Dependency Structures Representation: Meaning Text Theory's Deep-Syntax Encoding With Abstract Categorial Grammars

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

Abstract

We present an encoding of the meaning-text theory into abstract categorial grammars, a grammatical formalism based on λ-calculus. It adresses some shortcomings of a previous encoding (Cousin, 2025). This encoding shows some limitations, especially the articulation of dependencies within structures and modifiers behavior (predicative role of adjectives and adverbs at semantic and syntactic levels, number of modifiers, etc.). Reusing Cousin's (2025) grammars composition, we present a representation of syntactic dependency structures based on de Groote (2023) that overcomes these limitations, with a focus on deep-syntax.

1 Introduction

The work presented in this article takes place in a context of encoding the meaning-text theory (MTT) into abstract categorial grammars (ACG). It extends previous work presented in (Cousin, 2023b,a, 2025). While using the same ACG architecture, we provide here a new encoding of dependency structures based on (de Groote, 2022, 2023) that concerns MTT's deep-syntactic and surface-syntactic representations. This work falls within a text generation context with formal methods. Similar motivations are to be found in Grammatical Framework (Ranta, 2004). This work's aims are to have strong control over generated text in order to be sure that it indeed conveys the message to be expressed, and to implement MTT's linguistic structures through which this control operates. In order to do so, we rely on a formal model, ACGs.

1.1 MTT and ACGs

MTT (Mel'čuk et al., 2012, 2013, 2015, see section 2) is a linguistic theory describing the link between the meaning and the text of an utterance. MTT can be used for generation as well as for analysis purposes. It is composed of seven representation levels (including semantic, deep-syntactic, and surface-syntactic levels) and six transition modules between these levels. Each representation is composed of at least both predicative and communicative structures. MTT heavily relies on the key concepts of lexical functions (LF) and paraphrase. Systems presented in (Wanner et al., 2010) and (Lareau et al., 2018) are two examples of MTT applied to text generation. These systems use graph transducers to perform generation, while we use ACGs which have the advantage of being reversible.

ACGs (de Groote, 2001, see section 3) are a grammatical framework based on λ-calculus, enabling the representation of other grammatical formalisms. For instance, Pogodalla (2017) shows an implementation of TAGs into ACGs. ACGs can also be used in generation and analysis (Kanazawa, 2007), since they are reversible. Their reversibility is a property we take avantage of in this work.

In this article, we focus on an encoding of MTT into ACGs.

1.2 Related Work

1.2.1 MTT Into ACG Encoding

(Cousin, 2023b,a) shows such an encoding of MTT into ACGs, that only takes predicative structures into account and completely ignores communicative structures. This encoding enables several linguistic phenomena's handling, such as collocations or paraphrase generation. Such phenomena are enabled thanks to the encoding of LFs among others. This article reworks Cousin's (2023a) ACG architecture and composition.

(Cousin, 2025) reuses this implementation to take the communicative structure into account. Theme-rheme opposition (Mel'cuk, 2001; Polguère, 1990), included in communicative structures, plays a crucial role in the determination of the deep-syntactic tree corresponding to the semantic graph of an utterance. Indeed, depending on the communicative structure, for two semantic representations

sharing the same predicative structure, the obtained expression differ. It is the theme-rheme opposition, that, for instance, makes a copula appear (or not) in deep-syntax when an adjective modify a noun, and therefore decides if the adjective will be expressed as an attributive ("*[the **purple** dragon]*$_{Rheme}$") or as a subject complement ("*[the dragon]*$_{Theme}$ *[is **purple**]*$_{Rheme}$"). Two expressions obtained from the same semantic predicative structure are different if they have different communicative structures. In this case, the two expressions are usually not paraphrases.

Since the communicative structure plays an important role in MTT's generation process, we keep Cousin's (2025) addition of this structure. However, (Cousin, 2023b,a, 2025) show some limitations, like the inability to use multiple modifiers on one same predicate for instance. These limitations are detailed below.

1.2.2 ACGs and Dependency Strutures

de Groote (2022, 2023) presents an ACG encoding of a semantic-syntax interface to derive formal logical semantic representations *à la* Montague. The syntactic part of de Groote's interface uses dependency structures. We use here a similar approach to add non-mandatory dependencies to syntactic structures, such as adverbial and adjectival ones.

However, several aspects show differences:

- de Groote (2023) uses higher-order ACGs: their parsing complexity is not polynomial, and possibly only semi-decidable (de Groote, 2015). We want to use ACGs from a specific class, ACGs of order 2, to have polynomial complexity;
- de Groote's (2023) encoding is made toward analysis (syntax to semantics), we are working toward generation (semantic to syntax);
- de Groote (2023) defines the coherence principle: regardless of the computation order of dependents (given one common governor), computed structures are logically equivalent and yield the same semantic interpretation. The coherence principle is not respected here. Indeed, respecting de Groote's coherence principle would mean computationally intractable or even undecidable ACGs. Since we focus on the opposite direction to him (i.e., generation), and heavily rely on ACG parsing, it's something we choose to avoid.

1.3 Contributions

This article present a new version of MTT into ACG encoding, with a focus on deep-syntax realization. It reuses original aspects of MTT, such as paraphrase and lexical functions (that enable the representation and handling of idiomatic expressions for instance). We are interested in the way linguistic structures, especially MTT's ones, can be expressed in ACGs. The ACG encoding shows some advantages. First, it can be used in generation as well as analysis, since ACGs are reversible (see section 3). Second, some similarities with other formalisms can be found, which enable a comparison to them, and to reuse some aspects of these formalisms in our encoding. It is for example the case of modification presented in section 5 and inspired by Pogodalla's (2017) TAG into ACG encoding.

Our contributions are the following:

- syntactic dependency structures allowing the use of multiple modifiers (for both deep-syntax and surface-syntax; this article focuses on deep-syntax), less general than de Groote's (2023) ones, but which parsing is polynomial;
- decidable ACGs (as opposed to Cousin (2025) who had higher order ACGs);
- an encoding that naturally falls within Cousin's (2023a) ACG architecture, with a better encoding of predicative structures having multiple dependents expressed by modifiers (adjectives and adverbs).

Sections 2 and 3 present MTT and ACGs. Sections 4 and 5 detail the used ACG architecture and the deep-syntax encoding. Section 6 presents obtained results before the conclusion in section 7.

2 Meaning-Text Theory (MTT)

MTT (Mel'čuk et al., 2012, 2013, 2015) is a linguistic theory aiming at representing the link between the meaning and the textual representation of an utterance. The *meaning* is "a linguistic content to be communicated" (Milićević, 2006) and the *text* is "any fragment of speech" (Milićević, 2006). In order to do so, MTT uses a meaning-text model (MTM, see. figure 1) composed of seven representation levels and six transition modules. MTT uses key concepts of lexical functions (LFs, Mel'Čuk and Polguère, 2021) and paraphrase (Iordanskaja et al., 1991; Milićević, 2007). These concepts are not detailed here.

2.1 MTT Architecture

A MTM (see. figure 1) is composed of seven representation levels, each one corresponding to an eponym utterance representation, namely the semantic (SemR), deep-syntactic (DSyntR), surface-syntactic (SSyntR), deep-morphologic and surface-morphologic (resp. DMorphR and SMorphR), deep-phonologic and surface-phonologic (resp. DPhonR and SPhonR) representations.

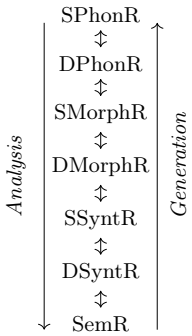


Figure 1: Schema of a MTM (Mel'čuk et al., 2012)

Between each pair of adjacent levels operates a transition module, named after the level (of the pair) nearest to the semantic level. For instance, the semantic transition module operates the transition between SemR and DSyntR.

Each representation level is made of several substructures, among which a predicative structure (graphs for SemR, trees for DSyntR and SSyntR, strings for the four remaining ones) and a communicative structure. The latter bears (among other communicative oppositions) the theme-rheme opposition (Polguère, 1990), that decorate and complement the predicative structure (see figure 3).

A transition module carry out all necessary operations between two levels to transform the previous representation into the next one. It handles structure changes, such as the semantic module that transforms semantic graphs into deep-syntactic trees, but can also perform other operations such as paraphrase steps (see section 2.3).

2.2 DSyntR

DSyntR is made of four substructures (see figure 2), namely: the deep-syntactic structure (DSyntS), the deep-syntactic communicative structure (DSynt-CommS), the deep-syntactic prosodic structure (DSynt-ProsS) and the deep-syntactic anaphoric structure (DSynt-AnaphS) (Mel'čuk et al., 2013). In this article, we only consider the first two ones, DSyntS and DSynt-CommS.

DSyntR =
⟨ DSyntS, DSynt-CommS, DSynt-ProsS, DSynt-AnaphS ⟩

Figure 2: Composition of a DSyntR

DSyntS are dependency tree structures, which nodes are deep-syntactic lexemes and edges are labeled with deep-syntactic relations (see figure 3). Several examples of DSyntS are given in section 5. Deep-syntactic relations are mainly actantial (*actant* **I**, *actant* **II**, etc.), attributive (**ATTR**), or coordinative (**COORD**) relations. Its lexemes are deep lexemes (or "rough" lexemes): collocations and idiomatic expressions are represented by one single lexeme (and not by as many lexemes as there are words composing the expression, like in SSyntS), and some lexical categories, like prepositions and determinants, are not represented (prepositions appear in SSyntS and determinants in the morphologic levels).

DSynt-CommS is represented by markers that decorate the DSyntS with communicative oppositions. For instance, theme and rheme markers are represented as boxes that encapsulates subtrees (see figure 3). The subtree in the theme-box (respectively rheme-box) is labeled as theme (resp. rheme) and therefore represents the theme (resp. rheme).

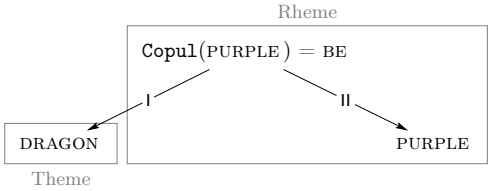


Figure 3: DSyntS and DSynt-CommS representing "*The dragon is purple.*"

2.3 Semantic Transition Module

The semantic module performs the transition between SemR and DSyntR. SemS are graphs which nodes are semantemes and which edges are semantic relations (labelled *1*, *2*, etc. depending on the semantic actant (first, second, etc.) the edge is pointing at). Hence, the semantic module performs structural operations to transform a SemS into a DSyntS. While doing so, some LF might appear, such as the ones expressing support verbs (like $\texttt{Oper}_i$), that are semantically empty and therefore no semanteme represent them in SemS, or the one indicating a copula (`Copul`) that is also missing from SemSs. Figure 3 shows an example of DSyntS bearing the `Copul` LF.

On top of such structural operations, the semantic module also performs paraphrase steps, such as semantic paraphrasing and deep-syntactic paraphrasing. Semantic paraphrasing can bee seen as unfolding the semantic graph with semanteme definitions. Deep-syntactic paraphrasing relies heavily on LFs. Both paraphrase steps are described in (Milićević, 2007).

3 Abstract Categorial Grammars (ACGs)

ACGs (de Groote, 2001, whose definitions are used here) are a grammatical framework based on λ-calculus and used to represent other grammatical formalisms. An ACG is composed of two vocabularies (abstract and object vocabularies) that are linked by a lexicon. The abstract language, build upon the abstract vocabulary, corresponds to the set of all abstract grammatical structures, such as analysis trees. The object language, build upon the object vocabulary, corresponds to the set of surface realizations of abstract language structures, such as strings. Section 3.1 gives necessary definitions to define ACGs in section 3.2. Section 3.3 present some ACGs operations and properties.

3.1 Types, Signatures, and Lexicons

Definition 1 *Be A 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 *A **higher order signature**. Σ is a tuple $\Sigma = \langle A, C, \tau \rangle$, where:*

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

$\vdash_{\Sigma_1} t : s$ expresses that the type of a λ-term t is s in the signature Σ (or $t : s$ if there is no ambiguity on the used signature). $\Lambda(\Sigma)$ denotes the set of λ-terms obtained using constants of C, variables, abstractions and applications.

For instance, $dragon^{dt} : T$ means that the constant $dragon^{dt}$ has type T and $invite^{ds} : \text{MOD} \to G' \to G \to G$ means that the constant $invite^{ds}$ has type[1] $\text{MOD} \to G' \to G \to G$, so that it expects a first argument of type MOD (a modifier), a second argument of type G' (an optional argument) and a third argument of type G (a mandatory argument) to obtain a term of type G (a graph). $\Sigma_{\textit{deep-syntactic-0tr}}$ and $\Sigma_{\textit{dsynt-tree}}$, illustrated in figure 5, are higher-order signatures which extracts are given in figures 8a (page 8) and 6 (page 6).

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), regardless of whether the used morphism is F and both t and γ are types or the used morphism is G and they are terms.

Then, $\mathcal{L}_{\textit{dsyntRel}}(dragon^{ds}) = \lambda\ A.\ A\ dragon^{dt}$ means that the constant $dragon^{ds}$ (from $\Sigma_{\textit{deep-syntactic-0tr}}$) is interpreted by $\mathcal{L}_{\textit{dsyntRel}}$ in $\Sigma_{\textit{dsynt-tree}}$ as the term $\lambda\ A.\ A\ dragon^{dt}$. Similarly, $\mathcal{L}_{\textit{dsyntRel}}(invite^{ds}_{rtr}) = \lambda\ A\ X\ Y.\ A\ (\lambda x.\ A_1(A_2\ invite^{dt}\ Y)\ x)\ X$ means that the constant $invite^{ds}_{rtr}$ of $\Sigma_{\textit{deep-syntactic-0tr}}$ is interpreted by $\mathcal{L}_{\textit{dsyntRel}}$ in $\Sigma_{\textit{dsynt-tree}}$ as the term $\lambda\ A\ X\ Y.\ A\ (\lambda x.\ A_1\ (A_2 invite^{dt}\ Y)\ x)\ X$. In figure 5, $\mathcal{L}_{\textit{dsyntRel}}$ is the lexicon from $\Sigma_{\textit{deep-syntactic-0tr}}$ to $\Sigma_{\textit{dsynt-tree}}$. Its extract is given in figure 9 (page 9).

3.2 ACG

We may now define notions of ACG, abstract language and object language:

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:*

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

This article uses $\beta\eta$-equivalence as equality between λ-terms.

[1] More detail about these types is given in section 5.

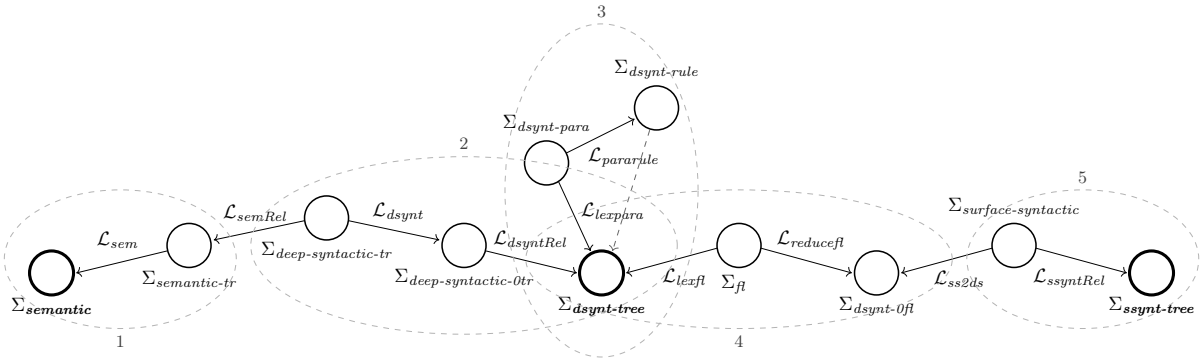


Figure 4: ACG composition of MTT encoding

Both abstract and object languages of an ACG are sets of λ-terms obtained by induction over eponym signatures of the ACG. For example, the set of deep-syntactic trees corresponds to the object language of the ACG (illustrated in figure 5) made of $\Sigma_{deep\text{-}syntactic\text{-}0tr}$ (its abstract signature), $\Sigma_{dsynt\text{-}tree}$ (its object signature), and $\mathcal{L}_{dsyntRel}$.

3.3 Operations and Properties

ACGs allow the use of three main operations, as illustrated in figure 5. Application is the operation of applying the lexicon of an ACG from its abstract signature to its object signature ($\mathcal{L}_{dsynt}(\gamma^{dst}) = \gamma^{ds}$). Lexicons are reversible, and the operation of reversing the lexicon from the object signature of an ACG to its abstract signature is called parsing ($\mathcal{L}^{-1}_{semRel}(\gamma^{str}) = \{\gamma^{dst}\}$). Both operations (application and parsing) can be composed to form a transduction operation ($\mathcal{L}_{dsynt}(\mathcal{L}^{-1}_{semRel}(\gamma^{str})) = \{\gamma^{ds}\}$). Two transduction examples are shown in figure 5.

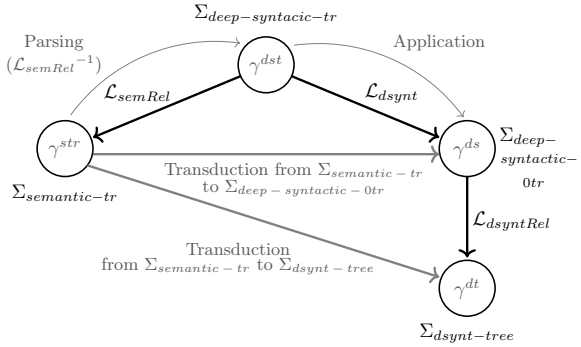


Figure 5: Example of application, parsing and transduction operations

An ACG is characterized by its order and its complexity. The complexity of an ACG is its parsing complexity.

Definition 6 *The order of a type $\tau \in \mathcal{T}(A)$ is inductively defined:*

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

The order of an abstract constant is the order of its type, and the ***order of an ACG*** *is the maximum of the order of its abstract constants.*

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 $ACG_{(\gamma,\eta)}$.*

We aim to use a specific class of ACGs, ACGs said of order two, since their parsing complexity is decidable and polynomial (Salvati, 2005; Kanazawa, 2017)

4 MTT Into ACG Encoding

We use the ACG architecture of (Cousin, 2025) (that uses Cousin, 2023b,a architecture while adding theme-rheme opposition to the semantic to deep-syntactic transition). However, we rewrite several ACGs to model DSyntR and SSyntR dependency trees based on de Groote's (2023) ACGs. We use the ACGtk software (Guillaume et al., 2024) to implement this model.

Figure 4 shows the architecture and composition of Cousin (2023b, 2025) that we use here. It is divided into five areas:

- Areas 1 - 2: transduction between $\Sigma_{semantic\text{-}tr}$ and $\Sigma_{dsynt\text{-}tree}$ allows transitions between SemR and DSyntR. Area 1 corresponds to semantic and area 2 corresponds to deep-syntax;
- Area 3: transduction between $\Sigma_{dsynt\text{-}tree}$ and $\Sigma_{dsynt\text{-}rule}$ performs deep-syntactic paraphrase steps;

- Area 4: transduction between $\Sigma_{dsynt\text{-}tree}$ and $\Sigma_{dsynt\text{-}0ft}$ realizes LFs eventually present in DSyntR;
- Area 5: transduction between $\Sigma_{dsynt\text{-}0ft}$ and $\Sigma_{ssynt\text{-}tree}$ transforms DSyntR (without LFs) into SSyntS. Area 5 corresponds to surface-syntax.

Our modifications concern areas 2, 4 and 5. This article only present the ones about the second area, *i.e.*, about deep-syntax.

Notations: In the following, $\Sigma_{deep\text{-}syntactic\text{-}tr}$ is referenced as Σ_{dst} (deep-syntax with thematicity), $\Sigma_{deep\text{-}syntactic\text{-}0tr}$ as Σ_{ds} (deep-syntax without thematicity), and $\Sigma_{dsynt\text{-}tree}$ as Σ_{dt} (deep-syntactic trees). A constant representing a lexeme LEX in the signature Σ_S is written lex^S. For instance, $dragon^{dt}$ is the constant representing the lexeme DRAGON in Σ_{dt}. A term representing an expression E in $\Lambda(\Sigma_S)$ is written γ^S_E. For instance, the term encoding the DSyntR associated to the expression (cpd): "*the calm purple dragon*" in Σ_{dt} is written γ^{dt}_{cpd}.

5 Deep-Syntax Encoding

We adapt de Groote's (2023) dependency structures. This section details how it is done for deep-syntactic dependency structures. Figures 8a, 8b, 6, 8c and 9 (on the following pages) respectively show extracts of Σ_{dst}, Σ_{ds}, Σ_{dt}, $\mathcal{L}_{dsynt}$ and $\mathcal{L}_{dsyntRel}$ that are used below.

5.1 DSyntS Encoding (Σ_{dt})

Σ_{dt} (deep-syntactic trees, see figure 6) enables DSyntS representation. DSyntS are dependency tree structures, which nodes are deep lexemes and branches are deep-syntactic relations, such as **I**, **II**, or **ATTR** for example. These three examples respectively correspond to relations expressing the first actant, the second actant, or an attributive dependent of a deep lexeme. A relation is represented as a branch in the deep-syntactic tree that starts at the governor and points toward its dependent.

Each deep lexeme is encoded in this implementation by a constant of type T (see (1)). In MTT, categories are distinguished at the deep-syntactic level. However, for simplification purposes, we chose to represent them only at the surface-syntactic level of the present encoding and not at its deep-syntactic level. Therefore, grammatically incorrect structures can be produced. However, they will be filtered out by lexicons and won't be generated at the surface syntactic level (see section 5.3). Hence, no distinction is made between grammatical categories in Σ_{dt}.

$$dragon^{dt}, invite^{dt} : T \tag{1}$$

Since a relation starts at one lexeme and points toward another, it is represented by a constant that takes two arguments of type T (being resp. the governor and the dependent), and that has a resulting type T. Hence, each deep-syntactic relation (such as the ones in (2)) is of type $T \to T \to T$.

$$\mathbf{I}, \mathbf{ATTR} : T \to T \to T \tag{2}$$

Since the resulting type of a relation is T, a dependency (*i.e.* a governor, the relation, and its dependent) is represented by a term of type T as well. This term can in turn be used as a governor (to add another dependency to its governor) or as a dependent (to make it depend from another lexeme). Hence, type T is used for constants and terms encoding trees, sub-trees, and single nodes.

$$\begin{array}{r} \Sigma_{dsynt\text{-}tree} \\ T : type \end{array}$$

$$\begin{array}{rl} \mathbf{I}, \mathbf{II}, \mathbf{III} & : T \to T \to T \\ \mathbf{ATTR} & : T \to T \to T \end{array}$$

$$Conv_{21} : T \to T \to T$$

$$\begin{array}{rl} calm^{dt} & : T \\ invite^{dt} & : T \\ dragon^{dt} & : T \\ often^{dt} & : T \\ purple^{dt} & : T \\ reason^{dt} & : T \\ specific^{dt} & : T \\ wyvern^{dt} & : T \end{array}$$

Figure 6: Extract of $\Sigma_{dsynt\text{-}tree}$ (noted Σ_{dt}, that enables the representation of deep-syntactic trees)

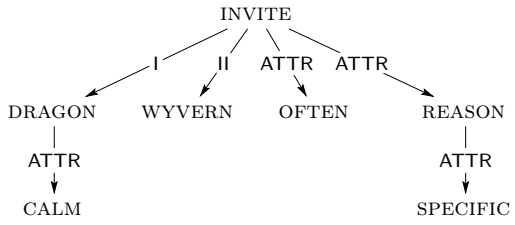
Figure 7 shows two examples of DSyntS along with terms of $\Lambda(\Sigma_{dt})$ encoding them. Figure 7a pictures a rather simple DSyntS, having only one dependency, while figure 7b pictures a more complex one, with several dependencies.

5.2 DSyntR Encoding (Σ_{dst} and Σ_{ds})

As stated in the introduction (section 1), a DSyntR contains a communicative structure, DSynt-CommS. This structure is absent from Σ_{dt} (deep-syntactic trees) and Σ_{ds} (deep-syntax without thematicity), but is present in Σ_{dst} (deep-syntax

$$\gamma^{dt}_{cd} = \mathbf{ATTR}\ dragon^{dt}\ calm^{dt} \\ :\ T$$

(a) "*the calm dragon*" (γ^{dt}_{cd})



$$\gamma^{dt}_{cdoiwfsr} = \mathbf{ATTR}\ (\mathbf{ATTR}\ (\mathbf{I}\ (\mathbf{II}\ invite^{dt} \\ wyvern^{dt})\ (\mathbf{ATTR}\ dragon^{dt}\ calm^{dt}))\\ often^{dt})\ (\mathbf{ATTR}\ reason^{dt}\ specific^{dt}) \\ :\ T$$

(b) "*The calm dragon often invites the wyvern for a specfic reason*" ($\gamma^{dt}_{cdoiwfsr}$)

Figure 7: Representation of DSyntS associated to expressions "*the calm dragon*" and "*The calm dragon often invites the wyvern for a specific reason*" and their associated object terms (from $\Lambda(\Sigma_{dt})$).

with thematicity). Indeed, theme-rheme opposition plays a crucial role in the determination of the DSyntR that corresponds to a SemR. In order to represent this role and enable the potential corresponding choices inherent to the theme-rheme opposition of a SemR, thematic annotations are encoded in the type of constants from Σ_{str} (semantic with theme-rheme opposition) and Σ_{dst} (deep-syntax with thematicity).

In both Σ_{dst} and Σ_{ds}, lexemes are encoded with constants of type G-like. These types stand for graph, subgraph or node and follows the same use principle as type T. Variants can be found, such as G_t or G_r, that respectively express a theme or a rheme marking. If a prime symbol is used (like G' or G'_t), it denotes an optionally expressible argument[2]. Type $G_{f,r}$ is used to indicate that a predicative structure contains a thematic opposition (i.e. both theme and rheme areas) with the index $-_f$, and that its dominant node (the future tree root) is marked as rheme with the index $-_r$.

For instance, $invite^{dst}_{rtr}$ (which type expression is given in (3)) is a constant encoding a verb which predicative structure expects two arguments, the first one being marked as theme (type G_t), the remaining part of the predicative structure being marked as rheme (type G_r). Its first argument (of type MOD) stands for a potential modifier[3], such as an adverbial group.

$$invite^{dst}_{rtr} :\ \text{MOD} \rightarrow G_t \rightarrow G_r \rightarrow G_{f,r} \tag{3}$$

Thematicity markers in the constants' types are lost between Σ_{dst} and Σ_{ds}. $\mathcal{L}_{dsynt}$ only purpose is to erase these types markings (see Figure 8c). For instance, $\mathcal{L}_{dsynt}(invite^{dst}_{rtr}) = invite^{ds}_{rtr}$ is given in (4). Therefore, expressions of terms from $\Lambda(\Sigma_{dst})$ and $\Lambda(\Sigma_{ds})$ are really similar (see figures 8a and 8b, page 8).

$$invite^{ds}_{rtr} :\ \text{MOD} \rightarrow G \rightarrow G \rightarrow G_f \tag{4}$$

It is $\mathcal{L}_{dsyntRel}$ (see Figure 9) that will really build all DSyntS in $\Lambda(\Sigma_{dt})$ from $\Lambda(\Sigma_{ds})$. In other words, Σ_{dst} is used to express and control thematicity contraints and optional arguments issues (see section 5.3), while $\mathcal{L}_{dsyntRel}$ realizes DSyntS. $\mathcal{L}_{dsynt}$ and Σ_{ds} are intermediary lexicon and signature used for encoding purposes.

Σ_{dst} bears the encoding of all constraints and is therefore used in examples from the following section about these constraints. However, for reading simplicity purposes, we use in section 5.5 examples from Σ_{ds} (see figure 8b). Their pendants from Σ_{dst} do not add any information for section 5.5 purpose and can be found in figures 8a, and their interpretation by $\mathcal{L}_{dsynt}$ in figure 8c (page 8).

5.3 Constraints Encoding (Σ_{dst} and $\mathcal{L}_{dsyntRel}$)

Since all lexemes of Σ_{dt} are encoded by constants of type T, incorrect structures can be build upon $\Lambda(\Sigma_{dt})$, like the one illustrated in (5). In this example, two major issues occur. First, in a DSyntR, according to well-formedness rules of MTT, each node (or lexeme) has at most one actancial relation of each kind; $dragon^{dt}$ can't be the governor of two **I** relations. Second, $invite^{dt}$ is a lexeme expecting at least one dependent ("*Y is invited*"), and usually two, since its predicative structure contains two arguments: $X\ invites\ Y$.

$$\gamma^{dt}_{illicit} = \mathbf{I}\ (\mathbf{I}\ dragon^{dt}\ wyvern^{dt})\ invite^{dt} \\ :\ T \tag{5}$$

[2]Optionally expressible arguments are detailed in section 5.3.

[3]Modifiers are described in section 5.5.

$\Sigma_{\textit{deep-syntactic-tr}}$

$G_r,\ G_t,\ G'_r : type$
$\text{MOD}_{rr} : type$
$\text{MOD}_{tr} : type$
$\text{MOD}'_{rr} : type$
$\text{MOD}j_{rr} : type$

$C_{r\to t} : G_r \to G_t$
$I^{dst}_{\text{MOD},rr} : \text{MOD}_{rr}$
$I^{dst}_{\text{MOD},tr} : \text{MOD}_{tr}$
$I^{dst}_{\text{MOD}',rr} : \text{MOD}'_{rr}$
$I^{dst}_{\text{MOD}j,rr} : \text{MOD}j_{rr}$

$calm^{dst}_{rr} : \text{MOD}_{rr} \to \text{MOD}j_{rr} \to \text{MOD}j_{rr}$
$invite^{dst}_{rtr} : \text{MOD}_{tr} \to G_t \to G_r \to G_{f,r}$
$invite^{dst}_{rrt} : \text{MOD}'_{rr} \to G'_r \to G_t \to G_{f,r}$
$dragon^{dst}_{r} : \text{MOD}j_{rr} \to G_r$
$often^{dst}_{tr} : \text{MOD}_{tr} \to \text{MOD}_{tr}$
$purple^{dst}_{rr} : \text{MOD}_{rr} \to \text{MOD}j_{rr} \to \text{MOD}j_{rr}$
$fsr^{dst}_{tr} : \text{MOD}_{tr} \to \text{MOD}_{tr}$
$wyvern^{dst}_{r} : \text{MOD}j_{rr} \to G_r$

(a) Extract of $\Sigma_{\textit{deep-syntactic-tr}}$ (noted Σ_{dst})

$\Sigma_{\textit{deep-syntactic-0tr}}$

$G,\ G' : type$
$\text{MOD} : type$
$\text{MOD}' : type$
$\text{MOD}j : type$

$I^{ds}_{\text{MOD}} : \text{MOD}$
$I^{ds}_{\text{MOD}'} : \text{MOD}'$
$I^{dst}_{\text{MOD}j} : \text{MOD}j$

$calm^{ds} : \text{MOD} \to \text{MOD}j \to \text{MOD}j$
$invite^{ds}_{rtr} : \text{MOD} \to G \to G \to G$
$invite^{ds}_{rrt} : \text{MOD}' \to G' \to G \to G$
$dragon^{ds} : \text{MOD}j \to G$
$often^{ds} : \text{MOD} \to \text{MOD}$
$purple^{ds} : \text{MOD} \to \text{MOD}j \to \text{MOD}j$
$fsr^{ds} : \text{MOD} \to \text{MOD}$
$wyvern^{ds} : \text{MOD}j \to G$

(b) Extract of $\Sigma_{\textit{deep-syntactic-0tr}}$ (noted Σ_{ds})

$\mathcal{L}_{dsynt}$

$G_r, G_t := G$
$G'_r := G'$
$\text{MOD}_{rr}, \text{MOD}_{tr} := \text{MOD}$
$\text{MOD}'rr := \text{MOD}'$
$\text{MOD}j_{rr} := \text{MOD}j$

$C_{r\to t} := \lambda p.\ p$
$I^{dst}_{\text{MOD},rr}, I^{dst}_{\text{MOD},tr} := I^{ds}_{\text{MOD}}$
$I^{dst}_{\text{MOD}',rr} := I^{ds}_{\text{MOD}'}$
$I^{dst}_{\text{MOD}j,rr} := I^{ds}_{\text{MOD}j}$
$calm^{dst}_{rr} := calm^{ds}$
$invite^{dst}_{rtr} := invite^{ds}_{rtr}$
$invite^{dst}_{rrt} := invite^{ds}_{rrt}$
$dragon^{dst}_{r} := dragon^{ds}$
$often^{dst}_{tr} := often^{ds}$
$purple^{dst}_{rr} := purple^{ds}$
$fsr^{dst}_{tr} := fsr^{ds}$
$wyvern^{dst}_{r} := wyvern^{ds}$

(c) Extract of $\mathcal{L}_{dsynt}$

Figure 8: Extracts of $\Sigma_{\textit{deep-syntactic-0tr}}$ (deep-syntax without thematicity), $\Sigma_{\textit{deep-syntactic-tr}}$ (deep-syntax with thematicity) and $\mathcal{L}_{dsynt}$, lexicon from $\Sigma_{\textit{deep-syntactic-tr}}$ to $\Sigma_{\textit{deep-syntactic-0tr}}$

In order to tackle these issues, two main strategies are used. Regarding the number of dependents (and their nature), *we use abstract signatures to control obtained structures by encoding the constraints in the constants types.* Hence, these constraints are dealt with in abstract signatures Σ_{dst} and Σ_{ds}. Figures 8a and 8b give extracts of Σ_{dst} and Σ_{ds} as examples.

Each predicate has a predicative structure that expects mandatory dependents. Among these dependents, some are necessarily expressible, and other may be omitted. Let's take the example of INVITE. When using this verb, we know that *someone*$_1$ invites another *someone*$_2$. However, sentences like "*Charlie is invited*" are correct (with "*Charlie*" being the invited *someone*$_2$). Hence INVITE has two mandatory dependents (*someone*$_1$ and *someone*$_2$), but the first one is optionally expressible. Such constraints are expressed in the type of constants of Σ_{dst}, with the prime notation indicating an optionally expressible argument. In the case of a noun like DRAGON, no dependent is expected, hence no argument is mandatory. (6) and (7) [4] give the types of constants $invite^{dst}_{rrt}$ and $dragon^{dst}_{r}$ illustrating the encoding of this constraint.

$$invite^{dst}_{rtr} : \text{MOD}_{tr} \to G_t \to G_r \to G_{f,r} \tag{6}$$

$$\begin{aligned} invite^{dst}_{rrt} &: \text{MOD}'_{rr} \to G'_r \to G_t \to G_{f,r} \\ dragon^{dst}_{r} &: \text{MOD}j_{rr} \to G_r \end{aligned} \tag{7}$$

Nevertheless, (7) shows two constants for INVITE, $invite^{dst}_{rtr}$ and $invite^{dst}_{rrt}$. The former has its first argument marked as theme, and will produce sentences like "*[The dragon]*$_{Theme}$ *[invites the wyvern]*$_{Rheme}$", where the first dependent of INVITE is not optionally expressible (since it is the theme, hence it can not be omitted). The latter has its second argument marked as theme, and will produce sentences like "*[The wyvern]*$_{Theme}$ *[is invited by the dragon]*$_{Rheme}$", where the first semantic dependent of INVITE, DRAGON, is marked as rheme and can easily be omitted ("*[The wyvern]*$_{Theme}$ *[is invited]*$_{Rheme}$"). So, on top of constraints inherent to the predicative structure (optionally expressible arguments), we need to take the thematic opposition and its consequences into account. This example (with INVITE) is one illustration of the role of the thematic opposition. This opposition is to be found in communicative structures of MTT, that complement its predicative structures. Theme-rheme opposition is not limited to active/passive distinction[5]. Another example of its implication can be found in the introduction (subsection 1.2.1)

[4] MOD-like types (MOD_{tr}, MOD'_{rr}, etc.) are modifiers types and explained in section 5.5. Their theme-rheme markings reflect the thematic markings of their semantic actants.

[5] See (Polguère, 1990) and (Mel'cuk, 2001) for further details about MTT's communicative structure and the thematic opposition. Some similarities can be found between theme-rheme opposition and information structure (Kruijff-Korbayova and Steedman, 2003; Steedman, 2024).

$$\mathcal{L}_{ssyntRel}$$

$$\begin{aligned}
G,\ G'_r &:= T \\
\text{MOD}, \text{MOD}' &:= (T \to T) \to (T \to T) \\
\text{MOD}j &:= T \to T \\
\\
I^{ds}_{\text{MOD}} &:= \lambda v.\ v \\
I^{ds}_{\text{MOD}'} &:= \lambda v.\ v \\
I^{ds}_{\text{MOD}j} &:= \lambda n.\ n \\
\\
calm^{ds} &:= \lambda M\ A\ P.\ (\textbf{ATTR}\ (A\ P)\ (M\ (\lambda m.\ m)\ calm^{dt})) \\
invite^{ds}_{rtr} &:= \lambda M\ X\ Y.\ M\ (\lambda x.\ \textbf{I}\ (\textbf{II}\ invite^{dt}\ Y)\ x)\ X \\
invite^{ds}_{rrt} &:= \lambda M\ X\ Y.\ M\ (\lambda x.\ \textbf{I}\ (\textbf{II}\ (Conv_{21}\ invite^{dt})\ x)\ Y)X \\
dragon^{ds} &:= \lambda A.\ A\ dragon^{dt} \\
often^{ds} &:= \lambda M\ PX.\ M\ (\lambda x.\ \textbf{ATTR}\ (P\ x)\ often^{dt})X \\
purple^{ds} &:= \lambda M\ A\ P.\ (\textbf{ATTR}\ (A\ P)\ (M\ (\lambda m.\ m)\ purple^{dt})) \\
fsr^{ds} &:= \lambda M\ PX.\ M\ (\lambda x.\ \textbf{ATTR}\ (P\ x)\ (\textbf{ATTR}\ reason^{dt}\ specific^{dt})\ X \\
wyvern^{ds} &:= \lambda A.\ A\ wyvern^{dt}
\end{aligned}$$

Figure 9: Extract of $\mathcal{L}_{dsyntRel}$, lexicon from $\Sigma_{deep\text{-}syntactic\text{-}0tr}$ to $\Sigma_{dsynt\text{-}tree}$

where it determine the apparition (or not) of a copula. As shown in (7), the thematic opposition is also encoded in the type of abstract constraints (see (Cousin, 2025) for further details on the thematic opposition encoding).

Regarding the well-formedness of DSyntR, it is enforced in $\mathcal{L}_{dsyntRel}$ (see Figure 9) that associates abstract constants from Σ_{ds} to their object terms in $\Lambda(\Sigma_{dt})$. (8)[6] shows the interpretations by $\mathcal{L}_{dsyntRel}$ of a predicate expecting two dependents, respectively an actant **I** and an actant **II** ($invite^{ds}_{rtr}$) and the interpretation of a noun which predicative structure does not except any dependents ($dragon^{ds}$).

$$\begin{aligned}
&\mathcal{L}_{dsyntRel}(invite^{ds}_{rtr}) \\
&:=\ \lambda\ M\ X\ Y.\ M\ (\lambda\ x.\ \textbf{I}\ (\textbf{II}\ invite^{dt}\ Y)\ x)\ X \\
&\mathcal{L}_{dsyntRel}(dragon^{ds}) \\
&:=\ \lambda\ A.\ A\ dragon^{dt}
\end{aligned} \tag{8}$$

The encoding of interpretations by the lexicon solves the issue of having only well-formed structures in $\Lambda(\Sigma_{dt})$. Since only correct interpretations are encoded, only correct structures will be generated. Reversely, if an incorrect structure is build upon Σ_{dt}, no antecedent will be found by parsing of $\mathcal{L}_{dsyntRel}$, and hence no SemR can be obtained (when continuing the analysis).

We therefore have the following equalities[7]:

$$\begin{aligned}
&\mathcal{L}_{dsyntRel}(\mathcal{L}_{dsynt}(dragon^{dst}_{r}\ (calm^{dst}_{rr}\ I^{dst}_{\text{MOD},rr}\ I^{dst}_{\text{MOD}j}))) \\
&= \textbf{ATTR}\ dragon^{dt}\ calm^{dt} \\
&\mathcal{L}_{dsyntRel}(\mathcal{L}_{dsynt}(invite^{ds}_{rtr}\ I^{dst}_{\text{MOD},rt}\ (C_{r\to t} \\
&(dragon^{dst}_{r}\ I^{dst}_{\text{MOD}j,rr}))(wyvern^{dst}_{r}\ I^{dst}_{\text{MOD}j,rr}))) \\
&= \textbf{I}\ (\textbf{II}\ invite^{dt}\ wyvern^{dt})\ dragon^{dt}
\end{aligned} \tag{9}$$

5.4 Discussion

This encoding presents some major differences with de Groote's (2023) one, since we made choices opposed to his, as stated earlier in section 1.2.2. He encodes dependency relations in the abstract signature, while we do so in the object signature. In (de Groote, 2023), relations (or rather, constants encoding them) constrain the grammatical categories of their governor and dependent. Here, we do not have grammatical categories, at least not at the deep-syntactic level. In the encoding presented here, lexemes themselves (and not relations) constrain their dependents: they force the mandatory ones to be represented (by a lexeme or a dummy constant indicating that the dependent is, in fact, not expressed) and the obligatory expressible constant to be expressed (in this case the dummy constant of type G' can't be used). Dependency relations are encoded in the object signature, and do not constrain anything nor bear any constraints.

On top of that, de Groote (2023) respects what he calls the coherence principle: regardless of the dependencies' computation order, obtained structures are logically equivalent. This result is obtained using affine constants. However, we want linear constants and ACGs of order 2. It is a constraint we fix ourselves for parsing computation and complexity issues, and in order to encode the model in

[6] Abstract varibles A and M encode modifiers. Their behavior is explained in section 5.5.

[7] Constants I^{dst}_{MOD} and its variants (that depends on the thematic markings of their governor predicative structure) have a MOD-like type (*i.e.*, a variant of MOD) and are interpreted by $\mathcal{L}_{dsyntRel} \circ \mathcal{L}_{dsynt}$ as the identity.

ACGtk to automate the computation process. Thus, respecting the coherence principle seems incompatible with this constraint. Moreover, the order in which relations are computed does matter in our case. At the deep-syntactic level, relations are not ordered, so one could think the order is of no importance. However, their order does matter at the surface syntactic level, since it will be used in the linearisation process toward the morphological levels. Consequently, two structures having the same dependencies but in different orders are to yield two different expressions in MTT. Such representations have to be considered different and not logically equivalent. Hence, the dependency order does matter at the surface-syntactic level. Then, even if the order doesn't matter at the deep-syntactic level, it is also useless to care for the coherence principle at that level since it will have to be broken at the next level. Additionally, not respecting the coherence principle makes both syntactic levels have homogeneous encoding. For all these reasons, ACGs presented here do not respect the coherence principle.

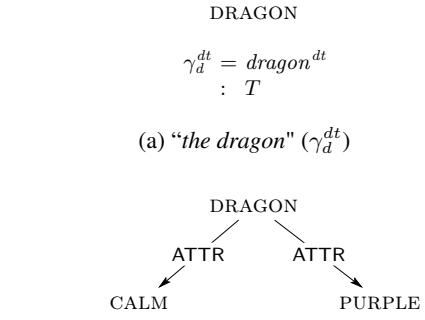
Next section describes the encoding of modifiers. One straightforward possibility to do so would have been to allow a bounded arbitrary number of optional dependents slots in the lexemes' lexical entry. This, however, would significantly complexify the deep-syntactic paraphrase part of the encoding (area 3 of figure 4), and one could always find a case where the arbitrary number of slots would be too small to represent the wanted expression. For these reasons, we chose to use a recursive solution, drawn from Pogodalla's (2017) TAG encoding, and described below.

5.5 Modifiers Encoding

Modifiers, such as adjectives and adverbial groups, have a specific behavior at the deep-syntactic level.

We may observe that they have a specific behavior in the semantic module as well. In a SemR, the modifier predicates over the entity that is modified (the modifier governs the predicate), while in a DSyntR, dependencies between a modifier and a predicate usually go from the predicate to the modifier: the predicate governs the modifier (some exceptions can be found, but that is the general case). Hence, a dependency inversion is to be found in the semantic module during the transition from SemR to DSyntR (Mel'čuk et al., 2013, p.248). We account for this, but since it is not our point here, we focus on DSyntR themselves.

In a DSyntR, modifiers are not mandatory dependents of a lexeme. A lexeme can have zero modifier (as illustrated in figure 10a), one modifier (as illustrated in figures 7a and 7b for DRAGON), or more (as illustrated for DRAGON in figure 10b and 7b for INVITE).



(a) "*the dragon*" (γ_d^{dt})

(b) "*the calm purple dragon*" (γ_{cpd}^{dt})

Figure 10: Representations of DSyntS associated to nominal expressions "*the dragon*" and "*the calm purple dragon*", and associated object terms (from $\Lambda(\Sigma_{dt})$).

In Montague semantics, a modifier like an adverb usually have a type like $(NP \to S) \to (NP \to S)$ (Carpenter, 1998). Since we encode constraints in the abstract signature and do not make distinctions between grammatical categories (see section 5.1), such a type would be translated with a type like $(G \to G) \to (G \to G)$, *i.e.*, a higher order type. To avoid such higher order types, we draw from Pogodalla's (2017) TAG encoding into ACGs and introduce MOD-like atomic types, used for modifiers in both Σ_{dst} and Σ_{ds}.

In Σ_{dt}, modifiers are considered like any other lexemes and encoded with a constant of type T (see figure 6, page 6). In Σ_{dst} and Σ_{ds}, MOD-like types are used.

5.5.1 Adverbial Modifiers

In a SemR, an adverbial group can govern up to two semantemes, namely the semanteme it modifies, and the first semantic actant of this semanteme (see figure 11, adverbial group "*for a specific reason*"). Therefore, an adverb is encoded with a constant of type[8] MOD $\to$ MOD, and MOD is interpreted by $\mathcal{L}_{dsyntRel}$ as $(T \to T) \to (T \to T)$, *i.e.*, a type similar to the classical one mentioned above. (10) gives the example of $often^{ds}$. Then, each constant of Σ_{ds} encoding a predicate that accepts an

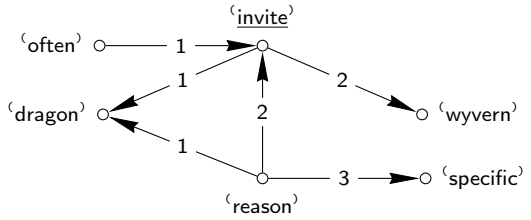


Figure 11: SemS associated to the expression "*The dragon often invites the wyvern for a specific reason*" that illustrate the semantic behavior of adverbial groups "*often*" and "*for a specific reason*".

adverbial modifier (*i.e.* verbs and adjectives[8]) have, in Σ_{ds}, its first argument being of type MOD (or MOD′ depending on the expressibility of the first argument), to enable a possible modification to take place. Following arguments have type G or G' (depending on the expressibility of arguments), and the final argument has type G (see (10)[9] for the example of $invite^{ds}_{rtr}$).

$$\begin{aligned} often^{ds} &: \text{MOD} \to \text{MOD} \\ invite^{ds}_{rtr} &: \text{MOD} \to G \to G \to G \\ \mathcal{L}_{dsyntRel}(\text{MOD}) &:= (T \to T) \to (T \to T) \\ \mathcal{L}_{dsyntRel}(often^{ds}) &:= \lambda\, M\ P\ X.\ M\ (\lambda\, x. \\ &\qquad \textbf{ATTR}\ (P\ x)\ often^{dt})\ X \end{aligned} \tag{10}$$

With such an encoding, adverbs are represented by constants of order 2, and several adverbs can be used to modify one predicate (provided its type accepts adverbial modifiers) as shown in (11)[10].

$$\begin{aligned} &invite^{ds}_{rtr}\ (often^{ds}\ I^{ds}_{\text{MOD}}) \\ &:\ G \to G \to G \\ &invite^{ds}_{rtr}\ (often^{ds}\ (fsr^{ds}\ I^{ds}_{\text{MOD}})) \\ &:\ G \to G \to G \end{aligned} \tag{11}$$

5.5.2 Adjectival Modifiers

Adjectives have the same encoding except that, in their SemR, they only have one dependent, the semanteme they modify. Hence, another MOD-like type is used for adjectives: MODj. An adjective can be modified by an adverb ("*the* ***often calm*** *dragon*"), so the first argument of a constant encoding an adjective is of type MOD, followed by the core of the adjective constant type: MOD$j \to$ MODj (see (12)[9]). Similarly to constants admitting adverbial modifiers, constants admitting adjectival modifiers also have their first argument being of type MODj.

$$\begin{aligned} purple^{ds} &: \text{MOD} \to \text{MOD}j \to \text{MOD}j \\ dragon^{ds} &: \text{MOD}j \to G \\ \mathcal{L}_{dsyntRel}(purple^{ds}) &:= \lambda M\ A\ P.\ (\textbf{ATTR}\ (A\ P) \\ &\qquad (M\ (\lambda m.\ m)\ purple^{dt})) \end{aligned} \tag{12}$$

We then have the following equalities (expressions of γ^{dt}_{d}, γ^{dt}_{cpd} and $\gamma^{dt}_{cdoiwfsr}$ are given in figures 10a, 10b and 7b respectively, along with their DSyntS representation):

$$\begin{aligned} \gamma^{ds}_{d} &= dragon^{ds}\ I_{\text{MOD}j} \\ \mathcal{L}_{dsyntRel}(\gamma^{ds}_{d}) &:= \gamma^{dt}_{d} \\ \gamma^{ds}_{cpd} &= dragon^{ds}(calm^{ds}\ I^{ds}_{\text{MOD}} \\ &\qquad (purple^{ds}\ I^{ds}_{\text{MOD}}\ I^{ds}_{\text{MOD}j})) \\ \mathcal{L}_{dsyntRel}(\gamma^{ds}_{cpd}) &:= \gamma^{dt}_{cpd} \\ \gamma^{ds}_{cdoiwfsr} &= invite^{ds}_{rtr}(often^{ds}(fsr^{ds}\ I^{ds}_{\text{MOD}})) \\ &\qquad (dragon^{ds}(calm^{ds}\ I^{ds}_{\text{MOD}}\ I^{ds}_{\text{MOD}j})) \\ &\qquad (wyvern^{ds}\ I^{ds}_{\text{MOD}j}) \\ \mathcal{L}_{dsyntRel}(\gamma^{ds}_{cdoiwfsr}) &:= \gamma^{dt}_{cdoiwfsr} \end{aligned} \tag{13}$$

6 Evaluation and Results

6.1 Evaluation

Lexicons and vocabularies used here have between 30 and 50 lexemes approximately. It's a toy grammar, but it would be possible to extract data from SUD annotated corpora (Gerdes et al., 2018) or from the RL-fr (ATILF, 2025) to extend our coverage. Syntactic structures of MTT are dependency structures that roughly correspond to the ones of SUD. Labels on relations are different, but tree structures of MTT surface-syntactic representations are very similar to SUD representations. Syntactic representations of the encoding presented here are in accordance with the ones of annotated corpora from Grew (Guillaume, 2021). These ACGs are tested on 63 terms build upon $\Lambda(\Sigma_{dst})$. Tests scripts perform transduction operations toward $\Sigma_{semantic}$ as well as toward $\Sigma_{ssynt\text{-}tree}$. For 63 input terms, 430 output terms (*i.e.*, surface-syntactic structures) are generated and manually verified.

[8] An adverbial modifier can modify another adverbial modifier. Hence, adverb type should be MOD → MOD → MOD. For reasons of readability and computational complexity we won't consider this case here and use type MOD → MOD instead. However, it can be treated in the exact same way as adverbial modification on adjectives (see subsection 5.5.2).

[9] Interpretations of $invite^{ds}_{rtr}$ and $dragon^{ds}$ by $\mathcal{L}_{dsyntRel}$ are given in (8)

[10] fsr^{ds} is a constant representing the adverbial group "*for a specific reason*"

6.2 Results

This encoding of MTT into ACGs enables the representation of concepts such as LFs, semantic paraphrasing, surface-syntactic paraphrasing, the communicative structure (theme-rheme opposition) and its role, as well as some deep-syntactic paraphrasing. It also allows the representation and handling of several lexical phenomena, among which synonymy, idiomatic expressions, and modifiers behavior. Several modifiers can be used to modify the same predicate. ACGs of this encoding are from the specific class of ACGs of order 2, which parsing is decidable and polynomial.

7 Conclusion

Inspired by (de Groote, 2022, 2023), the encoding of MTT into ACG presented in this article is closer to more classical encodings such as Pogodalla's (2017) TAG encoding into ACG. In contrast to previous encodings (Cousin, 2023b,a, 2025), this encoding is of order 2, and not of higher order. This enables, on top of polynomial ACG complexity, that all used lexicons can be parsed. Consequently, this encoding can be used for generation as well as analysis purposes. It is the case for every transition between two signatures, independently of whether the whole transition between semantic and surface-syntax is considered, or only an intermediate transition. Theme-rheme opposition is accounted for in semantic and deep-syntactic representation levels, what enables a more accurate SSyntS generation from a SemR. Additionally, the lexemes do no longer define and decide exactly how many dependents they will have. They only constrain their mandatory dependents, and do so via the type of abstract constants encoding them. The addition of optional dependents, such as modifiers (adjectives and adverbs), is permitted.

Acknowledgments

I would like to thank Sylvain Pogodalla (Université de Lorraine, CNRS, Inria, LORIA) and reviewers for their comments and constructive remarks, which helped to improve this article.

References

ATILF. 2025. Réseau lexical du français (rl-fr). ORTOLANG (Open Resources and TOols for LANGuage) –www.ortolang.fr.

Bob Carpenter. 1998. Chapter 4: Applicative categorial grammar. In *Type-Logical Semantics*. MIT Press.

Marie Cousin. 2023a. Meaning-text theory within abstract categorial grammars: Toward paraphrase and lexical function modeling for text generation. In *Proceedings of the 15th International Conference on Computational Semantics*, pages 134–143, Nancy, France. Association for Computational Linguistics.

Marie Cousin. 2023b. Vers une implémentation de la théorie sens-texte avec les grammaires catégorielles abstraites. In *Actes de CORIA-TALN 2023. Actes des 16e Rencontres Jeunes Chercheurs en RI (RJCRI) et 25e Rencontre des Étudiants Chercheurs en Informatique pour le Traitement Automatique des Langues (RÉCITAL)*, pages 72–86, Paris, France. ATALA.

Marie Cousin. 2025. Adding Communicative Structure to the MTT into ACG Encoding. A long version is in preparation.

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.

Philippe de Groote. 2015. Abstract categorial parsing as linear logic programming. In *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015)*, pages 15–25, Chicago, USA. Association for Computational Linguistics.

Philippe de Groote. 2022. Deriving Formal Semantic Representations from Dependency Structures. In *Logic and Engineering of Natural Language Semantics: 19th International Conference, LENLS19, Tokyo, Japan, November 19–21, 2022, Revised Selected Papers*, volume Lecture Notes in Computer Science, pages 157–172, Tokyo (JP), Japan. Springer.

Philippe de Groote. 2023. On the semantics of dependencies: Relative clauses and open clausal complements. In *Logic and Engineering of Natural Language Semantics - 20th International Conference, LENLS20, Osaka, Japan, November 18-20, 2023, Revised Selected Papers*, volume 14569 of *Lecture Notes in Computer Science*, pages 244–259. Springer.

Kim Gerdes, Bruno Guillaume, Sylvain Kahane, and Guy Perrier. 2018. SUD or surface-syntactic Universal Dependencies: An annotation scheme near-isomorphic to UD. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 66–74, Brussels, Belgium. Association for Computational Linguistics.

Bruno Guillaume. 2021. Graph matching and graph rewriting: GREW tools for corpus exploration, maintenance and conversion. In *Proceedings of the 16th*

Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations, pages 168–175, Online. Association for Computational Linguistics.

Maxime Guillaume, Sylvain Pogodalla, and Vincent Tourneur. 2024. ACGtk: A Toolkit for Developing and Running Abstract Categorial Grammars. In *Functional and Logic Programming. 17th International Symposium, FLOPS 2024*, volume Lecture Notes in Computer Science of *Functional and Logic Programming. 17th International Symposium, FLOPS 2024*, pages 13–30, Kumamoto, Japan. Springer.

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.

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.

Makoto Kanazawa. 2017. Parsing and generation as datalog query evaluation. *IfCoLog Journal of Logics and Their Applications*, 4(4):1103–1211.

Ivana Kruijff-Korbayova and Mark Steedman. 2003. Discourse and information structure. *Journal of Logic, Language and Information*, 12:249–259.

François Lareau, Florie Lambrey, Ieva Dubinskaite, Daniel Galarreta-Piquette, and Maryam Nejat. 2018. GenDR: A generic deep realizer with complex lexicalization. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Igor Mel'cuk. 2001. *Communicative Organization in Natural Language: The semantic-communicative structure of sentences*.

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.

J. Milićević. 2007. *La paraphrase: modélisation de la paraphrase langagière*. Sciences pour la communication. Lang.

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

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

Alain Polguère. 1990. *Structuration et mise en jeu procédurale d'un modèle linguistique déclaratif dans un cadre de génération de texte*. Ph.D. thesis.

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.

Mark Steedman. 2024. Combinatory categorial grammar: An introduction. Course notes from ESSLLI 2024.

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.