Multityped Abstract Categorial Grammars and their Composition

Pierre Ludmann, Sylvain Pogodalla, and Philippe de Groote

Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
{pierre.ludmann,sylvain.pogodalla,philippe.degroote}@loria.fr

Abstract. This article introduces multityped abstract categorial grammars and show that a suitable composition operation can be defined.

Keywords: Abstract Categorial Grammar · Grammar Composition · Overloaded Signature.

1 Introduction

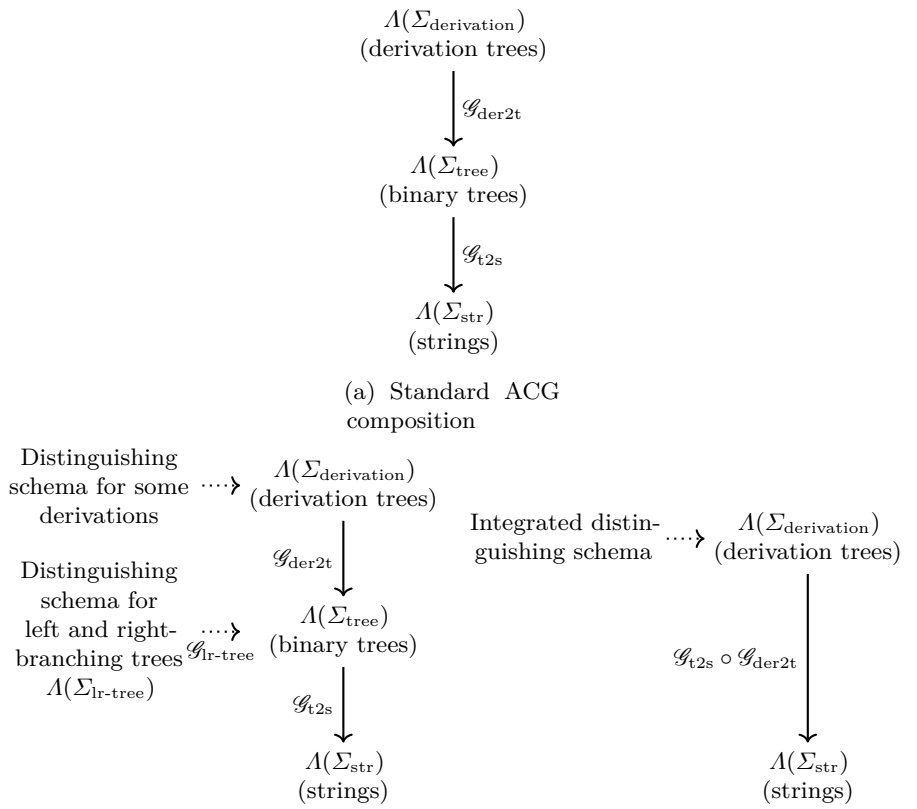
Abstract categorial grammars (ACGs, [4]) is an expressive compositional framework for both syntax and semantics. It is known to deeply encode several context-free formalisms, such as context-free grammars (CFGs) or tree-adjoining grammars (TAGs, [7]). However, just as for these symbolic grammatical formalisms, large-scale grammars exhibit a combinatorial explosion of parsing ambiguity. A widespread method to tackle this issue is to use statistics and probabilities, leading for instance to probabilistic CFGs (pCFGs, [11,2]) and probabilistic TAGs (pTAGs, [10,3]). An important goal would then be to also extend ACGs with probabilities or weights.

Yet, ACGs come with features that make this extension non-trivial. In particular, as Figure 1a illustrates with the main ACGs given as examples in this article, ACGs can be composed by making the parse structures of a grammar $\mathscr{G}_{\text{t2s}}$ the surface structures of another ACG $\mathscr{G}_{\text{der2t}}$, simply by *function composition*. The resulting composition $\mathscr{G} = \mathscr{G}_{\text{t2s}} \circ \mathscr{G}_{\text{der2t}}$ is an ACG.

Introducing preferences to some specific structures at a given level, eventually expressed by weights, requires to be able to distinguish between the structures at this level. For instance, distinguishing between left or right branching trees from other binary trees in $\Lambda(\Sigma_{\text{tree}})$ (done in Example 4), distinguishing between derivation trees of $\Lambda(\Sigma_{\text{derivation}})$ where adjunctions occur high or deep, distinguishing between strings containing some specific sequences in $\Lambda(\Sigma_{\text{str}})$, etc. Such distinctions, exemplified with dotted arrows in Figure 1 and informally called distinguishing schemas, typically rely on defining an additional ACG to control the generated structures, such as $\mathscr{G}_{\text{lr-tree}}$ that maps $\Lambda(\Sigma_{\text{lr-tree}})$ to $\Lambda(\Sigma_{\text{tree}})$ in Figure 1b, pretty much in the same way as adding states in an automaton allows for more fine-grained distinction between accepted inputs. This highlights an important new grammatical object: the composition of an ACG (plain arrows on Figure 1) with another ACG (dotted arrows) allowing for distinguishing between parse structures of the former ACG, e.g., $\mathscr{G}_{\text{t2s}} \circ \mathscr{G}_{\text{lr-tree}}$.

We then expect to have a well-behaved composition of these new grammatical objects as well. In particular the result should indeed be one of these new grammatical objects, as Figure 1c illustrates (also see its formally defined couterpart in Figure 5). We will see that the dotted arrows of Figure 1 correspond to specific ACGs, namely *relabelings*. The difficulty we address here is then to combine these new grammatical objects into equivalent ones that also make use of relabelings.

To this end, this article introduces *multityped* ACGs (mACGs). Multityped ACG are the underlying discrete mathematical structures that will support weighting extension (not presented here). It also shows that a suitable notion of composition can be defined for multityped ACGs.



(a) Standard ACG composition

(b) Addition of distinguishing schemas

(c) Composition taking distinguishing schemas into account

Fig. 1: Composition of ACGs

Section 2 sets the main mathematical notions used in the article and introduces examples to be used through the different sections. It then briefly but formally

introduces ACGs. Using these notions, Section 2.2 develops the motivations and the directions for extending ACGs. Section 3 defines *overloaded* signatures and *multityped* ACGs, and Section 4 addresses the problem of multityped ACG composition.

2 Abstract Categorial Grammars

ACGs were introduced by [4]. With respect to other grammatical formalisms (e.g., CFGs, TAGs, or combinatory categorial grammar, [13]), ACGs have the following discriminating key features:

- An ACG actually defines *two* languages: an *abstract* and an *object* one. The former can generally be seen as the set of admissible parse structures, while the latter can be seen as the set of corresponding surface structures in a broad sense: strings, for instance, but also logical formulas.
- The languages are sets of (linear) λ-terms, that generalize both strings and trees.
- A *lexicon* defines the relation between the abstract and the object language. It is a homomorphism that interprets abstract structures as object ones.
- Because both abstract and object languages are sets of λ-terms, ACGs have built-in support for grammar composition.

2.1 Definitions

Definition 1 (Linear Implicative Type). *Let A be a set of atomic types. The set $\mathscr{T}_A$ of* linear implicative types *built upon A is inductively defined as follows:*

- *if $a \in A$, then $a \in \mathscr{T}_A$;*
- *if $\alpha, \beta \in \mathscr{T}_A$, then $(\alpha \multimap \beta) \in \mathscr{T}_A$.*

In order to save parentheses, we use the usual convention of right association, i.e., we write $\alpha_1 \multimap \alpha_2 \multimap \cdots \alpha_n \multimap \alpha$ for $(\alpha_1 \multimap (\alpha_2 \multimap \cdots (\alpha_n \multimap \alpha) \cdots))$.

Definition 2 (Type Morphism and Relabeling). *Given two sets of atomic types, A and B, a mapping $h : \mathscr{T}_A \to \mathscr{T}_B$ is called a* type homomorphism *(or a* type substitution*) if it satisfies the following condition: $\forall \alpha, \beta \in \mathscr{T}_A, h(\alpha \multimap \beta) = h(\alpha) \multimap h(\beta)$.*

A type substitution mapping atomic types to atomic types is called a type relabeling*.*

Thus, a type morphism is fully defined by how it maps atomic types to linear implicative types.

Definition 3 (Skeleton of a Type). *Let A be a set of atomic types and let $☠ = \{\square\}$. The* skeleton *of a type $\alpha \in \mathscr{T}_A$ is $\mathrm{skel}_A(\alpha)$ where $\mathrm{skel}_A : \mathscr{T}_A \to \mathscr{T}_☠$ is the relabeling mapping atomic types of $\mathscr{T}_A$, i.e., elements of A, to the unique atomic type $\square$.*

For instance, $\mathrm{skel}((NP \multimap S) \multimap NP) = (\square \multimap \square) \multimap \square$.

Definition 4 (Higher-Order Linear Signature). *A* higher-order linear signature *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;*
- *$\tau : C \to \mathscr{T}_A$ is a function that assigns to each constant in C a linear implicative type in $\mathscr{T}_A$.*

Given, a higher-order linear signature Σ, we write A_Σ, C_Σ, and τ_Σ for its respective components.

Definition 5 (Linear λ-Term). *Let X be an infinite countable set of variables and Σ be a higher-order linear signature (X and C_Σ disjoint). The set $\Lambda(\Sigma)$ of* linear λ-terms *built upon Σ is inductively defined as follows:*

- *if $c \in C_\Sigma$, then $c \in \Lambda(\Sigma)$;*
- *if $x \in X$, then $x \in \Lambda(\Sigma)$;*
- *if $x \in X$, $t \in \Lambda(\Sigma)$, and x occurs free in t exactly once, then $(\lambda x.\, t) \in \Lambda(\Sigma)$;*
- *if $t, u \in \Lambda(\Sigma)$, and the sets of free variables of t and u are disjoint, then $(t\, u) \in \Lambda(\Sigma)$.*

$\Lambda(\Sigma)$ is provided with the usual notions of α-conversion, β-reduction, and η-reduction [1]. In this paper, all linear λ-terms are identified up to α-conversion. Let t and u be linear λ-terms. We write $t \to_\beta u$ and $t =_\beta u$ for the relations of β-reduction and β-equivalence, respectively.

Definition 6 (λ-Term Homomorphism and Relabeling). *Let Σ_1 and Σ_2 be two signatures. We say that a mapping $h : \Lambda(\Sigma_1) \to \Lambda(\Sigma_2)$ is a* λ-term homomorphism *if it satisfies the following conditions: $\forall t, u \in \Lambda(\Sigma_1), x \in X, h(x) = x$, $h(\lambda x.\, t) = \lambda x.\, h(t)$, and $h(t\, u) = h(t)\,(h(u))$.*

A λ-term homomorphism that maps constants of Σ_1, i.e., elements of C_{Σ_1}, to constants of Σ_2, i.e., elements of C_{Σ_2}, is called a term relabeling.

Thus, a λ-term homomorphism is fully defined by how it maps constants to linear λ-terms.

Definition 7 (Typing Context and Judgment). *Let Σ be a higher-order linear signature. A* typing context *Γ is a finite multiset of pairs $x : \alpha$ such that x is a variable and $\alpha \in \mathscr{T}_{A_\Sigma}$.*

A typing judgment *$\Gamma \vdash_\Sigma t : \alpha$ assigns the type $\alpha \in \mathscr{T}_{A_\Sigma}$ to the term $t \in \Lambda(\Sigma)$ in the context Γ if $\Gamma \vdash_\Sigma t : \alpha$ is the root of a derivation tree obeying the rules of Table 1.*

We note $t :_\Sigma \alpha$ (or simply $t : \alpha$ when Σ is explicit from the context) for $\vdash_\Sigma t : \alpha$, i.e., when the typing context is empty. We also note $\Gamma \vdash u : \alpha =_\beta \Delta \vdash v : \beta$ for $u =_\beta v$, $\alpha = \beta$ and $\Gamma = \Delta$.

$$\frac{}{x : \alpha \vdash_\Sigma x : \alpha}(\text{var}) \qquad \frac{\Gamma, x : \alpha \vdash_\Sigma t : \beta}{\Gamma \vdash_\Sigma \lambda x.\, t : \alpha \multimap \beta}(\text{abs})$$

$$\frac{}{\vdash_\Sigma c : \tau_\Sigma(c)}(\text{cst}) \qquad \frac{\Gamma_1 \vdash_\Sigma t : \alpha \multimap \beta \qquad \Gamma_2 \vdash_\Sigma u : \alpha}{\Gamma_1, \Gamma_2 \vdash_\Sigma t\, u : \beta}(\text{app})$$

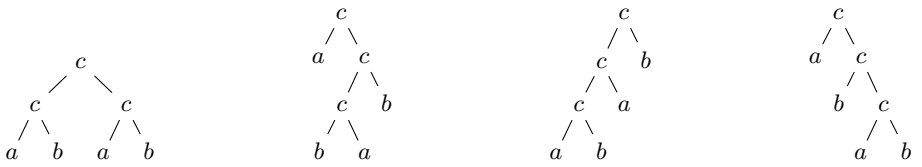
Table 1: Inference rules, given a signature Σ

Example 1 (Tree Signature). We can define a signature Σ_{tree} for binary trees with a binary symbol c and two nullary symbols a and b as follows: $\Sigma_{\text{tree}} = \langle A_{\Sigma_{\text{tree}}}, C_{\Sigma_{\text{tree}}}, \tau_{\Sigma_{\text{tree}}} \rangle$ where: $A_{\Sigma_{\text{tree}}} = \{T\}$; $C_{\Sigma_{\text{tree}}} = \{a, b, c\}$; and $\tau_{\Sigma_{\text{tree}}}(a) = T$, $\tau_{\Sigma_{\text{tree}}}(b) = T$, $\tau_{\Sigma_{\text{tree}}}(c) = T \multimap T \multimap T$.

Terms of $\Lambda(\Sigma_{\text{tree}})$ of type T include, for instance, the ones defined in Equations (1a–1d) and which correspond to the trees of Figure 2.

$$t_{2a} = c\,(c\,a\,b)\,(c\,a\,b) \quad (1a) \qquad t_{2c} = c\,(c\,(c\,a\,b)\,a)\,b \quad (1c)$$

$$t_{2b} = c\,a\,(c\,(c\,b\,a)\,b) \quad (1b) \qquad t_{2d} = c\,a\,(c\,b\,(c\,a\,b)) \quad (1d)$$



(a) Balanced tree

(b) Mixed tree

(c) Left-branching tree

(d) Right-branching tree

Fig. 2: Terms of $\Lambda(\Sigma_{\text{tree}})$

Example 2 (String Signature). We define a signature Σ_{str} for strings using the a and b string symbols as follows: $\Sigma_{\text{str}} = \langle A_{\Sigma_{\text{str}}}, C_{\Sigma_{\text{str}}}, \tau_{\Sigma_{\text{str}}} \rangle$ where:

- $A_{\Sigma_{\text{str}}} = \{o\}$ will allow us to define the complex type of strings $\sigma = o \multimap o$ for which concatenation is functional composition (the λ-term $+ = \lambda f\, g.\, \lambda z.\, f\,(g\,z) : \sigma \multimap \sigma \multimap \sigma$) and the empty string is the identity $(\lambda x.\, x : \sigma)$
- $C_{\Sigma_{\text{str}}} = \{a, b\}$
- $\tau_{\Sigma_{\text{str}}}(a) = \tau_{\Sigma_{\text{str}}}(b) = \sigma$

For instance, the string ab corresponds to the term $\lambda z.\, a\,(b\,z)$ of $\Lambda(\Sigma_{\text{str}})$; and it duplication $abab$ is $\lambda z.\, a\,(b\,z) + \lambda z.\, a\,(b\,z) = \lambda z.\, a\,(b\,(a\,(b\,z)))(= a + b + a + b)$.

Definition 8 (Simple Lexicon and Relabeling). *Let $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ be two higher-order signatures. A* lexicon $\mathscr{L} : \Sigma_1 \to \Sigma_2$ *is a pair $\langle F, G \rangle$ such that:*

- $F : A_1 \to \mathscr{T}_{A_2}$ *is a type homomorphism that interprets linear implicative types built upon* Σ_1 *as linear implicative types built upon* Σ_2*;*
- $G : C_1 \to \Lambda(\Sigma_2)$ *is a term homomorphism that interprets linear* λ*-terms built upon* Σ_1 *as linear* λ*-terms built upon* Σ_2*;*
- *the interpretation functions are compatible with the typing relation, i.e., for any* $c \in C_1$, $\vdash_{\Sigma_2} G(c) : F(\tau_1(c))$ *is derivable.*

When F *and* G *are respectively type and term relabelings, the lexicon is called a relabeling.*

In the sequel, given such a lexicon $\mathscr{L} = \langle F, G\rangle$, $\mathscr{L}(a)$ will stand for either $F(a)$ or $G(a)$, according to the context and $a := b$ will stand for $\mathscr{L}(a) = b$. We also write $\mathscr{L}(x_1 : \alpha_1, \dots, x_n : \alpha_n)$ for the context $x_1 : \mathscr{L}(\alpha_1), \dots, x_n : \mathscr{L}(\alpha_n)$, and $\mathscr{L}(\Gamma \vdash_{\Sigma_1} u : \alpha)$ for $\mathscr{L}(\Gamma) \vdash_{\Sigma_2} \mathscr{L}(u) : \mathscr{L}(\alpha)$.

Lemma 1. *Let* $\mathscr{R}$ *be a relabeling from* Σ_1 *to* Σ_2*. Let* $t \in \Lambda(\Sigma_2)$ *and* $N \in \Lambda(\Sigma_1)$ *such that* $\mathscr{R}(N) =_\beta t$*. There exists a unique* $M =_\beta N$ *such that* $\mathscr{R}(M) = t$*.*

Proof. The β-equivalence boils down to the two cases where either $\mathscr{R}(N)$ or t is a redex and reduction is performed in one step.

The first case is $\mathscr{R}(N) = (\lambda x.u)v \to_\beta t$. Since relabelings preserve structures, N has the form $(\lambda x.N_0)N_1$. Thus $M = N_0[x := N_1]$.

The second case is $t = (\lambda x.u)v \to_\beta \mathscr{R}(N)$. Since relabelings preserve structures, N has the form $N_0[x := N_1]$. Thus $M = (\lambda x.N_0)N_1$.

Definition 9 (Abstract Categorial Grammar). *An* abstract categorial grammar *is a quadruple,* $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, S\rangle$*, where:*

- Σ_1 *and* Σ_2 *are two higher-order linear signatures. They are called the* abstract vocabulary *and the* object vocabulary*, respectively. Terms of* $\Lambda(\Sigma_1)$ *(resp.* $\Lambda(\Sigma_2)$*) are called* abstract terms *(resp.* object terms*).*
- $\mathscr{L} : \Sigma_1 \to \Sigma_2$ *is a lexicon from the abstract vocabulary to the object vocabulary.*
- S *is a set of* distinguished *abstract types of the abstract vocabulary.*

Example 3 (Tree Interpretation). Using the signatures defined in Examples 1 and 2, we can now define an ACG $\mathscr{G}_{\text{t2s}} = \langle \Sigma_{\text{tree}}, \Sigma_{\text{str}}, \mathscr{L}_{\text{t2s}}, \{T\}\rangle$ where

- $\mathscr{L}_{\text{t2s}}(T) = \sigma$, i.e., a tree (a term of $\Lambda(\Sigma_{\text{tree}})$) is interpreted as string.
- $\mathscr{L}_{\text{t2s}}(a) = a$ and $\mathscr{L}_{\text{t2s}}(b) = b$.
- $\mathscr{L}_{\text{t2s}}(c) = \lambda l\, r.\, l + r$, where $+$ is the concatenation of strings.

We then check that $\mathscr{L}_{\text{t2s}}(t_{2a}) = \mathscr{L}_{\text{t2s}}(t_{2b}) = \mathscr{L}_{\text{t2s}}(t_{2c}) = \mathscr{L}_{\text{t2s}}(t_{2d}) = a + b + a + b$.

Definition 10 (Abstract and Object Languages). *Let* $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, S\rangle$ *be an ACG. The* abstract language *of* $\mathscr{G}$ *is* $\mathcal{A}(\mathscr{G}) = \{t \in \Lambda(\Sigma_1) | \exists \alpha \in S, \vdash_{\Sigma_1} t : \alpha\}$*. The* object language *of* $\mathscr{G}$ *is* $\mathcal{O}(\mathscr{G}) = \{u \in \Lambda(\Sigma_2) | \exists t \in \mathcal{A}(\mathscr{G}), u = \mathscr{L}(t)\}$*.*

Given an ACG $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, S\rangle$, the ACG parsing of a term u of $\Lambda(\Sigma_2)$ amounts to finding $t \in \mathcal{A}(\mathscr{G})$ such that $\mathscr{L}(t) = u$.

Example 4 (ACG Composition). The ACG $\mathscr{G}_{\text{t2s}}$ of Example 3 defines the object language $\mathcal{O}(\mathscr{G}_{\text{t2s}}) = (a|b)^+$.

Regarding the abstract language $\mathcal{O}(\mathscr{G}_{\text{t2s}})$, it contains any binary tree, such as the ones exemplified in Figure 2. For instance, $a + b + a + b$ has five antecedents by $\mathscr{L}_{\text{t2s}}$ (among which the terms corresponding to the trees of Figure 2).

Let us now assume we are only interested in left or right-branching trees. We are going to define an ACG $\mathscr{G}_{\text{lr-tree}} = \langle \Sigma_{\text{lr-tree}}, \Sigma_{\text{tree}}, \mathscr{L}_{\text{lr-tree}}, S \rangle$ in which terms of the abstract language are left or right-branching trees, and compose it with $\mathscr{G}_{\text{t2s}}$ so that we do not need to redefine the string interpretation.

We define a signature $\Sigma_{\text{lr-tree}}$ with $A_{\Sigma_{\text{lr-tree}}} = \{T_u, T_l, T_r\}$ and $C_{\Sigma_{\text{lr-tree}}} = \{a_u, b_u, a_l, b_l, a_r, b_r, c_l, c_r\}$. The type assigment $\tau_{\Sigma_{\text{lr-tree}}}$ is such that: $\tau_{\Sigma_{\text{lr-tree}}}(v_w) = T_w$ for $v \in \{a, b\}$ and $w \in \{u, l, r\}$, while $\tau_{\Sigma_{\text{lr-tree}}}(c_l) = T_l \multimap T_u \multimap T_l$ and $\tau_{\Sigma_{\text{lr-tree}}}(c_r) = T_u \multimap T_r \multimap T_r$.

The definition of $\mathscr{L}_{\text{lr-tree}} : A_{\Sigma_{\text{lr-tree}}} \to \Lambda(\Sigma_{\text{tree}})$ is straightforward: $\mathscr{L}_{\text{lr-tree}}(T_v) = T$ for any $v \in \{u, l, r\}$ and $\mathscr{L}_{\text{lr-tree}}(x_v) = x$ for any $x \in \{a, b, c\}$ and $v \in \{u, l, r\}$. Terms of $\mathcal{A}(\mathscr{G}_{\text{lr-tree}})$ (when $S = \{T_l, T_r\}$) then only consists of left-branching or right-branching trees, and they can be interpreted as strings by $\mathscr{L}_{\text{t2s}} \circ \mathscr{L}_{\text{lr-tree}}$. For instance, $a + b + a + b$ now only has two antecedents by $\mathscr{L}_{\text{t2s}} \circ \mathscr{L}_{\text{lr-tree}}$: the terms of $\mathcal{A}(\mathscr{G}_{\Sigma_{\text{lr-tree}}})$ that are interpreted by $\mathscr{L}_{\text{lr-tree}}$ as terms corresponding to the trees of Figures 2c and 2d.

2.2 Distinguishing between Structures

Albeit through quite a simple lexicon, Example 4 illustrates the composition ability of ACGs. The lexicon $\mathscr{L}_{\text{lr-tree}}$ is actually an example of a *relabeling*. From the Σ_{tree} point of view, it means that c is not considered with the full generality of the $T \multimap T \multimap T$ type, but either with the more specific type $T_l \multimap T_u \multimap T_l$ or the more specific type $T_u \multimap T_r \multimap T_r$. On the other hand, its interpretation by $\mathscr{L}_{\text{t2s}}$ remains unchanged.

We decided to restrict the admissible parse structures to left-branching and right-branching trees. However, we eventually are interested in *preferences* and *ranking*, not only in membership. So we want to distinguish left and right branching trees (and possibly give them preference), but we don't want to exclude some other tree structures. For instance, we could add a constant c'_l with type $T_r \multimap T_u \multimap T_l$, and a constant c'_r with type $T_u \multimap T_l \multimap T_r$. This is reminiscent to adding states in tree automata to distinguish transitions triggered by a same symbol. Because all these constants are interpreted as c, we could equivalently assign c this set of types and possibly add a weight to each of its typings, pretty much as weights can be added to transition of tree automata [6,9].

Note that, although ACGs in the examples feature abstract constants with parameters of atomic type, the approach also extends to higher-order signatures where abstract constants may have functions as parameters.

However, we first have to precisely define multityped ACGs, and check what the implications on ACG operations such as composition are. In particular, the relabelings can introduce distinction between structures at any level when composing ACGs. We therefore expect them to combine in order to define an equivalent distinction for the resulting composed ACG.

3 Multityped Abstract Categorial Grammars

3.1 Overloaded Signatures

The main differences between simple signatures and overloaded ones is the relational nature of the type assignment, and the addition of the skeleton constraint.

Definition 11 (Overloaded Signature). *An* overloaded signature *is a triple* $\Pi = \langle A, C, \tau^+ \rangle$*, where:*

- A *is a finite set of atomic types;*
- C *is a finite set of constants;*
- $\tau^+ : C \to \mathcal{P}_f(\mathscr{T}_A)$ *is a function that assigns to each constant in* C *a non-empty finite set of linear implicative types in* $\mathscr{T}_A$*;*
- *a constant's types share a skeleton:* $\forall c \in C, \forall \alpha, \beta \in \tau^+(c), \mathrm{skel}(\alpha) = \mathrm{skel}(\beta)$

Given a signature $\Pi = \langle A, C, \tau^+ \rangle$*, we write* $\mathscr{T}_\Pi$ *for* $\mathscr{T}_A$ *and* $\Lambda(\Pi)$ *for* $\Lambda(C)$*.*

Example 5. We can now formally define the overloaded signature Π mentioned in Section 2.2 as a follow up to Example 4. $\Pi = \langle A, C, \tau^+ \rangle$ where: $A = \{T_u, T_l, T_r\}$; $C = \{a, b, c\}$; τ^+ is given by Table 2.

Note that Π satisfies the skeleton constraint. It would not be the case, for instance, if a was additionally given an implicative type such as $T_l \multimap T_r$.

$$a : \begin{Bmatrix} T_u, \\ T_l, \\ T_r \end{Bmatrix} \qquad b : \begin{Bmatrix} T_u, \\ T_l, \\ T_r \end{Bmatrix} \qquad c : \begin{Bmatrix} T_l \multimap T_u \multimap T_l, \\ T_u \multimap T_r \multimap T_r \end{Bmatrix}$$

Table 2: The overloaded signature Π

Typing judgments are still defined by induction on the typing rules featured in Table 1, but for the typing rule of constants which is slightly amended as follows: $\dfrac{\alpha \in \tau_\Pi(c)}{\vdash_\Pi c : \alpha}(\mathrm{cst}+)$.

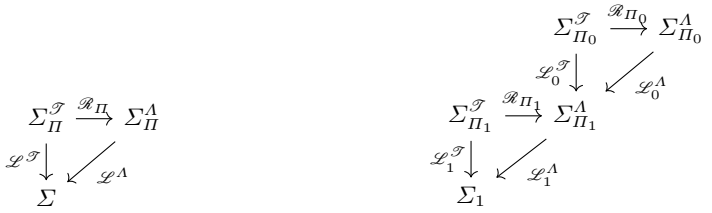
It can be shown (see Section 3.3) that to any overloaded signature Π corresponds a triple $(\Sigma_\Pi^\mathscr{T}, \Sigma_\Pi^\Lambda, \mathscr{R}_\Pi)$ where $\Sigma_\Pi^\mathscr{T}$ (the *type carrier* of Π) and Σ_Π^Λ (the *term carrier* of Π) are simple signatures and $\mathscr{R}_\Pi : \Sigma_\Pi^\mathscr{T} \to \Sigma_\Pi^\Lambda$ is a relabeling, such that $\Gamma \vdash_\Pi M : \alpha$ iff there exists $t \in \Lambda(\Sigma_\Pi^\mathscr{T})$ such that $\mathscr{R}_\Pi(t) = M$ and $\Gamma \vdash_{\Sigma_\Pi^\mathscr{T}} t : \alpha$ (see Lemma 2). The reverse also holds (see Lemma 3).

3.2 Lexicons and Multityped ACGs

We now want to define a lexicon $\mathscr{L}^+$ between an overloaded signature Π and a simple signature Σ. A guiding principle is to refer to the carriers of Π and to the relabeling $\mathscr{R}_\Pi : \Sigma_\Pi^\mathscr{T} \to \Sigma_\Pi^\Lambda$ that simulates Π (see Lemma 2). We can

then consider the simple lexicons $\mathscr{L}^{\mathscr{T}} : \Sigma_{\Pi}^{\mathscr{T}} \to \Sigma$ and $\mathscr{L}^{\Lambda} : \Sigma_{\Pi}^{\Lambda} \to \Sigma$. These two lexicons respectively act on the linear implicative types, and on the linear λ-terms built upon Π.

To preserve typing judgments, we require $\mathscr{L}^{\mathscr{T}}$ and $\mathscr{L}^{\Lambda}$ to comply with the diagram in Figure 3a: for any $\Gamma \vdash_{\Sigma_{\Pi}^{\mathscr{T}}} t : \alpha$, $\mathscr{L}^{\mathscr{T}}(\Gamma \vdash_{\Sigma_{\Pi}^{\mathscr{T}}} t : \alpha) =_{\beta} \mathscr{L}^{\Lambda} \circ \mathscr{R}_{\Pi}(\Gamma \vdash_{\Sigma_{\Pi}^{\mathscr{T}}} t : \alpha)$. From Section 3.3, this is equivalent to requiring that for any $\Gamma \vdash_{\Pi} M : \alpha$, there exists $t \in \Lambda(\Sigma_{\Pi}^{\mathscr{T}})$ such that $\mathscr{R}_{\Pi}(t) = M$ and $\mathscr{L}^{\mathscr{T}}(\Gamma) \vdash_{\Sigma} \mathscr{L}^{\mathscr{T}}(t) : \mathscr{L}^{\mathscr{T}}(\alpha) =_{\beta} \mathscr{L}^{\Lambda}([\Gamma]_{\sim_{\Pi}}) \vdash_{\Sigma} \mathscr{L}^{\Lambda}(M) : \mathscr{L}^{\Lambda}([\alpha]_{\sim_{\Pi}})$ where $\Gamma = x_1 : \alpha_1, \ldots, x_n : \alpha_n$ and $[\Gamma]_{\sim_{\Pi}} = x_1 : [\alpha_1]_{\sim_{\Pi}}, \ldots, x_n : [\alpha_n]_{\sim_{\Pi}}$.



(a) Diagram of a lexicon $\mathscr{L}^{+} : \Pi \to \Sigma$

(b) Maps defined by two ACGs with a shared vocabulary

Fig. 3: Diagrams describing mACGs

Thus, we define the lexicon $\mathscr{L}^{+}$ from Π to Σ such that $\forall \alpha \in \mathscr{T}_{\Pi}, \mathscr{L}^{+}(\alpha) = \mathscr{L}^{\mathscr{T}}(\alpha)$ and $\forall M \in \Lambda(\Pi), \mathscr{L}^{+}(M) = \mathscr{L}^{\Lambda}(M)$. In particular, $\mathscr{L}^{+}$ maps Π-equivalent (atomic) types to the same type.

Definition 12 (Lexicon). *Let $\Pi = \langle A, C, \tau^{+} \rangle$ be an overloaded signature and Σ a simple signature. A* lexicon *from Π to Σ is a pair $\mathscr{L}^{+} = \langle F, G \rangle$ such that:*

- *$F : \mathscr{T}_{\Pi} \to \mathscr{T}_{\Sigma}$ is a type homomorphism that interprets linear implicative types built upon Π as linear implicative types built upon Σ;*
- *$G : \Lambda(\Pi) \to \Lambda(\Sigma)$ is a term homomorphism that interprets linear λ-terms built upon Π as linear λ-terms built upon Σ;*
- *the interpretation homomorphisms are compatible with the typing relation: $\forall c \in C, \forall \alpha \in \tau^{+}(c), \vdash_{\Pi} G(c) : F(\alpha)$*
- *the type homomorphism maps the skeleton equivalence to the equality: $\forall a, b \in A, a \sim_{\Pi} b \Rightarrow F(a) = F(b)$*

We write $\mathscr{L}^{+}(\alpha)$ for $F(\alpha)$, $\mathscr{L}^{+}(M)$ for $G(M)$, $\mathscr{L}^{+}(\Gamma)$ for $F(\Gamma)$, and $\mathscr{L}^{+}(\Gamma \vdash_{\Pi} M : \alpha)$ for $\mathscr{L}^{+}(\Gamma) \vdash_{\Sigma} \mathscr{L}^{+}(M) : \mathscr{L}^{+}(\alpha)$.

As for simple lexicons (see Definition 8), it is sufficient to define a lexicon on the domain signature.

Conversely, given a lexicon $\mathscr{L}^+ : \Pi \to \Sigma$, one defines $\mathscr{L}^{\mathscr{T}} : \Sigma_\Pi^{\mathscr{T}} \to \Sigma$ the *type lexicon of* $\mathscr{L}^+$ and $\mathscr{L}^\Lambda : \Sigma_\Pi^\Lambda \to \Sigma$ the *term lexicon of* $\mathscr{L}^+$ as follows:

$$\mathscr{L}^{\mathscr{T}}(t) = \mathscr{L}^+ \circ \mathscr{R}_\Pi(t) \text{ if } t \in \Lambda(\Sigma_\Pi^{\mathscr{T}}) \qquad \mathscr{L}^{\mathscr{T}}(\alpha) = \mathscr{L}^+(\alpha) \text{ if } \alpha \in \mathscr{T}_{\Sigma_\Pi^{\mathscr{T}}}$$
$$\mathscr{L}^\Lambda(M) = \mathscr{L}^+(M) \text{ if } M \in \Lambda(\Sigma_\Pi^\Lambda) \qquad \mathscr{L}^\Lambda([\alpha]_{\sim_\Pi}) = \mathscr{L}^+(\alpha) \text{ if } [\alpha]_{\sim_\Pi} \in \mathscr{T}_{\Sigma_\Pi^\Lambda}$$

Then: $\mathscr{L}^+$ and $\mathscr{L}^{\mathscr{T}}$ act the same on linear implicative types; $\mathscr{L}^+$ and $\mathscr{L}^\Lambda$ act the same on linear λ-terms; and $\mathscr{L}^{\mathscr{T}}$ and $\mathscr{L}^\Lambda$ comply with the Figure 3a.

Example 6 (Lexicon from an Overloaded Signature). Using the overloaded signature of Table 2 and the simple signature Σ_{str} defined in Example 2, we can define the lexicon:

$$T_u, T_l, T_r := \sigma \qquad a := a \qquad b := b \qquad c := \lambda l\, r.\, l + r$$

Because a is assigned the three atomic types T_u, T_l, and T_r these types are skeleton equivalent and are required to be interpreted by the same type of $\mathscr{T}_{A_{\Sigma_{\text{str}}}}$.

Definition 13 (Multityped ACG). *A* multityped abstract categorial grammar (mACG) *is a quadruple* $\mathscr{G}^+ = \langle \Pi, \Sigma, \mathscr{L}^+, S \rangle$ *where:*

- Π *is an overloaded signature;*
- Σ *is a simple signature;*
- $\mathscr{L}^+ : \Pi \to \Sigma$ *is a lexicon from* Π *to* Σ*;*
- S *is a set of distinguished abstract types of* $\mathscr{T}_\Pi$*.*

Definition 14 (Abstract and Object Languages of a Multityped ACG). *Let* $\mathscr{G}^+ = \langle \Pi, \Sigma, \mathscr{L}^+, S \rangle$ *be an ACG. The* abstract language *of* $\mathscr{G}^+$ *is* $\mathcal{A}(\mathscr{G}^+)$*, where* $\mathcal{A}(\mathscr{G}^+) = \{t \in \Lambda(\Pi) | \exists \alpha \in S, \vdash_\Pi t : \alpha\}$.

The object language *of* $\mathscr{G}^+$ *is* $\mathcal{O}(\mathscr{G}^+) = \{u \in \Lambda(\Sigma) | \exists t \in \mathcal{A}(\mathscr{G}^+), u = \mathscr{L}^+(t)\}$.

Example 7 (Multityped ACG). We can now define the multityped ACG $\mathscr{G}^+ = \langle \Pi, \Sigma_{\text{str}}, \mathscr{L}^+, \{T_l, T_r\} \rangle$ from the lexicon of Example 6. As in Example 3, $\mathscr{L}^+(t_{2a}) = \mathscr{L}^+(t_{2b}) = \mathscr{L}^+(t_{2c}) = \mathscr{L}^+(t_{2d}) = a + b + a + b$ (the t_i are defined in Equations 1a–1d). However, only $t_{2c} = c\,(c\,(c\,a\,b)\,a)\,b$ and $t_{2d} = c\,a\,(c\,b\,(c\,a\,b))$ can be given a type in Π (T_l and T_r, resp.). They both belong to $\mathcal{A}(\mathscr{G}^+)$.

3.3 Multityped ACGs as Commutative Diagrams

This section shows that overloaded signatures are actually as expressive as simple signatures. Indeed, as illustrated in Figure 3a, an overloaded signature Π can be simulated with two simple ones linked with a relabeling $\mathscr{R}_\Pi : \Sigma_\Pi^{\mathscr{T}} \to \Sigma_\Pi^\Lambda$. As a result, overloading preserves properties of simple higher-order linear signatures.

We identify types that can occur at the same position in terms of $\Lambda(\Pi)$.

Definition 15 (Skeleton Equivalence). *Let* Π *be an overloaded signature. Then* $\sim_\Pi$ *is the smallest equivalence relation such that:*

- *if there exists a constant* $c \in \Pi$ *such that* $\alpha, \beta \in \tau^+(c)$*, then* $\alpha \sim_\Pi \beta$*;*

– *if* $\alpha_0 \multimap \alpha_1 \sim_\Pi \beta_0 \multimap \beta_1$*, then* $\alpha_0 \sim_\Pi \beta_0$ *and* $\alpha_1 \sim_\Pi \beta_1$*;*
– *if* $\alpha_0 \sim_\Pi \beta_0$ *and* $\alpha_1 \sim_\Pi \beta_1$*, then* $\alpha_0 \multimap \alpha_1 \sim_\Pi \beta_0 \multimap \beta_1$*;*

We say α *and* β *are* Π*-equivalent for* $\alpha \sim_\Pi \beta$*.*

Note that skeleton equivalence implies skeleton equality. Then, we define the relabeling, its domain and its codomain.

Definition 16 (Carriers). *Let* $\Pi = \langle A, C, \tau^+ \rangle$ *be an overloaded signature.*

– *The* type carrier $\Sigma_\Pi^{\mathscr{T}} = \langle A, C', \tau_0 \rangle$ *of* Π *is the simple signature where* $C' = \biguplus_{c \in C} \{c^\alpha \mid \alpha \in \tau^+(c)\}$ *and* $\tau_0 : c^\alpha \mapsto \alpha$*.*
– *The* term carrier $\Sigma_\Pi^{\Lambda} = \langle {}^{A}/_{\sim_\Pi}, C, \tau_1 \rangle$ *of* Π *is the simple signature where* $\tau_1 : c \mapsto [\alpha]_{\sim_\Pi}$ *and* $\alpha \in \tau_c^+$ *is any type of* c*;*
– *The* carrier relabeling $\mathscr{R}_\Pi : \Sigma_\Pi^{\mathscr{T}} \to \Sigma_\Pi^{\Lambda}$ *of* Π *is the relabeling such that* $\forall c^\alpha \in C', \mathscr{R}_\Pi(c^\alpha) = c$ *and* $\forall a \in A, \mathscr{R}_\Pi(a) = [a]_{\sim_\Pi}$*.*

The domain $\Sigma_\Pi^{\mathscr{T}}$ of $\mathscr{R}_\Pi$ is called the type carrier of Π, for it builds the same linear implicative types as Π. Moreover, to each constant c of Π corresponds $|\tau^+(c)|$ constants $c^\alpha : \alpha$ in $\Sigma_\Pi^{\mathscr{T}}$, one for each $\alpha \in \tau^+(c)$. Therefore, type-checking a term $u : \beta \in \Lambda(\Pi)$ corresponds to checking whether there is some $t : \beta \in \Lambda(\Sigma_\Pi^{\mathscr{T}})$ such that $\mathscr{R}_\Pi(t) = u$, as Lemma 2 shows.

The codomain Σ_Π^{Λ} of $\mathscr{R}_\Pi$ is called the term carrier of Π, for it builds the same linear λ-terms as Π. So that Σ_Π^{Λ} is a simple signature, its set of linear implicative types is the quotient of $\mathscr{T}_\Pi$ by the skeleton equivalence $\sim_\Pi$.

As for the carrier relabeling $\mathscr{R}_\Pi$, it maps the duplicated constants c^α to their generators c and the types α to their equivalence classes $[\alpha]_{\sim_\Pi}$.

Lemma 2 (Carriers). *Let* $\Pi = \langle A, C, \tau^+ \rangle$ *be an overloaded signature. Let* $\Sigma_\Pi^{\mathscr{T}} = \langle A, C', \tau_0 \rangle$ *be its type carrier,* $\Sigma_\Pi^{\Lambda} = \langle {}^{A}/_{\sim_\Pi}, C, \tau_1 \rangle$ *be its term carrier and* $\mathscr{R}_\Pi : \Sigma_\Pi^{\mathscr{T}} \to \Sigma_\Pi^{\Lambda}$ *be its carrier relabeling.*

Then $\Gamma \vdash_\Pi M : \alpha$ *iff there exists* $t \in \Lambda(C')$ *such that* $\mathscr{R}_\Pi(t) = M$ *and* $\Gamma \vdash_{\Sigma_\Pi^{\mathscr{T}}} t : \alpha$*.*

Lemma 3 proves that, in turn, any relabeling defines an overloaded signature satisfying a similar property.

Lemma 3 ($\mathscr{R}$-Overloaded Signature). *Let* $\Sigma_0 = \langle A_0, C_0, \tau_0 \rangle$ *and* $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ *be simple signatures and* $\mathscr{R} : \Sigma_0 \to \Sigma_1$ *a relabeling. Define the* signature overloaded with $\mathscr{R}$ *(or* $\mathscr{R}$-overloaded signature*) as* $\Pi_\mathscr{R} = \langle A_0, C_1, \tau^+ \rangle$ *the overloaded signature with* $\tau^+ : c \mapsto \tau_0(\mathscr{R}^{-1}(c))$*.*

Then $\Gamma \vdash_{\Pi_\mathscr{R}} M : \alpha$ *iff there exists* $t \in \Lambda(C_0)$ *such that* $\mathscr{R}(t) = M$ *and* $\Gamma \vdash_{\Sigma_0} t : \alpha$*.*

Notice the signature overloaded with the carrier relabeling of Π is Π, i.e., for any overloaded signature Π, $\Pi_{\mathscr{R}_\Pi} = \Pi$. Conversely, given the simple signature Σ_0 and Σ_1, and the relabeling $\mathscr{R} : \Sigma_0 \to \Sigma_1$, we have that Σ_0 and $\Sigma_{\Pi_\mathscr{R}}^{\mathscr{T}}$ are isomorphic, Σ_1 and $\Sigma_{\Pi_\mathscr{R}}^{\Lambda}$ are isomorphic, and $\mathscr{R}$ and $\mathscr{R}_{\Pi_\mathscr{R}}$ are isomorphic.

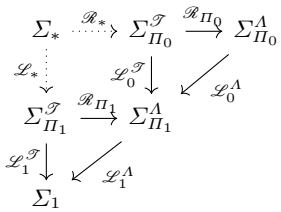
4 The Composition Problem

Lexicons of multityped ACGs map overloaded signatures to simple signatures. Therefore, composition of ACGs cannot be as simple as lexicon composition since their domains and codomains are of different kinds. We show that a well-behaved composition for mACGs can be defined using a pullback in a specific construction. This pullback turns out to be the construction provided by [8] to show ACGs are closed by intersection with regular languages. This section reminds this construction and shows that it is indeed a pullback.

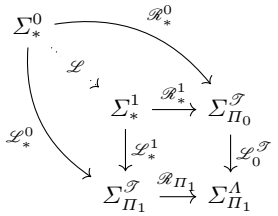
4.1 Composition as Pullback

In this section, consider two mACGs: $\mathscr{G}_0^+ = \langle \Pi_0, \Sigma_0, \mathscr{L}_0^+, S_0 \rangle$ on the one hand and $\mathscr{G}_1^+ = \langle \Pi_1, \Sigma_1, \mathscr{L}_1^+, S_1 \rangle$ on the other hand. We aim to provide a well-behaved definition of their composition $\mathscr{G}_1^+ \circ \mathscr{G}_0^+$.

Figure 3b illustrates the composition problem. We look to extract a triangle defining a lexicon, like in Figure 4a, from this diagram. The term lexicon of this triangle should be $\mathscr{L}_1^\Lambda \circ \mathscr{L}_0^\Lambda$.



(a) A solution to the composition problem with Σ_*, the lexicon $\mathscr{L}^{\mathscr{T}} = \mathscr{L}_1^{\mathscr{T}} \circ \mathscr{L}_*$, and the relabeling $\mathscr{R} = \mathscr{R}_{\Pi_0} \circ \mathscr{R}_*$



(b) A solution $\langle \Sigma^1_*, \mathscr{L}^1_*, \mathscr{R}^1_* \rangle$ is better than a solution $\langle \Sigma^0_*, \mathscr{L}^0_*, \mathscr{R}^0_* \rangle$ if there exists a unique $\mathscr{L} : \Sigma^0_* \to \Sigma^1_*$

Fig. 4: Solutions to the composition problem

A possibility is to find a simple signature Σ_* which is mapped to both type carriers $\Sigma^{\mathscr{T}}_{\Pi_0}$ and $\Sigma^{\mathscr{T}}_{\Pi_1}$. A lexicon $\mathscr{L}_* : \Sigma_* \to \Sigma^{\mathscr{T}}_{\Pi_1}$ means the overloading of Π_1 is accounted for in Σ_*. And, with a relabeling $\mathscr{R}_* : \Sigma_* \to \Sigma^{\mathscr{T}}_{\Pi_0}$, Σ_* is a finer type carrier than $\Sigma^{\mathscr{T}}_{\Pi_0}$. For Σ_*, $\mathscr{L}_*$, and $\mathscr{R}_*$ are not given but to be found, Figure 4a depicts them with dotted arrows. Thus, we obtain a new lexicon $\mathscr{L}^+$ from the $(\mathscr{R}_{\Pi_0} \circ \mathscr{R}_*)$-overloaded signature (see Lemma 3) to the simple signature Σ_1: the type lexicon of $\mathscr{L}^+$ is $\mathscr{L}_1^{\mathscr{T}} \circ \mathscr{L}_*$, and the term lexicon of $\mathscr{L}^+$ is $\mathscr{L}_1^\Lambda \circ \mathscr{L}_0^\Lambda$.

Now, a given situation of our composition problem may lead to several solutions for Σ_*, $\mathscr{L}_*$, and $\mathscr{R}_*$. To discriminate them, we say the solution triple $\langle \Sigma^1_*, \mathscr{L}^1_*, \mathscr{R}^1_* \rangle$ is a better solution than the solution triple $\langle \Sigma^0_*, \mathscr{L}^0_*, \mathscr{R}^0_* \rangle$ if there exists a unique simple lexicon $\mathscr{L} : \Sigma^0_* \to \Sigma^1_*$ such that $\mathscr{L}^0_* =_\beta \mathscr{L}^1_* \circ \mathscr{L}$ and $\mathscr{R}^0_* =_\beta \mathscr{R}^1_* \circ \mathscr{L}$. See Figure 4b for the corresponding diagram.

The best solution of the composition problem in Figure 4a with respect to the order in Figure 4b is actually called *the pullback*. Pullbacks are universal constructions in category theory [12]. The category at hand here is the one where objects are the (simple) signatures and arrows are the lexicons.

However, the pullback as the best solution to the composition problem in Figure 4a comes with two issues. First, the pullback may not exist in some situations. Second, the pullback is actually the minimum for the order in Figure 4b when the $\mathscr{R}_*^i$ are not restricted to relabelings (the arrows of our category are simple lexicons). As such, there is no reason *a priori* for $\mathscr{R}_*$ to be a relabeling in the pullback triple $\langle \Sigma_*, \mathscr{L}_*, \mathscr{R}_* \rangle$.

In Section 4.2, we revisit a signature construction by [8]. We name it *fiber product signature*, for this construction is built upon the fiber product of atomic types and linear implicative types, and the fiber product of constants and linear λ-terms. We show the fiber product signature is not only a solution to Figure 4a but, furthermore, the pullback Σ_*, and the associated $\mathscr{R}_*$ is indeed a relabeling.

Moreover, [8] shows the associated $\mathscr{L}_*$ generates the object language of $\mathscr{G}_0^+$, up to the relabeling $\mathscr{R}_{\Pi_1}$: $\mathcal{O}(\langle \Sigma_*, \Sigma_{\Pi_1}^{\mathscr{T}}, \mathscr{L}_*, \mathscr{R}_*^{-1}(S_0) \rangle) = \mathscr{R}_{\Pi_1}^{-1}(\mathcal{O}(\mathscr{G}_0^+))$

Thus, defining the lexicon $\mathscr{L}^+ = \langle \mathscr{L}_1^{\mathscr{T}} \circ \mathscr{L}_*, \mathscr{L}_1^{\Lambda} \circ \mathscr{L}_0^{\Lambda} \rangle$, the multityped ACGs $\mathscr{G}_0^+$ and $\mathscr{G}_1^+$ compose into the multityped ACG $\mathscr{G}_1^+ \circ \mathscr{G}_0^+ = \langle \Sigma_*, \Sigma_1, \mathscr{L}^+, \mathscr{R}_*^{-1}(S_0) \rangle$ (here Σ_* is seen as the $(\mathscr{R}_{\Pi_0} \circ \mathscr{R}_*)$-overloaded signature; see Lemma 3).

Notice that, similarly to simple ACGs, the object language of the composition is the image of the abstract language under the composition:

$$\mathcal{O}(\mathscr{G}_1^+ \circ \mathscr{G}_0^+) = \mathscr{L}_1^+(\mathcal{O}(\mathscr{G}_0^+)) = \mathscr{L}_1^+ \circ \mathscr{L}_0^+(\mathcal{A}(\mathscr{G}_0^+))$$

Example 8 (Composition of Multityped ACGs). Example 7 introduced a multityped ACG for which admissible abstract structures are left and right-branching trees. We now want to enforce these trees to be the derived trees of a tree-adjoining grammar. To this end, we follow the construction provided by [5] (not detailed here) and define $\Sigma_{\text{derivation}}$ with the atomic types $\{T_a, T\}$ (T_a stands for the place where adjunction is possible) and the constants of Table 3.

We then define the ACG $\mathscr{G}_{\text{der2t}} = \langle \Sigma_{\text{derivation}}, \Sigma_{\text{tree}}, \mathscr{L}_{\text{der2t}}, \{T\} \rangle$ where $\mathscr{L}_{\text{der2t}}$ is the lexicon of Table 3 (with the corresponding initial and auxiliary trees of the equivalent tree-adjoining grammar on the right-hand side). The (usual) composition of ACGs $\mathscr{G}_{\text{t2s}} \circ \mathscr{G}_{\text{der2t}}$ generates binary trees that are, in addition, the derived tree of a given tree-adjoining grammar.

For instance, the term $t_{2c} = c\,(c\,(c\,a\,b)\,a)\,b$ corresponding to the tree of Figure 2c has two antecedents by $\mathscr{G}_{\text{der2t}}$: $b_l\,I\,(a_r\,(a\,I)\,b_u)$ and $b_l\,I\,(b_l\,(a\,I)\,a_u)$. Both are, of course, antecedents of $a + b + a + b$ by $\mathscr{G}_{\text{t2s}} \circ \mathscr{G}_{\text{der2t}}$.

The construction of Section 4.2 will allow us to define the composition of the multityped ACG $\mathscr{G}^+$ and $\mathscr{G}_{\text{der2t}}$ (see Example 9 for the resulting lexicon), where, in order to avoid even more types and constants, $\Sigma_{\text{derivation}}$ can be considered as a trivial multityped ACG with the identity as relabeling. The admissible parse structures are derivation trees that are furthermore interpreted as left and right-branching derived trees.

$$a_r, b_l : T_a \multimap T \multimap T \qquad a_u, b_u : T \qquad a, b : T_a \multimap T_a \qquad I : T_a$$

$$T := T \qquad T_a := T \multimap T \qquad a_u := a \qquad b_u := b \qquad I := \lambda x.\, x$$

$$a_r := \lambda r\, t.\, r\,(c\, a\, t) \qquad b_l := \lambda r\, t.\, r\,(c\, t\, b) \qquad a := \lambda r\, t.\, r\,(c\, t\, a) \qquad b := \lambda r\, t.\, r\,(c\, b\, t)$$

Table 3: The signature $\Sigma_{\text{derivation}}$ and the lexicon $\mathscr{L}_{\text{der2t}}$

4.2 Kanazawa's Construction

[8] defines the construction of a simple ACG to demonstrate the closure of string-encoding (resp. tree-encoding) ACGs under intersection with regular languages (resp. regular tree languages). The construction actually generalizes to any simple signature that is the codomain of both a lexicon and a relabeling. In this paper, we name this construction the *fiber product* and introduce a dedicated notation.

Definition 17 (Fiber Product, [8]). *Let Σ, Σ_1, and $\Sigma_0 = \langle A_0, C_0, \tau_0\rangle$ be simple signatures. Let $\mathscr{L} : \Sigma_0 \to \Sigma$ be a simple lexicon and $\mathscr{R} : \Sigma_1 \to \Sigma$ a relabeling. Their* fiber product signature *is the signature $\Sigma_0 \times_\Sigma \Sigma_1 = \langle A, C, \tau\rangle$ with the lexicon $\mathscr{L}_* : \Sigma_0 \times_\Sigma \Sigma_1 \to \Sigma_1$ and the relabeling $\mathscr{R}_* : \Sigma_0 \times_\Sigma \Sigma_1 \to \Sigma_0$ such that:*

- *the set of atomic types of the fiber product $\Sigma_0 \times_\Sigma \Sigma_1$ is the fiber product of the atomic types of Σ_0 and the linear implicative types built upon Σ_1 over $\mathscr{L}$ and $\mathscr{R}$: $A = \{a^\beta \mid a \in A_0, \beta \in \mathscr{T}_{\Sigma_1}, \mathscr{L}(a) = \mathscr{R}(\beta)\}$*
- *the set of constants of the fiber product $\Sigma_0 \times_\Sigma \Sigma_1$ is the fiber product of the constants of Σ_0 and the linear λ-terms built upon Σ_1 over $\mathscr{L}$ and $\mathscr{R}$: $C = \{c_{N:\beta} \mid c \in C_0, \vdash_{\Sigma_1} N : \beta$ such that $\mathscr{L}(\vdash_{\Sigma_0} c : \tau_0(c)) = \mathscr{R}(\vdash_{\Sigma_1} N : \beta)\}$*
- *τ assigns a constant $c_{N:\beta}$ to $\text{anti}(\tau_0(c), \beta)$, a most specific common anti-instance of $\tau_0(c)$ and β, with $\text{anti}(\alpha_0 \multimap \alpha_1, \beta_0 \multimap \beta_1) = \text{anti}(\alpha_0, \beta_0) \multimap \text{anti}(\alpha_1, \beta_1)$ and $\text{anti}(a, \beta) = a^\beta$ for $a \in A_0$*
- *$\mathscr{L}_*$ forgets the baseline: $\forall a^\beta \in A, \mathscr{L}_*(a^\beta) = \beta$; $\forall c_{N:\beta} \in C, \mathscr{L}_*(c_{N:\beta}) = N$*
- *$\mathscr{R}_*$ forgets the adornement: $\forall a^\beta \in A, \mathscr{R}_*(a^\beta) = a$; $\forall c_{N:\beta} \in C, \mathscr{R}_*(c_{N:\beta}) = c$*

Example 9 (Fiber Product Signature Resulting from the Composition of mACGs in Example 8). Table 4 provides the signature $\Sigma_{\text{derivation}} \times \Sigma_{\text{tree}}$ as in Example 8. It follows the construction given in Definition 17 above with $\Sigma_0 = \Sigma_{\text{derivation}}$, $\Sigma_1 = \Sigma_{\text{lr-tree}}$, and $\Sigma = \Sigma_{\text{tree}}$. It corresponds to the diagram of Figure 5. Note that, in order to avoid having even more types and constants, we are in the special case where the top most signature $\Pi_0 = \Sigma_{\text{derivation}}$ is simple, i.e. the relabeling is the identity on $\Sigma_{\text{derivation}}$.

[8] showed that the fiber product signature makes the square diagram in Figure 6 commute: $\mathscr{L} \circ \mathscr{R}_* = \mathscr{R} \circ \mathscr{L}_*$. Thus, $\Sigma_0 \times_\Sigma \Sigma_1$ is a solution to our

$$
\begin{aligned}
a_{u\,a_u:T_u}, b_{u\,b_u:T_u} &: \mathrm{anti}(T, T_u) = T^{T_u} \\
a_{u\,a_l:T_l}, b_{u\,b_l:T_l} &: T^{T_l} \\
a_{u\,a_r:T_r}, b_{u\,b_r:T_r} &: T^{T_r} \\
a_{r\,\lambda r\,t.\,r\,(c_l\,a_l\,t):(T_l \multimap T_u) \multimap T_u \multimap T_u} &: T_a^{T_l \multimap T_u} \multimap T^{T_u} \multimap T^{T_u} \\
a_{r\,\lambda r\,t.\,r\,(c_l\,a_l\,t):(T_l \multimap T_l) \multimap T_u \multimap T_l} &: T_a^{T_l \multimap T_l} \multimap T^{T_u} \multimap T^{T_l} \\
a_{r\,\lambda r\,t.\,r\,(c_l\,a_l\,t):(T_l \multimap T_r) \multimap T_u \multimap T_r} &: T_a^{T_l \multimap T_r} \multimap T^{T_u} \multimap T^{T_r} \\
a_{r\,\lambda r\,t.\,r\,(c_r\,a_u\,t):(T_r \multimap T_u) \multimap T_r \multimap T_u} &: T_a^{T_r \multimap T_u} \multimap T^{T_r} \multimap T^{T_u} \\
a_{r\,\lambda r\,t.\,r\,(c_r\,a_u\,t):(T_r \multimap T_l) \multimap T_r \multimap T_l} &: T_a^{T_r \multimap T_l} \multimap T^{T_r} \multimap T^{T_l} \\
a_{r\,\lambda r\,t.\,r\,(c_r\,a_u\,t):(T_r \multimap T_r) \multimap T_r \multimap T_r} &: T_a^{T_r \multimap T_r} \multimap T^{T_r} \multimap T^{T_r} \\
b_{l\,\lambda r\,t.\,r\,(c_l\,t\,b_u):(T_l \multimap T_u) \multimap T_l \multimap T_u} &: T_a^{T_l \multimap T_u} \multimap T^{T_l} \multimap T^{T_u} \\
b_{l\,\lambda r\,t.\,r\,(c_l\,t\,b_u):(T_l \multimap T_l) \multimap T_l \multimap T_l} &: T_a^{T_l \multimap T_l} \multimap T^{T_l} \multimap T^{T_l} \\
b_{l\,\lambda r\,t.\,r\,(c_l\,t\,b_u):(T_l \multimap T_r) \multimap T_l \multimap T_r} &: T_a^{T_l \multimap T_r} \multimap T^{T_l} \multimap T^{T_r} \\
b_{l\,\lambda r\,t.\,r\,(c_r\,t\,b_r):(T_r \multimap T_u) \multimap T_u \multimap T_u} &: T_a^{T_r \multimap T_u} \multimap T^{T_u} \multimap T^{T_u} \\
b_{l\,\lambda r\,t.\,r\,(c_r\,t\,b_r):(T_r \multimap T_l) \multimap T_u \multimap T_l} &: T_a^{T_r \multimap T_l} \multimap T^{T_u} \multimap T^{T_l} \\
b_{l\,\lambda r\,t.\,r\,(c_r\,t\,b_r):(T_r \multimap T_r) \multimap T_u \multimap T_r} &: T_a^{T_r \multimap T_r} \multimap T^{T_u} \multimap T^{T_r} \\
a_{\lambda r\,t.\,r\,(c_l\,t\,a_u):(T_l \multimap T_u) \multimap T_l \multimap T_u} &: T_a^{T_l \multimap T_u} \multimap T_a^{T_l \multimap T_u} \\
a_{\lambda r\,t.\,r\,(c_l\,t\,a_u):(T_l \multimap T_l) \multimap T_l \multimap T_l} &: T_a^{T_l \multimap T_l} \multimap T_a^{T_l \multimap T_l} \\
a_{\lambda r\,t.\,r\,(c_l\,t\,a_u):(T_l \multimap T_r) \multimap T_l \multimap T_r} &: T_a^{T_l \multimap T_r} \multimap T_a^{T_l \multimap T_r} \\
a_{\lambda r\,t.\,r\,(c_r\,t\,a_r):(T_r \multimap T_u) \multimap T_u \multimap T_u} &: T_a^{T_r \multimap T_u} \multimap T_a^{T_u \multimap T_u} \\
a_{\lambda r\,t.\,r\,(c_r\,t\,a_r):(T_r \multimap T_l) \multimap T_u \multimap T_l} &: T_a^{T_r \multimap T_l} \multimap T_a^{T_u \multimap T_l} \\
a_{\lambda r\,t.\,r\,(c_r\,t\,a_r):(T_r \multimap T_r) \multimap T_u \multimap T_r} &: T_a^{T_r \multimap T_r} \multimap T_a^{T_u \multimap T_r} \\
b_{\lambda r\,t.\,r\,(c_l\,b_l\,t):(T_l \multimap T_u) \multimap T_u \multimap T_u} &: T_a^{T_l \multimap T_u} \multimap T_a^{T_u \multimap T_u} \\
b_{\lambda r\,t.\,r\,(c_l\,b_l\,t):(T_l \multimap T_l) \multimap T_u \multimap T_l} &: T_a^{T_l \multimap T_l} \multimap T_a^{T_u \multimap T_l} \\
b_{\lambda r\,t.\,r\,(c_l\,b_l\,t):(T_l \multimap T_r) \multimap T_u \multimap T_r} &: T_a^{T_l \multimap T_r} \multimap T_a^{T_u \multimap T_r} \\
b_{\lambda r\,t.\,r\,(c_r\,b_u\,t):(T_r \multimap T_u) \multimap T_r \multimap T_u} &: T_a^{T_r \multimap T_u} \multimap T_a^{T_r \multimap T_u} \\
b_{\lambda r\,t.\,r\,(c_r\,b_u\,t):(T_r \multimap T_l) \multimap T_r \multimap T_l} &: T_a^{T_r \multimap T_l} \multimap T_a^{T_r \multimap T_l} \\
b_{\lambda r\,t.\,r\,(c_r\,b_u\,t):(T_r \multimap T_r) \multimap T_r \multimap T_r} &: T_a^{T_r \multimap T_r} \multimap T_a^{T_r \multimap T_r} \\
I_{\lambda x.\,x:T_u \multimap T_u} &: T_a^{T_u \multimap T_u} \\
I_{\lambda x.\,x:T_l \multimap T_l} &: T_a^{T_l \multimap T_l} \\
I_{\lambda x.\,x:T_r \multimap T_r} &: T_a^{T_r \multimap T_r}
\end{aligned}
$$

Table 4: Constants of $\Sigma_{\text{derivation}} \times \Sigma_{\text{tree}}$

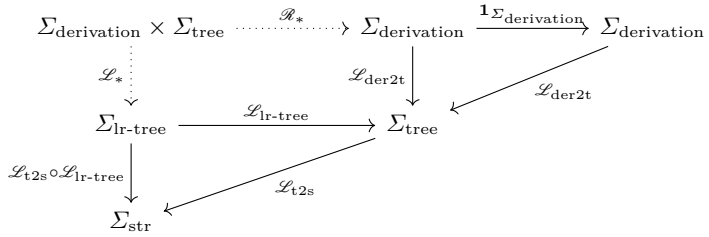


Fig. 5: Solution to the composition problem for Example 8

composition problem pictured in Figure 4a. For the reminder of this section, let consider an arbitrary fiber product signature as depicted in Figure 6.

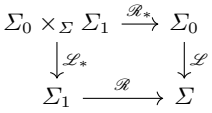


Fig. 6: The fiber product signature $\Sigma_0 \times_\Sigma \Sigma_1$

To prove the fiber product signature is the pullback, we prove the stronger property that, for any pair of Σ_0 and Σ_1-judgments mapped to the same Σ-judgment under $\mathscr{L}$ and $\mathscr{R}$, there exists a unique typing judgment built upon $\Sigma_0 \times_\Sigma \Sigma_1$ which is mapped to that pair under $\mathscr{R}_*$ and $\mathscr{L}_*$.

First, we prove this property for types. The unique type built upon the fiber product is the most specific common anti-instance built for the fiber product.

Lemma 4. *Let $\alpha \in \mathscr{T}_{\Sigma_0}$ and $\beta \in \mathscr{T}_{\Sigma_1}$ such that $\mathscr{L}(\alpha) = \mathscr{R}(\beta)$. Then $\gamma = \text{anti}(\alpha, \beta)$ is the unique $\gamma \in \mathscr{T}_{\Sigma_0 \times_\Sigma \Sigma_1}$ such that $\mathscr{R}_*(\gamma) = \alpha$ and $\mathscr{L}_*(\gamma) = \beta$.*

Proof. For the proof legibility, we will write $\text{anti}(\Gamma, \Delta)$ for the context $x_1 : \text{anti}(\alpha_1, \beta_1), \ldots, x_n : \text{anti}(\alpha_n, \beta_n)$ when $\Gamma = x_1 : \alpha_1, \ldots, x_n : \alpha_n$ and $\Delta = x_1 : \beta_1, \ldots, x_n : \beta_n$. In particular, $\text{anti}(\varnothing, \varnothing) = \varnothing$.

By induction over α.

When α is atomic, so is γ. By construction of the fiber product, $\gamma = \alpha^\beta = \text{anti}(\alpha, \beta)$.

When α is an arrow type, so is $\mathscr{R}(\beta)$. Since inverse relabelings preserve syntactic structures, so is β, and the induction hypothesis applies twice.

On typing judgments, the property actually has more interest when formulated up to β-equivalence.

Lemma 5. *Let $\pi_0 = \Gamma \vdash_{\Sigma_0} t : \alpha$ and $\pi_1 = \Delta \vdash_{\Sigma_1} N : \beta$ such that $\mathscr{L}(\pi_0) =_\beta \mathscr{R}(\pi_1)$. Then $\pi_* = \text{anti}(\Gamma, \Delta) \vdash_{\Sigma_0 \times_\Sigma \Sigma_1} M : \text{anti}(\alpha, \beta)$ is the unique π_* up to β-equivalence such that $\mathscr{R}_*(\pi_*) =_\beta \pi_0$ and $\mathscr{L}_*(\pi_*) =_\beta \pi_1$.*

Proof. By induction over the Σ_0-judgment.

When t is a constant, Γ is an empty context. Since $\mathscr{R}_*$ preserves structures, suitable judgments feature terms β-equivalent to a constant. By construction of the fiber product, $\vdash_{\Pi_0} c_{N:\beta} : \alpha^\beta$ is the unique solution.

When t is a variable x, then $N = x$, $\Gamma = x : \alpha$ and $\Delta = x : \beta$. Necessarily, $M = x$ and Lemma 4 applies.

When $t = \lambda x.\, u$, then $\alpha = \alpha_0 \multimap \alpha_1$ by inversion lemma. Since $\mathscr{R}_*$ preserves structures, M is β-equivalent to an abstraction over x. Since $\mathscr{R}$ preserves structures, $N =_\beta \lambda x.\, P$. By subject equivalence, $\Gamma \vdash_{\Sigma_1} \lambda x.\, P : \beta$. By inversion lemma, $\beta = \beta_0 \multimap \beta_1$. The induction hypothesis applies to $\Gamma, x : \alpha_0 \vdash_{\Sigma_0} u : \alpha_1$ with $\Delta, x : \beta_0 \vdash_{\Sigma_1} P : \beta_1$ and yields $\Theta, x : \gamma_0 \vdash_{\Sigma_0 \times_\Sigma \Sigma_1} Q : \gamma_1$. By construction of anti, $\gamma_0 \multimap \gamma_1 = \mathrm{anti}(\alpha, \beta)$.

When $t = u\, v$, then Lemma 1 yields $N =_\beta N_u\, N_v$. Since N is linear, subject equivalence applies: $\Delta \vdash_{\Sigma_1} N_u\, N_v : \beta$. Thus, the last rule in this judgment is an application and the induction hypothesis applies to both premises. It yields $\Theta_u \vdash_{\Sigma_0 \times_\Sigma \Sigma_1} M_u : \mathrm{anti}(\alpha' \multimap \alpha, \beta' \multimap \beta)$ and $\Theta_v \vdash_{\Sigma_0 \times_\Sigma \Sigma_1} M_v : \mathrm{anti}(\alpha', \beta')$. By construction of anti, the typing judgment $\Theta_u, \Theta_v \vdash_{\Pi_0} M_u\, M_v : \mathrm{anti}(\alpha, \beta)$ holds.

This property on typing judgments also holds for syntactic equality. One could even drop the linearity of the λ-terms, for this version does not need subject expansion.

Theorem 1. *The fiber product signature is the pullback.*

Proof. Let Σ' be a simple signature with two lexicons F and G such that $F \circ \mathscr{R} =_\beta G \circ \mathscr{L}$. From Lemma 5, we build the lexicon (F, G) (see Figure 4b). Given a Σ'-judgment $\Gamma \vdash_{\Sigma'} t : \alpha$, we define the lexicon $(F, G)(\Gamma \vdash_{\Sigma'} t : \alpha)$ as the unique $\Sigma_0 \times_\Sigma \Sigma_1$-judgment Lemma 5 builds from the premises $F(\Gamma \vdash_{\Sigma'} t : \alpha)$ and $G(\Gamma \vdash_{\Sigma'} t : \alpha)$. The lexicon (F, G) can take no other value, for it would go against the uniqueness proven in Lemma 5.

5 Conclusion

We have introduced multityped ACGs and showed how to compose them. The need to address the issue of the combinatorial explosion of parsing ambiguity by means of probabilistic or weighted ACGs motivated this extension. The proposed construction, as a fiber product, shows how the resulting ACG uses the overloadings of each of the ACG. For instance, in Figure 4a, Σ_* and the relabeling $\mathscr{R}_*$ depend both on $\mathscr{R}_{\Pi_1}$ and $\mathscr{R}_{\Pi_0}$. Since, as discussed in Section 2.2, the overloaded signatures seem to be the relevant place to introduce weights, it shows that the weights of a parse structure of an ACG $\mathscr{G}_1^+$, when considered as an object structure of an ACG $\mathscr{G}_0^+$, should be able to play a role in the parse structure of the resulting ACG $\mathscr{G}^+ = \mathscr{G}_1^+ \circ \mathscr{G}_0^+$. Further work will take advantage of this feature, which fully apply to higher-order ACGs as well, to explore ways towards weighted ACGs.

References

1. Barendregt, H.P.: The lambda calculus: Its syntax and semantics, Studies in Logic and the Foundations of Mathematics, vol. 103. North-Holland (1984)
2. Booth, T., Thompson, R.: Applying probability measures to abstract languages. IEEE Transactions on Computers **C-22**(5), 442–450 (1973). https://doi.org/10.1109/T-C.1973.223746
3. Chiang, D.: Statistical parsing with an automatically-extracted Tree Adjoining Grammar. In: Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics. pp. 456–463. Association for Computational Linguistics, Hong Kong (10 2000). https://doi.org/10.3115/1075218.1075276, ACL anthology: P00-1058
4. de Groote, P.: Towards Abstract Categorial Grammars. In: Proceedings of 39th Annual Meeting of the Association for Computational Linguistics. pp. 148–155 (2001), ACL anthology: P01-1033
5. 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). pp. 145–150. Università di Venezia (2002), ACL anthology: W02-2220
6. Fülöp, Z., Vogler, H.: Weighted tree automata and tree transducers. In: Droste, M., Kuich, W., Vogler, H. (eds.) Handbook of Weighted Automata, pp. 313–403. Springer Berlin Heidelberg, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01492-5_9
7. Joshi, A.K., Schabes, Y.: Tree-adjoining grammars. In: Rozenberg, G., Salomaa, A.K. (eds.) Handbook of formal languages, vol. 3, chap. 2. Springer (1997). https://doi.org/10.1007/978-3-642-59126-6_2
8. Kanazawa, M.: Abstract families of abstract categorial languages. Electronic Notes in Theoretical Computer Science **165**, 65–80 (2006). https://doi.org/10.1016/j.entcs.2006.05.037, proceedings of the 13th Workshop on Logic, Language, Information and Computation (WoLLIC 2006)
9. Maletti, A., Satta, G.: Parsing algorithms based on tree automata. In: Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09). pp. 1–12. Association for Computational Linguistics, Paris, France (10 2009), ACL anthology: W09-3801
10. Resnik, P.: Probabilistic Tree-Adjoining Grammar as a framework for statistical natural language processing. In: COLING 1992 Volume 2: The 14th International Conference on Computational Linguistics (1992). https://doi.org/10.3115/992133.992135, ACL anthology: C92-2065
11. Salomaa, A.: Probabilistic and weighted grammars. Information and Control **15**(6), 529–544 (1969). https://doi.org/10.1016/S0019-9958(69)90554-3
12. Simmons, H.: An Introduction to Category Theory. Cambridge University Press (2011). https://doi.org/10.1017/CBO9780511863226
13. Steedman, M., Baldridge, J.: Combinatory categorial grammar. In: Borsley, R., Börjars, K. (eds.) Non-Transformational Syntax: Formal and Explicit Models of Grammar, chap. 5. Wiley-Blackwell (2011)