THE SYNTAX-SEMANTICS INTERFACE: FROM MONTAGUE GRAMMAR TO TODAY

Aleksandre Maskharashvili

Ohio State University, Ohio, USA
maskharashvili.1@osu.edu

Abstract

In this paper, we overview logical studies of the syntax-semantics interface, with the focus on Montague's seminal works in the field, called Montague Grammar. We also discuss more recent developments of Montague Grammar on the example of Abstract Categorial Grammars. We briefly illustrate how recent developments in Montague's line of research contribute to the studies in classical natural language problems such as parsing and generation. We also sketch an approach that makes use of Abstract Categorial Grammars to establish interrelations between the surface form and the meaning of a discourse.

1 Introduction

Studying regularities of forms across natural language expressions was a subject of study already in the first known grammatical system, created by Pāṇini for Sanskrit. One of the first interest in mathematical study of natural language syntax emerged in the works of Ajdukiewicz [1] who proposed a formal system of rules, called Categorial Grammar, which was supposed to represent the ways natural language sentences are built. One of the main insights in Ajdukiewicz's [1] categorial grammar is to use *functor categories* (*Funktoren-Kategorien*) and *basic categories* (*Grundkategorie*), where a functor category carries functional characteristics and a basic category is anything that is not a functor category. Bar-Hillel [2] further elaborated ideas of Ajdukiewicz [1].

Advances in proof theory motivated Lambek [16] to make use of the ideas of Ajdukiewicz and Bar-Hillel to provide a logical, proof-theoretic model of natural language syntax.

Chomsky [5] proposed Context-Free Grammars (CFGs) to study natural language syntax. His approach was based on Bloomfield's [4] principle of immediate constituents.

Shaumyan [30, 31] developed another line of study in natural language syntax in a setting based on Curry's theory of types and combinators.

In concordance with that, he called his framework Applicative Universal Grammar (AUG).

As studies in syntax had been developing in various directions, Montague [21, 19, 20] proposed a program of studying meaning in natural language. He designed a way to translate syntactic expressions to their semantic forms (meanings). Montague made use of Ajdukiewicz's syntactic calculus as a model of syntax. To encode semantic representations, Montague has developed his language, which from today's perspective can be seen as a version of higher-order logic (HOL). Below, we may refer to Montague's works as Montague Grammar (MG).

2 AB-Calculus

We follow Moot and Retoré [22] to define Ajdukiewicz's [1] (and Bar-Hillel's [2]) version of categorial grammar, called AB-calculus or AB-grammar. In an AB-calculus, an expression is a category (type) if and only if it is derivable from a set of basic categories, denoted by P, in following way:

$$L := P \mid (L\backslash L) \mid (L/L) \tag{1}$$

If v and u are categories, then $u\backslash v$ and v/u are such *functor* categories which take an expression of type u as its argument and produce an expression of type v. The difference between $u\backslash v$ and v/u is *directional*, that is, $u\backslash v$ receives an argument from *right*, whereas v/u receives it from *left*. These possibilities of receiving arguments (from right and from left) are encoded with the following rules (called the right and left elimination rules, respectively):

$$\begin{array}{ll} u\ (u\backslash v) \to v & (\backslash_e) \\ (v/u)\ u \to v & (/_e) \end{array} \tag{2}$$

Formally, an AB-grammar is a function $\mathscr{L}$ which maps each word (a terminal symbol) a finite set of types (in a case of an ambiguous word, several but finite number of types may correspond to it).

Definition 1. We say that a string $w_1 \dots w_n$ is *derivable* in an AB-grammar, and its category is u if and only if for each w_i there exist t_i, $i = 1, 2 \dots n$ such that t_i is a category of w_i (i.e. $t_i \in \mathscr{L}(w_i)$) and $t_1 \dots t_n \to u$. The set of strings which are derivable by a given Ab-grammar is called the language produced by that AB-grammar.

CAT	Description	Expressions
S	Sentences	John walks, A woman smiles, etc.
IV	Intransitive Verbs	walk, sleep, smile, etc.
CN	Common Nouns	woman, book, tie, man, etc.
T = S/IV	Noun Phrases	a woman, John, every man, he_0, he_1, etc.
TV = IV/T	Transitive Verbs	love, see, meet, etc.

Table 1: Categories and their use in syntax and semantics

3 Glance at Montague Grammar

In order to provide systematic translations of syntactic expression into logical ones, Montague [21] assumes the following:

- Every lexical entry in the grammar has a syntactic category that has a corresponding semantic category.
- Compositionality Principle: The meaning of a compound expression is a function of the meanings of its parts and of the syntactic rule that combines these parts.

3.1 Syntax

Montague [21] employs an AB-calculus at the level of syntax. He defines rules that enable one to produce new categories out of the given ones as follows:[1]

Definition 2. CAT, the set of categories, is the smallest set such that:

- *S, IV, CN* belong to CAT.
- If A and B are elements CAT, then A/B belongs to CAT as well.[2]

S, stands for sentences, IV for intransitive verbs, and CN for common nouns. Table 1 describes how Montague defines other categories in terms of these three categories.

The syntactic derivation process in Montague's approach is the same as in AB-calculus. B_A denotes the set of lexical items of category A. For instance, B_{T} contains *a man*, *all dogs*, together with he_i for every natural number i.

[1] We do not provide the exact version of Montague's original grammar, but follow a more recent one, namely, Dowty [11].

[2] Montague [21] introduces another syntactic type, denoted as A//B, that has the same semantic meaning (i.e. translation) as A/B, but they model different syntactic categories. That is why, we will only use A/B categories as it is done, for instance, by Gamut [13].

Montague [21] proposes seventeen syntactic rules, which are classified as follows: basic rules, rules of functional application, rules of conjunction and disjunction, rules of quantification, rules of tense and sign. Each of these syntactic rules has the corresponding semantic rule. Let us consider some of Montague's syntactic rules and in the next section we discuss their semantic translations.

S1 is the first rule in MG, which is one of basic rules:

S1 B_A being the set of lexical elements of category A is contained in the set of all expressions of category A denoted as P_A, that is $B_A \subset P_A$

The another basic rule is the following:

S2 If $\zeta \in P_{\mathrm{CN}}$, then $F_0(\zeta), F_1(\zeta), F_2(\zeta) \in P_{\mathrm{T}}$ where:

- $F_0(\zeta) = \textit{every}\ \zeta$
- $F_1(\zeta) = \textit{the}\ \zeta$
- $F_2(\zeta)$ is $a\ \zeta$ or $an\ \zeta$ according as the first word in ζ takes a or an

Some of rules of functional application are as follows:

S4 If $\alpha \in P_{\mathrm{S/IV}}$ and $\delta \in P_{\mathrm{IV}}$, then $F_4(\alpha, \delta) \in P_{\mathrm{S}}$, where $F_4(\alpha, \delta) = \alpha\delta'$ and δ' is the result of replacing the first *verb* in δ by its third person singular present

The following rules are for quantification:

S14 If $\alpha \in P_{\mathrm{T}}$ and $\phi \in P_{\mathrm{S}}$, then $F_{10,n}(\alpha, \phi) \in P_{\mathrm{S}}$, where either:

1. α does not have the form he_k, and $F_{10,n}(\alpha, \phi)$ comes from ϕ by replacing the first occurrence of he_n or him_n by α and all other occurrences of he_n or him_n by $\left\{ \begin{array}{l} he \\ she \\ she \end{array} \right\}$ or $\left\{ \begin{array}{l} him \\ her \\ it \end{array} \right\}$ respectively, according as the gender of the first B_{CN} or B_{T} in α is $\left\{ \begin{array}{l} \text{masc.} \\ \text{fem.} \\ \text{neuter} \end{array} \right\}$, or
2. $\alpha = he_k$, and $F_{10,n}(\alpha, \phi)$ comes from ϕ by replacing all occurrences of he_n or him_n by he_k or him_k respectively

S15 If $\alpha \in P_{\mathrm{T}}$ and $\zeta \in P_{\mathrm{T}}$, then $F_{10,n}(\alpha, \zeta) \in P_{\mathrm{CN}}$

(1) Every man loves Mary.

Fig. 1 shows an exemplifying derivation of a sentence (1) using MG. A node of the tree in Fig. 1 shows the rule that is applied to the expressions from its daughter nodes in order to obtain the expression standing as the current node. If one views the derivation from a bottom up perspective, then the first rule used is **S5**; as a result, one produces out of *loves* of category IV/T and *Mary* of category IV/T the expression *loves Mary* of category IV. Using the rule **S4** leads to he_0 *loves Mary* of category P_{S}. On the other hand, by using the rule **S2**, one produces *every man*. We can then employ the rule **S14** in order to produce the expression $F_{10,0}(\textit{every man}, he_0 \textit{ loves Mary})$ of category S.

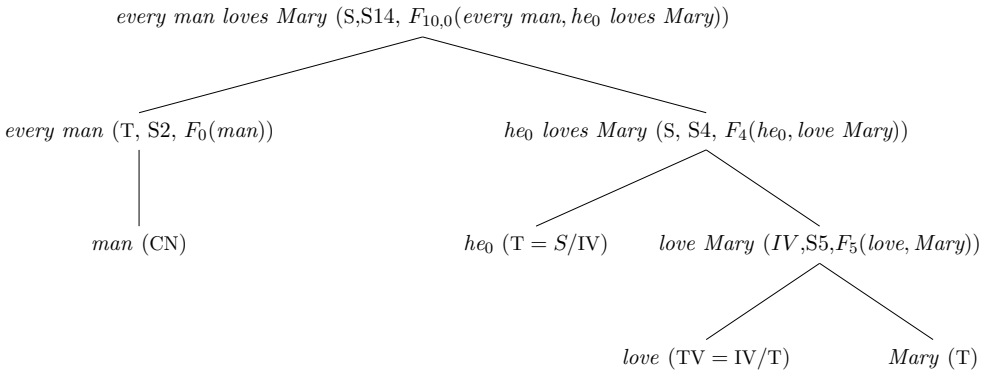


Figure 1: MG Derivation of *Every man loves Mary*

3.2 Translating Syntax to Semantics

Montague [21] shows how to translate syntactic expressions into the corresponding semantic expressions by translating his rules governing syntax into the corresponding semantic ones.[3]

S category for sentences translates to t. Also note that the syntactic categories IV and CN, corresponding to intransitive verbs and common nouns respectively, translate to the same semantic type, $e \to t$.

Montague [21] translates the other syntactic types using the following rule:

$$f \textit{ is a translation function of types such that } f(A/B) = f(B) \to f(A) \tag{3}$$

[3]While Montague's original translation is an *intensional* one, we only present extensional translations as it is sufficient for our current purposes.

Using the rule (3), one can derive Montague's translation[4] of the categories of noun phrases and transitive verbs as follows:

$$\begin{array}{rl} \textit{Noun phrases}: & f(\mathrm{T}) = f(\mathrm{S/IV}) = (e \to t) \to t \\ \textit{Transitive verbs}: & f(\mathrm{TV}) = f(\mathrm{IV/T}) = f(\mathrm{T}) \to f(\mathrm{IV}) = \\ & = f(\mathrm{S/IV}) \to f(\mathrm{IV}) = ((e \to t) \to t) \to e \to t \end{array} \tag{4}$$

In the MG semantic alphabet, one has individual constants and variables of type e, such as **j** for John, **m** for Mary etc.

To encode semantic representations of intransitive verbs, Montague introduces constants such as **sleep** of type $e \to t$, **cry** of type $e \to t$, etc. In words, these constants are one-place predicates whose argument is of type e. The same is true for common nouns (e.g. *woman*, *book*) and adjectives (e.g. *smart*, *interesting*, *tall*), in MG they are also treated as one-place predicates of type $e \to t$.

One of the insights in MG is Montague's [21] higher-order interpretations of noun phrases: all noun-phrases are of type $(e \to t) \to t$. For instance, (a) *John* is translated as $\lambda P.P\, j$, where j is of type e and P is of type $e \to t$; (b) A noun phrase such as *every man* is translated as $\lambda P.\forall x.\mathbf{man}\, x \supset P\, x$. The syntactic variables he_n, for $n = 0, 1, \ldots$, translate to $\lambda P.P\, x_n$, where x_n is a variable of type e. (c) A noun phrase, which is built with the help of an indefinite article, e.g. *a woman* is translated as $\lambda P.\exists x.\mathbf{woman}\, x \wedge P\, x$.

Consider the translation of transitive verbs in MG on the example of the verb *loves*. MG translates *loves* as follows:[5]

$$\lambda T.\lambda x.T(\lambda y.\mathbf{love}\, x\, y) : ((e \to t) \to t) \to e \to t \tag{5}$$

In order to illustrate a way of using the semantic translations of the syntactic rules, let us compute the meaning of the sentence *Every man loves Mary* (1), whose MG derivation is depicted in Fig. 1. For that, we consider Montague's [21] translation of the syntactic rules **S2**, **S4**, **S5**, and **S14**, which we encounter in the derivation of the sentence (1). We read the derivation tree in Fig. 1 from the left to right and bottom to top perspective.

We have already discussed the semantic translation of *every man*, it remains to see how to translate the syntactic rules **S4**, **S5**, and **S14**.

[4]By convention, type constructors such as $\to$ associate to the right, that is $\alpha \to \beta \to \gamma$ means $\alpha \to (\beta \to \gamma)$. In Montague's notations $\alpha \to \beta$ is $\langle \alpha, \beta \rangle$.

[5]One may observe an asymmetry between the object and subject encodings in the translation (5): the variable T standing for the object of *loves* is of a higher order type,$(e \to t) \to t$, whilst the variable modeling a subject, x, is of type e. The asymmetry in the formula $\lambda T \lambda x.T(\lambda y.\mathbf{love}\, x\, y)$ is due to the syntactic analyzes that Montague makes use of and not because of his semantic approach.

As both **S4** and **S5** are rules for *application*, their semantic translations are very similar.

T4 If $\alpha \in P_{\mathrm{T}}$ (i.e. $\alpha \in P_{\mathrm{S/IV}}$) and $\beta \in P_{\mathrm{IV}}$, and their translations are $(f\,\alpha)$ and $(f\,\beta)$ then $F_4(\alpha, \beta)$ is translated as $(f\,\alpha)\,(f\,\beta)$;

T5 If $\alpha \in P_{\mathrm{IV/T}}$ and $\beta \in P_{\mathrm{T}}$, and their translations are $f(\alpha)$ and $f(\beta)$ then $F_5(\alpha, \beta)$ is translated as $f(\alpha)(f(\beta))$.

In words, the *application of a functor to its argument* at the syntax level translates into *functional application* (of lambda calculus)[6] of their corresponding terms at the semantic level.

Hence, one can translate he_0 *loves Mary*. By following the derivation shown in Fig. 1, first we translate *loves Mary*; then, the translation of he_0 will apply to the translation of *loves Mary*. Thus, we translate *loves Mary* as follows:

$$f(\textit{loves Mary}) = (\lambda T.\lambda x.T(\lambda y.\mathbf{love}\,x\,y))\,(\lambda P.P\,\mathbf{m}) \twoheadrightarrow_\beta \lambda x.\mathbf{love}\,x\,\mathbf{m} \quad (6)$$

Now, we apply $\lambda P.P\,x_n$ to $\lambda x.\mathbf{love}\,x\,\mathbf{m}$:

$$\lambda P.P\,x_n(\lambda x.\mathbf{love}\,x\,\mathbf{m}) \to_\beta (\lambda x.\mathbf{love}\,x\,\mathbf{m})\,x_n \;\to_\beta\; \mathbf{love}\,x_n\,\mathbf{m} \quad (7)$$

Let us look up the semantic counterpart of **S14**. It is as follows:

T14 If $\alpha \in P_{\mathrm{T}}$ and $\phi \in P_{\mathrm{S}}$, and their translations are $f(\alpha)$ and $f(\phi)$ then $F_{10,n}(\alpha, \phi) \in P_{\mathrm{S}}$ is translated as $f(\alpha)(\lambda x_n f(\phi))$

Note that in **T14** application $f(\alpha)$ to $(\lambda x_n f(\phi))$ is in concordance with the fact that $f(\alpha)$ is of type $(e \to t) \to t$, which is why it can only take an argument of type $e \to t$; however, since $f(\phi)$ is of type t (as type of ϕ is P_{S}, which translates to t), $f(\phi)$ cannot be an argument of $f(\alpha)$, but $\lambda x_n f(\phi)$ can since its type is $e \to t$ (as x_n is of type e).[7]

Thus, we translate the results of the last step of the derivation given in Fig. 1. The syntactic expression $F_{10,0}(\textit{every man}, he_0\ \textit{loves Mary})$ is translated as:

[6] We refer readers to [3] for more details about lambda calculus and its variants. We may write either $f(x)$ and $(f\ x)$ for denoting application of f to x.

[7] According to S14, $F_{10,n}(\alpha, \phi)$ is the result of substituting by α the first occurrence of he_n in ϕ, whereas the other occurrences of he_n in ϕ are substituted by he/she/it or him/her/it (depending on the gender of α). From a semantic point of view, it means that all the occurrences of he_n should be substituted by α, because those occurrences of he_n that are substituted by he/she/it or him/her/it are antecedents of the first occurrence of he_n, which is substituted by α.

$\lambda P.\forall x.\mathbf{man}(x) \supset P\,x$ applied to $\lambda x_0.\mathbf{love}\,x_0\,\mathbf{m}$, which is computed as follows:

$$(\lambda P.\forall x.\mathbf{man}\,x \supset P\,x)(\lambda x_0.\mathbf{love}\,x_0\,\mathbf{m}) \twoheadrightarrow_\beta \forall x.\mathbf{man}\,x \supset \mathbf{love}\,x\,\mathbf{m} \quad (8)$$

If we had *a woman* instead of *Mary*, the translation would be as follows:

$$\begin{aligned}&(\lambda P.\forall x.\mathbf{man}\,x \supset P\,x)(\lambda x_0.\exists y.\mathbf{woman}\,y \wedge \mathbf{love}\,x_0\,y)\\ \twoheadrightarrow_\beta \quad &\forall x.\mathbf{man}\,x \supset (\exists y.\mathbf{woman}\,y \wedge \mathbf{love}\,x\,y)\end{aligned} \quad (9)$$

The formula (9) in words means that *for every man there is a woman whom he loves*. On the other hand, the initial sentence could also mean that *there is a woman whom every man loves*. However, a formula that would give rise to the second interpretation cannot be obtained using the derivation we have used. In order to obtain the second reading, Montague [21] proposes a technique, known as *Montague's trick*.

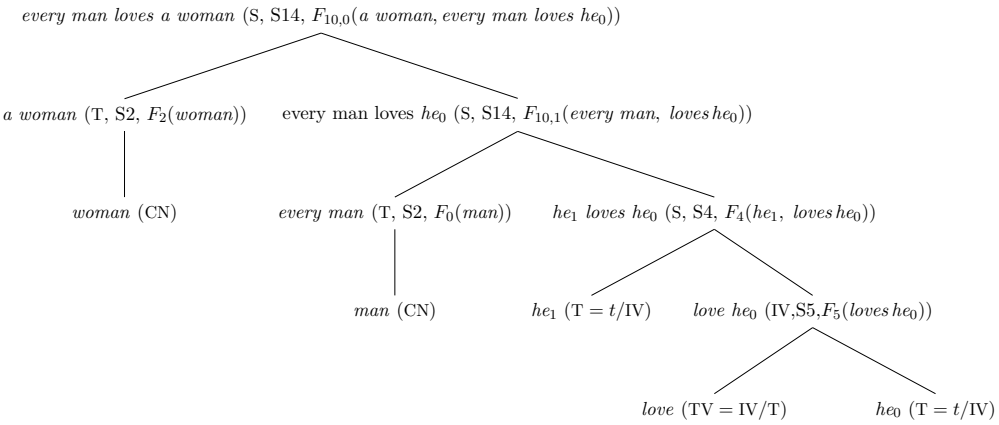


Figure 2: Montague's Trick: Deriving Second Reading

The idea behind of Montague's trick is to analyze a sentence in a way as it is shown in Fig. 2: The desired reading is obtained as a result of *reversing scopes*, i.e., by making *a woman* to scope over *every man loves somebody*.

We compute the formula corresponding to the final step of the derivation in Fig. 2 as follows:

$$\begin{aligned}&(\lambda Q.\exists y.\mathbf{woman}\,y \wedge Q\,y)(\lambda x_0.\lambda P.\forall x.\mathbf{man}\,x \supset \mathbf{love}\,x\,x_0) \twoheadrightarrow_\beta\\ &\twoheadrightarrow_\beta \exists y.\mathbf{woman}\,y \wedge \forall x.\mathbf{man}\,x \supset \mathbf{love}\,x\,y\end{aligned} \quad (10)$$

4 Abstract Categorial Grammars

Abstract Categorial Grammars (ACGs) by de Groote [9] are grammatical framework based on Curry's idea of having two levels of grammar [6],

pheno and tecto. At the pheno-grammatical level, natural language expressions are produced, whereas the tecto-grammatical one is responsible for production of those expressions with the help of rules that do not involve any information about surface forms. Another source of inspiration of ACGs comes from MG, namely the use of a typed lambda calculus and a compositional interpretation of syntax into semantics.

The following preliminary notions help to define Abstract Categorial Grammars (ACGs).

- A higher-order linear signature (HOS) is a triple $\Sigma = \langle A, C, \tau \rangle$ where:
  - A is a finite set of atomic types;
  - C is a finite set of constants;
  - $\tau : C \longrightarrow \mathcal{T}(A)$ is type assignment function mapping each constant from C to a linear implicative type built upon A.
- The order of a type ξ, denoted as $ord(\xi)$ is defined as:

$$ord(\xi) = \begin{cases} 1 \text{ if } \xi \text{ is atomic.} \\ max(1 + ord(\alpha), ord(\beta)) \text{ if } \xi = \alpha \to \beta \end{cases}$$

- Linear λ-terms $\Lambda^l(\Sigma)$over a HOS $\Sigma = \langle A, C, \tau \rangle$ is the set containing all and only elements defined as follows:
  - If $t_1, t_2 \in \Lambda^l(\Sigma)$ the $(t_1 t_2) \in \Lambda^l(\Sigma)$.
  - If $t \in \Lambda^l(\Sigma)$, then $\lambda x.t \in \Lambda^l(\Sigma)$, where x is a variable.
  - For any subterm $\lambda x.p$ of a term $t \in \Lambda^l(\Sigma)$, x is free in p.
  - for any subterm $t_1 t_2$ of $t \in \Lambda^l(\Sigma)$, t_1 and t_2 have no common free variable.

Definition 3. An ACG is a quadruple $G = (\Sigma_a, \Sigma_o, \mathcal{L}, S)$ where:

- $\Sigma_a = \langle A_a, C_a, \tau_a \rangle$ is a higher order signature, called the abstract signature;
- $\Sigma_o = \langle A_o, C_o, \tau_o \rangle$ is a higher order signature, called the object signature;
- $\mathcal{L}$ is a mapping from C_a to $\Lambda^l(\Sigma_o)$, called the lexicon of the grammar G, which is uniquely lifted to a homomorphism from $\Lambda^l(\Sigma_a)$ to $\Lambda^l(\Sigma_o)$ (which we denote again with $\mathcal{L}$) that has the following properties:
  - $\mathcal{L}(x) = x$ where x is a variable;

- $\mathcal{L}(t_1 t_2) = \mathcal{L}(t_1)\mathcal{L}(t_2)$;
- $\mathcal{L}(\lambda x.t) = \lambda x.\mathcal{L}(t)$.

- S is a type of Σ_a, called the distinguished type of G.

With the ACG $G = (\Sigma_a, \Sigma_o, \mathcal{L}, S)$, we associate two languages, defined as follows:

The *abstract language*: $\mathcal{A}(G) = \{u \in \Lambda(\Sigma_a) \mid \vdash_{\Sigma_o} u : s \quad \text{is derivable}\}$
The *object language*: $\mathcal{O}(G) = \{v \in \Lambda(\Sigma_o) \mid \exists u \in \mathcal{A}(G) : v = \mathcal{L}(u)\}$

In words, the object language is the image of the abstract language by the lexicon.

We call an ACG $G = (\Sigma_a, \Sigma_o, \mathcal{L}, S)$ of order n (or n-th order) if n is the maximum of orders of the types of the constants in the abstract signature Σ_a.

5 Montague Grammar as an ACG

We can build a HOS, Σ_1, to model the syntax used by Montague. More specifically, the terms over Σ_1 model the derivation trees. The derivation shown in Fig. 3 is modeled by the term u defined as follows:

$$u = \; C_{every}\, C_{man}\, (C_{loves}\, (C_a\, C_{woman})) : \mathrm{S}$$

We can view the term u as an instantiation of the Montague's rule **S14**, which allows us to produce the expression $F_{10,0}(\textit{every man}, \textit{he}_0 \textit{ loves Mary})$ of category S.

To be able to build (logical) semantic formulas, we construct another HOS, Σ_2, whose constants are shown in Fig. 4.

C_{every}, C_a, C_{some}	$: \mathrm{CN} \to \mathrm{IV} \to \mathrm{S}$
C_{mary}, C_{john}	$: \mathrm{IV} \to \mathrm{S}$
C_{walks}, C_{sleeps}	$: \mathrm{IV}$
C_{woman}, C_{man}	$: \mathrm{CN}$
C_{loves}, C_{meets}	$: (\mathrm{IV} \to \mathrm{S}) \to \mathrm{IV}$

Figure 3: Σ_1: Derivation Trees

$\mathbf{run}, \mathbf{woman}, \mathbf{man}$	$: e \to t$
$\mathbf{love}, \mathbf{meet}, \mathbf{read}$	$: e \to e \to t$
$\wedge, \supset$	$: t \to t \to t$
$\forall, \exists$	$: (e \to t) \to t$

Figure 4: Σ_2: Logical Alphabet

By mapping the term u under the lexicon f, we get:

$$\begin{aligned} f(u) = \; & f(C_{every})\, f(C_{man})\, (f(C_{loves})\, (f(C_a)\, f(C_{woman}))) \twoheadrightarrow_\beta \\ & \forall x.(\mathbf{man}\, x) \supset \exists y.(\mathbf{woman}\, y) \wedge (\mathbf{love}\, x\, y) \end{aligned} \tag{11}$$

$$\begin{array}{lll} \text{S} & := t & C_{every} := \lambda P.\lambda Q.\forall x.Px \supset Qx \\ \text{IV} & := e \to t & C_a, C_{some} := \lambda P.\lambda Q.\forall x.Px \wedge Qx \\ \text{CN} & := e \to t & C_{walks} := \mathbf{walk} \\ & & C_{woman} := \mathbf{woman} \\ & & C_{loves} := \lambda T.\lambda x.T(\lambda y.\mathbf{love}\, x\, y) \end{array}$$

Figure 5: f: mapping From Derivation Trees to Logical Formulas

Montague's trick Note that the current state of the grammar does not allow to obtain the second reading of the sentence (1). We can introduce new constants to build a term that could be mapped to the second reading. To mimic Montague's trick, we first combine the verb and the subject and the resultant term is then combined with the object. So, we extend the abstract signature with the constant $C^2_{loves} : (\text{IV} \to \text{S}) \to \text{IV}$ whose semantic interpretation is the following one:

$$f(C^2_{loves}) = \lambda S.\lambda x.S(\lambda y.\mathbf{love}\, y\, x)$$

The term u_{trick} is defined as follows:

$$u_{trick} = C_a\, C_{woman}\, (C^2_{loves}\, (C_{every}\, C_{man})) : \text{S}$$

The semantic interpretation of u_{trick} is as follows:

$$\begin{aligned} f(u) = f(C_a)\, f(C_{woman})\, (f(C^2_{loves})\, (f(C_{every})\, f(C_{man}))) \twoheadrightarrow_\beta \\ \exists x.(\mathbf{woman}\, x) \wedge \forall y.(\mathbf{man}\, y) \supset (\mathbf{love}\, y\, x) \end{aligned} \tag{12}$$

6 Montague Grammar in Natural Language Processing

Parsing for ACGs of order three is an NP-complete problem, as showed by Salvati [29, 28]. The grammar based on Montague's syntax, which we proposed in the previous section, is of third order.

For second order ACGs, parsing is of polynomial complexity as shown by Salvati [27] and Kanazawa [15]. Second order ACGs can encode a number of formalisms, including CFGs, Tree-Adjoining Grammars (TAGs) by Joshi [14] and Linear Context-Free Rewriting Systems by Weir [33].

6.1 The Syntax-Semantics interface: TAG for Syntax

TAG were found to be useful for modeling natural language phenomena that CFGs cannot, like certain kind of long-distance dependencies [32]. At

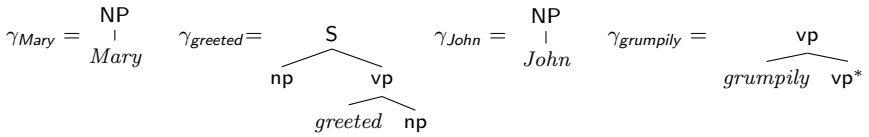
the same time, for TAGs, as it for CFGs, polynomial parsing algorithms are available. Subsequently TAGs found a number of applications across natural language processing problems.

6.2 Tree Adjoining Grammars

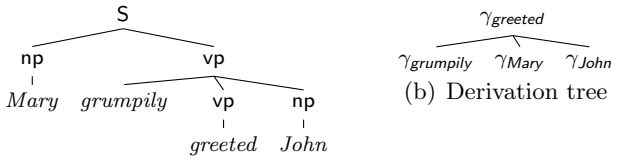
TAG is a tree generating formalism. The TAG tree language is generated by combining *elementary* trees. There are two kinds of elementary trees, *initial* and *auxiliary* ones. There are two ways of combining elementary trees, by *substitution* and by *adjunction*.

Substitution is a replacement of a frontier node of a tree with an initial tree that has the same root label as the given node.

Adjunction is like substitution, but in this case one can also substitute an internal node (i.e. a node which has children) of a tree with an auxiliary tree whose root node has the same label as the substituted node does. Since an internal node, call it n, of a tree (call this tree γ) has children (by definition), those children would be left orphan as a result of adjoining an auxiliary tree (call it β) on n. As a consequence, the tree structure would be lost. To avoid that, a TAG auxiliary tree β has a frontier node, marked with $*$, which has the same label as the root of β (and thus the same label as n). This frontier node, called the *foot* node of β, becomes mother to the children of the node n. For example, $\gamma_{greeted}$, γ_{John} and γ_{Mary} are initial trees, whereas $\gamma_{grumpily}$ is an auxiliary tree. Substituting γ_{John} and γ_{Mary} into $\gamma_{greeted}$ on the frontier labeled with np and adjoining $\gamma_{grumpily}$ into $\gamma_{greeted}$ on the node with label vp produces the derived tree shown in Figure 6(a).



The process of the production of the derived tree 6(a) is recorded by the corresponding *derivation* tree, which is represented as the tree 6(b).



(a) Derived tree (b) Derivation tree

Figure 6: *Mary grumpily greeted John*

6.3 TAG with Semantics as ACGs

While TAGs proved to be successful for encoding a number of syntactic phenomena, it was not clear how one could design a compositional semantic approach with TAGs when there is a mismatch between the derivational scope (parent-child relations in a derivation tree) and the semantic (logical) one.

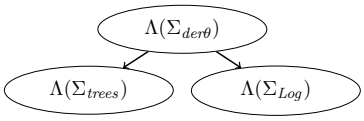


Figure 7: ACG architecture for TAG

In the ACG encoding of TAG of de Groote [10], TAG derivation trees are represented as the abstract language, whereas TAG derived trees as the object one. Pogodalla [25, 24] provided an encoding of TAG with Montagovian semantics. In Pogodalla's approach, TAG derivation trees are realized as an abstract language (like it is in [10]), whereas logical formulas are modeled as another object language. This architecture is pictorially represented in Fig. 7: there are the following signatures and lexicons involved: a signature $\Sigma_{der\theta}$, where we model TAG derivation trees; a signature Σ_{trees} where TAG derived trees are encoded; $\mathscr{L}_{\textit{d-ed trees}} : \Sigma_{der\theta} \longrightarrow \Sigma_{trees}$, it maps derivation trees to derived trees; Σ_{Log} where we define HOL terms encoding Montague semantics; $\mathscr{L}_{Log} : \Sigma_{der\theta} \longrightarrow \Sigma_{Log}$ maps derivation trees to Montague semantics.

$\Sigma_{der\theta}$: Its atomic types include S, vp, np, S_A, $\mathsf{vp}_A \ldots$ where the X types stand for the categories X of the nodes where a substitution can occur while the X_A types stand for the categories X of the nodes where an adjunction can occur. For each elementary tree $\gamma_{\textit{lex. entry}}$ it contains a constant $\mathrm{C}_{\textsf{lex. entry}}$ whose type is based on the adjunction and substitution (see Table 2). It additionally contains constants $I_X : X_A$ that are meant to provide a fake auxiliary tree on adjunction sites labeled with X where no adjunction actually takes place in a TAG derivation.

Σ_{trees}: Its unique atomic type is τ the type of trees. For any X of arity n belonging to the ranked alphabet describing the elementary trees of the TAG, Σ_{trees} has a constant $X_n : \overbrace{\tau \multimap \cdots \multimap \tau}^{n \text{ times}} \multimap \tau$

$\mathscr{L}_{\textit{d-ed trees}}(X_A) = \tau \multimap \tau$ and for any other type X, $\mathscr{L}_{\textit{d-ed trees}}(X_A) = \tau$. Table 2 illustrates the way $\mathscr{L}_{\textit{d-ed trees}}$ interprets constants of $\Sigma_{der\theta}$.

Constants of Σ_{Log} are shown in Table 3. We have two atomic types in Σ_{Log}, e for entities and t for propositions.

The lexicon $\mathscr{L}_{Log} : \Sigma_{der\theta} \longrightarrow \Sigma_{Log}$ is given in Table 4.

(2) Mary grumpily greeted John.

The term v models the TAG derivation tree on Fig. 6. By mapping v with $\mathscr{L}_{\textit{d-ed trees}}$, one obtains the term representation of the derived tree for (2); and by mapping v with $\mathscr{L}_{Log}$, one obtains Montague style HOL semantics of (2).

$$
\begin{aligned}
v &= \mathrm{C}_{\mathsf{greeted}}\ \mathrm{I}_{\mathsf{S}}\ (\mathrm{C}^{\mathsf{V}}_{\mathsf{grumpily}}\ \mathrm{I}_{\mathsf{S}})\ \mathrm{C}_{\mathsf{Mary}}\ \mathrm{C}_{\mathsf{John}} \\
\mathscr{L}_{\textit{d-ed trees}}(v) &= \mathsf{S}_2\ (\mathsf{np}_1\ \textit{Mary})\ (\mathsf{v}_2\ (\mathsf{v}_2\ \textit{grumpily}\ (\mathsf{v}_1\ \textit{greeted}))\ (\mathsf{np}_1\ \textit{John})) \\
\mathscr{L}_{Log}(v) &= \mathbf{grumpily}\,(\mathbf{greet}\,\mathbf{m}\,\mathbf{j})
\end{aligned}
$$

Abstract constants $\Sigma_{der\theta}$	Their images by $\mathscr{L}_{\textit{d-ed trees}}$	The corresponding TAG trees
$\mathrm{C}_{\mathsf{John}} : \mathsf{np}$	$c_{\textit{John}} \begin{array}{l} :\tau \\ = \mathsf{np}_1\ \textit{John} \end{array}$	$\gamma_{\textit{John}} =$ NP | *John*
$\mathrm{C}^{\mathsf{V}}_{\mathsf{grumpily}} : \mathsf{vp}_A \multimap \mathsf{vp}_A$	$c^{\mathsf{vp}}_{\textit{grumpily}} \begin{array}{l} :(\tau \multimap \tau) \multimap (\tau \multimap \tau) \\ = \lambda^{\mathrm{o}} adv_v\, x.adv_v\ (\mathsf{vp}_2\ \textit{grumpily}\ x) \end{array}$	$\gamma_{\textit{grumpily}} =$ vp: *grumpily* vp*
$\mathrm{C}_{\mathsf{greeted}} : \begin{array}{l} \mathsf{S}_A \multimap \mathsf{v}_A \multimap \\ \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S} \end{array}$	$c_{\textit{greeted}} = \begin{array}{l} : \begin{array}{l} (\tau \multimap \tau) \multimap (\tau \multimap \tau) \\ \multimap \tau \multimap \tau \multimap \tau \end{array} \\ \lambda^{\mathrm{o}} adv_s\, adv_v\, subj\, obj.\ adv_s \\ (\mathsf{S}_2\ subj\ (adv_v\ (\mathsf{vp}_2\ greeted\ obj))) \end{array}$	$\gamma_{\textit{greeted}} =$ S: np, vp: *greeted*, np
$\mathrm{I}_X : X_A$	$\lambda x.x : \tau \multimap \tau$	

Table 2: TAG with Semantics as ACGs: $\mathscr{L}_{\textit{d-ed trees}}$ lexicon

$\mathbf{j}, \mathbf{m}$	$: e$	**because**	$: t \to t \to t$
woman, **smart**, **work-out**	$: e \to t$	**greet**, **love**	$: e \to e \to t$
grumpily	$: t \to t$	**fast**	$: (e \to t) \to e \to t$
$\wedge$	$: t \to t \to t$	$\vee$	$: t \to t \to t$
$\Rightarrow$	$: t \to t \to t$	$\neg$	$: t \to t$
$\exists$	$: (e \to t) \to t$	$\forall$	$: (e \to t) \to t$

Table 3: Constants in the semantic vocabulary Σ_{Log}

6.4 Discourse Grammars as ACGs

Several authors have proposed grammatical approaches to discourse based on TAGs [12, 7]. Each of these grammars consist of two levels, a discourse-level one and a sentence (clause) level one. An ACG approach to discourse facilitated to overcome some problems that TAG based grammars experience when modeling an important discourse phenomenon of *clause-medial adverbials*. Consider the following discourse:

(3) Mary worked out. She *then* watched TV.

In (3), *then* is a discourse adverbial. At the discourse level, it has two places: two discourse units which it connects rhetorically (with a temporal relation of succession), which are *Mary worked out* and *Mary watched TV*. But at the sentence level, it operates on the verb phrase *watched TV*, which is its only argument. Thus, there is a mismatch between the discourse level and sentence level descriptions of discourse adverbials.

To distinguish a discourse meaning and a surface structure, we build two ACGs, one for studying structural properties of discourse and the other one to study discourse meaning. Let us sketch the constraint we are modeling: in discourse meaning, *then* is a two-place predicate whose arguments are discourse units, but in syntax, it is a verb-phrase modifier.

Let us first model the case when then is fronted: $d_{then} : \mathsf{DU} \rightarrow \mathsf{DU} \rightarrow \mathsf{DU}$, which encodes that a discourse connective takes two arguments that are pieces of discourses (in our modeling, they are terms of type DU); the resulting term also models a discourse (its type is again DU). We can interpret the constant $d_{then} : \mathsf{DU} \rightarrow \mathsf{DU} \rightarrow \mathsf{DU}$ into TAG derived trees as follows:

$$d_{then} := \lambda s_1.\, \lambda\, s_2.\mathsf{S}_3\ s_1\ \mathbf{dot}\ (\mathsf{S}_2\ (\mathit{Then},\ s_2)$$

In the case of a clause-medial connective *then*, we introduce a constant d^m_{then} typed as $d^m_{then} : \mathsf{DU} \rightarrow (\mathsf{S}_\mathsf{A} \rightarrow \mathsf{DU}) \rightarrow \mathsf{DU}$. It indeed models the fact that the one of its arguments needs an adjunction in order to become a discourse unit. Then we can easily interpret it as TAG derived trees. However note that while theoretically it is perfectly possible to have d^m_{then} defined as it is now, it makes the ACG we are building of order three (and thus polynomial parsing cannot be guaranteed).

As it was shown by Danlos et al. [8], it is possible to build a second order ACG that would be still able to express the needed constraint. Let us therefore introduce a new type DU_m, which we use in order to model clause-medial connectives. In this new approach, the type of d^m_{then} is $\mathsf{DU} \rightarrow$

Constants of $\Sigma_{der\theta}$	Their interpretations by $\mathscr{L}_{Log}$
$\mathrm{C}_{\mathsf{woman}} : \mathsf{n}_\mathsf{A} \multimap \mathsf{np}$	$\lambda D.\lambda q\,.D\ \mathbf{woman}\ q$
$\mathrm{C}_{\mathsf{Mary}} : \mathsf{np}$	$\lambda D.D\ \mathbf{m}$
$\mathrm{C}_{\mathsf{grumpily}} : \mathsf{V}_\mathsf{A} \multimap \mathsf{V}_\mathsf{A}$	$\lambda a_v.\lambda r.\ a_v\ (\lambda x.\mathbf{grumpily}(r\,x))$
$\mathrm{C}_{\mathsf{greeted}} : \mathsf{S}_\mathsf{A} \multimap \mathsf{V}_\mathsf{A} \multimap \mathsf{np} \multimap \mathsf{np} \multimap \mathsf{S}$	$\lambda\, s\, a\, S\, O.s(S(a(\lambda x.O(\lambda y.(\mathbf{greet}\ x\ y)))))$
$\mathrm{C}_{\mathsf{every}}, \mathrm{C}_{\mathsf{each}} : \mathsf{n}_\mathsf{A}$	$\lambda\, P.\, \lambda Q\,.\, \forall\, x.\, (P\, x)\ \supset\ (Q\, x)$
$\mathrm{C}_{\mathsf{a}}, \mathrm{C}_{\mathsf{some}} : \mathsf{n}_\mathsf{A}$	$\lambda\, P.\, \lambda Q\,.\, \exists\, x.\, (P\, x)\ \wedge\ (Q\, x)$

Table 4: Interpretations by $\mathscr{L}_{Log}$

$\mathsf{DU}_m \rightarrow \mathsf{DU}$. In order to interpret it in TAG derived trees, we first interpret it in TAG derivation trees. The constraint that the constant $d^m_{then} : \mathsf{DU} \rightarrow \mathsf{DU}_m \rightarrow \mathsf{DU}$ models can be expressed in TAG derivation trees as follows:

$$d^m_{then} := \lambda s_1.\lambda s_2.\ C_+\ s_1\ (s_2\ C^{\mathsf{vp}}_{then})$$

Indeed, we expressed that the second argument of d^m_{then} must receive an adjunction that would place *then* in its VP (a clause-medial position). (The constant C_+ stands for a concatenation: it can be interpreted in TAG derived trees as an initial tree with two substitution sites separated by a dot.) It means that we have to allow adjunction on the second argument; the first argument, however, does not need it. To achieve that, we interpret types DU and DU_m as follows:

$$\begin{array}{ll} \mathsf{DU} & := \mathsf{S} \\ \mathsf{DU}_m & := \mathsf{vp}_A \rightarrow \mathsf{S} \end{array}$$

Since we have already built the lexicon interpreting TAG derivation trees into TAG derived trees, we can compose that lexicon with the above defined interpretations to obtain interpretation of discourse into TAG derived trees.

On the meaning side of the discourse, d^m_{then} and d_{then} behave in the same way: both model the *succession* rhetorical relation. We therefore interpret them as follows: $d^m_{then}, d_{then} := \textsc{succession} : t \rightarrow t \rightarrow t$. As one can see, we interpret types DU and DU_m, both as t.

7 Discussion and Summary

There are a number of insights that Montague's works offer to the studies in semantics and in particular to the studies of the syntax-semantics interface, such as, for instance, use of typed lamda calculus and higher-order interpretations of noun phrases. Another important point in MG is its uniform treatment of intransitive verbs, common nouns and adjectives. As we already mentioned above, all of them are treated as one-place predicates of type $e \rightarrow t$. While this approach to intransitive verbs, common nouns and adjectives has found a number of uses, there are other ways of their modeling which were offered within other theories. Ranta [26] and Luo [17, 18] propose grammars within frameworks of dependent types. In their grammars, common nouns are not treated as predicates but types. That is, *human*, *book*, *tree* etc. are types. The verbs are predicates. For instance, *talk* is of type *human* $\rightarrow t$. This makes sure that one would not have *a tree talks*, because *a tree* is not of type *human*. Pkhakadze [23] argues in favor of distinguishing between nouns, adjectives and verbs. In

his approach, verbs are predicates. A noun is either a set, or a constant ranging on the set defined by that noun.

Meaning in natural language claimed attention of various communities, including linguistics, mathematics (logics, computer science), philosophy, and psychology, and the syntax-semantics interface is only one way of studying it, treating syntax as a pivot to semantics. In the studies focused on the syntax-semantics interface though, the main question is still whether a syntactic analysis is sufficient to obtain a semantic one, or to put it another way, is syntax a pure pivot for semantics, or syntax is also shaped by semantics? After all, we, speakers, use natural language to create and convey meanings, which allows us to think and communicate by means of a language. In MG we already see an emergence of some semantics inspired syntactic rules. (Montague's trick shows this spirit as it offers a new syntactic analysis in order to obtain one of the possible readings.) A later developments of MG, such as ACGs, have been making use of semantics inspired syntax in order to model various phenomena in studying form-meaning relations.

References

1. Ajdukiewicz, K. Die syntaktische Konnexität. *Stud. Philos. 1* (1935), 1–27.

2. Bar-Hillel, Y. A quasi-arithmetical notation for syntactic description. *Language 29*, 1 (1 1953), 47–58.

3. Barendregt, H. P. Lambda calculi with types. In *Handbook of Logic in Computer Science (Vol. 2)*, S. Abramsky, D. M. Gabbay, and S. E. Maibaum, Eds. Oxford University Press, Inc., New York, NY, USA, 1992, pp. 117–309.

4. Bloomfield, L. *Language*. University of Chicago Press, Chicago, 1933.

5. Chomsky, N. On certain formal properties of grammars. *Information and Control 2*, 2 (1959), 137 – 167.

6. Curry, H. B. Some logical aspects of grammatical structure. *Journal of Symbolic Logic 25*, 4 (1960), 341–341.

7. Danlos, L. D-STAG : un formalisme d'analyse automatique de discours basé sur les TAG synchrones. *Revue TAL 50*, 1 (2009), 111–143.

8. Danlos, L., Maskharashvili, A., and Pogodalla, S. Interfacing sentential and discourse TAG-based grammars. In *Proceedings of the 12th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+12)* (Düsseldorf, Germany, June 2016), pp. 27–37.

9. de Groote, P. Towards abstract categorial grammars. In *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter* (Toulouse, France, July 2001), pp. 148–155. Colloque avec actes et comité de lecture. internationale.

10. de Groote, P. Tree-Adjoining Grammars as Abstract Categorial Grammars. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6)* (2002), Università di Venezia, pp. 145–150.

11. Dowty, D. R., Wall, R. E., and Peters, S. *Introduction to Montague Semantics*. Reidel, Dordrecht, 1981.

12. Forbes, K., Miltsakaki, E., Prasad, R., Sarkar, A., Joshi, A. K., and Webber, B. L. D-LTAG system: Discourse parsing with a Lexicalized Tree-Adjoining Grammar. *Journal of Logic, Language and Information 12*, 3 (2003), 261–279. Special Issue: Discourse and Information Structure.

13. Gamut, L. *Logic, Language, and Meaning: Intensional logic and logical grammar*. Logic, Language, and Meaning. University of Chicago Press, 1991.

14. Joshi, A. K., and Schabes, Y. Tree-adjoining grammars. In *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa, Eds. Springer Berlin Heidelberg, 1997, pp. 69–123.

15. Kanazawa, M. Parsing and generation as datalog queries. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL)* (Prague, Czech Republic, June 2007), Association for Computational Linguistics, pp. 176–183.

16. Lambek, J. The mathematics of sentence structure. *Americal Mathematical Monthly 65* (1958), 154–170.

17. Luo, Z. Common nouns as types. In *Logical Aspects of Computational Linguistics* (Berlin, Heidelberg, 2012), D. Béchet and A. Dikovsky, Eds., Springer Berlin Heidelberg, pp. 173–185.

18. Luo, Z. Formal semantics in modern type theories with coercive subtyping. *Linguistics and Philosophy 35*, 6 (Nov 2012), 491–513.

19. Montague, R. English as a formal language. In *Linguaggi Nella Società e Nella Tecnica*, B. Visentini, Ed. Edizioni di Communita, 1970, pp. 188–221.

20. Montague, R. Universal grammar. *Theoria 36*, 3 (1970), 373–398.

21. Montague, R. The proper treatment of quantification in ordinary English. In *Formal Philosophy: Selected Papers of Richard Montague*, R. Thomason, Ed. Yale University Press, New Haven, CT, 1973, pp. 247–270.

22. Moot, R., and Retoré, C. *The Logic of Categorial Grammars: A Deductive Account of Natural Language Syntax and Semantics*. FoLLI-LNCS. Springer, July 2012.

23. Pkhakadze, K. About logical declination and lingual relations in georgian. *Georgian language and logic 1*, 1 (2005), 19–77.

24. Pogodalla, S. Advances in Abstract Categorial Grammars: Language Theory and Linguistic Modeling. ESSLLI 2009 Lecture Notes, Part II. ESSLLI 2009 Lecture Notes, 2009.

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

26. Ranta, A. *Type-theoretical Grammar*. Indices (Claredon). Clarendon Press, 1994.

27. Salvati, S. *Problèmes de filtrage et problèmes d'analyse pour les grammaires catógorielles abstraites*. PhD thesis, Institut National Polytechnique de Lorraine, 2005.

28. Salvati, S. On the complexity of Abstract Categorial Grammars. In *10th conference on Mathematics of Language (MOL 10), Los Angeles, CA* (July 2007), M. Kracht, G. Penn, and E. Stabler, Eds.

29. Salvati, S. A note on the complexity of abstract categorial grammars. In *The Mathematics of Language* (Berlin, Heidelberg, 2010), C. Ebert, G. Jäger, and J. Michaelis, Eds., Springer Berlin Heidelberg, pp. 266–271.

30. Shaumyan, S. *Structural Linguistics*. Nauka, 1965.

31. Shaumyan, S. *A semiotic theory of language*. Advances in semiotics. Indiana University Press, 1987.

32. Shieber, S. M. Evidence against the context-freeness of natural language. *Linguistics and Philosophy 8*, 3 (1985), 333–343.

33. Weir, D. J. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA, 1988. Supervisor: Aravind K. Joshi.