Meaning-Text Theory within Abstract Categorial Grammars: Towards Paraphrase and Lexical Function Modeling for Text Generation

Marie Cousin
Université de Lorraine, CNRS, Inria, LORIA / F-54000 Nancy, France
marie.cousin@loria.fr

Abstract

The meaning-text theory is a linguistic theory aiming to describe the correspondence between the meaning and the surface form of an utterance with a formal device simulating the linguistic activity of a native speaker. We implement a version of a model of this theory with abstract categorial grammars, a grammatical formalism based on λ-calculus. This implementation covers the syntax-semantic interface of the meaning-text theory, i.e., not only the three semantic, deep-syntactic and surface-syntactic representation levels of the theory, but also their interface (i.e., the transformation from one level to another). This implementation hinges upon abstract categorial grammars composition in order to encode level interfaces as transduction operate.

1 Introduction

We present in this article our implementation of a model of the meaning-text theory (MTT, Mel'čuk et al., 2012; Milićević, 2006) with abstract categorial grammars (ACG, de Groote, 2001), focusing on the meaning to text direction, i.e., generation. MTT is a linguistic theory that has already been used in a generation context, while ACGs are a grammatical framework known to encode a range of various grammatical formalisms.

MTT aims to describe the link between the meaning and the textual representation of an utterance. This description is made possible thanks to a formal device, a meaning-text model (MTM), that simulates the linguistic activity of a native speaker. It uses, among others, a dependency syntax and the key concepts of paraphrase and lexical functions (LF) (see Section 2.2). The latter enables a text to be more natural: the syntagmatic LFs for instance encode collocations or support verbs. They play an important role, especially for surface representations, and when producing text. Some formalisations and implementations focusing on text generation already exist, such as GUST (Kahane and Lareau, 2005) or MARQUIS (Wanner et al., 2010).

ACGs are a grammatical framework based on λ-calculus. They enable the implementation of other grammatical formalisms, and have many advantages. ACGs are reversible. We can therefore use them in generation or analysis (Kanazawa, 2007). Their capacities to encode other formalisms, such as tree adjoining grammars (TAG, Pogodalla (2017a)), and to be used in generation were employed to generalize the G-TAG model (Danlos et al., 2014). They are also currently used in an industrial context by Yseop. We aim in this article to use these properties with another linguistic theory that was already proven usefull for generation systems: MTT.

We may now wonder if ACGs are a relevant tool to implement a linguistic theory based on a dependency syntax by assessing it on a text generation task. This article presents the feasibility of such an implementation, based on a restricted number of examples which illustrate several specificities of MTT. As a grammatical framework, ACGs can provide the grammatical formalisms they encode with their generic algorithms, making it unnecessary to develop and implement specific information. They also offer a reversible encoding so that, for instance, we get here both synthesis and analysis for MTT.

Because we wish to have a fined-grained control over the generated text, we choose to focus on text generation with formal methods. The same motivations can be found in Grammatical Framework (Ranta, 2004), that use a type-theoretic system too. The encoding and links to other formalisms in ACGs have been well studied. Indeed, Table 1 below highlights the expressive power of ACGs. A hierarchy of ACGs is used in this table, based on two notions (order and complexity of an ACG) which are defined in Section 3.

ACG	generated language
$\mathrm{ACG}_{(1,n)}$	finite languages
$\mathrm{ACG}_{(2,1)}$	regular languages
$\mathrm{ACG}_{(2,2)}$	context-free grammars (CFG)
$\mathrm{ACG}_{(2,3)}$	well-nested multiple CFG
$\mathrm{ACG}_{(2,4)}$	mildly context-sensitive grammars
$\mathrm{ACG}_{(2,4+n)}$	$\mathrm{ACG}_{(2,4)}$
$\mathrm{ACG}_{(3,n)}$	MELL decidability

Table 1: Expressive power of ACGs (Pogodalla, 2017b).

We aim at encoding a MTM within ACGs, and especially the linguistic structures used by MTT, even thought other formalisms close to the ones used by MTT exist. MTT uses graphs for its semantic representation for instance, to represent predicate-arguments structures, and is therefore close to AMR (Banarescu et al., 2013). Regarding the deep-syntactic representation, MTT uses dependency trees with labels that can differ from other dependency formalisms. We are here interested by how the linguistic structures of MTT relate to each other and the unity of the whole.

2 Meaning-text theory, lexical functions and paraphrase

2.1 MTT

MTT (Mel'čuk et al., 2012; Milićević, 2006) describes the meaning-text correspondance of an utterance. The meaning is "a linguistic content to be communicated" (Milićević, 2006), and is not directly observable, while the text is "any fragment of speech" (Milićević, 2006), immediately perceptible.

MTM consists of 7 representation levels (see Figure 1): the semantic (SemR), deep-syntactic (DSyntR), surface-syntactic (SSyntR), deep-morphologic (DMorphR), surface-morphologic (SMorphR), deep-phonetic (DPhonR) and surface-phonetic (SPhonR) representation levels.

It also has 6 transition modules between these levels. Between each pair of (neighbor) representation levels lay one module, which performs the transition from one of these adjacent representation levels to the other one. They take as input one representation of the former level, perform the transition, and output the obtained representations of the next level. For example, if we give the semantic module (between the semantic level and the deep-syntactic level) the SemR represented Figure 6a as an input, it will output the DSyntR represented Figure 6b.

On top of this transition, they may also perform paraphrase steps, that are transformations inside the same level. It is the case of the deep-syntactic module (between the deep-syntactic and surface-syntactic levels) that performs deep-syntactic paraphrasing and LFs realization on top of DSyntR to SSyntR structure transition.

Thus, depending on the chosen direction, the MTM enables the generation (from SemR to SPhonR) or the analysis (from SPhonR to SemR) of an utterance (see Figure 1).

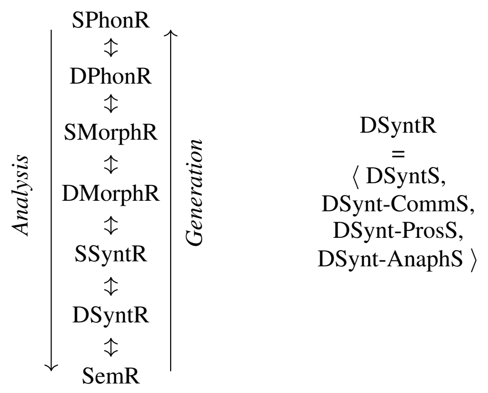


Figure 1: Schema of the MTM and detail of the substructures of DSyntR (Mel'čuk et al., 2012).

Each one of these representations has substructures: one main structure and other structures that add information to the main one.

In this article and our implementation we use mainly DSyntS (the main structure of DSyntR), and not the other three substructures of DSyntR. Therefore, we will not give further detail about them, and only work with DSyntS. The same applies to the substructures of the other representation levels.

- SemS is the main structure of SemR. It is a directed graph whose nodes are semantemes and arcs are labeled with numbers (starting with 1). For each semantic predicate, the numbers on the arcs leading to its arguments indicate their order (see Figure 6a).
- DSyntS is the main structure of DSyntR (see Figure 6b). It is a dependency tree, whose nodes are deep-syntactic lexemes and branches are labelled with deep-syntactic relations. Deep-syntactic relations include actantial relations (labelled from I to VII), attributive relations (labelled ATTR and ATTR_{descr}),

other subordinate relations (labelled APPEND) and coordinative relations (labelled COORD and QUASI-COORD) relations.

- SSyntS is the main structure of SSyntR. It is also a dependency tree, whose nodes are surface-syntactic lexemes, and branches are labelled with surface-syntactic relations (see Figure 6c for an example).

- The main structures of all other levels are represented by strings.

Regarding DSyntR (see Figure 1), its substructures are the deep-syntactic structure (DSyntS), deep-syntactic communicative structure (DSynt-CommS), deep-syntactic prosodic structure (DSynt-ProsS) and deep-syntactic anaphoric structure (DSynt-AnaphS). Mel'čuk et al. (2013) give further detail about the construction rules of such a structure. DSynt-CommS consists of markers of communicative opposition, such as the theme of the DSyntR (see Milićević (2006), page 15 where an example is detailed). DSynt-ProsS consists of a set of markers of prosodies, such as "declarative" or "ironic" for instance. DSynt-AnaphS contains the links of co-referentiality between the node of a DSyntS.

We focus in this article on the deep-syntactic module, more precisely on the LFs realization.

2.2 LF and paraphrase

In the transition modules as well as at some representation levels MTT uses the key concepts of paraphrase and LFs.

LFs (Mel'Čuk and Polguère, 2021) aim at describing linguistic phenomena that exist in all languages. Indeed, they are functions describing relations between lexical units (LU). They associate with that LU the set of all other alternative choices of LUs consistent with the relation they describe. They hinge on semantics, syntax and morphology. That means that they are part of the lexicon of the language, as well as part of its grammar. They are used in MTT in the explanatory combinatorial dictionary (we will not describe that part, see Mel'čuk et al. (2012, 2013) for further detail), and to perform linguistic paraphrase.

LFs are classified in two main categories: paradigmatic and syntagmatic LFs. The former ones express relations of semantic derivation between LUs, while the latter express the combinatorial properties of LUs. For instance, `anti` is a paradigmatic LF associating a LU with its contrary: `anti`(CALM) = {UPSET, RESTLESS} (based on Mel'čuk et al. (2013)) and `causFunc` is a syntagmatic LF which associates a LU with a support verb meaning *make it exist*: `causFunc`(ATTENTION) = DRAW (in the expression "*to draw attention*") (Milićević, 2006). LFs are useful to encode lexical phenomena such as collocations or support verbs. Further detail is given in Mel'Čuk and Polguère (2021).

As for the paraphrase, there are different kinds of paraphrases that can occur at different levels:

(a) at the semantic level with the definition of semantemes by simpler semantemes,

(b) at the deep-syntactic level with the transformation of the dependency tree into another one thanks to lexical paraphrasing rules and restructuring paraphrasing rules (that supports the lexical ones) (Mel'čuk et al., 2013) making some LFs appear (cf. Figures 2 and 3),

(c) between the deep-syntactic and surface-syntactic levels, when choosing how to realize a LF when more than one value of LU is possible (cf. Figures 2 and 3),

(d) at the surface-syntactic level.

Both types (b) and (c) of paraphrase are often considered to be part of the deep-syntactic paraphrase. We want here to make a distinction between what we call deep-syntactic paraphrase (i.e., type (b)) and what we call LF realization (i.e., type (c)), even if both of them occur in the deep-syntactic module.

Iordanskaja et al. (1991) gives further detail on the different paraphrase types. We will here evoke mechanisms to encode the deep-syntactic paraphrase (type (b)) and highlight the one concerning LFs (type (c)). Figure 2 shows an example of paraphrase that uses both types (b) and (c), i.e., deep-syntactic paraphrase and LF realization (it will be further explained in Section 5).

3 Abstract categorial grammars

ACGs (de Groote (2001), whose definitions we use here) are a grammatical formalism based on λ-calculus. An ACG is composed of two languages, linked together by a lexicon. The first langage is called the abstract language and is the set of abstract grammatical structures, such as analysis

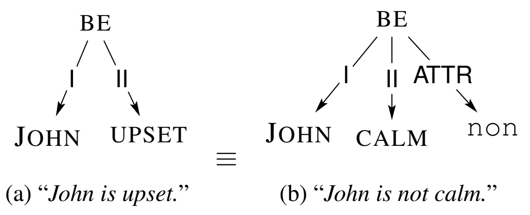


Figure 2: Representation of two deep-syntactic structures representing the paraphrase of "*John is upset*". That paraphrase uses lexical and restructuration deep-syntactic rules as well as the relation `anti`(UPSET) = CALM (based on Mel'čuk et al. (2013)). The dependency tree obtained between deep-syntactic paraphrase and LF realization is given in figure 3a.

trees. The second one, called the object language, is the set of the surface representations generated by the abstract language, such as strings or logical representations in the form of a graph. Each one of these languages is a set of λ-terms obtained by induction over a signature.

Definition 1 *Let A be a set of atomic types. $\mathcal{T}(A)$ is the set of* ***linear implicative types****, obtained inductively over A:*

- *if $a \in A$ then $a \in \mathcal{T}(A)$*
- *if $\alpha, \beta \in \mathcal{T}(A)$ then $(\alpha \to \beta) \in \mathcal{T}(A)$*

Definition 2 *Let Σ be a* ***higher order signature****. Σ is of the form $\Sigma = \langle\, A,\ C,\ \tau \,\rangle$, where:*

- *A is a set of atomic types,*
- *C a set of constants,*
- *$\tau : C \longrightarrow \mathcal{T}(A)$ a function.*

We express with $\vdash_{\Sigma_1} t : s$ that the type of a λ-term t is s in the signature Σ (or $t : s$ if there is no ambiguity).

We express $\Lambda(\Sigma)$ the set of λ-terms obtained using the constants of C, the variables, the abstractions and the applications.

Definition 3 *Let Σ_1 and Σ_2 be two signatures. A* ***lexicon*** *$\mathcal{L}_{12}$ from Σ_1 to Σ_2 is a pair of morphisms $\langle F, G\rangle$ such that $F : \tau(A_1) \longrightarrow \tau(A_2)$ and $G : \Lambda(\Sigma_1) \longrightarrow \Lambda(\Sigma_2)$.*

We write $\mathcal{L}_{12}(t) = \gamma$ to express that γ is the interpretation of t by $\mathcal{L}_{12}$ (or $t := \gamma$ if there is no ambiguity on the used lexicon).

The signatures (like Σ_{dsynt_tree}, see Figure 2) we describe here use almost linear λ-terms. We will not explain this notion, for it is not of great interest for what we say (the used variables being neither discarded nor duplicated in the lexicons). Nevertheless we use the notation λ^o and λ for the linear and non-linear abstractions respectively.

Moreover, an interesting property of ACGs is that when they are second order almost linear, the morphisms inversions (see below) are decidable in a polynomial time (Salvati, 2005), and when they are not almost linear, they remain decidable, even though the complexity is not polynomial anymore (Salvati, 2010). We therefore were careful to use as much as possible second order almost linear ACGs in this implementation.

We may thereby define Σ_{dsynt_tree} in Figure 2 that corresponds to the DSyntR level. Indeed, the constants of this signature enable to build the deep-syntactic trees of DSyntS, like the ones in Figure 3.

- $A_{dsynt\_tree} = \{T, rel, l\}$,
- $C_{dsynt\_tree} = \{c^{dt}_{John}, c^{dt}_{be}, c^{dt}_{restless}, c^{dt}_{upset}, c^{dt}_{calm}, A_1, A_2, ATTR, lex_0, lex_2, lex_3,$ `Anti`, `Non`$\}$,
- τ_{dsynt_tree} is given by Table 2 below.

Constant		Type
c^{dt}_{John}	:	l
c^{dt}_{be}	:	l
$c^{dt}_{restless}$	:	l
c^{dt}_{upset}	:	l
c^{dt}_{calm}	:	l
A_1	:	rel
A_2	:	rel
$ATTR$	:	rel
lex_0	:	$l \to T$
lex_2	:	$l \to rel \to T \to rel \to T \to T$
lex_3	:	$l \to rel \to T \to rel \to T \to rel \to T \to T$
`Anti`	:	$l \to l$
`Non`	:	T

Table 2: τ_{dsynt_tree}

In Table 2, all the constants of the form c^X_L encode LUs, while all constants of the form A_i and lex_i encode the dependency tree structure. The constants `anti` and `non` encode the eponyms LFs.

Due to space considerations, we will not define A and C in the following paragraphs and sections anymore, they can be deduced from the table representing τ. We may now define the notions of ACG, abstract and object languages:

Definition 4 *An* ***abstract categorial grammar*** *is a tuple $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}_{12}, s\rangle$ where:*

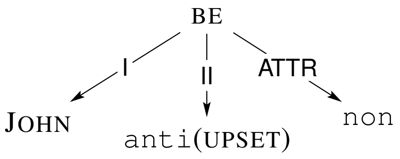
- $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ *and* $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ *are two higher order signatures,*
- $\mathcal{L}_{12} = \Sigma_1 \longrightarrow \Sigma_2$ *is the lexicon,*
- $s \in \mathcal{T}(A_1)$ *is the distinguished type of the grammar.*

Definition 5 *The* ***abstract language*** $\mathcal{A}$ *and the* ***object language*** $\mathcal{O}$ *of an ACG* $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}_{12}, s \rangle$ *are:*

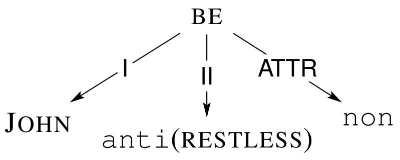
- $\mathcal{A} = \{t \in \Lambda(\Sigma_1) | \vdash_{\Sigma_1} t : s$ *is derivable*$\}$
- $\mathcal{O} = \{t \in \Lambda(\Sigma_2) | \exists u \in \mathcal{A}(\mathcal{G})$ *such that* $t = \mathcal{L}_{12}(u)\}$

In this article we use $\beta\eta$*-equivalence as equality between* λ*-terms.*

Σ_{dsynt_tree} illustrated above gives the needed constants to build the dependency trees of Figure 3.



(a) "*John is not not upset.*", encoded by equation (1) below using Table 2 and corresponding to γ^{dt}_{au} in Figure 4.



(b) "*John is not not restless.*", encoded by equation (2) below using Table 2 and corresponding to γ^{dt}_{ar} in Figure 4.

Figure 3: Representation of two deep-syntactic structures representing the deep-syntactic paraphrases of "*John is not calm*" (based on Mel'čuk et al. (2013)).

But, in order to do so with an ACG, we need another signature as well as a lexicon: we now introduce the abstract signature $\Sigma_{deep\_syntactic}$ (see Table 3), to define $\mathcal{L}_{dsyntRel}$ (see Table 4). $\langle \Sigma_{deep\_syntactic}, \Sigma_{dsynt\_tree}, \mathcal{L}_{dsyntRel}, \mathrm{G} \rangle$ is the ACG that builds the dependency trees of DSyntS in Σ_{dsynt_tree} (see the articulation of these signatures in Figure 5).

We define the notion of order and complexity of an ACG as well. They are used in Table 1 which describes the expressive power of ACGs.

Constant		Type
c^{ds}_{John}	:	G
c^{ds}_{be}	:	$G \to G \to G$
c^{ds}_{upset}	:	G
$c^{ds}_{restless}$	:	G
c^{ds}_{calm}	:	G

Table 3: $\tau_{deep\_syntactic}$

$\Sigma_{deep\_syntactic}$		Σ_{dsynt_tree}
G	:=	T
c^{ds}_{John}	:=	$lex_0\ c^{dt}_{John}$
c^{ds}_{be}	:=	λ^o X Y. $lex_2\ c^{dt}_{be}\ A_1$ X lex_2 Y
c^{ds}_{upset}	:=	$lex_0\ c^{dt}_{upset}$
$c^{ds}_{restless}$	:=	$lex_0\ c^{dt}_{restless}$
c^{ds}_{calm}	:=	$lex_0\ c^{dt}_{calm}$

Table 4: $\mathcal{L}_{dsyntRel}$

Definition 6 (Pogodalla, 2017b) *The* ***order*** *of an ACG is the maximum of the order of its abstract constants. The order of an abstract constant is the order of its type* τ*. The order of a type* $\tau \in \mathcal{T}(A)$ *is inductively defined:*

- $order(\tau) = 1$ *if* $\tau \in A$,
- $order(\alpha \to \beta)$
  $= max(1 + order(\alpha), order(\beta))$ *else.*

The ***complexity*** *of an ACG is the maximum of the orders of its atomic types realizations. An ACG of order* γ *and of complexity* η *is written* $ACG_{(\gamma,\eta)}$.

In the following paragraphs and sections, we use the notation c^X_L for the constant of Σ_X encoding the LU L. If there are two different constants representing the same LU L in Σ_X, we write c^X_{L1} and c^X_{L2} to distinguish them. We also use γ^X_i for the complex λ-term of Σ_X indexed by i. These complex λ-terms being the encoding of possible representations for an expression, the index i indicate this expression. Therefore, we will use au for "*John is not not upset*" (cf. Figure 3a), ar for "*John is not not restless*" (cf. Figure 3b), and $c1$, $c2$ for "*John is not calm*".

We use dt and $d0f$ instead of $dsynt\_tree$ and $dsynt\_0\_fl$.

That being said, we define the complex λ-terms:

$$\gamma^{dt}_{au} = lex_3\ c^{dt}_{be}\ A_1\ (lex_0\ c^{dt}_{John})\ A_2\ (lex_0(\mathtt{anti}\ c^{dt}_{upset}))\ ATTR\ c^{dt}_{non} \quad (1)$$

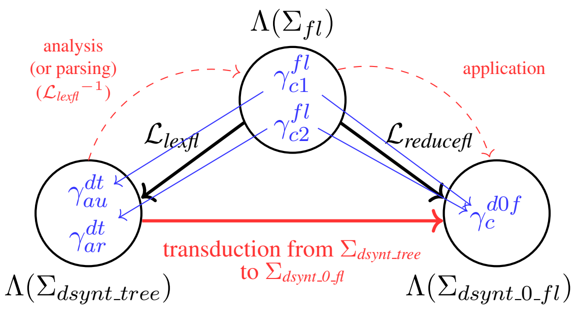
$$\gamma^{dt}_{ar} = lex_3\ c^{dt}_{be}\ A_1\ (lex_0\ c^{dt}_{John})\ A_2\ (lex_0(\mathtt{anti}\ c^{dt}_{restless}))\ ATTR\ c^{dt}_{non} \quad (2)$$

$$\gamma^{fl}_{c1} = lex_3\ c^{fl}_{be}\ A_1\ (lex_0\ c^{fl}_{John})\ A_2\ (lex_0\ c^{fl}_{calm1})\ ATTR\ c^{fl}_{non} \quad (3)$$

$$\gamma^{fl}_{c2} = lex_3\ c^{fl}_{be}\ A_1\ (lex_0\ c^{fl}_{John})$$
$$A_2\ (lex_0\ c^{fl}_{calm2})\ ATTR\ c^{fl}_{non} \quad (4)$$
$$\gamma^{d0f}_{c} = lex_3\ c^{d0f}_{be}\ A_1\ (lex_0\ c^{d0f}_{John})$$
$$A_2\ (lex_0\ c^{d0f}_{calm})\ ATTR\ c^{d0f}_{non} \quad (5)$$

γ^{dt}_{au} encodes the upper tree of Figure 3 while γ^{dt}_{ar} encodes the lower tree of Figure 3.

Another advantageous property of ACGs is their ability to be composed. A specific case of composition is transduction: given two ACGs sharing the same abstract signature, transduction (see Figure 4) is the composition of the analysis (or the inversion) of a morphism (like $\mathcal{L}_{lexfl}{}^{-1}$) and the application of a morphism (as $\mathcal{L}_{reducefl}$) using both ACGs. Consequently, transduction is very useful since it enables two terms of two object signatures to be in relation with each other.



$\Lambda(\Sigma_{fl})$
analysis (or parsing) ($\mathcal{L}_{lexfl}{}^{-1}$)
application
γ^{fl}_{c1}
γ^{fl}_{c2}
$\mathcal{L}_{lexfl}$
$\mathcal{L}_{reducefl}$
γ^{dt}_{au}
γ^{dt}_{ar}
γ^{d0f}_{c}
transduction from Σ_{dsynt_tree} to $\Sigma_{dsynt\_0\_fl}$
$\Lambda(\Sigma_{dsynt\_tree})$
$\Lambda(\Sigma_{dsynt\_0\_fl})$

Figure 4: Transduction from DSyntR to a deep-syntactic representation where the LFs would be realized. $\mathcal{L}_{reducefl}$ is such that $\mathcal{L}_{reducefl}(\gamma^{fl}_{c1}) = \mathcal{L}_{reducefl}(\gamma^{fl}_{c2})$.

It is indeed a method to make possible the transition between the representation levels (represented here by signatures) and perform the transformation of the structures: given an initial λ-term, transduction gives a second λ-term, without modifying the first one. MTT being like a suite of structure transformations (see Figure 1), transduction seems well adapted to implement a MTM. This suite of structure transformations also appears in the overview of the ACG architecture of our implementation in Figure 5 presented in Section 4, especially between the areas 1, 2, 4, and 5 of Figure 5.

Moreover, we can produce a lot of structures inside a signature. But, they do not all have an antecedent in the abstract signature. Indeed, when parsing a structure of an object signature of an ACG, if it has an antecedent, it will be found (for parsing is decidable, see above). If it does not have one, then nothing happens. That means that, when applying transduction between two object signatures (or two representation levels, in the case of our implementation), if one structure should not have a correspondance in the next object signature, it will not have one: no new structure will be produced.

4 Overview of our implementation

In order to represent the MTM, at least from SemR to SSyntR, we implemented signatures and lexicons (see Figure 5) and experimented our encoding with ACGtk (Pogodalla, 2016), a piece of software allowing for defining grammars and using the associated parsing and interpretation operation. Nevertheless, for simplification purpose we did not implement the other substructures than the main one for each representation level, like the communicatives structures (see Section 2).

This implementation uses transduction (see Figure 4) which is the heart of our implementation, for it is used to perform all transformations (see Figure 5, where we can guess the use of transduction).

We can see that the third area (see Figure 5) looks like a detour. That is due to the fact that, on one hand, MTT does not modify a structure when going from a representation level to another (which transduction also does), and, on the other hand, during some steps of the generation process, we want to keep the former structures as well as the newly obtained ones. That is the case of the deep-syntactic paraphrasing, for we want to keep all possible dependency trees that would lead to a sentence, so all the possible paraphrases. In other words, MTT makes other structures appear inside of one representation level, and we want them all to reach the next representation level, not only the last one. For this is not the goal of transduction, deep-syntactic paraphrasing is hard to perform in the current state of our implementation, and is performed by this third area, although it is problematic, as we will explain later.

Our implementation encodes the different levels of representation of a MTM (see Figure 5): Σ_{sem} corresponds to SemR, Σ_{dsynt_tree} to DSyntR, and Σ_{ssynt_tree} to SSyntR. As we can see, we use the transduction between:

- $\Sigma_{semantic}$ and Σ_{dsynt_tree}: to perform the semantic paraphrasing and to make the transition from SemR to DSyntR (areas 1 and 2),
- Σ_{dsynt_tree} and Σ_{dsynt_rule}: to perform the deep-syntactic paraphrasing (area 3),

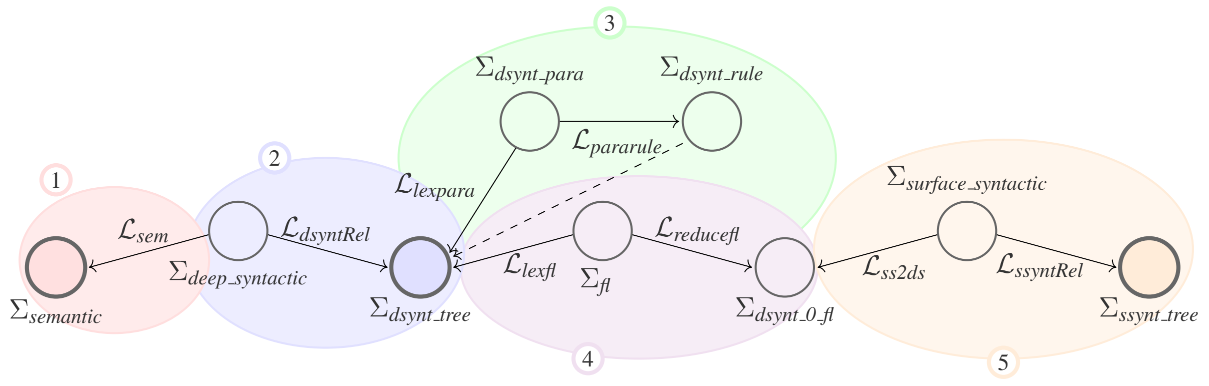


Figure 5: Overview of the ACG architecture. Area 1 corresponds to the semantic paraphrasing, area 2 to the transition between SemR and DSyntR, area 3 to the deep-syntactic paraphrasing, area 4 to the LFs realization step, and area 5 to the transition to SSyntR.

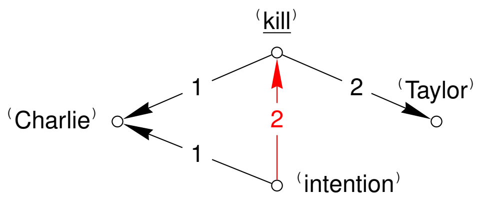
- Σ_{dsynt_tree} and $\Sigma_{dsynt\_0\_fl}$: to realize LFs (area 4, this part will be detailed in Section 5),

- $\Sigma_{dsynt\_0\_fl}$ and Σ_{ssynt_tree}: to make the transition from a deep-syntactic representation without LFs anymore to SSyntR.

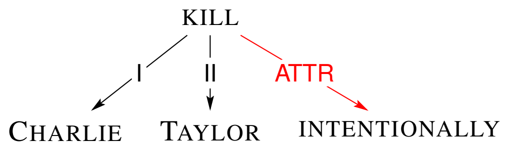
It was tested on a sample of example sentences. This sample is short, but covers many lexical phenomena, like collocations, the use and realization of LFs, semantic or syntactic equivalences, as well as obligatory arguments optionally expressible. As stated at the end of Section 3, inside a representation level (or a signature), the structures that should not have a correspondance in the next representation level (because they are incorrect for example) do not have one: they will not find an antecedent by reversing the lexicon during transduction (Table 5 highlights this). If a structure should not lead to another one regarding MTT formalism, then our implementation of ACGs is such that, by transduction, no antecedent will be found.

The code and the examples are available at https://inria.hal.science/hal-04104453.

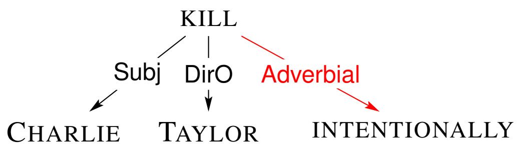
On top of that, this implementation also deals with adverbial groups, that have a specific treatment inspired by the work on TAG of Pogodalla (2017a). Their treatment is indeed not the same depending on the signature we look at. In SemR, i.e., in $\Lambda(\Sigma_{semantic})$, the arc between the adverbial group and the verb it modifies is directed toward the verb, while in DSyntR, so $\Lambda(\Sigma_{dsynt\_tree})$, it is directed toward the adverbial group. This is because adverbial groups are modifiers (Mel'čuk et al., 2013, 2015). We used the same approach as Candito and Kahane (1998) for the dependency inversions between derived trees and derivation trees in TAG (see Figure 6). The manipulation of obligatory arguments of a SemR that are optionally expressible is also possible, and was inspired by Blom et al. (2011). It is done thanks to some constants in $\Sigma_{deep\_syntactic}$. These last two points will not be detailed here, but you can find further detail in (Cousin, 2022). Nevertheless, the Section 5 will explain the last step of the paraphrase illustrated on Figure 2, that is the realization of LFs. It is a good example to show the different kinds of possible equivalence inside an ACG, as well as how LFs are used in this modeling.



(a) SemS for "*Charlie kills Taylor intentionally.*"



(b) DSyntS for "*Charlie kills Taylor intentionally.*"



(c) SSyntS for "*Charlie kills Taylor intentionally.*"

Figure 6: Dependency inversion for the adverb "*intentionally*" in "*Charlie kills Taylor intentionally*".

Expression linked to the semantic graph	"*Charlie kills Taylor intentionally*"	"*John is calm*"
SemR ($\Sigma_{semantic}$)	1	1
DSyntR before deep-syntactic paraphrasing ($\Sigma_{dsynt\text{-}tree}$)	2	1
of which will be accepted at the next stage	2	1
DSyntR after deep-syntactic paraphrasing ($\Sigma_{dsynt\text{-}tree}$)	2	6[1]
of which will be accepted at the next stage	2	3
DSyntR after FLs realization ($\Sigma_{dsynt\text{-}0\text{-}fl}$)	2	5
of which will be accepted at the next stage	2	3
SSyntR ($\Sigma_{ssynt\text{-}tree}$)	2	3

Table 5: Number of obtained structures by generation step for two initial semantic graphs, corresponding to the expressions "*Charlie kills Taylor intentionally*" and "*John is calm*".

5 Example

This section gives a detailed example on how the transduction works and how we realize LFs in our implementation. We consider here the signatures Σ_{dsynt_tree} (see Table 2), Σ_{fl} and $\Sigma_{dsynt\_0\_fl}$ only, as well as the lexicons $\mathcal{L}_{lexfl}$ and $\mathcal{L}_{reducefl}$. We saw in Figure 2 a paraphrase example using deep-syntactic paraphrase as well as LF realization. We will explain the LFs realization in this section.

We consider the following sentences:

(6) a. "*John is upset*"

b. "*John is not calm*"

c. "*John is restless*"

We consider the example of the paraphrase between expressions (6a) and (6b). We may remember that anti(CALM)={UPSET, RESTLESS}, so we also have anti(UPSET) = CALM = anti(RESTLESS) (among other values, but we are interested in CALM here). After the deep-syntactic paraphrasing of sentence (6a), before realizing LFs, we have a dependency tree such as Figure 3a. But, (6a) is a paraphrase of (6b), and so is (6c) (illustrated in Figure3b). They both have the same paraphrase (6b), so these two sentences (6a) and (6c) are paraphrases of each other themselves (depending on the context, but that point will be explained in the conclusion).

Therefore, we want for our implementation to allow this link, this equivalence between the expressions (6c) and (6a). Thus, we want to obtain an equivalence such as (7):

$$\gamma^{dt}_{au} \equiv \gamma^{dt}_{ar} \qquad (7)$$

Indeed, we may remember that γ^{dt}_{au} represents the DSyntS of expression (6a), and γ^{dt}_{ar} the DSyntS of expression (6c). Because both expressions are paraphrases of (6b) and because two different images of a morphism cannot have the same antecedent, we will have to use transduction here. The equivalence between (6a) (or (6c)) and (6b) will take two steps, i.e., parsing and application. We want, if we write $\mathcal{T}$ for the transduction relation, our implementation to allow equations such as (8):

$$\mathcal{T}(\gamma^{dt}_{au}, \gamma^{d0f}_{c}) \text{ and } \mathcal{T}(\gamma^{dt}_{ar}, \gamma^{d0f}_{c}) \qquad (8)$$

Hence, we want to use two ACGs sharing the same abstract signature, to have the following equations (9), (10) and (11) (see Figure 4 that illustrates it) in order to have equations (8) and (7):

$$\mathcal{L}_{reducefl}(\gamma^{fl}_{c1}) = \gamma^{d0f}_{c} = \mathcal{L}_{reducefl}(\gamma^{fl}_{c2}) \qquad (9)$$

$$\mathcal{L}_{lexfl}(\gamma^{fl}_{c1}) = \gamma^{dt}_{au} \qquad (10)$$

$$\mathcal{L}_{lexfl}(\gamma^{fl}_{c2}) = \gamma^{dt}_{ar} \qquad (11)$$

Tables 2, 6a, 6b and 6c define the constants of the signatures and lexicon we use in this section in order to do so. They are simplified and show only relevant information for this example. The constants such as lex_i and A_i (see Section 3, Table 2) are not specified anymore, for they do not change from one signature to another.

We implement the realization of LFs with the transduction and the properties of λ-calculus, like β-reduction. Indeed, one expression may have different representations in Σ_{dsynt_tree} (see Table 2), but only one in $\Sigma_{dsynt\_0\_fl}$ (see Table 6a). To realize LFs, we use different levels of equivalencies, that this example highlights.

The different levels of equivalencies are the following (see Figure 4):

- inside of a signature, and by application or parsing of a lexicon, two representations may

[1]In fact, more structures are obtained, but they are incorrect by construction. They will therefore not be considered here (and do not have an antecedent in $\Sigma_{surface\text{-}syntactic}$).

Constant		Type	Constant		Type
c^{fl}_{John}	:	l	c^{d0f}_{John}	:	l
c^{fl}_{be}	:	l	c^{d0f}_{be}	:	l
c^{fl}_{calm0}	:	l	c^{d0f}_{calm}	:	l
c^{fl}_{calm1}	:	l	Non	:	t
c^{fl}_{calm2}	:	l			
Non	:	t			

(a) τ_{fl} (left) and $\tau_{dsynt\_0\_fl}$ (right)

Σ_{fl}		Σ_{dsynt_tree}
$t := T, r := rel, l := l$		
c^{fl}_{John}	:=	c^{dt}_{John}
c^{fl}_{be}	:=	c^{dt}_{be}
c^{fl}_{calm0}	:=	c^{dt}_{calm}
c^{fl}_{calm1}	:=	$\texttt{anti}(c^{dt}_{upset})$
c^{fl}_{calm2}	:=	$\texttt{anti}(c^{dt}_{restless})$
Non	:=	Non

(b) $\mathcal{L}_{lexfl}$

Σ_{fl}		$\Sigma_{dsynt\_0\_fl}$
$t := t, r := r, l := l$		
c^{fl}_{John}	:=	c^{d0f}_{John}
c^{fl}_{be}	:=	c^{d0f}_{be}
c^{fl}_{calm0}	:=	c^{d0f}_{calm}
c^{fl}_{calm1}	:=	c^{d0f}_{calm}
c^{fl}_{calm2}	:=	c^{d0f}_{calm}
Non	:=	Non

(c) $\mathcal{L}_{reducefl}$

Table 6: τ_{fl}, $\tau_{dsynt\_0\_fl}$, $\mathcal{L}_{lexfl}$ and $\mathcal{L}_{reducefl}$.

be equal thanks to β-reduction. Nevertheless, this example does not show it. This β-equivalence inside of a signature is used in $\Sigma_{semantic}$ but not illustrated in this article due to space restrictions.

- by parsing a lexicon, for instance $\mathcal{L}_{reducefl}$: the sentence "*John is not calm*" has one representation in $\Sigma_{dsynt\_0\_fl}$, while it has two different ones in Σ_{fl}. Thus, we obtain thanks to the parsing the equality (9).

- by transduction: the two dependency trees γ^{dt}_{au} and γ^{dt}_{ar} in Σ_{dsynt_tree} are equivalent. Indeed, when parsing and applying the lexicons $\mathcal{L}_{reducefl}$ and $\mathcal{L}_{lexfl}$, we obtain (as wanted) the equations (9), (10) and (11), then (8) and finally (7) by transduction. Equations (10) and (11) show that `anti`(UPSET) and `anti`(RESTLESS) will be realized as CALM. Because one antecedent cannot have two different images, we need the second object signature $\Sigma_{dsynt\_0\_fl}$ in order to have one unique constant per lexeme (here CALM). Hence we have in $\Sigma_{dsynt\_0\_fl}$ deep-syntactic dependency trees where LFs are realized.

Thus transduction between signatures Σ_{dsynt_tree} and $\Sigma_{dsynt\_0\_fl}$ allows to realize LFs, and to perform the third type of paraphrase (see Section 2).

6 Conclusion and future work

We have shown a possible implementation of a MTM with ACGs. This implementation models the SemR to SSyntR levels of MTM. Even though this implementation uses only the main structures of the representation levels of MTT and not the other substructures (like the communicatives ones), when tested over a sample of example sentences, their SSyntS are correctly obtained (see Table 5). Indeed, for a given representation level, in the direction of the generation, the incorrect structures are not produced, for they do not have an antecedent by parsing the lexicon to the next abstract signature. Moreover, if we take the direction of analysis, we obtain the wanted semantic graphs.

The implemented model enables the semantic paraphrase to take place, as well as the transitions between the representation levels thanks to transduction, the realization of LFs thanks to transduction too, the handling of obligatory semantic arguments optionally expressible, and the handling of adverbial groups.

However, this implementation has some limitations. Indeed, the deep-syntactic paraphrasing is not optimal. It is actually made possible by the detour of area 3 in Figure 5, but we need to manually iterate the process. We need to save the previous structures, perform the paraphrasing loop, and apply again the process for the newly generated structures until no new correct structure is obtained. Transduction is showing some limitations here: this mechanism is not well suited for the deep-syntactic paraphrasing because rewritten structures still need to be processed as well as resulting structures.

Furthermore, we have not exploited all the possible types of paraphrasing (Iordanskaja et al., 1991) in our implementation yet: we also want to continue in this direction to implement all of them. Moreover, we want to include, for each level, other substructures, such as the communicative structures, to have more information about the theme, the rheme, and the speaker intentions, in order to have an implementation nearer to MTT than what it currently is.

Acknowledgments

I would like to thank Sylvain Pogodalla and the reviewers for their comments and constructive remarks, which helped to improve this article. I would also like to thank my two thesis supervisors, Philippe de Groote and Sylvain Pogodalla, for their supervision and our exchanges which enabled me to write this article.

References

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria. Association for Computational Linguistics.

Chris Blom, Philippe de Groote, yoad Winter, and Joost Zwarts. 2011. Implicit Arguments: Event Modification or Option Type Categories? In *18th Amsterdam Colloquium on Logic, Language and Meaning*, volume 7218 of *Lecture Notes in Computer Science*, pages 240–250, Amsterdam, Netherlands. Springer.

Marie-Hélène Candito and Sylvain Kahane. 1998. Can the TAG derivation tree represent a semantic graph? an answer in the light of meaning-text theory. In *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*, pages 21–24, University of Pennsylvania. Institute for Research in Cognitive Science.

Marie Cousin. 2022. Génération de texte avec les grammaires catégorielles abstraites et la théorie sens-texte. Master's thesis, Grenoble INP Ensimag, September.

Laurence Danlos, Aleksandre Maskharashvili, and Sylvain Pogodalla. 2014. Text generation: Reexamining G-TAG with abstract categorial grammars (génération de textes : G-TAG revisité avec les grammaires catégorielles abstraites) [in French]. In *Proceedings of TALN 2014 (Volume 1: Long Papers)*, pages 161–172, Marseille, France. Association pour le Traitement Automatique des Langues.

Philippe de Groote. 2001. Towards abstract categorial grammars. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 252–259, Toulouse, France. Association for Computational Linguistics.

Lidija Iordanskaja, Richard Kittredge, and Alain Polguère. 1991. *Lexical Selection and Paraphrase in a Meaning-Text Generation Model*, pages 293–312. Springer US, Boston, MA.

Sylvain Kahane and François Lareau. 2005. Grammaire d'Unification Sens-Texte : modularité et polarisation. pages 23–32. Grammaire d'Unification Sens-Texte : modularité et polarisation. Dourdan.

Makoto Kanazawa. 2007. Parsing and generation as datalog queries. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 176–183, Prague, Czech Republic. Association for Computational Linguistics.

Igor Mel'Čuk and Alain Polguère. 2021. Les fonctions lexicales dernier cri. In Sébastien Marengo, editor, *La Théorie Sens-Texte. Concepts-clés et applications*, Dixit Grammatica, pages 75–155. L'Harmattan.

I.A. Mel'čuk, I.A. Mel'čuk, D. Beck, and A. Polguère. 2012. *Semantics: From Meaning to Text*, volume 1 of *Semantics: From Meaning to Text*. John Benjamins Publishing Company.

I.A. Mel'čuk, I.A. Mel'čuk, D. Beck, and A. Polguère. 2013. *Semantics: From Meaning to Text*, volume 2 of *Semantics: From Meaning to Text*. John Benjamins Publishing Company.

I.A. Mel'čuk, I.A. Mel'čuk, D. Beck, and A. Polguère. 2015. *Semantics: From Meaning to Text*, volume 3 of *Semantics: From Meaning to Text*. John Benjamins Publishing Company.

Jasmina Milićević. 2006. A short guide to the meaning-text linguistic theory. *Journal of Koralex*, 8:187–233.

Sylvain Pogodalla. 2016. ACGtk : un outil de développement et de test pour les grammaires catégorielles abstraites (ACG TK : a toolkit to develop and test abstract categorial grammars). In *Actes de la conférence conjointe JEP-TALN-RECITAL 2016. volume 5 : Démonstrations*, pages 1–2, Paris, France. AFCP - ATALA.

Sylvain Pogodalla. 2017a. A syntax-semantics interface for Tree-Adjoining Grammars through Abstract Categorial Grammars. *Journal of Language Modelling*, 5(3):527–605.

Sylvain Pogodalla. 2017b. Abstract Categorial Grammars as a Model of the Syntax-Semantics Interface for TAG. In *FSMNLP 2017 and TAG+13 conference*, Umeå, Sweden.

Aarne Ranta. 2004. Grammatical framework. *J. Funct. Program.*, 14:145–189.

Sylvain Salvati. 2005. *Problèmes de filtrage et problème d'analyse pour les grammaires catégorielles abstraites*. Ph.D. thesis, Institut National Polytechnique de Lorraine. Thèse de doctorat dirigée par Philippe de Groote.

Sylvain Salvati. 2010. On the membership problem for non-linear abstract categorial grammars. *Journal of Logic, Language and Information*, 19(2):163–183.

Leo Wanner, Bernd Bohnet, Nadjet Bouayad-Agha, François Lareau, and Daniel Nicklaß. 2010. Marquis: Generation of user-tailored multilingual air quality bulletins. *Applied Artificial Intelligence*, 24(10):914–952.