of $\mathcal{L}$ with the variable $N_{\mathcal{L}}$ and the current position of the examined node in the left frontier of $\mathcal{R}$ with the variable $N_{\mathcal{R}}$. We initialise $N_{\mathcal{L}}$ and $N_{\mathcal{R}}$ with two nodes of the respective frontiers. One of the nodes must be a root, and $N_{\mathcal{L}} \sim N_{\mathcal{R}}$ must hold. In our example, we have only one possibility: $N_{\mathcal{L}}$ and $N_{\mathcal{R}}$ are the roots $S$ of the two frontiers.

Since we aim at coordinating constituents at the lowest level in the syntactic structure, we try to merge as much as possible the subtrees rooted at $N_{\mathcal{L}}$ and $N_{\mathcal{R}}$. To examine if $N_{\mathcal{L}}$ and $N_{\mathcal{R}}$ can be merged, we have to look at their daughters. We start with their daughter nodes $D_{\mathcal{L}}$ and $D_{\mathcal{R}}$ on the frontiers. Both have the same category `VP` and they verify the condition $D_{\mathcal{L}} \sim D_{\mathcal{R}}$. Then, we have to look at the right sisters of $D_{\mathcal{L}}$ and the left sisters of $D_{\mathcal{R}}$. There is only a left daughter of $D_{\mathcal{R}}$, a substitution node `NP`. This node can be saturated by merging it with the corresponding left sister of $D_{\mathcal{L}}$. All conditions for merging $N_{\mathcal{L}}$ and $N_{\mathcal{R}}$ hold and we merge them as their daughter nodes `NP`.

The current values of $N_{\mathcal{L}}$ and $N_{\mathcal{R}}$ become the nodes $D_{\mathcal{L}}$ and $D_{\mathcal{R}}$. Then, we repeat the same step of computation from the new values of $N_{\mathcal{L}}$ and $N_{\mathcal{R}}$. The new values of $D_{\mathcal{L}}$ and $D_{\mathcal{R}}$ are the `V` daughters. The first condition for merging the new values of $N_{\mathcal{L}}$ and $N_{\mathcal{R}}$ holds but not the second one: the right sister of $D_{\mathcal{L}}$, a substitution node `NP`, cannot merge with a right sister node of $D_{\mathcal{R}}$. The process of merging halts and the proper coordination process starts.

The current nodes $N_{\mathcal{L}}$ and $N_{\mathcal{R}}$ represent the two constituents having to be coordinated. For this, we insert a new node $C$ between $N_{\mathcal{L}}$, $N_{\mathcal{R}}$ and their common mother node. We add a new subtree of $C$ between $N_{\mathcal{L}}$ and $N_{\mathcal{R}}$ with a unique daughter node, the anchor of the conjunction, *but*.

Then, the descent along the two frontiers continues. $D_{\mathcal{L}}$ has a right sister $S$, which is a substitution node `NP`. This node must be saturated, the constraints between the yields of $N_{\mathcal{L}}$ and $N_{\mathcal{R}}$ must be obeyed. They must be adjacent. As a consequence, $S$ is saturated by merging it with a node $M$ of the right frontier of the subtree rooted at $N_{\mathcal{R}}$. This frontier is `VP`, `S`, `S`, `VP`, `NP`, `chocolate`. Because of the constraint $S \sim M$, the only possible value for $M$ is node `NP`. Note that the path from $N_{\mathcal{L}}$ to $S$ does not have the same length and the same labeling as the path from $N_{\mathcal{R}}$ to $M$. The two paths nonetheless obey certain constraint, expressed as an equivalence relation between paths, which will be discussed later.

After merging $S$ and $M$, the descent continues. The current values of $N_{\mathcal{L}}$ and $N_{\mathcal{R}}$ become the nodes $D_{\mathcal{L}}$ and $D_{\mathcal{R}}$. The new values of $D_{\mathcal{L}}$ and $D_{\mathcal{R}}$ are the respective anchors `likes` and `knows`. Since they have no sister node, the algorithm ends, and the resulting syntactic structure is that of Fig. 1b.

### 5.2. The algorithm

As the previous example illustrates it, there are two stages in the algorithm:

- In the *context merging* stage, the two sub-RDAGs rooted at the initial values of $N_{\mathcal{L}}$ and $N_{\mathcal{R}}$ are merged as deeply as possible up to the coordinated nodes;
- These two nodes are the starting point of the *argument sharing* stage, in which substitutions and adjunctions to be realized on the right of the right

```
choose_one (N_L, N_R) in L_↘ × R_↙ the right frontier of L and the left frontier of R, such that N_L ∼ N_R and one of them is a root
    while not FINISH do
        D_L ← the daughter nodes of N_L on the right frontier;
        D_R ← the daughter nodes of N_R on the left frontier;
        if D_L ∼ D_R then
            H_L ← the list of the left sister nodes of D_L;
            H_R ← the list of the left sister nodes of D_R;
            if H_L and H_R have the same length n then
                for 1 ≤ k ≤ n do
                    if the subtrees rooted at H_L[k] and H_R[k] can be merged then
                        merge them
                    else
                        FINISH ← True; BREAK
                    end
                end
            else
                FINISH ← True
            end
        else
            FINISH ← True
        end
        if not FINISH and D_L and D_R and not anchors then
            N_L ← D_L; N_R ← D_R
        end
    end
    Insert a new node C between N_L and N_R and their eventual common mother node
end
```

**Algorithm 1:** Left context merging

frontier of $\mathcal{L}$ are performed by sharing with realized substitutions and adjunctions on the right frontier of $\mathcal{R}$; in a symmetrical way, substitutions and adjunctions on the left of the left frontier of $\mathcal{R}$ are realized by sharing with substitutions and adjunctions on the left frontier of $\mathcal{L}$.

The first stage is presented in Algorithm 1. To simplify the presentation, only left merging, as in the example, is considered.

The second stage of the algorithm for argument sharing is shown in details in Algorithm 2. Again, only right argument sharing is considered. It starts at the end of the first stage with $N_{\mathcal{L}_0}$ and $N_{\mathcal{R}_0}$ being the roots of the two coordinated structures $\mathcal{C}_\mathcal{L}$ and $\mathcal{C}_\mathcal{R}$. From $N_{\mathcal{L}_0}$, we go down along the right frontier of $\mathcal{C}_\mathcal{L}$ until we find a node which has substitution leaves as daughter nodes at the right of the daughter node on the frontier, or where a right adjunction is allowed. The current position along this frontier is represented with the variable $N_\mathcal{L}$. To

simplify the presentation of the algorithm, we assume that $N_{\mathcal{L}}$ has one daughter at most that is on the right of the frontier.

In parallel, from $N_{\mathcal{R}_0}$, we go down along the right frontier of $\mathcal{C}_{\mathcal{R}}$ to find a node $M_{\mathcal{R}}$ that shares a sub-RDAG with $N_{\mathcal{L}}$. The current position from which we look for $M_{\mathcal{R}}$ is represented with the variable $N_{\mathcal{R}}$.

Here is how the algorithm deals with substitution and adjunction sharing:

**Substitution sharing.** When $N_{\mathcal{L}}$ has a rightmost daughter which is a substitution leaf $N_r^{\downarrow}$, we search top-down for the first node $M_{\mathcal{R}}$ in the right frontier of the sub-RDAG rooted at $N_{\mathcal{R}}$ that has a rightmost daughter $M_r$ able to fill the substitution leaf $N_r^{\downarrow}$. The paths from $N_{\mathcal{L}}$ to $N_r^{\downarrow}$ and from $N_{\mathcal{R}}$ to $M_r$ must be equivalent in a sense that will be specified shortly. Nodes $M_r$ and $N_r^{\downarrow}$ are merged.

**Adjunction sharing.** When $N_{\mathcal{L}}$ allows for an adjunction, the only possible adjunction is a right adjunction[3] we search top-down for the first node $M_{\mathcal{R}}$ in the right frontier of the sub-RDAG rooted at $N_{\mathcal{R}}$ that results from a right adjunction and that can share this adjunction with $N_{\mathcal{L}}$. To perform it, an equivalence condition on paths in the same sense as for substitution must be verified. If we find such a node $M_{\mathcal{R}}$, then we have to decide (line 5 of the algorithm) whether to share the adjunction (lines 6 to 9) or not.

Our presentation of the algorithm makes it non-deterministic: a choice is made among the possible conjuncts in stage 1, and choices are made as well between possible substitutions and adjunctions in stage 2. A deterministic formulation of the algorithm would simply need to enumerate the list of solutions.

Graph modifications the algorithm can use for substitution and for adjunction sharing are shown in Fig. 2. The top and bottom parts of the figures describe the graph before and after the modifications; grey nodes identify $N_{\mathcal{L}}$ and $N_{\mathcal{R}}$ in the top part, and the next iteration's starting point in the bottom part.

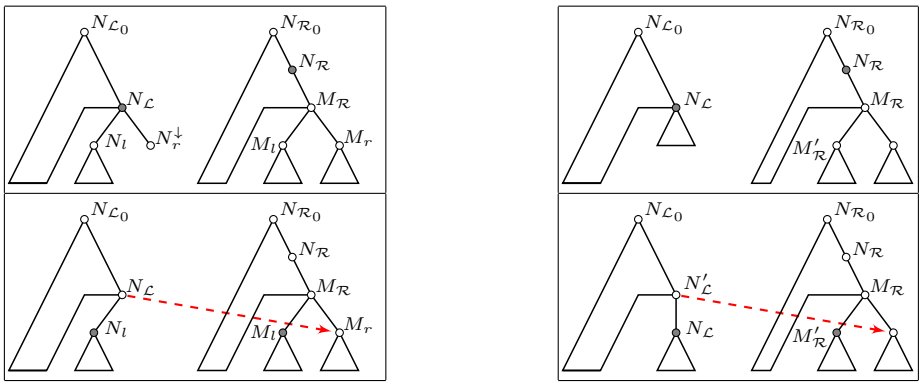



Fig. 2: Substitution sharing (on the left) and adjunction sharing (on the right)

[3] A right adjunction comes from a right auxiliary tree, that is a tree where the foot node is the leftmost leaf;

```
Initialise N_L and N_R with the roots N_L0 and N_R0 of the structures being
coordinated after the stage of context merging;
while not FINISH do
    if some adjunction is allowed at N_L then
        choose_one M_R in N_R↘ such that path→(N_L0, N_L) ≈ path→(N_R0, M_R)
        and M_R is the root of a right auxiliary tree
            choose
                (see adjunction sharing, Fig. 2);
                Insert a node N'_L between N_L and its mother node with the
                same category;
                Add all daughters of M_R on the right of its foot node M'_R as
                right daughters of N'_L;
                N_R ← M'_R
            or Do nothing
        end
    else if N_L has a rightmost daughter which is a substitution node N_r then
        (see substitution sharing, Fig. 2);
        choose_one M_R in N_R↘ such that path→(N_L0, N_L) ≈ path→(N_R0, M_R)
        and M_R has a rightmost daughter M_r ∼ N_r
            N_L ← N_l where N_l is the immediate left sister of N_r;
            N_R ← M_l where M_l is the immediate left sister of M_r;
            Merge M_r and N_r
        end
    else neither adjunction sharing nor substitution sharing
        if N_L is a leaf then
            FINISH
        else
            N_L ← the rightmost daughter of N_L
        end
    end
end
```

**Algorithm 2:** Right argument sharing

We now define the equivalence relation between paths $\approx$. It depends on the language and on the choice of the representation of syntactic trees. There is no room to define the relation exhaustively for a given language and choice of syntactic representation here, but we can sketch it. In our settings, it is the smallest equivalence relation such that: *a*) if $N \sim M$ then $\{N\} \approx \{M\}$, *b*) $p \approx p.(\mathtt{S}, \mathrm{obj}).(\mathtt{VP}, \mathrm{head})$, and *c*) $p.(X, \mathit{funct}) \approx p.(X, \mathit{funct}).(X, \mathit{funct})$. The condition *b* is meant to go through object clauses, whereas *c* is meant to go through constituents with adjoined modifiers, such as in sentence (8).

## 6. Verb gapping and argument clusters

Verb gapping is a grammatical construction in which two clauses are coordinated and the verb of the second clause is replaced with a gap. This means that the verb of the first conjunct is picked up again as the verb of the second conjunct.

Argument clusters is another grammatical construction in which the second conjunct is a cluster of constituents, which are all arguments of the same predicate which is absent from this conjunct. It is a resumption of the predicate that is the head of the first conjunct.

Sentences (9) and (10) illustrate verb gapping and argument cluster. The projections of the left and right RDAGs are between square brackets. The part of the left segment that is replaced with a gap in the right segment is in a box.

(9) [*John* $\boxed{\textit{buys}}$ *a car*] and [*Maria*] [*a shower*].

(10) [$\boxed{\textit{Nicolas carries goods}}$ *from Paris to Lyon*] and [*from Lyon*] [*to Nancy*].

Verb gapping and argument clusters are processed with a common algorithm because, in both situations, the head of the syntactic structure of the second conjunct is lacking and one has to reuse the head of the first conjunct to build this syntactic structure.

The input of the algorithm is a RDAG $\mathcal{L}$ representing the syntactic structure of the parsed phrase on the left of the conjunction and a list of RDAGs $\mathcal{R}_1, \ldots, \mathcal{R}_n$ representing the partial parse of the phrase on the right, ordered with respect to the linear order of the sentence. The first step of the algorithm is to establish a bijection between $\mathbf{root}(\mathcal{R}_1), \ldots, \mathbf{root}(\mathcal{R}_p)$ of the $p$ first RDAGs from the $n$ right RDAGs and $p$ nodes $N_1, \ldots, N_p$ from $\mathcal{L}$, verifying the following properties:

- For any $i$ such that $1 \leq i \leq p$, $N_i \sim \mathbf{root}(\mathcal{R}_i)$;
- The projections on the sentence of $N_1, \ldots, N_p$ constitutes two continuous segments[4] immediately on the left of the conjunction, separated by a word $w$.

The second step is to select a node $N$ that dominates $N_1, \ldots, N_p$ in $\mathcal{L}$, and whose head is $w$. This node is necessarily a node from the right frontier of $\mathcal{L}$ and it represents the left conjunct. The third step is to duplicate the sub-RDAG $\mathcal{T}$ rooted at $N$ with $N_1, \ldots, N_p$ and $w$ as leaves. In the clone $\mathcal{T}'$, the leaves $N_1, \ldots, N_p$ are replaced with $\mathbf{root}(\mathcal{R}_1), \ldots, \mathbf{root}(\mathcal{R}_p)$ but $w$ is shared with $\mathcal{T}$. Finally, $\mathcal{T}$ and $\mathcal{T}'$ are coordinated as in Section 4.

## 7. Results

The algorithm was simulated on the 14 sentences from section 0 of the Penn Treebank that contain either peripheral sharing or gapping. The algorithm returned at least a parse for all sentences but two, which both involved coordination of unlikes, which our algorithm cannot handle. An average of 1.3 parses was returned per sentence. The parse from the Penn Treebank was always among the

[4] The first one can be empty.

returned parses, and all but three parses arguably described genuine ambiguity in the scope of coordination[5].

## 8. Conclusion

In this article we presented an algorithm that allows for parsing of coordination outside the scope of grammar. This algorithm is able to interact with any parser using a coordination-less grammar, thus returning incomplete parses, and reconstruct a complete parse graph from the fragments returned by the parser. While we used TAGs to aid with the presentation and evaluation, we believe this algorithm could be applied to other formalisms with little adaptation.

A first evaluation shows that the algorithm deals appropriately with most types of coordination, including right node raising and gapping. Unsurprisingly, the only cases where it failed to parse coordination were with coordination of unlikes. The algorithm adds little ambiguity, and most of it is justified as genuine coordination scope ambiguity.

Future works will involve a more full-fledged evaluation within different grammatical frameworks, with special respect for the effect of different equivalence relations between nodes of two graphs.

## References

1. Beavers, J., Sag, I.: Coordinate Ellipsis and Apparent Non-Constituent Coordination. 11th Intl. Conf. on HPSG (2004)
2. Bourigault, D.: Un analyseur syntaxique opérationnel : SYNTEX. Habilitation thesis, Université Toulouse–Le Mirail (2007)
3. Frank, R.: Coordinating parsing and grammar. GLOT Int'l 1(4), 4–8 (1995)
4. Joshi, A., Schabes, Y.: Tree-adjoining grammars. Handbook of Formal Languages, pp. 69–123 (1997)
5. Kaplan, R., Maxwell, J.: Constituent coordination in lexical-functional grammar. Proc. of the 12th conf. on Computational linguistics. pp. 303–305 (1988)
6. Kübler, S., McDonald, R., Nivre, J.: Dependency Parsing. Morgan and Claypool (2009)
7. Le Roux, J., Perrier, G.: La coordination dans les grammaires d'interaction. TAL 47(3), 89–113 (2006)
8. Mouret, F.: Grammaire des constructions coordonnées. Coordinations simples et coordinations à redoublement en français contemporain. Ph.D. thesis, Université Paris 7 (2007)
9. Sarkar, A., Joshi, A.: Coordination in tree adjoining grammars: Formalization and implementation. 16th conf. on Computational linguistics. pp. 610–615. ACL (1996)
10. Steedman, M.: Dependency and Coordination in the Grammar of Dutch and English. Language 61(3), 523–568 (1985)
11. White, M.: Efficient Realization of Coordinate Structures in Combinatory Categorial Grammar (2004)

[5] Detailed results: `http://wikilligramme.loria.fr/doku.php?id=ecp:ecp`