From Semantic Graphs to Logic Formulae

Guy Perrier, Mathieu Morey, Maxime Amblard

From Semantic Graphs to Logic Formulae

Guy Perrier, Mathieu Morey, Maxime Amblard
LORIA - Nancy-Université
Nancy, France
{guy.perrier, mathieu.morey, maxime.amblard}@loria.fr

Abstract

We present a general two-step algorithm which transforms a graph expressing the semantic dependencies between the words of an utterance into logic formulae representing the different semantic interpretations of the utterance.

The algorithm focuses on the scopal elements of the utterance, i.e. quantifiers and scopal predicates. First, the scope of every scopal element is computed as a subgraph of the whole graph, given an order on these elements. Second, we use these scopes with their order to build a logic formula incrementally from the most internal to the most external scopal elements.

1 Introduction

There are two opposite directions in the representation of the semantics of utterances in natural language: the use of graphs which directly express the semantic dependencies between the words from their syntactic dependencies or the use of logic formulae which express the truth conditions of the utterances.

- The first approach is generally associated with Dependency Grammars (Mel'cuk, 1988). Its advantage is that semantic graphs are easily computed from the syntax. They are also very readable and easy to manipulate. Finally, they avoid combinatory explosion by factorizing different readings of an utterance. This last advantage is at the same time a major drawback because there is no simple way of computing a logical representation from such a graph. This is not surprising since logic is often viewed as external to the field of linguistics in that approach (Mel'cuk, 1988).
- The use of logic formulae presents a major significance: it allows reasoning from natural language utterances. Categorial Grammars are one of the most characteristic illustrations of this approach (Carpenter, 1998). Contrary to the first approach, every word is associated in a lexicon with a syntactic type and with a semantic term. The syntactic type gives the way of combining the word with other words to build a sentence. The semantic term represents the contribution of the word to the whole logical formula representing the semantics of the sentence. The computation is based on the compositionality principle: the logical formula is built step by step under the control of the syntax. The composition rules being very simple, the difficulty of the task is concentrated in finding the appropriate semantic terms for the words in the lexicon and it is bounded by the specific limitations of the composition principle. To overcome these limitations, the lexical entries are complexified, using continuations for instance (de Groote, 2006; Barker and Shan, 2008).

There are intermediate approaches, in particular formalisms using underspecified trees (Copestake et al., 2005; Bunt and Muskens, 2007). Underspecified trees represent underspecified logic formulae in a flat setting. They are viewed as sets of logic formulae constituting their models and expressing the different readings of the corresponding utterance. The problem is that such trees often overgenerate and the only solution to solve this problem is to enumerate the correct models but it looses the interest of underspecification (Ebert, 2005).

To benefit from both of the first approaches, we propose semantic graphs that represent the semantic dependencies between the words of utterances, in such a way that they allow an easy translation

into logic formulae. A crucial point for this translation is the management of quantifiers. Classically, a quantifier is formalized as a function with three arguments : a *variable individual* represents the quantification target; a property, called its *restriction*, delimits the range of the individual; another property is an assertion about this individual and we call it its *body*, according to (Hobbs and Shieber, 1987). The union of the restriction and the body represents the *scope* of the quantifier[1].

The main originality of our graphs with respect to the semantic graphs used in Dependency Grammars lies in the representation of a quantified individual as an argument of a predicate : if the predicate contributes to the restriction of the quantifier, the representation is not the same as if it contributes to the body. Consider a quantifier Q, which is represented with a node N_Q, and a predicate P, which is represented with a node N_P:

- if P contributes to the body of Q, then N_P is a predecessor of N_Q;
- if P represents the semantic head to which quantification applies, N_P is the unique direct successor of N_Q; it means that the predicate is reified and it represents the range of individuals to which quantification applies; P can also contribute to restrict or to transform this range; in this case, N_P is a predecessor of the direct successor of N_Q, but not a predecessor of N_Q itself.

Such a distinction allows the scopes of quantifiers to be computed from the structure of semantic graphs, but it has to be completed with an order between the scopal elements, i.e quantifiers and scopal predicates: scopal predicates are predicates with a fixed scope over one of their arguments, which is a proposition and which can be quantified. In other terms, the scope of a quantifier can be inserted between a scopal predicate and its propositional argument. If the scope of every scopal predicate is fixed according to some constraints, and if an order between the scopal elements is chosen, all scopes can be computed deterministically from the most internal elements to the most external elements. Of course, not every order can be chosen because of some constraints

[1]In several works, the scope of a quantifier denotes what we call its body. In our use, the word *scope* rather corresponds to its classical meaning in first order logical formulae.

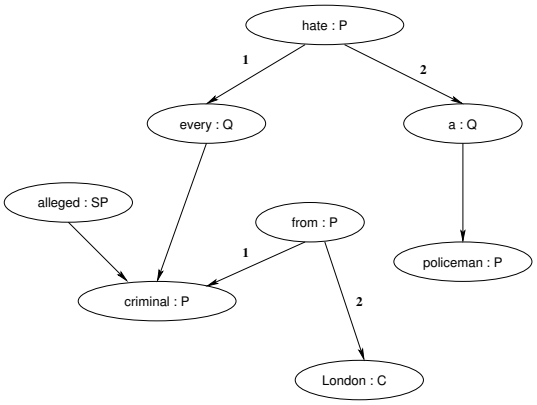

Figure 1: Semantic graph for the sentence **Every alleged criminal from London hates a policeman.**

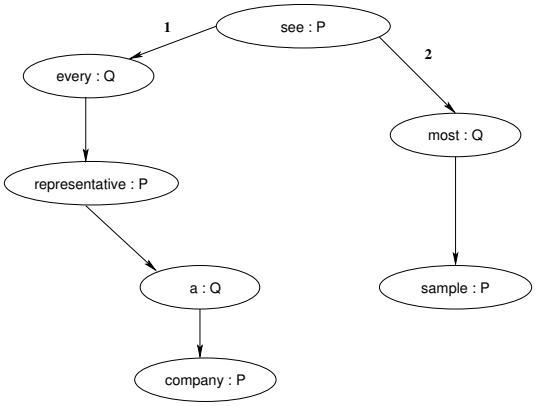

Figure 2: Semantic graph for the sentence **Every representative of a company saw most samples.**

derived from the structure of the semantic graph but also from linguistic considerations.

All scopes being determined, it is possible to compute a logic formula representing the truth conditions of the utterance dynamically. The idea is to attach formulae to the vertices of the semantic graph and to enrich them by visiting all quantifiers according to their order from the most internal to the most external one.

2 The definition of semantic graphs

A semantic graph, denoted *sem-graph*, is a DAG with vertices and edges respectively representing *semantemes* and *semantic roles*. Figure 1 shows an example of semantic graph.

A vertex *N* is labelled with a *name*, denoted *name(N)*, and every name has a *type* chosen among four types:

- *P for predicates:* a predicate may take from 0 to n arguments to become a proposition

which can be true or false. A vertex labelled with a predicate is called a P-vertex. The arguments of a P-vertex *N* are represented by its direct successors and their set is denoted *arg(N)*. To distinguish their roles in the predicate, their incident edges are labelled with positive integers: 1, 2, 3 ...

- *SP for scopal predicates:* a scopal predicate *N* is a predicate with a specific argument *target(N)*, which is a predicate. For instance, in figure 1, **alleged** is a scopal predicate with **criminal** as its target. The specificity of a scopal predicate is that quantifier scopes can be inserted between the predicate and its argument. A vertex labelled with a scopal predicate is called SP-vertex. Its target is one of its direct successors.

- *Q for quantifiers:* a quantifier requires three arguments: a variable individual, which is concerned in quantification, its *restriction* and its *body*. A vertex labelled with a quantifier is called a Q-vertex. A Q-vertex *N* has exactly one direct successor *target(N)*, which is a P-vertex and which represents the *kernel of its restriction*. In figure 1, quantifiers **every** and **a** respectively have **criminal** and **policeman** as their target. The whole restriction of a Q-vertex is computed deterministically. The field is also computed but there is a part of non determinism in this computation.

- *C for constant individuals:* they generally represent named entities. A vertex labelled with a constant individual is called a C-vertex. It has no successor. In figure 1, there is one C-vertex, **London**.

A vertex that is either a Q-vertex or an SP-vertex is called an S-vertex.

Well-formed semantic graphs must respect the following constraints :

- Every P-vertex without argument has exactly one Q-vertex as a direct predecessor.

- Every P-vertex with arguments has at most one Q-vertex as a direct predecessor[2].

- Every SP-vertex is a source node of the whole graph.

[2]If a P-vertex has one Q-vertex as a direct predecessor, then it represents a reified predicate: the predicate is taken as a class of individuals.

To show the fundamental difference between our semantic graphs and the semantic graphs used in Dependency Grammars, let us consider again the sentence **every alleged criminal from London hates a policeman**. Its semantics would be represented in Dependency Grammars with a graph similar to that one from figure 1, with one essential difference: the direct successors of the P-vertex **hate** would be P-vertices **criminal** and **policeman**. There would be no distinction between **hate** and **from** in their contribution to the quantifier **every**. To indicate that **hate** contributes to the body of **every**, whereas **from** contributes to its restriction, we distinguish their direct successors: in the first case, we choose the Q-vertex and in the second case, we choose its target.

3 Semantic graphs and scopes

Computing formulae which represent the semantics of an utterance is problematic because of the different possible sources of ambiguities this utterance can contain. In this paper, we focus on scopal ambiguity and thus do not consider lexical ambiguity.

In our work, scopes are defined as subgraphs of the original sem-graph. SP-vertices scopes are fixed non deterministically. Q-vertices scopes are determined in two steps: the structure of the sem-graph implies deterministically the *restriction* and non deterministically the *body*. This indetermination is lifted by fixing a partial government order on the S-vertices.

Definitions - Scope: In a graph (or subgraph), a vertex without any incoming edge is called a *source*. The set of source vertices of a graph (or subgraph) G is denoted by *source(G)*. Let *down-subgraph* be the function which associates to a set of vertices the subgraph G' which is accessible from them and such that for all Q-vertices in G', there is no edge coming out of them in G'[3].

The scope of an S-vertex is a *down-subgraph* of the considered graph. Consequently, calculating a scope consists in determining its *source*, given a specific order on scopal elements.

Scope of an SP-vertex: The maximal scope of an SP-vertex N_{sp}, denoted *max-scope(N_{sp})*, is the biggest subgraph over which N_{sp} can scope. *Max-scope(N_{sp})* is the *down-subgraph* of either its

[3]Remark that the *down-subgraph* of any Q-vertex is a part of its restriction.

sources or *target(N_{sp})* if *source(max-scope(N_{sp}))* is empty. A node *M* is in *source(max-scope(N_{sp}))* iff it is in *source(G)* and there exists a path from *M* to *target(N_{sp})* which does not contain any Q-vertex. The scope of an SP-vertex N_{sp}, denoted *scope(N_{sp})*, is the *down-subgraph* of an arbitrarily chosen subset of *source(max-scope(N_{sp}))*, which must contain at least *down-subgraph(target(N_{sp}))*.

Scope of a Q-vertex: Every Q-vertex N_q has a *restriction*, denoted *restrict(N_q)* and a *body*, denoted *body(N_q)*. The restriction of N_q is the *down-subgraph* accessible from both *restrict-ker(N_q)* and the source vertices of the sem-graph which are predecessors of *restrict-ker(N_q)*, except N_q itself and its predecessors.

The *minimal body* of N_q, denoted *min-body(N_q)* is the *down-subgraph* that has the direct predecessors of N_q as its source. The *minimal scope* of N_q, denoted *min-scope(N_q)*, is the union of *restrict(N_q)*, *min-body(N_q)* and the edge from N_q to *restrict-ker(N_q)*.

Govern order: The set of SP-vertices of a semantic graph (or subgraph) G is called *Scopal Predicate Nodes*, denoted *SPN*(G). The set of Q-vertices of a semantic graph (or subgraph) G is called *Quantifier Nodes*, denoted *QN*(G). The set of S-vertices of a semantic graph (or subgraph) G is called *Scopal Nodes*, denoted *SN*(G).

Let G be a semantic graph, *govern* is the relation of inclusion on SN(G) scopes. This relation is denoted by $\succ^*$. If the scope of b is in the scope of a then $a \succ^* b$. The relation of direct governance, denoted by $\succ$, is called the *direct-govern*. $a \succ b$ implies that there is no intermediary S-vertex between a and b. This relation is used here to define partial orders.

The order on the elements of SN(G), which is use in the algorithm, is based on the *govern* relation. This order is not completely arbitrary but depends on both the structure of G and linguistic knowledge which are beyond the scope of this paper. This order is determined by the following rules which use the restriction, min-body and max-scope of all elements of SN(G):

If an S-vertex A governs an S-vertex B (A$\succ$B), then one of the following assertions holds:

1. A is a Q-vertex, B is a SP-vertex and (B $\in$*restrict*(A) or A $\in$ *scope*(B))
2. A and B are Q-vertices and B $\in$ *min-body*(A)
3. A is a SP-vertex, B $\in$ *scope*(A)
4. There is a S-vertex C such that A$\succ^*$C and B$\succ^*$C

Conversely:

I. If A is a Q-vertex, B is an SP-vertex and B $\in$ *restrict*(A) then A$\succ^*$B

II. If A is a Q-vertex, B is an SP-vertex and *scope*(B)$\cap$*min-body*(A)$\neq$ $\emptyset$ then A$\succ^*$B or B$\succ^*$A

III. If A and B are Q-vertices and (B $\in$ *min-body*(A) or B $\in$ *restrict*(A)) then A$\succ^*$B or B$\succ^*$A

IV. If A and B are SP-vertices and *scope*(A) $\cap$ *scope*(B) $\neq \emptyset$ then A$\succ^*$B or B$\succ^*$A

V. If A and B are S-vertices and A$\succ^*$C and B$\succ^*$C then A$\succ^*$B or B$\succ^*$A

In the example of figure 1, we distinguish one SP-vertex (**alleged**) and two Q-vertices (**every** and **a**). Rule I. implies that **every**$\succ^*$**alleged**. Rule III. implies that **a** and **every** must be in a govern relation (**a**$\succ^*$**every** or **every**$\succ^*$**a**). Two acceptable orders derive from what precedes:

- **a**$\succ$**every**$\succ$**alleged**
- **every**$\succ$**alleged** and **every**$\succ$**a**

In the following, we will present the computation with the following order: **every**$\succ^*$**alleged** and **every**$\succ^*$**a**.

According to (Hobbs and Shieber, 1987), from the sem-graph fig. 2 associated to the classical example *Every representative of a company saw most samples*, we must derive only five formulae. Using our rules to determine acceptable orders, we derive from rule IV. that on one hand **every** and **a** and on the other hand **every** and **most** must be ordered:

- **a**$\succ$**every**$\succ$**most**,
- **most**$\succ$**every**$\succ$**a**,
- **every**$\succ$**a** and **every**$\succ$**most**,
- **a**$\succ^*$**every** and **most**$\succ^*$**every**, for which rule V. implies that **a** and **most** are ordered, thus:
  - **a**$\succ$**most**$\succ$**every**
  - **most**$\succ$**a**$\succ$**every**

We have derived five acceptable orders to which the second algorithm associates a formula.

Body of a Q-vertex: Given an acceptable order on the S-vertices, we define the *body* of a Q-vertex, denoted *body(N_q)* recursively with the following rules:

- If N_q does not govern any S-vertex, then *body(N)* is equal to *min-body(N)*.
- If N_q directly governs an SP-vertex which belongs to its restriction, then *body(N)* remains the same.
- If N_q directly governs an SP-vertex M_{sp} which does not belong to its restriction, then *body(N)* is augmented with *scope(M_{sp})*.
- If N_q directly governs a Q-vertex M_q, then *body(N)* is augmented with the part of *scope(M_q)* which is outside *restrict(N_q)*.

The scope of a Q-vertex N_q, denoted *scope(N_q)*, is the subgraph which is the union of *restrict(N_q)*, *body(N_q)*, the edge from N_q to *restrict-ker(N_q)* and the restriction of all the Q-vertices in its restriction.

4 The construction of scopes from an order on scopal elements

This section presents the algorithm which determines the scope of the Q-vertices. The full algorithm is presented in algorithm 1, page 6.
Input: a sem-graph G, the *restriction* and *min-body* of its Q-vertices, the scope of its SP-vertices and a govern order on the S-vertices[4].
Output: the body of every Q-vertex of the sem-graph.
One by one, each Q-vertex is taken out of the set of non-explored Q-vertices, denoted NE, and its scope is computed. The order in which Q-vertices are taken depends on the govern order: the chosen Q-vertex must be a minimal element of the govern order on the Q-vertices which still belong to NE.

In our example, given the semantic graph 1, the sets *source(restriction)* and *source(min-body)* for the two Q-vertices are:

	source(restrict)	*source(min-body)*
every	{**alleged**, **from**}	{**hate**}
a	{**policeman**}	{**hate**}

[4]Note that for a sem-graph, this algorithm runs for every combination between an order and the possible scopes of the SP-vertices.

The source of the scope of the SP-vertex **alleged** could be either **criminal** or **criminal** and **from**. To illustrate the way our algorithm works, we choose the latter, as well as the following order on the scopal vertices: **every** $\succ^*$ **a** and **every** $\succ^*$ **alleged**.

For every Q-vertex *N* the algorithm computes its body. We could decompose the process in three different parts: extend the body upwards (lines 6 to 11), fit the body by removing the restriction and adding the body of the Q-vertices that N directly governs (lines 12 to 16) and adding the scope of the SP-vertices that N directly governs (lines 17 to 20).

To extend the body upwards, we replace each source s of its *min-body* by one of its predecessors p. p is chosen non deterministically and such that there is a path without any Q-vertex between s and p.

After this extension process has occurred, all the vertices in the scope of an S-vertex v_2 which is directly governed by a Q-vertex v_1 must belong to the scope of v_1.

In particular, if v_2 is a Q-vertex, there can be some vertices which are both in *source(body(v_1))* and in the *restriction* of v_2. These vertices must be removed from *source(body(v_1))* because the information they provide is already added by the direct govern relation.
Moreover, if v_2 is a Q-vertex, *body(v_2)* must be a subgraph of *scope(v_1)*. Thus, all source vertices of *body(v_2)* which do not already belong to *scope(v_1)* have to be added to *source(body(v_1))*.

If v_2 is an SP-vertex, *scope(v_2)* must be a subgraph of *scope(v_1)*. If v_2 is in *restrict(v_1)* then *scope(v_2)* is a subgraph of *restrict(v_1)*. Thus, all source vertices of *scope(v_2)* which do not already belong to *body(v_1)* have to be added to *source(body(v_1))*. Before adding a vertex to *source(body(v_1))*, the algorithm removes all the successors of this vertex[5].

In the example showed in fig. 1, the minimal element for the govern order is **a** for which: *source(restrict*(**a**)*)*={**policeman**} and *source(min-body*(**a**)*)*={**hate**}. NE_{local} is initialized with the source **hate**. The only predecessor of **a** which is not a Q-vertex is **hate**. We add **hate** to *source(body(**a**))*. The Q-vertex **a** does not govern any S-vertex, then its treatment is done.

The next minimal element is the **every** Q-vertex

[5]Remark that adding a predecessor of a source vertex s in a subgraph transforms s to a simple vertex in the extended subgraph.

for which: *source(restrict*(**every**)*)*=$\{$**alleged**, **from**$\}$ and *source(min-body*(**every**)*)*=$\{$**hate**$\}$. For *body*(**every**) and *restriction*(**a**) are disjoint and *body*(**a**) is included in *scope*(**every**), and *scope*(***alleged***) is in *restrict*(***every***), the algorithm states that *source(body*(**every**)*)*=*source(min-body*(**every**)*)*.

Input: a Sem-graph G, a govern order on G and scopes of SP-vertices
Output: complete description of scope for G

1 NE$\leftarrow$ QN(G);
2 **while** *NE* $\neq \emptyset$ **do**
3 Select $N \in$ NE such that **N** is minimal for the govern relation on NE;
4 Delete **N** from NE;
5 *source(body(N))* $\leftarrow$ { };
6 NE$_{local}$ $\leftarrow$ *source(min-body(N))*;
7 **while** *NE*$_{local}$ $\neq \emptyset$ **do**
8 Choose M $\in$ NE$_{local}$;
9 Choose a non Q-vertex M' such that M' is a predecessor of M without Q-vertex between them;
10 Add M' to *source(body(N))*;
11 Delete M from NE$_{local}$;
12 **for** *every Q-vertex M directly governed by N* **do**
13 Delete all elements from *source(body(N))* which are also in *restriction(*M*)*;
14 **for** *every vertex M' in source(body(M)) such that M'* $\notin$ *scope(N)* **do**
15 Delete all successors of M' from *source(body(N))* ;
16 add M' to *source(body(N))*;
17 **for** *every SP-vertex M directly governed by* N **do**
18 **for** *every vertex M' in source(scope(M)) such that M'* $\notin$ *scope(N)* **do**
19 Delete all successors of M' from *source(body(N))* ;
20 add M' to *source(body(N))*;

Algorithm 1: Define scope for Q-vertices

Finally, the algorithm produces the following *source(body)* and *source(scope)*:

	source(body)	*source(scope)*
every	$\{$**hate**$\}$	$\{$**hate**, **from**, **alleged**$\}$
a	$\{$**hate**$\}$	$\{$**hate**$\}$
alleged	—	$\{$**from**$\}$

5 The construction of the formula

The main goal of the whole process is to compute formulae which are associated with the original graph. We call *semantic representation of the sentence* the final formulae resulting from these two algorithms.

The second algorithm takes as input the complete description of the scopes of a sem-graph and returns a formula. For each node v of the sem-graph, the algorithm computes its semantics denoted *Sem(*v*)*, depending on the type of its neighbours.

From the lexicon, we associate to each P- or SP-vertex v a λ-term which represents the atomic meaning of the predicate. This λ-term is called *logical form*, denoted *LF* and is used to initialise *Sem(*v*)*. The semantics associated to a C-vertex (a constant) is automatically set to the *LF* provided by the lexicon. For the example of fig. 1, LF(**London**) $=$ *London*.

There are three distinct parts in this algorithm. The first one computes the *primary form*, denoted *PF*, of each vertex: it is the composition of the neighbouring *LFs* by the functional application of the λ-calculus. The second one uses the *primary forms* and the scope decomposition obtained with the first algorithm to compose quantified formulae. The third one builds the final formula by substituting each meta-variable of a scopal predicate with the overall semantics of its scope.

In what follows, we describe more precisely the different parts of this algorithm. Before, we define a denotation: for a P- or SP-vertex N, its successors are denoted $N_1, \cdots, N_p$ and the semantics associated to each of them is defined by $\alpha_i = Sem(N_i)$ or if N_i is the restriction kernel of a Q-vertex M then $\alpha_i = x_M$.

Primary forms: The first part of the algorithm (lines 1 to 11) initialises the semantics of each vertex ; the *Primary Form* of the vertex N is the result of this first state of Sem(N). For each Q-vertex N, Sem(N) is initialised with a fresh variable x_N and its list of dependant quantifiers, ListQ(N), is created as an empty list. The use of ListQ(N) will be explained in more detail later in this section.
For each P-vertex N, Sem(N) is initialised with the functional application of LF(N) to the semantics of each one of its successors. If N is the direct successor of a Q-vertex M, then it is a reified predicate and the variable x_M has to be the first argument of the application.
For each SP-vertex N, Sem(N) is initialised with the functional application of LF(N) to the semantics of each one of its successors, except that the semantics of target(N) is replaced by a fresh meta-variable X_N. This meta-variable will be replaced later in the algorithm by the realization of the whole scope of N, because at this point its exact content has not been computed yet.

Quantifier realization: The second part (lines 12 to 24) computes the realization of quantifiers in the logic formula, using a list of non-explored Q-vertices, denoted NE. One by one, starting with the minimal ones for the govern relation, the Q-vertices are taken out of NE and their realization is computed.
In most cases, the realization of N is computed by the LF function, according to the restriction and body of N. This realization is set as the semantics of every source vertex of body(N). Lines 16 to 20 enable the algorithm to process sem-graphs with particular shared structures, which requires use of ListQ(N).

Substitution of meta-variables in SP-vertices: The last part (lines 25 to 29) computes the realization of scopal predicates in the formula, using a list of non-explored SP-vertices, denoted MV because they still contain a meta-variable. One by one, starting with the ones which depend on no other meta-variable (as is checked in the semantics of the source vertices of the scope), the SP-vertices are taken out of MV. Each realization is computed by replacing X_N with the conjunction of the semantics of the source vertices of its scope.

Finally (line 30), the resulting formula is the conjunction of the semantics of all the source vertices of the sem-graph G which are not in the restriction of any Q-vertex.

Delaying the realisation of quantifiers - ListQ: A sem-graph can contain subgraphs which are common to the scope of some Q-vertices, but the semantics of these subgraphs must appear only once in the final logic formula. In most cases the standard process is sufficient to respect this principle, but for phenomena such as donkey sentences it is not. More precisely it fails when, for a Q-vertex N, *body(N)* originates partly in the restriction and partly in the scope of a Q-vertex M governing N. For the realization of N in the final formula depends on its surrounding quantifier M, listQ(M) contains the default quantifier associated to N, denoted quant(N), and the default quantifiers associated to the other Q-vertices like N. This way, the realization of N is delayed until its context has been entirely computed. The function $\overline{\text{LF}}$ computes the realization of such embedded Q-vertices without their quantifier. This realization without quantifier is established as the semantics associated to all vertices in the common part of

Input: complete description of scopes of G
Output: logical formulae
1 **for** *every Q-vertex N* **do**
2 Sem(N) ← a fresh variable x_N ;
3 ListQ(N) ← []
4 **for** *every P-vertex N* **do**
5 **if** *N is the direct successor of a Q-vertex M* **then**
6 Sem(N) ← LF(N)($x_M, \alpha_1, \cdots, \alpha_p$);
7 **else**
8 Sem(N) ← LF(N)($\alpha_1, \cdots, \alpha_p$);
9 **for** *every SP-vertex N* **do**
10 Choose a fresh meta-variable X_N;
11 Sem(N) ← LF(N)($\alpha_1, \cdots, \alpha_{i-1}, X_N, \cdots, \alpha_p$) where $N_i = target(N)$;
12 NE ← QN(G);
13 **while** *NE* $\neq \emptyset$ **do**
14 Choose N ∈ NE which is minimal for the govern relation on NE;
15 Delete **N** from NE;
16 **if** *source(body(N)) is distributed between the restriction and the body of a Q-vertex M governing N* **then**
17 Add (quant(N),Sem(N)) as the head of ListQ(M);
18 F ← $\overline{LF}(N)$(ListQ(N), restrict(N), body(N) ∩ restrict(M));
19 **for** *every M' ∈ source(body(N)) ∩ restrict(M)* **do**
20 Sem(M') ← F;
21 **else**
22 F ← LF(N)(ListQ(N), restrict(N), body(N));
23 **for** *every M' in source(body(N))* **do**
24 Sem(M') ← F;
25 MV ← SPN(G);
26 **while** *MV* $\neq \emptyset$ **do**
27 Choose M ∈ MV such that X_M is not instantiated AND $\bigwedge_{M' \in source(scope(M)) AND M' \neq M}$ Sem(M') contains no meta-variable X_p;
28 Delete **M** from MV;
29 $X_M \leftarrow \bigwedge_{M' \in source(M)}$ Sem(M');
30 SemFinal ← $\bigwedge_{N \in source(G)}$*outside every restriction* $Sem(N)$

Algorithm 2: Compute the logical formula

source(body(N)) and *restrict(M)*. The quantifier associated to N in the final formula is added by the LF function of M using ListQ(M).

In the example, the treatment of Q-vertices in the algorithm associates to **every**: Sem(**every**) = x and ListQ(**every**)= [] and to **a**: Sem(**a**) = y and ListQ(**a**)= []. Then, P-vertices are processed: Sem(**hate**) = $hate(x, y)$, Sem(**criminal**) = $criminal(x)$ because **criminal** is the direct successor of **every**. In the same way, Sem(**policeman**) = $policeman(y)$, Sem(**from**) = $from(x, London)$ because the first successor of **from** is **criminal** which is the restriction kernel of **every** and thus, the variable associated to

every is used. Lastly, the only SP-vertex gives: Sem(**alleged**) $= alleged(X_{alleged})$.

The next step of the algorithm is to compute the realization of Q-vertices. The minimal element for the govern order is **a** which is not distributed between the *restriction* and *body* of **every**. Then Sem(**a**)$= \exists y.policeman(y) \wedge hate(x, y)$. There is only one element in NE: Sem(**every**) $= \forall x.alleged(X_{alleged}) \Rightarrow \exists y.policeman(y) \wedge hate(x, y)$

Finally, the meta-variable is substituted with the conjunction of the semantics of the sources of its scope, then $X_{alleged} = criminal(x) \wedge from(x, London)$, and the final formula is $\forall x.alleged(criminal(x) \wedge from(x, London)) \Rightarrow \exists y.policeman(y) \wedge hate(x, y)$ which corresponds to the particular reading of the utterance implied by our choices.

6 Related works

The literature about quantifier scoping in semantics is very rich but there are very few works that aim at throwing a bridge between the view of semantics with graphs and the logical view.

The closest work to ours is (Hobbs and Shieber, 1987). From the syntax of an utterance, they build a first term that represents the predicate-argument relations and the restrictions of quantifiers but not their bodies. Then, an algorithm transforms this term step by step, computing the body of each quantifier, the order of computation corresponding to an inclusion order between the scopes of quantifiers. The computation order is not completely free: quantifiers that belong to restrictions of other quantifiers are taken into account only after the others and every bound variable must remain in the scope of its binder. The resulting terms represent logical formulae corresponding to different orders between the quantifier scopes. The interest is that not all permutations of quantifiers are permitted: the algorithm acts as a first filter that rejects some bad orders.

The main originality of our work with respect to (Hobbs and Shieber, 1987), is that we take advantage of the graph representation to define the restriction and the body of quantifiers and to put constraints on the order between scopal elements. First, we recover the same filters on the orders between quantifiers as they do but instead of using a total order, we use a partial order, which better reflects linguistic reality. Second, we are able to express sharing between structures, which is necessary to the representation of coreferences, present for instance in donkey sentences. In (Hobbs and Shieber, 1987), information sharing is expressed in a very limited way with variables shared between predicates.

(Marlet, 2008) proposes another bridge between semantic graphs and logical semantics. More precisely, semantic graphs are those produced by Dependency Grammars (Mel'cuk, 1988) and logical semantics uses the underspecified formulae of MRS (Copestake et al., 2005). Since those semantic graphs make no distinction between the body and the restriction of a quantifier, Marlet must add a function to define the restriction of every quantifier. With a relatively simple syntax, he cannot represent some complex constructions, which are taken into account in our approach such as donkey sentences or imbrication of restrictions like in the sentence **a man that I know a child of has arrived**. Moreover, his interpretation of the order between quantifiers with the equality modulo quantifiers of MRS is not consistent with the corresponding examples because this interpretation forbids a quantifier to be dependent on another quantifier and to be in its restriction at the same time.

7 Prospects and Advances

In this article, we have described a process which, from the semantic graph associated to an utterance, generates a set of formulae corresponding to the different readings of this utterance. The distinguishing features of our proposal are the following: first, the restriction and body of quantifiers are structurally different in our sem-graphs; second, it enables structure sharing and not merely variable sharing; third, it uses partial orders on quantifiers where having total orders is not only unnecessary but also problematic.

These three key features make it possible to treat some widely studied phenomena in a simple manner, but we do not know the exact covering and limits of our approach. The main reason for that is the lack of any evaluation corpus, which cannot be overcome until we define a syntax-semantics interface to generate a semantic graph from any utterance. Exploring the limits of our proposal and defining a syntax-semantics interface is still subject to ongoing research and will be detailed in future publications.

References

C. Barker and C. Shan. 2008. Donkey anaphora is in-scope binding. *Semantics & Pragmatics*, 1:1–42.

H.C. Bunt and R.A. Muskens. 2007. *Computing meaning, Volume 3*. Number 83 in Studies in Linguistics and Philosophy. Springer Dordrecht.

B. Carpenter. 1998. *Type-logical Semantics*. MIT Press, Cambridge, Massachusetts.

A. Copestake, D. Flickinger, C. Pollard, and I.A. Sag. 2005. Minimal Recursion Semantics - an Introduction. *Research on Language and Computation*, 3:281–332.

P. de Groote. 2006. Towards a Montagovian account of dynamics. In *Semantics and Linguistic Theory 16*, Tokyo, Japan.

C. Ebert. 2005. *Formal investigations of underspecified representations*. Ph.D. thesis, King'sCollege.

J.R. Hobbs and S.M. Shieber. 1987. An algorithm for generating quantifier scopings. *Computational Linguistics*, 13:13–1.

R. Marlet. 2008. Un sens logique pour les graphes sémantiques. In *Traitement Automatique des Langues Naturelles (TALN 2008)*, pages 498–507.

I. Mel'cuk. 1988. *Dependency Syntax: Theory and Practice*. Albany, N.Y.: The SUNY Press.