Simulating Feature Structures with Simple Types

Valentin D. Richard
LORIA, UMR 7503
Université de Lorraine, CNRS, Inria
54000 Nancy, France
valentin.richard@loria.fr

Abstract

Feature structures have been several times considered to enrich categorial grammars in order to build fine-grained grammars. Most attempts to unify both frameworks either model categorial types as feature structures or add feature structures on top of categorial types. We pursue a different approach: using feature structure as categorial atomic types. In this article, we present a procedure to create, from a simplified HPSG grammar, an equivalent abstract categorial grammar (ACG). We represent a feature structure by the enumeration of its totally well-typed upper bounds, so that unification can be simulated as intersection. We implement this idea as a meta-ACG preprocessor[1].

1 Introduction

Feature structures (FSs) (Carpenter, 1992) have been widely used to represent natural language syntax, particularly by HPSGs (*Head-driven Phrase Structure Grammars*, (Pollard and Sag, 1987, 1994)).

In the original ideas of categorial grammars (Ajdukiewicz, 1935; Bar-Hillel, 1953; Lambek, 1958), only a few number of atomic categories are taken, and complex categories are built on them as simple types. This approach makes it less flexible to capture fine-grained morpho-syntactic phenomena (e.g. agreement or case). Grammatical systems combining categorial and feature approaches have been developed, aiming at recovering these fine structures and grammatical interactions, but also allowing a better lexicon organization (e.g. hierarchy inheritance) (Moortgat, 1997).

According to Moortgat (1997), first generation hybrid systems (Zeevat, 1988; Bouma, 1988; Uszkoreit, 1986) encode categorial logic in feature logic.

By contrast, second generation hybrid systems (Dörre et al., 1996; Dörre and Manandhar, 1995) preserve the categorial inferential system by adding a layer of feature structures to categorial type atoms.[2]

While the general framework of feature logic may suffer from Turing-completeness when regarding time complexity of parsing (Carpenter, 1991), second generation hybrids bypass this issue by restricting feature structure power to subtyping (Buszkowski, 1988). However, this restriction forbids the latter to exploit structure-sharing (i.e. reentrancy).

More recent systems fall in either generation. Unification-based General Categorial grammars (Villavicencio, 2002; Baldridge, 2002) encode Combinatory Categorial Grammars (Steedman, 1988) as feature structures using asymmetric default unification. Extensions of Abstract Categorial Grammars (de Groote, 2001) to dependent product, variant types and records model feature logic inside type theory (de Groote and Maarek, 2007). However, these extensions make it undecidable (de Groote et al., 2007).

In this article, we advocate for a different, yet intuitive combination of categorial logic and feature logic: representing feature structures as atomic categorial types with no additional operation. Unification is not implemented, but simulated by set intersection. This proposal is based on two ideas:

1. Restrictions on appropriateness allows us to enumerate a representative set of any FS
2. The labor is divided into a preprocessor, handling FS combinatorics, and the grammar engine, performing categorial operations

This framework resembles second generation systems, because it creates a layer between feature

[1] Source code is available at https://doi.org/10.12763/VWKNSA

[2] Steedman (1990) and Muskens (2001) could also be put in the second generation. Moreover, we could mention Kraak (1995), who models FSs via modalities (Moortgat, 1996).

Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Student Research Workshop, pages 113 - 122

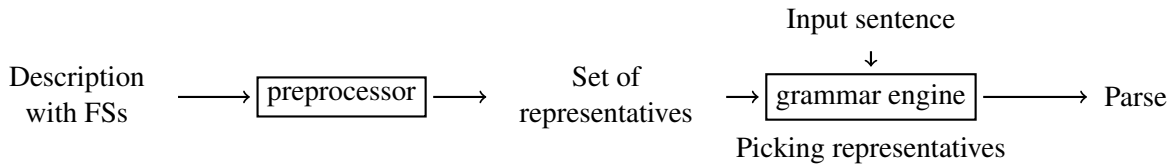


Figure 1: Division of labor between the preprocessor and the grammar engine

logic and categorial logic. However, there is no need to resort to unification, and it can deal with structure-sharing. Although it does not provide a different grammatical system, this solution has the advantage to be easier to implement.

We focus here on Abstract Categorial Grammars (ACGs). We present a first implementation of the preprocessor, called meta-ACG preprocessor. As feature structure are not yet implemented in ACGtk (Pogodalla, 2016), this program brings the possibility to work with ACGs and FSs. We also mention how it reduces labor when defining a grammar.

The motivation of this work is thus twofold:

1. Formalize a way to work with features structures and categorial logic, in particular ACGs
2. Improve ACGtk to be able to define feature structures and reduce some grammar design labor

In section 2, we present our system and its formal proof of work. We exemplify it by exhibiting a transformation from simplified HPSG grammars into ACG grammars. In section 3, we present the meta-ACG preprocessor.

2 Simulating feature structures

2.1 Feature structures as atoms

The idea of adding refinements of categorial atomic types goes back to Lambek (1958). He distinguishes third-person singular nouns n from third-person plural nouns n^*, and the verb *work* has two possible types: $n\backslash s$ and $n^*\backslash s$.

In systems where unification is not taken as granted, using FSs as atoms is a cheap solution: e.g. PP_{to} vs. PP_{about} in (Morrill et al., 2011), $NP\_NUM{=}PL$ in (Maršík, 2013), and npe (existentially quantified *np*) vs. npu (universally quantified) in (Amblard et al., 2021).

This technique relies on the grammar engine to select the right featured type when parsing. Therefore, no unification system has to be added. However, the main drawback is the combinatorial explosion due to the many possible values the attributes can take. For example, writing a grammar including all possible rules for *NP-VP* agreement would not only be long, but it also increases the risks of making typos. Maršík (2013) suggests to use meta-variables to, at least, present these rules more compactly.

We advocate for a more generic solution: automatizing the process of generation of constants and rules with FSs as atoms. For example, from a given description

$$\texttt{np}[\texttt{AGR} = \texttt{x}] \to \texttt{vp}[\texttt{AGR} = \texttt{x}] \to \texttt{s}$$

we would like to generate

$$\begin{array}{l} \texttt{np}[\texttt{AGR} = [1, \texttt{sg}]] \to \texttt{vp}[\texttt{AGR} = [1, \texttt{sg}]] \to \texttt{s} \\ \texttt{np}[\texttt{AGR} = [1, \texttt{pl}]] \to \texttt{vp}[\texttt{AGR} = [1, \texttt{pl}]] \to \texttt{s} \\ \texttt{np}[\texttt{AGR} = [2, \texttt{sg}]] \to \texttt{vp}[\texttt{AGR} = [2, \texttt{sg}]] \to \texttt{s} \\ \vdots \end{array} \tag{1}$$

where `np[AGR=[1,sg]]`,... are taken as atomic types.

The system we introduce works as depicted in Fig. 1. Given a set of descriptions, the preprocessor generates a set of representatives (like in (1)) out of any (underspecified) input FS. Then, the grammar engine can pick in this set when trying to parse a sentence.

In part 2.2 we define the set selected representatives are based on. Part 2.3 introduces ranked appropriateness, the hypothesis enabling this set to be enumerable. Finally, we present the transformation of simplified HPSG grammars into ACG grammars in part 2.4.

2.2 Set of representatives

We begin with some semi-formal reminders about feature structures.

Set $\langle \mathsf{T}, \sqsubseteq \rangle$ an inheritance hierarchy[3], and Att a finite set of attributes. By $\tau \sqsubseteq \sigma$, we mean that type τ is more general than type σ.

[3]Complementary formal definitions can be found in appendix B.

Type	Rank	Specification	Description
j-τ-*list*	0		list of at most j elements ($j \geq 0$)
j-τ-*ne-list* ($j \geq 1$)	$r(\tau)+1$	HEAD : τ TAIL : $(j-1)$-τ-*list*	list of length between 1 and j

Table 1: Data structure simulating lists of at most m elements of type τ. 0-τ-*list* is the empty list (aka. *e-list*) The inheritance hierarchy is given in Fig. 2.

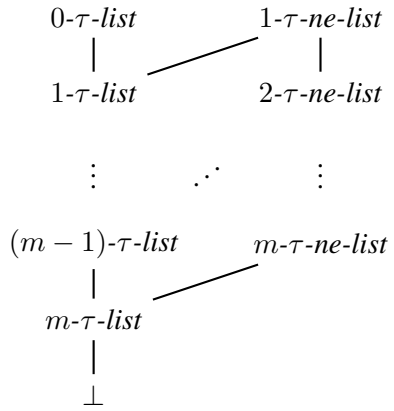
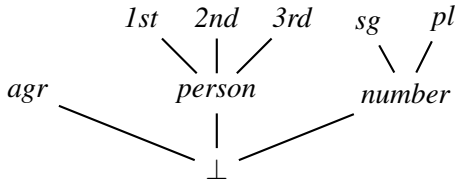


Figure 2: Inheritance hierarchy of types simulating lists of at most m elements of type τ. The appropriateness specification and ranks are given in Tab. 1.

Let us illustrate this here with *NP-VP* agreement, using $\mathsf{Att} = \{\text{P}, \text{N}\}$ and the following inheritance hierarchy:



More general types are placed here at the bottom, e.g. *person* $\sqsubseteq$ *1st*. The most general type (i.e. the minimum) is $\bot$.

A feature structure (FS) is a pair of a type and a list of features. A feature is a pair of an attribute and a feature structure. We usually represent FSs as attribute-value matrices, like in (2). Subsumption $\sqsubseteq$ can be extended to FSs. The unification of two FSs F and G is the most general FS $F \sqcup G$ which is subsumed by F and G, if it exists. We only consider well-typed feature structures, i.e. having restrictions on the values a feature can take. These restrictions are expressed via an appropriateness specification.

By $X \rightharpoonup Y$ we denote the set of partial functions f from X to Y, and we write $f(x)\!\downarrow$ if $x \in \mathsf{dom}\, f$, i.e. if x belongs to the definition domain of f.

Definition 1 (Appropriateness specification (Carpenter, 1992))**.** *An appropriateness specification is partial function* $\mathsf{Approp} : \mathsf{Att} \times \mathsf{T} \rightharpoonup \mathsf{T}$ *such that*

Feature introduction: *For every* $\text{A} \in \mathsf{Att}$*, there exists* $\mathsf{Intro}(\text{A}) \in \mathsf{T}$ *s.t.* $\mathsf{Approp}(\text{A}, \mathsf{Intro}(\text{A}))\!\downarrow$

Monotonicity: *If* $\mathsf{Approp}(\text{A}, \sigma)\!\downarrow$ *and* $\sigma \sqsubseteq \tau$*, then* $\mathsf{Approp}(\text{A}, \tau)\!\downarrow$ *and* $\mathsf{Approp}(\text{A}, \sigma) \sqsubseteq \mathsf{Approp}(\text{A}, \tau)$

$\mathsf{Approp}(\text{A}, \tau) = \sigma$ means that a FS of type τ can have attribute A valued by a FS of type σ or more specific. The following notion of totally well-typed FSs allows us to talk about completely specified FSs.

Definition 2. *A feature structure is totally well-typed when all its appropriate attributes are valued.*

The appropriateness specification of our example is P : *person*, N : *number* for type *agr* (i.e. $\mathsf{Approp}(\text{P}, agr) = person$ and $\mathsf{Approp}(\text{N}, agr) = number$, and undefined elsewhere). For instance, both FSs below are well-typed, but only the one on the right is totally well-typed.

$$\begin{bmatrix} agr & \\ \text{P} & 1st \end{bmatrix} \qquad \begin{bmatrix} agr & \\ \text{P} & 1st \\ \text{N} & number \end{bmatrix} \tag{2}$$

First-order terms can be represented by their sets of subsumed ground terms. Similarly we could take, to represent a potentially underspecified FS in ACGs, its maximal (resp. or grounded) upper bounds. However, (Carpenter, 1992) points out that this fails because some feature structures can have the same set of maximal (resp. grounded) upper bounds, but still be different.

To solve this issue, we use totally well-typed (non-necessarily sort-resolved, grounded or maximal) upper bounds of a FS F to define the representative set of F.

Definition 3 (Totally well-typed upper-set)**.** *We call* $\mathcal{U}(F)$ *the set of totally well-typed upper bounds of* F*.*

This enables us to characterize unification as set intersection.

Proposition 1. *$F \sqcup G$ exists iff $\mathcal{U}(F) \cap \mathcal{U}(G) \neq \emptyset$, and in this case $\mathcal{U}(F \sqcup G) = \mathcal{U}(F) \cap \mathcal{U}(G)$.*

Proofs are given in appendix B.

2.3 Finite generation

We plan to model a feature structure F by adding a kind of copy of $\mathcal{U}(F)$ to an ACG grammar. The set $\mathcal{U}(F)$ has then to be finite. Therefore, we need FSs to be acyclic. Moreover, there must be no appropriateness (subsuming) loop, i.e. no type τ and path $\mathrm{w} \in \mathsf{Att}^*$ such that $\mathsf{Approp}(\mathrm{w}, \tau) \sqsubseteq \tau$. To enforce this, we require types to be ranked.

Definition 4. *Specification* Approp *is ranked if there exists a function $r : \mathsf{T} \to \mathbb{N}$ such that, for all $\tau \in \mathsf{T}$,*

1. *for all σ, if $\tau \sqsubseteq \sigma$ then $r(\tau) \leq r(\sigma)$*
2. *for all $\mathrm{A} \in \mathsf{Att}$ and σ, if $\mathsf{Approp}(\mathrm{A}, \tau) \sqsubseteq \sigma$, then $r(\tau) > r(\sigma)$*

$r(\tau)$ is the rank of τ.

Ranked appropriateness specifications allow us to proceed by induction on the set of well-typed feature structures.

Proposition 2. *If* Approp *is ranked, then the set of well-typed FSs is finite.*

A proof is given in appendix B.3.

Ranking restricts the expressive power of feature structures. However, we can still create a data structure resembling finite lists. Set τ a type and m an positive integer. We define τ-lists of at most m elements as in Tab. 1 and Fig. 2.

Ranking forbids potentially infinite elements, like lists of arbitrary length. This limit is actually not so restrictive because, supposing there is a reasonable maximal number of words a sentence can have, we could always resort to lists of a predefined maximal length.

2.4 Simple HPSG into ACG

The goal of this part is to illustrate our approach on a selected pair of language grammar formalisms based on feature structures and categorial types respectively.

We want to code a HPSG grammar $\mathcal{G}$ in an ACG grammar $\mathsf{ACG}(\mathcal{G})$. We focus on simple HPSG characteristics, following a context-free backbone. For simplicity, we do not take headedness and lexical rules into account. We also assume that the appropriateness specification of $\mathcal{G}$ is ranked (except for DTRS and PHON).

$$\frac{}{w \vdash \boxed{a} : \boxed{0}}\ (*)_{\boxed{a}}$$

$$\frac{u_1 \vdash \boxed{1'} : s_1 \quad \ldots \quad u_n \vdash \boxed{n'} : s_n}{u_1 \ldots u_n \vdash \boxed{c} : \boxed{0}}\ (**)_{\boxed{b}}$$

if there exists $\boxed{c} = \boxed{b} \sqcup \left[\text{DTRS} \quad \left\langle \boxed{1'}, \ldots, \boxed{n'} \right\rangle\right]$

for all $\boxed{a}, \boxed{b}$ in the grammar

Figure 3: Simplified HPSG deduction system

$$\frac{\mathsf{c}_F \in \mathcal{R}(\boxed{a})}{w \vdash \mathsf{c}_F : \boxed{0}}\ (*)_{\boxed{a},F}$$

$$\frac{u_1 \vdash M_1 : s_1 \quad \ldots \quad u_n \vdash M_n : s_n}{u_1 \ldots u_n \vdash \mathsf{c}_F\ M_1\ \ldots\ M_n : \boxed{0}}\ (**)_{\boxed{b},F}$$

if $\mathsf{c}_F \in \mathcal{R}(\boxed{b})$ and $\mathsf{c}_F\ M_1\ \ldots\ M_n : \mathsf{t}_{F_0}$ is well-typed

for all $\boxed{a}, \boxed{b}$ in the grammar and FS F

Figure 4: Image ACG deduction system. $\mathsf{c}_F\ M_1\ \ldots\ M_n$ is λ-application.

We assume lexical items and phrases are of the form $(*)$ and $(**)$.

$$\boxed{a} \begin{bmatrix} \textit{word} & \\ \text{PHON} & w \\ \text{SYNSEM} & \boxed{0} \end{bmatrix} \qquad (*)$$

$$\boxed{b} \left[\begin{array}{ll} \textit{phrase} & \\ \text{PHON} & u_1 \ldots u_n \\ \text{SYNSEM} & \boxed{0} \\ \text{DTRS} & \left\langle \boxed{1}\left[\text{PHON}\ u_1\right], \ldots, \boxed{n}\left[\text{PHON}\ u_n\right] \right\rangle \end{array}\right] \qquad (**)$$

Feature structures of type *word* $(*)$ are lexical units. Attribute PHON specifies the phonological realization (here the spelling), and SYSSEM the syntactic and semantics properties.

Feature structures of type *phrase* $(**)$ represent phrases with contiguous daughters (DTRS) $\boxed{1}$,..., $\boxed{n}$. The concatenation of the phonological realizations of the daughters make up the PHON of the phrase. The syntactic and semantics properties of

the phrase also depend on the ones of the daughters via structure sharing (i.e. reentrancy).

See appendix A for instance examples.

The constraints on HPSG parsing can be rephrased as the deduction system in Fig. 3 (using the notation of (∗) and (∗∗)).

We translate this system into the ACG deduction system in Fig. 4, using the representative sets defined in def. 5. Phrase FSs are represented by a set of second-order typed constants.

Definition 5. *Given a word* $\boxed{a}$ *as in* (∗) *or a phrase* $\boxed{b}$ *as in* (∗∗)*, its set of representatives is defined by induction on its rank, as the set of ACG typed constants:*

$$\begin{aligned}\mathcal{R}(\boxed{a}) = &\ \{c_F : t_F \mid F \in \mathcal{U}(\boxed{a})\}\\ \mathcal{R}(\boxed{b}) = &\ \{c_F : t_{F_1} \to \ldots \to t_{F_n} \to t_{F_0} \mid\\ &\ F \in \mathcal{U}(\boxed{b}) \textit{ consistent with}\\ &\ F_i \in \mathcal{U}(\boxed{i}) \textit{ for all } 0 \leq i \leq n\}\end{aligned} \tag{3}$$

using the same $\boxed{i}$*'s as in* (∗∗).

Fig. 4 presents an ACG grammar in the style of λ-grammars (Muskens, 2001). We give in appendix C an alternative presentation of this grammar using the format used by de Groote (de Groote, 2001).

Proposition 3. *$\mathcal{G}$ and* $\mathsf{ACG}(\mathcal{G})$ *have the same string language.*

A proof is given in appendix B.4. A derivation instance is displayed in appendix A.

A sample HPSG grammar modeling simple English questions in the meta-ACG language is provided in the `example` folder of the enclosed program.

3 Implementation

3.1 Meta-ACG preprocessor

ACGtk (Pogodalla, 2016) is a toolkit offering an environment to develop and test ACG grammars. Feature structures have not been implemented yet in this program.

We implement the preprocessor presented in part 2.1 as a python program called `macg`. Given an input file written in a specially designed language, called meta-ACG language, this program generates an ACG grammar. This output consists in tree files: deep syntax signature, surface syntax signature and surface lexicon (see definition 11).

The syntax of the meta-ACG language is greatly inspired by NLTK (Bird et al., 2009), except that variables are declared with `@`. See Fig. 5 for an example minimal code.

```
Type:  person < 1st, 2nd, 3rd
Type:  number < sg, pl
Type:  tense  < prst, past
Type:  agr
  P : person
  N : number
Type:  np
  AGR : agr      # agreement
  PRO : bool     # pronominal
Type:  vp
  AGR : agr
  T : tense
Type:  s
  T : tense
Constant:  Proper nouns
  Ash :  np[agr[3rd,sg],-PRO]
Constant:  Intransitive verbs
  sleeps :  vp[agr[3rd,sg],prst]
  slept  :  vp[past]
Rule:  Clause
  np[AGR=@a] -> vp[AGR=@a,T=@t] \
    -> s[T=@t]
```

Figure 5: Sample code in the meta-ACG language, exemplifying NP – VP agreement. Italics is put on comments. Boldface identifies control keywords. `bool` is the predefined type of booleans.

The meta-ACG preprocessor has two main goals:

1. Making it possible to develop and test ACG grammars with feature structures
2. Reducing the redundancy of ACGtk grammar design

Goal 1 is obtained through an iterator able to generate all unfolded totally well-typed upper bounds of a feature structure description. These upper bounds are written as distinct atomic types in the output files. For example, constant `slept` of Fig. 5 yields $4 \times 3 = 12$ deep syntax constant:

```
SLEPT_person_number_past  :  vp_person_number_past
SLEPT_person_sg_past      :  vp_person_sg_past
SLEPT_person_pl_past      :  vp_person_pl_past
SLEPT_1st_number_past     :  vp_1st_number_past
                          :
SLEPT_3rd_pl_past         :  vp_3rd_pl_past
```

Similarly, rules are mapped to deep syntax constants of empty surface realization for every possible variable assignment. For example, the clause rule of Fig. 5 generates $(4 \times 3) \times 3 \times 3 = 108$ constants (i.e. every person, number, time, and pronominality type).

The ranking condition is ensured by the order in which the types and their appropriateness specifications are declared.

Goal 2 is obtained by two means. As a script language, the meta-ACG language aims at being light. The main contribution, however, revolves on the way ACG conventions are coded in the preprocessor. Even if ACGtk is able to handle a large variety of ACG grammars, most actually written test grammars follow the same pattern and code norms:

- a deep syntax constant in uppercase is mapped to its surface representation in lowercase
- the order in which the source types are declared is the same as the surface order of the respective arguments

This way, taking these conventions as default helps gain some time at the grammar design phase.

3.2 Limitations and future prospects

The `macg` program is still under development. We intend to add morphological rules and macros to facilitate even more the lexicon organization. Inequalities, default values and constraint equations could also be added in the future.

Although Tab. 1 gives an implementation of lists in our setting, the current meta-ACG language lacks primitives, like concatenation, to work with lists more easily. Technically, list concatenation can be written down by enumerating all element-wise operations as different rules. But this is not convenient. This also holds for sets, which are commonly used on LOCAL features in HPSG (e.g. SLASH).

Because of FS enumeration, there is an inevitable combinatorial explosion. This affects parsing time complexity exponentially in the number of attributes and the highest rank. In practice, we observe that our program actually runs slowly if complex type structures (e.g. lists as presented here) are involved. For instance, it took 1 hour to run `macg` on the very short `hpsg.macg` included example grammar, creating an intermediary grammar of several gigabytes. Therefore, this preprocessor approach might not be well suited for large-scale grammars. However, it offers a valuable tool for a quick development of experimental fragment grammars and prototypes.

Finally, we are planning to add the possibility to define a lexicon to type-theoretic semantics.

4 Conclusion

We introduced and formalized a novel way to include feature structures in categorial grammars. Our method consists in automatizing the idea of taking feature structures as categorial atomic types. The labor is divided into two separate modules: a preprocessor and a grammar engine. For every type with a feature structure, the preprocessor generates a representative set of categorial types. This creates an intermediary grammar given to the grammar engine. The latter works on these representative categorial types and just have to select right ones when parsing a sentence.

We proved that this approach of simulating feature structures by a set of representatives is sound and complete by showing that unification amounts to intersection of these representative sets. Having such a preprocessor avoids adding a unification module inside the grammar engine. It is modular and also easier to implement.

We evaluated this proposal by implementing a preprocessor for the grammar engine ACGtk working on abstract categorial grammars (ACG). This provides the first implementation of feature structures in an ACG toolkit. Example grammars show the well functioning of this method.

However, example grammars with a complex system of type hierarchy outlines the limits of the "enumeration-and-intersection" approach. Because of combinatorial explosion, the intermediary grammar can get really voluminous and take time to be created. This may restrict uses of such a preprocessor to toy ACG grammars only, waiting for a more efficient implementation of feature structures in ACGtk.

References

Kazimierz Ajdukiewicz. 1935. Die syntaktische Konexität. *Studia Philosophica*, 1:1–27.

Maxime Amblard, Maria Boritchev, and Philippe de Groote. 2021. An inquisitive account of wh-questions through event semantics. In *LACL 2021 - Logical Aspects of Computational Linguistics*, Montpellier (online), France.

Jason Baldridge. 2002. *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. Ph.D. thesis, Institute for Communicating and Collaborative Systems. University of Edinburgh.

Yehoshua Bar-Hillel. 1953. A Quasi-Arithmetical Notation for Syntactic Description. *Language*, 29(1):47–58.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media Inc.

Gosse Bouma. 1988. *Nonmonotonicity and Categorial Unification Grammar*. Ph.D. thesis, Rijksuniversiteit Groningen.

Wojciech Buszkowski. 1988. Generative Power of Categorial Grammars. In Richard T. Oehrle, Emmon Bach, and Deirdre Wheeler, editors, *Categorial Grammars and Natural Language Structures*, pages 69–94. Springer Netherlands, Dordrecht.

Bob Carpenter. 1991. The generative power of categorial grammars and head-driven phrase structure grammars with lexical rules. *Computational Linguistics*, 17(3):301–313.

Robert L. Carpenter. 1992. *The Logic of Typed Feature Structures: With Applications to Unification Grammars, Logic Programs and Constraint Resolution*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge.

Philippe de Groote. 2001. Towards abstract categorial grammars. In *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter*, pages 148–155, Toulouse, France.

Philippe de Groote and Sarah Maarek. 2007. Type-theoretic extensions of abstract categorial grammars. In *New Directions in Type-theoretic Grammars (NDTTG 2007). ESSLLI 2007 Workshop*, Dublin. Working paper or preprint.

Philippe de Groote, Sarah Maarek, and Ryo Yoshinaka. 2007. On Two Extensions of Abstract Categorial Grammars. In *Logic for Programming, Artificial Intelligence, and Reasoning*, Lecture Notes in Computer Science, pages 273–287, Berlin, Heidelberg. Springer.

Jochen Dörre, Esther König, and Dov Gabbay. 1996. Fibred Semantics for Feature-Based Grammar Logic. *Journal of Logic, Language, and Information*, 5(3/4):387–422.

Jochen Dörre and Suresh Manandhar. 1995. On Constraint-Based Lambek Calculi. In Patrick Blackburn and Marrten de Rijke, editors, *Logic, Structures and Syntax*. Reidel, Dordrecht.

Esther Kraak. 1995. French Clitics: A Categorial Perspective. Master's thesis, Universiteit Utrecht.

Joachim Lambek. 1958. The mathematics of sentence structure. *American Mathematical Monthly*, pages 154–170.

Jiří Maršík. 2013. Towards a Wide-Coverage Grammar : Graphical Abstract Categorical Grammars. Master's thesis, Université de Lorraine, June.

Michael Moortgat. 1996. Multimodal linguistic inference. *Journal of Logic, Language and Information*, 5(3/4):349–385.

Michael Moortgat. 1997. Categorial Type Logics. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*. Elsevier.

Glyn Morrill, Oriol Valentín, and Mario Fadda. 2011. The Displacement Calculus. *Journal of Logic, Language and Information*, 20(1):1–48.

Reinhard Muskens. 2001. Categorial Grammar and Lexical-Functional Grammar. In *Proceedings of the LFG01 Conference*, pages 259–279, University of Hong Kong.

Sylvain Pogodalla. 2016. ACGtk : un outil de développement et de test pour les grammaires catégorielles abstraites. In *Actes de la conférence conjointe JEP-TALN-RECITAL 2016. volume 5 : Démonstrations*, pages 1–2, Paris, France. AFCP - ATALA.

Carl Pollard and Ivan A. Sag. 1987. *Information-Based Syntax and Semantics, Vol. 1: Fundamentals*. The Center for the Study of Language and Information Publications, Stanford, CA.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago.

Mark Steedman. 1988. Combinators and Grammars. In Richard T. Oehrle, Emmon Bach, and Deirdre Wheeler, editors, *Categorial Grammars and Natural Language Structures*, pages 417–442. Springer Netherlands, Dordrecht.

Mark J. Steedman. 1990. Gapping as constituent coordination. *Linguistics and Philosophy*, 13(2):207–263.

Hans Uszkoreit. 1986. Categorial Unification Grammars. In *COLING 1986*, pages 187–194, Bonn.

Aline Villavicencio. 2002. *The Acquisition of a Unification-Based Generalised Categorial Grammar*. Ph.D. thesis, University of Cambridge, Cambridge.

Henk Zeevat. 1988. Combining Categorial Grammar and Unification. In U. Reyle and C. Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, pages 202–229. Springer Netherlands, Dordrecht.

A Further examples

We provide here an example to illustrate section 2.4. By lack of space, let us consider a very simplified toy HPSG grammar able to parse the sentence "*Ash slept*". It is based on the example code given in Fig.5. This grammar includes the two word FSs $\boxed{1'}$ and $\boxed{2'}$ below, as well as the phrase FS $\boxed{b}$ (Fig. 6) used to create a basic sentence with *NP-VP* agreement.

$$\boxed{1'}\begin{bmatrix} \textit{word} & & \\ \text{PHON} & \textit{Ash} & \\ \text{SYNSEM} & \boxed{0_{1'}}\begin{bmatrix} \textit{np} & \\ \text{AGR} & \begin{bmatrix} \text{P} & \textit{3} \\ \text{N} & \textit{sg} \end{bmatrix} \\ \text{PRO} & \text{-} \end{bmatrix} \end{bmatrix}$$

$$\boxed{2'}\begin{bmatrix} \textit{word} & & \\ \text{PHON} & \textit{slept} & \\ \text{SYNSEM} & \boxed{0_{2'}}\begin{bmatrix} \textit{vp} & \\ \text{AGR} & \textit{agr} \\ \text{T} & \textit{past} \end{bmatrix} \end{bmatrix}$$

The derivation of "*Ash slept*" in the deduction system of Fig. 3 is given in (4). FS $\boxed{c}$ is like $\boxed{b}$ but with $\boxed{1}$ and $\boxed{2}$ unified with $\boxed{1'}$ and $\boxed{2'}$ respectively.

$$\dfrac{\dfrac{}{\text{Ash} \vdash \boxed{1'} : \boxed{0_{1'}}}\,(*)_{\boxed{1'}} \qquad \dfrac{}{\text{slept} \vdash \boxed{2'} : \boxed{0_{2'}}}\,(*)_{\boxed{2'}}}{\text{Ash slept} \vdash \boxed{c} : \boxed{0_c}}\,(**)_{\boxed{b}} \tag{4}$$

Its transformation in the ACG system of Fig. 4 as described by the proof of proposition 3 is written in (5).

$$\dfrac{\dfrac{}{\text{Ash} \vdash \mathsf{c}_{\boxed{1'}} : \boxed{0_{1'}}}\,(*)_{\boxed{1'},\boxed{1'}} \qquad \dfrac{}{\text{slept} \vdash \mathsf{c}_{\boxed{2'}} : \boxed{0_{2'}}}\,(*)_{\boxed{2'},\boxed{2'}}}{\text{Ash slept} \vdash \mathsf{c}_{\boxed{b}}\,\mathsf{c}_{\boxed{1'}}\,\mathsf{c}_{\boxed{2'}} : \boxed{0_c}}\,(**)_{\boxed{b},\boxed{c}} \tag{5}$$

B Formal definitions and proofs

We provide here complementary formal definitions and proofs of the propositions stated in the main part.

B.1 Definitions

The following definitions are retrieved from Carpenter (1992).

Definition 6 (Inheritance hierarchy)**.** *An inheritance hierarchy $\langle \mathsf{T}, \sqsubseteq \rangle$ is a finite bounded complete partial order, i.e. a finite partial order such that every subset $S \subseteq \mathsf{T}$ having an upper bound has a least upper bound (aka. a join) $\bigsqcup S \in \mathsf{T}$.*

In particular, the empty set has a least upper bound noted $\bot$, which is then the minimum of T.

Definition 7 (Well-typed FS)**.** *A well-typed feature structure is a tuple $F = \langle Q, \overline{q}, \theta, \delta \rangle$ where*

- *Q is a finite non-empty tree of root $\overline{q} \in Q$*
- *$\theta : Q \to \mathsf{T}$ is a total node typing function*
- *$\delta : \mathsf{Att} \times Q \rightharpoonup Q$ is a feature partial function*
- *for every q, A such that $\delta(\mathrm{A}, q)\downarrow$, then $\mathsf{Approp}(\mathrm{A}, \theta(q))\downarrow$ and*

$$\mathsf{Approp}(\mathrm{A}, \theta(q)) \sqsubseteq \theta(\delta(\mathrm{A}, q))$$

$\mathcal{TF}$ is the set of well-typed feature structures.

Here we only consider well-typed feature structures (FS), and up to alphabetic variance.

Subsumption $\sqsubseteq$ and unification $\sqcup$ can be extended to well-typed feature structures.

Definition 8 (Subsumption of FS)**.** *$F = \langle Q, \overline{q}, \theta, \delta \rangle$ subsumes $F' = \langle Q', \overline{q}', \theta', \delta' \rangle$, written $F \sqsubseteq F'$, if there exists a function $h : Q \to Q'$ called morphism meeting the following conditions*

- *$h(\overline{q}) = \overline{q}'$*
- *for every $q \in Q$, $\theta(q) \sqsubseteq \theta'(h(q))$*
- *for every q, A, if $\delta(\mathrm{A}, q)\downarrow$, then $h(\delta(\mathrm{A}, q)) = \delta'(\mathrm{A}, h(q))$*

Subsumption is a partial ordering on $\mathcal{TF}$.

Definition 9 (Unification of FS)**.** *The unification of two well-typed FSs F, F' is, if it exists, the least upper bound of F and F' inside $\mathcal{TF}$.*

Here is the formal definition of totally well-typed FSs.

Definition 10 (Totally well-typed FS)**.** *A well-typed FS is totally well-typed if for all $q \in Q$ and $\mathrm{A} \in \mathsf{Att}$, if $\mathsf{Approp}(\mathrm{A}, \theta(q))\downarrow$, then $\delta(\mathrm{A}, q)\downarrow$.*

B.2 Proof of proposition 1

Proof. Set two feature structures F and G.

• Suppose $F \sqcup G$ exists. As $\mathcal{U}$ is clearly antitonic, and $F, G \sqsubseteq F \sqcup G$, we have $\mathcal{U}(\mathcal{F} \sqcup G) \subseteq \mathcal{U}(F), \mathcal{U}(G)$, so

$$\mathcal{U}(\mathcal{F} \sqcup G) \subseteq \mathcal{U}(F) \cap \mathcal{U}(G)$$

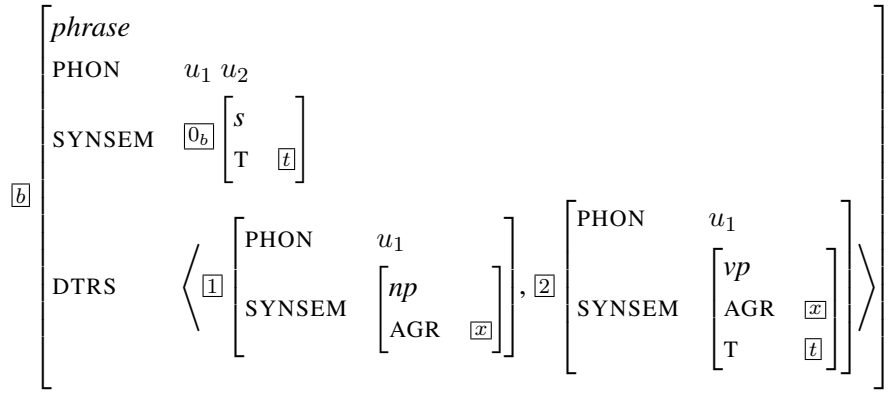


Figure 6: Feature structure for simple *NP-VP* phrase.

Moreover, by theorem 6.15 of Carpenter (1992), as Approp has no loop because of ranking, there exists at least one totally well-typed FS H such that $F \sqcup G \sqsubseteq H$. Therefore, $\mathcal{U}(\mathcal{F} \sqcup G) \neq \emptyset$, and so $\mathcal{U}(F) \cap \mathcal{U}(G) \neq \emptyset$.

• Now suppose there exists $H \in \mathcal{U}(F) \cap \mathcal{U}(G)$. As H is an upper bound of F and G, by theorem 6.9 of Carpenter (1992) they have a well-typed unification $\mathcal{F} \sqcup G$.

Moreover, we have $F \sqcup G \sqsubseteq H$ by minimality of the unification. As H is totally well-typed, H belongs to $\mathcal{U}(\mathcal{F} \sqcup G)$ too. Therefore

$$\mathcal{U}(F) \cap \mathcal{U}(G) \subseteq \mathcal{U}(\mathcal{F} \sqcup G)$$

In consequence, we proved that $\mathcal{F} \sqcup G$ exists iff $\mathcal{U}(F) \cap \mathcal{U}(G) \neq \emptyset$, and that in this case

$$\mathcal{U}(F) \cap \mathcal{U}(G) = \mathcal{U}(\mathcal{F} \sqcup G)$$

□

B.3 Proof of proposition 2

Proof. We write $\mathsf{T}_n = r^{-1}(n)$, which is finite because T is so.

By induction on $n \in \mathbb{N}$, let us prove that the set $\mathcal{TF}_n$ of FSs F of type $\tau \in \mathsf{T}_n$ is finite.

If $n = 0$, condition 2 of def. 4 implies that τ is appropriate for no attribute. As T_0 is finite, so is $\mathcal{TF}_0$.

If $n > 1$, then for all A such that $\delta(\mathrm{A}, \overline{q})\downarrow$, $\mathsf{Approp}(\mathrm{A}, \tau) \sqsubseteq \theta(\delta(\mathrm{A}, \overline{q}))$. Therefore

$$n = r(\tau) > r(\theta(\delta(\mathrm{A}, \overline{q})))$$

by condition 2 again.

So we can apply the induction hypothesis on $r(\theta(\delta(\mathrm{A}, \overline{q})))$. As Att is finite, so is the set of FSs of type τ. Then, as T_n is finite, so is $\mathcal{TF}_n$.

Since T is finite, there is a finite number of n such that $\mathsf{T}_n \neq \emptyset$. Therefore $\mathcal{TF} = \biguplus_{n \in \mathbb{N}} \mathcal{TF}_n$ is finite.

□

B.4 Proof of proposition 3

Proof. Let us begin with showing that

$$\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\mathsf{ACG}(\mathcal{G}))$$

$\boxed{\subseteq}$ Suppose string u is parsed by $\mathcal{G}$. There exists a derivation π of Fig. 3. We propagate the unification steps to the leaves and infer total type (Carpenter, 1992, thm. 6.15). From that, we construct a proof π' of Fig. 4 of same precedent and type, by induction on π:

If axiom $\pi = (*)_{\boxed{a}}$ exposes FS F, $F \in \mathcal{U}(\boxed{a})$. So we take $\pi' = (*)_{\boxed{a},F}$. This axioms has the same precedent w and type $\boxed{0}$ as π.

Suppose $\pi = (**)_{\boxed{b}}(\pi_1, ..., \pi_n)$ exposes FS F. Construct derivations $\pi'_1, ..., \pi'_n$ from $\pi_1, ..., \pi_n$ respectively, by induction hypothesis. We have $F \in \mathcal{U}(\boxed{c})$.

Moreover, from proposition 1 we deduce $\mathcal{U}(\boxed{c}) \subseteq \mathcal{U}(\boxed{b})$, because $\boxed{c} = \boxed{b} \sqcup \boxed{d}$ for some $\boxed{d}$. Therefore $F \in \mathcal{U}(\boxed{b})$.

As unification has been propagated, we have

$$F\left[\text{DTRS} \quad \left\langle F_1 \, , \, ..., \, F_n \right\rangle\right]$$

where F_i is the FS exposed at π_i, and π'_i has term M_i.

We thus have $c_F : \mathsf{t}_{F_1} \to ... \to \mathsf{t}_{F_n} \to \mathsf{t}_{F_0}$ with $c_F \in \mathcal{R}(\boxed{b})$, therefore term $\mathsf{c}_F \; M_1 \; ... \; M_n : \mathsf{t}_{F_0}$ is well-typed. As a result, the derivation $\pi' = (**)_{\boxed{b},F}(\pi'_1, ..., \pi'_n)$ is well-formed and has the same precedent $u_1 ... u_n$ and type $\boxed{0}$ as π.

As the root sequent of π is a finite sentence, its type is S, and so is the type of π'. Therefore $u \in \mathcal{L}(\mathsf{ACG}(\mathcal{G}))$.

Now let us show that

$$\mathcal{L}(\mathcal{G}) \supseteq \mathcal{L}(\mathsf{ACG}(\mathcal{G}))$$

$\boxed{\supseteq}$ Suppose string u is parsed by $\mathsf{ACG}(\mathcal{G})$. There exists a derivation π of Fig. 4 with precedent u. We construct a proof π' of Fig. 3 by induction on π by replacing axioms $(*)_{\boxed{a},F}$ by axioms $(*)_{\boxed{a}}$ and rules $(**)_{\boxed{b},F}$ by rules $(**)_{\boxed{b}}$. Each sequent $v \vdash M : s$ of π is mapped to $v \vdash \boxed{b} : s$ in π' with the λ-head c_F of M belonging to $\mathcal{R}(\boxed{b})$, so $F \in \mathcal{U}(\boxed{b})$. Therefore, π' is well-formed, has a sentence type, and thus $u \in \mathcal{L}(\mathcal{G})$.

□

C Alternative presentation of image ACG grammar

We give here an alternative presentation of the ACG grammar defined in Fig. 4 using the format used by de Groote (2001).

Definition 11. *Set Σ_1 the abstract signature where*

- *types are the* SYNSEM $\boxed{0}$ *of the word FSs and phrase FSs of $\mathcal{G}$*
- *constants are the representatives c_F of the word FSs or phrase FSs F of $\mathcal{G}$*
- *the type of c_F is the* SYNSEM *of F*

Set Σ_2 the signature of strings (de Groote, 2001, sec. 4), where constants are the phonological representations w of word FSs.

$$\begin{array}{c}\Sigma_1 \\ \downarrow\mathcal{Y} \\ \Sigma_2\end{array}$$

We define the ACG grammar $\mathsf{ACG}(\mathcal{G}) = \langle \Sigma_1, \Sigma_2, \mathcal{Y}, S\rangle$ with $\mathcal{Y} : \Sigma_1 \to \Sigma_2$ the lexicon mapping

1. *$\mathsf{c}_F \mapsto w$ if $F \in \mathcal{U}(\boxed{a})$ for some $\boxed{a}$ as in $(*)$*
2. *$\mathsf{c}_F \mapsto \lambda x_1, ..., x_n.\ x_1\ ...\ x_n$ if $F \in \mathcal{U}(\boxed{b})$ for some $\boxed{b}$ as in $(**)$*

and S is the feature structure of sentences[4] (Pollard and Sag, 1994):

$$\left[\text{LOC} \mid \text{CAT} \left[\begin{array}{ll}\text{HEAD} & \textit{verb}\left[\text{VFORM}\ \ \textit{fin}\right] \\ \text{SUBCAT} & \textit{e-list}\end{array}\right]\right]$$

As the appropriateness specification of $\mathcal{G}$ is ranked, $\mathsf{ACG}(\mathcal{G})$ is a well-defined ACG.

[4] Actually, there may be several FSs S of finite sentence (e.g. with different tenses). As the traditional definition of ACGs only allows one distinguished type, we could add a single extra abstract type s_d and abstract constants $T_S : S \to \mathsf{s}_d$ mapped to $\lambda x.\ x$ for every S.