Electronic Notes in Theoretical Computer Science 53 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume53.html 13 pages

The emptiness of intersection problem for languages of k-valued categorial grammars (classical and Lambek) is undecidable

Annie Foret [1]

IRISA-University of Rennes1
Rennes, FRANCE

Abstract

This paper is concerned with usual decidability questions on grammars for some classes of categorial grammars that arise in the field of learning categorial grammars. We prove that the emptiness of intersection of two langages is an undecidable problem for the following classes : k-valued classical categorial grammars, and k-valued Lambek categorial grammars, for each positive k.

1 Introduction

Categorial grammars have been studied in the domain of natural language processing, we focus here on classical (or basic) categorial grammars that were introduced in [1] and on Lambek categorial grammars [7] which are closely connected to linear logic introduced by Girard [3]. These grammars are lexicalized grammars that assign types (or categories) to the lexicon; they are called *k-valued*, when each symbol in the lexicon is assigned to at most k types; they are also called *rigid* when 1-valued. Such k-valued grammars are of particular interest in recent works on learnability [6] [11]. In this context, it is important to acquire a good understanding of the properties of the class of grammars in question.

In this paper we consider the problem of emptiness of intersection, that is given two k-valued categorial grammars G_1 and G_2, is the intersection of $L(G_1)$ and $L(G_2)$ empty? This usual question on grammars is also undecidable in general for categorial grammars since they correspond to the class of context-free grammars. We show that this problem remains

[1] Email: mailto:foret@irisa.fr

undecidable for k-valued grammars, for any $k \geq 1$ in particular when restricted to rigid grammars, that is for $k = 1$. This result indicates in particular that these subclasses are not trivial (wide). Our proof consists in an encoding of Post's correspondence problem inspired from the treatment for context-free languages [4]; it relies on a specific class introduced here as PCP-grammars, a subclass of unidirectional grammars, for which we establish several properties.

2 Background

2.1 Categorial Grammars

In this section, we introduce basic definitions. The interested reader may also consult [2,10,13,12] for further details.

Let Σ be a fixed alphabet.

Types. *Types* are constructed from Pr (set of *primitive types*) and two binary connectives / and \ . Tp denotes the set of types. Pr contains a *distinguished type*, written t, also called the *principal type*.

Classical categorial grammar. A *classical categorial grammar* over Σ is a finite relation G between Σ and Tp. If $< c, A > \in G$, we say that G *assigns* A to c, and we write $G : c \mapsto A$. We write $SubTp(G)$ the set of subformulas of types that are assigned by G to some symbol in Σ.

Notation. A sequence of types in Tp^* may be written using commas or concatenation or simple juxtaposition (this should not be confusing, since we consider grammars without product types).

Derivation $\vdash_{AB}$. The relation $\vdash_{AB}$ is the smallest relation $\vdash$ between Tp^+ and Tp, such that for all $\Gamma, \Delta \in Tp^+$ and for all $A, B \in Tp$:

$A \vdash A$
if $\Gamma \vdash A$ and $\Delta \vdash A \setminus B$ then $\Gamma, \Delta \vdash B$ (Backward application)
if $\Gamma \vdash B \,/\, A$ and $\Delta \vdash A$ then $\Gamma, \Delta \vdash B$ (Forward application)

We consider Lambek calculus restricted to the two binary connectives $\setminus$ and / .

We give a formulation consisting in introduction rules on the left and on the right of a sequent.

Lambek Derivation $\vdash_L$. The relation $\vdash_L$ is the smallest relation $\vdash$ between Tp^+ and Tp, such that for all $\Gamma \in Tp^+, \Delta, \Delta' \in Tp^*$ and for all $A, B \in Tp$ (Γ is non-empty) :

$A \vdash A$
if $A, \Gamma \vdash B$ then $\Gamma \vdash A \setminus B$ ($\setminus right$)
if $\Gamma, A \vdash B$ then $\Gamma \vdash B \,/\, A$ ($/right$)

if $\Gamma \vdash A$ and $\Delta, B, \Delta' \vdash C$ then $\Delta, \Gamma, A \setminus B, \Delta' \vdash C$ ($\setminus left$)
if $\Gamma \vdash A$ and $\Delta, B, \Delta' \vdash C$ then $\Delta, B \mathbin{/} A, \Gamma, \Delta' \vdash C$ ($/left$)

We recall that the cut rule is satisfied by both $\vdash_{AB}$ and $\vdash_L$.

Language. Let G be a classical categorial grammar over Σ. G *generates* a string $c_1 \ldots c_n \in \Sigma^+$ iff there are types $A_1, \ldots, A_n \in Tp$ such that : $G : c_i \mapsto A_i$ $(1 \leq i \leq n)$ and $A_1, \ldots, A_n \vdash_{AB} t$.

The *language of* G, is the set of strings generated by G and is denoted $L(G)$.

We define similarly $LL(G)$ by replacing $\vdash_{AB}$ with $\vdash_L$ in the definition of $L(G)$.

Rigid and k-valued grammars. Categorial grammars that assign at most k types to each symbol in the alphabet are called k-*valued grammars*; 1-valued grammars are also called *rigid* grammars.

Example 2.1 Let $\Sigma_1 = \{John, Mary, likes\}$ and let $Pr = \{t, n\}$ for sentences and nouns respectively.

Let $G_1 = \{John \mapsto n,\ Mary \mapsto n,\ likes \mapsto n \setminus (t \mathbin{/} n)\}$
We get $(John\ likes\ Mary \in L(G_1))$ since $(n,\ n \setminus (t \mathbin{/} n),\ n \vdash_{AB} t)$
G_1 is a rigid (or 1-valued) grammar.

2.2 Post's problem (PCP)

Post's correspondence problem (PCP in short) is a problem based on pairs of strings (see [4] or [8] for details). Let X be an alphabet (with two or more letters). *Post's correspondence problem* is to determine, given a finite sequence $D = < (u_1, v_1), ..., (u_k, v_k) >$ of pairs of non-empty strings on X, whether there exists a finite non-empty sequence of indices $i_1, ...i_m$ among $\{1, \ldots, k\}$ (with $m > 0$) such that :

$$u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}$$

Theorem 2.2 *Post's correspondence problem is undecidable.*

3 Encoding PCP into classical rigid categorial grammars

Given an instance $< (u_1, v_1), ..., (u_k, v_k) >$ of PCP, we construct two similar grammars : for the u_i's and for the v_i's. The key idea is to consider, for the first grammar, (similarly for the second one) any possible writing of a word as a succession of u_i's, and to encode it as a sequence of types with two parts Γ_1, Δ_1 such that Γ_1 encodes the entire word, Δ_1 encodes the decomposition using a succession of indices and corresponding u_i's and such that $\Gamma_1, \Delta_1 \vdash_{AB} t$.

We construct grammars that belong to a specific class of grammars (later called PCP-grammars).

3.1 A specific class of grammars and some properties

Let $\tilde{w}$ denote the miror image of word w and let $\tilde{\Gamma}$ denote the sequence of types of Γ in reverse order.

Rigid injective grammars. When the grammar G is rigid, let τ_G denote its type assignment on Σ ; we extend τ_G in a natural way on Σ^* by : $\tau_G(x_1 x_2 \dots x_q) = \tau_G(x_1), \tau_G(x_2), \dots, \tau_G(x_q)$ *where* $x_i \in \Sigma$
We also write for a set X of words : $\tau_G(X) = \{\tau_G(x) : x \in X\}$

For a rigid grammar, let us call the *grammar injective*, when the type assignment on Σ^* is injective.

Definition 3.1 [PCP-grammars] Let us call a *PCP-grammar*, a classical categorial grammar over an alphabet Σ, (with primitives types Pr, and a distinguished type t), that assigns types A (to symbols in Σ) only of the following shape :

$A = t_1 \setminus (t_2 \setminus (\dots t_{q-1} \setminus t_q))$ *where* $(q \geq 1)$ *and* $(\forall i : t_i \in Pr)$
where t_q is called the ***right-most type*** of A and $(t_r \setminus (\dots t_{q-1} \setminus t_q)$ are its *right-subformulas* (for $1 \leq r \leq q$).
We define *Lambek-PCP-grammars* similarly.

Definition 3.2 [Code-type] Given a non-empty sequence Γ of types A_i in Tp (not necessarily primitive), we associate to it a type written $C(\Gamma)$ called its *Code-type* defined as follows :

$C(A_1, A_2, \dots, A_{q-1}, A_q) = A_1 \setminus (A_2 \setminus (\dots A_{q-1} \setminus A_q))$
(with $C(A_1) = A_1$)

Example 3.3 [Using code-types] Let $Pr = \{a, b, 1, 2, t\}, u_1 = ab, u_2 = abb$, then $C(1u_12) = 1 \setminus (a \setminus (b \setminus 2))$. Note that using u_1 and $\tilde{u_1}$:

$$\tilde{u_1} 1 C(1u_1 2) \vdash_{AB} 2$$

and that if we iterate using u_2 and $\tilde{u_2}$ we get :

$$\tilde{u_2}\tilde{u_1} 1 C(1u_1 2) C(2u_2 t) \vdash_{AB} \tilde{u_2} 2 C(2u_2 t) \vdash_{AB} t$$

We shall iterate such situations so as to mimick PCP, using words, indices and delimiters.

Proposition 3.4 (Code-types) *Let G be a categorial grammar between Σ and Tp :*

(1) for $\Gamma \in Tp^*, \Gamma' \in Tp^+$ sequences of types (Γ' non-empty) :

$$\underbrace{\tilde{\Gamma}, C(\Gamma,}\Gamma') \vdash_{AB} C(\Gamma')$$

(2) for $k \geq 1, 1 \leq j \leq k, 1 \leq i \leq k+1, \Gamma_j \in Tp^*$ (possibly empty) and $A_i \in Tp$:

$$\underbrace{\tilde{\Gamma}_k, \tilde{\Gamma}_{k-1}, \dots, \tilde{\Gamma}_1}, A_1, \underbrace{C(A_1, \Gamma_1, A_2), \dots, C(A_{k-1}, \Gamma_{k-1}, A_k), C(A_k, \Gamma_k, A_{k+1})} \vdash_{AB} A_{k+1}$$

Notation. We use underbraces for ease of presentation only.

Proposition 3.5 (Rigid PCP-grammars) *Let G be a rigid categorial grammar between Σ and Tp, then :*

(3) if G is a rigid PCP-grammar then for $w \in \Sigma^+$ and $A \in Tp$:

$$\tau_G(w) \vdash_{AB} A \ \ implies \ \ A \in SubTp(G)$$

(4) if G is a rigid PCP-grammar and if A is not a strict right-subformula in $SubTp(G)$ then for $w \in \Sigma^+$:

$\tau_G(w) \vdash_{AB} A \ \ implies \ \ \tau_G(w) = A$

Proofs are given in Appendix.

3.2 Construction of the grammars encoding a PCP-instance

Let $D =< (u_1, v_1), ..., (u_n, v_n) >$ be an instance of PCP over a fixed alphabet $X = \{a, b\}$. Let $X' = Pr_D = X \cup \{1, \ldots, n\} \cup \{t, \#\}$ (numbers and # are intended as special marks). We associate to D, two grammars G_{1D} and G_{2D} over an alphabet Σ_D as follows :

$$\Sigma_D = \{c_a, c_b, c_\#\} \cup \{c_{i,j} : i \in \{1, \ldots, n\},\ j \in \{1, \ldots, n\} \cup \{t\}\} \\ \cup \{d_{i,j} : i \in \{1, \ldots, n\},\ j \in \{1, \ldots, n\} \cup \{t\}\}$$

Definition 3.6 We define G_{1D} as the following assignments, (where $u_i \in \{a, b\}^*$) :

$c_a \mapsto a$ $c_b \mapsto b$ $c_\# \mapsto \#$	$c_{i,j} \mapsto C(iu_ij) \ : \ for \ \ i \in \{1, \ldots, n\},\ j \in \{1, \ldots, n\} \cup \{t\}$ $d_{i,j} \mapsto C(\#u_ij) \ : \ for \ \ i \in \{1, \ldots, n\},\ j \in \{1, \ldots, n\} \cup \{t\}$

We define G_{2D} similarly, by exchanging the roles of all u_i and v_i.

Proposition 3.7 *G_{1D} and G_{2D} are both rigid injective PCP-grammars.*

Example 3.8 Let $D_1 =< (ab, abbb), (bb, b) >$ we get $Pr_{D_1} = \{a, b, 1, 2, t, \#\}$ and G_{1D_1} as follows :

$c_a \mapsto a$ $c_b \mapsto b$ $c_\# \mapsto \#$	$c_{1,1} \mapsto C(1ab1)$ $c_{1,2} \mapsto C(1ab2)$ $c_{1,t} \mapsto C(1abt)$ $c_{2,1} \mapsto C(2bb1)$ $c_{2,2} \mapsto C(2bb2)$ $c_{2,t} \mapsto C(2bbt)$	$d_{1,1} \mapsto C(\#ab1)$ $d_{1,2} \mapsto C(\#ab2)$ $d_{1,t} \mapsto C(\#abt)$ $d_{2,1} \mapsto C(\#bb1)$ $d_{2,2} \mapsto C(\#bb2)$ $d_{2,t} \mapsto C(\#bbt)$

We observe that $abbbbb$ admits two decompositions (ab.bb.bb=abbb.b.b) according to indices : 1, 2, 2. A correspondence between this solution and $L(G_{1D_1})$ is illustrated by the following derivation :

$$\begin{array}{llllllll}
w & = c_b c_b & c_b c_b & c_b c_a & c_\# & d_{1,2} & c_{2,2} & c_{2,t} \quad \in L(G_{1D_1}) \\
\tau_{G_{1D_1}}(w) & = bb & bb & ba & \# & C(\#ab2) & C(2bb2) & C(2bbt) \\
\tau_{G_{1D_1}}(w) & \vdash bb & bb & 2 & & & C(2bb2) & C(2bbt) \\
\tau_{G_{1D_1}}(w) & \vdash bb & 2 & & & & & C(2bbt) \\
\tau_{G_{1D_1}}(w) & \vdash t & & & & & &
\end{array}$$

The following technical proposition is useful to describe the languages of the above grammars.

Proposition 3.9 (Type descriptions) *Let G_{1D} be associated to D with type assignment τ_{1D} :*

(5) if $A \in \tau_{1D}(\Sigma_D)$ (ie A is an assigned type) then for $w \in \Sigma_D^+$:

$$\tau_{1D}(w) \vdash_{AB} A \ \ implies \ \ \tau_{1D}(w) = A$$

(6) if $A \notin \tau_{1D}(\Sigma_D)$ (ie A is not an assigned type) then for $w \in \Sigma_D^+$: [2]

$$\begin{array}{l}
\tau_{1D}(w) \vdash_{AB} A \ \ implies \\
\qquad \exists k > 0 \ \ \exists i_1, \ldots i_k \in \{1, \ldots, n\} \ \ \exists u' \ \ (possibly \ \ empty) \ : \\
\qquad \tau_{1D}(w) = \tilde{u}' \underbrace{\tilde{u}_{i_{k-1}} \ldots \tilde{u}_{i_1}} \# \underbrace{C(\# u_{i_1} i_2) \ldots C(i_{k-1} u_{i_{k-1}} i_k)} C(y_k u' A) \\
\qquad such \ \ that \ \ \exists t_p \ldots t_q \in Pr_D \ (1 \le p \le q) \ : \begin{cases} u_{i_k} = u' t_p \ldots t_{q-1} \\ A = C(t_p \ldots t_{q-1} t_q) \end{cases} \\
\qquad where \quad y_1 = \# \quad and \ \ if \ \ k > 1 : y_k = i_k
\end{array}$$

Proofs are given in Appendix; (5) is a corollary of (4) ; (6) is more technical (using (3) (4) (5)).

3.3 The correspondence

We now describe[3] the languages of G_{1D} and G_{2D} (with type-assignment τ_{1D} and τ_{2D}) associated to a PCP-instance $D =< (u_1, v_1), ..., (u_n, v_n) >$.

Proposition 3.10 (Language description) *The language $L(G_{1D}) = \{w : \tau_{1D}(w) \vdash_{AB} t\}$ associated to G_{1D} can be described as follows ($L(G_{2D})$ can be described similarly) :* [4]

$$\begin{array}{l}
L(G_{1D}) = \{w : \ \tau_{1D}(w) = \underbrace{\tilde{u}_{i_k} \tilde{u}_{i_{k-1}} \ldots \tilde{u}_{i_1}} \# \underbrace{C(\# u_{i_1} i_2)} \ldots \underbrace{C(i_{k-1} u_{i_{k-1}} i_k)} \underbrace{C(y_k u_{i_k} t)}, \\
\qquad \qquad and \ \ i_1, \ldots, i_k \in \{1, \ldots, n\}, \quad y_1 = \# \quad and \ \ if \ \ k > 1 : y_k = i_k\}
\end{array}$$

---

[2] in the degenerate case when $k = 1$, $\tau_{1D}(w)$ is as follows : $\tau_{1D}(w) = \tilde{u}' \# C(\# u' A)$

[3] proofs are corollaries of (2) and (6) : see Appendixx.

[4] in the degenerate case when $k = 1$, $\tau_{1D}(w)$ is as follows : $\tau_{1D}(w) = \tilde{u_{i_1}} \# C(\# u_{i_1} t)$

Note that $\tau_{1D}(w)$ consists in two main different parts separated by a $\#$ whose left part has no $\backslash$ operator and whose right part is made of code-types. The intended meaning is as follows : for a PCP-instance, the left part encodes the writing of a full word, while the right part encodes the succession of indices and the respective decompositions.

Proposition 3.11 (Simulation) *$L(G_{1D}) \cap L(G_{2D}) \neq \emptyset$ iff D is a positive instance of PCP.*

Corollary 3.12 (Main) *The emptiness of intersection problem for k-valued categorial grammars is undecidable for any $k \geq 1$ (in particular for rigid injective PCP-grammars).*

4 Extension to k-valued Lambek grammars

We show a similar result for k-valued Lambek grammars. This relies on the following property :

Proposition 4.1 *Let G denote a PCP-grammar, $\forall t_0 \in Pr$ (primitive) : $\tau_G(w) \vdash_{AB} t_0$ iff $\tau_G(w) \vdash_L t_0$*

Corollary 4.2 *For a PCP-grammar, $L(G)$ with respect to $\vdash_{AB}$ and $LL(G)$ with respect to $\vdash_L$ coincide.*

Corollary 4.3 (Main) *The emptiness of intersection problem for k-valued Lambek categorial grammars is undecidable for any $k \geq 1$ (in particular for rigid injective Lambek-PCP-categorial grammars).*

Note. This result seems to extend similarly to the non-associative version, but not to the commutative one.

This result clearly applies to the Lambek calculus with product [7] (by the sub-formula property and Cut elimination, the language of a PCP-grammar is the same for $\vdash_L$ with or without product). A similar argument also holds for $L\diamond$ [9,5] the Lambek calculus extended by a pair of residuation modalities ($L\diamond$ also enjoys the sub-formula property and Cut elimination).

5 Conclusion

This paper has answered a decidability question concerning each class of k-valued classical categorial grammars, and each class of k-valued Lambek grammars : the emptiness of intersection of two langages is an undecidable problem for each class. The proof relies on a specific class introduced here as PCP-grammars, a subclass of unidirectional grammars, for which we establish several properties. In particular, the problem we have focused on is undecidable for this subclass (thus not trivial).

For future work, we keep interested in closure, decidability and complexity issues concerning k-valued categorial grammars. In particular we leave open the decidability question of the inclusion problem.

References

[1] Y. Bar-Hillel. A quasi arithmetical notation for syntactic description. *Language*, 29:47–58, 1953.

[2] W. Buszkowski. Mathematical linguistics and proof theory. In van Benthem and ter Meulen [14], chapter 12, pages 683–736.

[3] Jean-Yves Girard. Linear logic: its syntax and semantics. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Notes*, pages 1–42. Cambridge University Press, 1995.

[4] John E. Hopcroft and Jeffrey Ullman. *Introduction to automata theory, languages and computation*. Addison Wesley, 1979.

[5] Gerhard Jager. On the generative capacity of multimodal categorial grammars. *To appear in Journal of Language and Computation*, 2001.

[6] Makoto Kanazawa. *Learnable classes of categorial grammars*. Studies in Logic, Language and Information. FoLLI & CSLI, 1998. distributed by Cambridge University Press.

[7] Joachim Lambek. The mathematics of sentence structure. *American mathematical monthly*, 65:154–169, 1958.

[8] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the theory of computation*. Prentice Hall, 1981.

[9] Michael Moortgat. Multimodal linguistic inference. *Journal of Logic, Language and Information, vol. 5, no 3/4, pp 349-385*, 1996.

[10] Michael Moortgat. Categorial type logic. In van Benthem and ter Meulen [14], chapter 2, pages 93–177.

[11] Jacques Nicolas. *Grammatical inference as unification*. Rapport de Recherche RR-3632, INRIA, 1999. With associated web site `http://www.inria.fr/RRRT/publications-eng.html`.

[12] Mati Pentus. Lambek grammars are context-free. In *Logic in Computer Science*. IEEE Computer Society Press, 1993.

[13] Christian Retoré. Calcul de lambek et logique linéaire. *Traitement Automatique des Langues*, 37(2):39–70, 1996.

[14] J. van Benthem and A. ter Meulen, editors. *Handbook of Logic and Language*. North-Holland Elsevier, Amsterdam, 1997.

A APPENDIX

Proof of (1) by induction on the length $|\Gamma| \geq 0$ of sequence Γ

- case $|\Gamma| = 0$ is $C(\Gamma') \vdash_{AB} C(\Gamma')$ that is an axiom
- case $|\Gamma| > 0$, let $\Gamma = \Gamma_1, A_1$ where A_1 is a type of Tp :

$$\tilde{\Gamma}, C(\Gamma, \Gamma') = A_1, \tilde{\Gamma}_1, C(\Gamma_1, A_1, \Gamma')$$

By induction applied on $|\Gamma_1|$, where $|\Gamma'| > 0$:

$$\tilde{\Gamma}_1, C(\Gamma_1, A_1, \Gamma') \vdash_{AB} \underbrace{C(A_1, \Gamma')}_{= A_1 \setminus C(\Gamma')}$$

From which, by backward application together with axiom $(A_1 \vdash_{AB} A_1)$:

$$A_1, \underbrace{\tilde{\Gamma}_1, C(\Gamma_1, A_1, \Gamma')}_{\vdash_{AB} A_1 \setminus C(\Gamma')} \vdash C(\Gamma')$$

which is the desired result

$\square$

Proof of (2) by induction on the number k of sequences Γ_j. For ease of presentation, let us write :
$\Delta_k = \tilde{\Gamma}_k, \tilde{\Gamma}_{k-1}, \ldots, \tilde{\Gamma}_1, A_1, C(A_1, \Gamma_1, A_2), \ldots, C(A_{k-1}, \Gamma_{k-1}, A_k), C(A_k, \Gamma_k, A_{k+1})$
then (2) also rewrites to $\Delta_k \vdash_{AB} A_{k+1}$.

- case $k = 1$ is a subcase of (1) with $A_2 = C(A_2)$: $\tilde{\Gamma}_1, A_1, C(A_1, \Gamma_1, A_2) \vdash A_2$
- case $k > 1$, by induction for $k - 1$: $\Delta_{k-1} \vdash_{AB} A_k$, that is :

$$\underbrace{\tilde{\Gamma}_{k-1}, \ldots, \tilde{\Gamma}_1}, A_1, \underbrace{C(A_1, \Gamma_1, A_2), \ldots, C(A_{k-1}, \Gamma_{k-1}, A_k)} \vdash_{AB} A_k$$

by backward application with axiom $A_k \vdash_{AB} A_k$: [5]

$$\underbrace{\Delta_{k-1}}_{\vdash_{AB} A_k}, \underbrace{C(A_k, \Gamma_k, A_{k+1})}_{= A_k \setminus C(\Gamma_k, A_{k+1})} \vdash_{AB} C(\Gamma_k, A_{k+1})$$

by (1) where $C(A_{k+1}) = A_{k+1}$:

$$\underbrace{\tilde{\Gamma}_k, C(\Gamma_k}, A_{k+1}) \vdash_{AB} A_{k+1}$$

then by CUT on $C(\Gamma_k, A_{k+1})$:

$$\tilde{\Gamma}_k, \underbrace{\Delta_{k-1}, C(A_k, \Gamma_k, A_{k+1})}_{\vdash_{AB} C(\Gamma_k, A_{k+1})} \vdash_{AB} A_{k+1}$$

which is a writing of the desired result $\Delta_k \vdash_{AB} A_{k+1}$ [6]

$\square$

[5] where $C(A_k, \Gamma_k, A_{k+1}) = A_k \setminus C(\Gamma_k, A_{k+1})$
[6] when Γ_k is empty $C(\Gamma_k, A_{k+1})$ is $C(A_{k+1})$

Proof of (3) (τ_G is written as τ) by easy induction on the length $|\mathcal{D}|$ of a derivation $\mathcal{D}$ ending in $\tau(w) \vdash_{AB} A$

- case $|\mathcal{D}| = 0$, it is an axiom with $\tau(w) = A$ and clearly $w \in \Sigma$ therefore $A \in SubTp(G)$
- case $|\mathcal{D}| > 0$, if the last rule is forward application : then the induction hypothesis would lead to a type with $/$ in $SubTp(G)$ which is not possible for PCP-grammars.
- case $|\mathcal{D}| > 0$, if the last rule is backward application : the antecedents of $\mathcal{D}$ are of the form, where $\tau(w) = \Gamma, \Delta$ and $\exists w_1, w_2 \in \Sigma^+ : \tau(w_1) = \Gamma, \tau(w_2) = \Delta$:

  $$\Gamma \vdash A_1 \text{ and } \Delta \vdash A_1 \setminus A$$

  by induction hypothesis, $A_1 \setminus A \in SubTp(G)$ which implies $A \in SubTp(G)$ by definition of SubTp

$\square$

Proof of (4) (τ_G is written as τ) by induction on the length $|\mathcal{D}|$ of a derivation $\mathcal{D}$ ending in $\tau(w) \vdash_{AB} A$

- case $|\mathcal{D}| = 0$, it is an axiom, and clearly $\tau(w) = A$
- case $|\mathcal{D}| > 0$, the last rule of $\mathcal{D}$ is backward application (as in (3) , forward application is not possible) the antecedents of $\mathcal{D}$ are of the form :

  $$\Gamma \vdash A_1 \text{ and } \Delta \vdash A_1 \setminus A$$

  where Γ and Δ are non-empty and $\Gamma, \Delta = \tau(w)$; but in this case $\exists w_1 : \tau(w_1) = \Delta$ and by (3) : $A_1 \setminus A \in SubTp(G)$ hence A would be a strict right-subformula in $SubTp(G)$, which is not possible by assumption

$\square$

Proof of (5) this is a particular case of (4) specialized to grammars G_{1D}, such that by construction : if $A \in \tau_{1D}(\Sigma_D)$ ($\{a, b, \#\}$ if primitive) then A is not a strict right-subformula in $SubTp(G_{1D})$ $\square$

Proof of (6) by induction on the length $|\mathcal{D}|$ of a derivation $\mathcal{D}$ of $\tau_{1D}(w) \vdash A$; suppose $A \notin \tau_{1D}(\Sigma_D)$

- case $|\mathcal{D}| = 0$, it is an axiom : $\tau_{1D}(w) = A$, which implies $w \in \Sigma_D$ but this is impossible since A is not an assigned type.
- case $|\mathcal{D}| > 0$, the last rule of $\mathcal{D}$ is backward application, (as in (3) , forward application is not possible) the antecedents of $\mathcal{D}$ are of the form :

  $$\Gamma \vdash A_1 \text{ and } \Delta \vdash A_1 \setminus A \qquad \text{with } \Gamma, \Delta = \tau_{1D}(w)$$

  by (3) $A_1 \setminus A \in SubTp(G_{1D})$ and by construction of G_{1D} : $A_1 \in (Pr_D - \{t\}) = \{a, b\} \cup \{1, \ldots, n\} \cup \{\#\}$.

  We now discuss according to whether $A_1 \in \tau_{1D}(\Sigma_D)$ or not.
  · subcase $A_1 \in \tau_{1D}(\Sigma_D)$ (it is also primitive), then $A_1 \in X \cup \{\#\} =$

$\{a, b, \#\}$ and by (4) we get (since $\Gamma = \tau_{1D}(w_1)$ for some prefix w_1 of w)

$$\Gamma = A_1$$

—On the other hand, if $A_1 \neq \#$ by induction hypothesis (6) applied to $\Delta \vdash A_1 \setminus A$ we get :

$$\exists k > 0 \ \ \exists i_1, \ldots i_k \in \{1, \ldots, n\} \ \ \exists u'_1 \ \ (possibly \ \ empty) \ :$$
$$\Delta = \tilde{u_1}' \underbrace{\tilde{u}_{i_{k-1}} \ldots \tilde{u}_{i_1}} y_1 \underbrace{C(y_1 u_{i_1} y_2) \ldots C(y_{k-1} u_{i_{k-1}} y_k)} C(y_k u'_1 \ A_1 \setminus A)$$

$$where \ \ \exists t_p \ldots t_q \in Pr_D \ : \begin{cases} u_{i_k} = u'_1 t_p \ldots t_{q-1} \\ A_1 \setminus A = C(t_p \ldots t_{q-1} t_q) \\ 1 \leq p \leq q \end{cases}$$

$$y_1 = \# \ \ , \ \ and \ \ (\forall i \in \{2, \ldots k\} : y_j = i_j)$$

For ease of presentation, let us write :
$\Delta_k = \underbrace{\tilde{u}_{i_{k-1}} \ldots \tilde{u}_{i_1}} y_1 \underbrace{C(y_1 u_{i_1} y_2) \ldots C(y_{k-1} u_{i_{k-1}} y_k)}$ (with $\Delta_1 = y_1$)
we then rewrite :

$$\Delta = \tilde{u_1}' \Delta_k C(y_k u'_1 \ A_1 \setminus A)$$

we first observe that $A_1 = t_p$, and $A = C(t_{p+1} \ldots t_{q-1} t_q)$ with $1 \leq p+1 \leq q$;
then by adjoining $\Gamma = A_1$, if we let $u' = u'_1 A_1 = u'_1 t_p$ we get the desired result as follows :

$$\tau_{1D}(w) = \underbrace{\Gamma}_{=A_1}, \Delta = \underbrace{\tilde{u}'}_{A_1 \tilde{u_1'}} \Delta_k \qquad \underbrace{C(y_k u' A)}_{=C(y_k u'_1 A_1 A) = C(y_k u'_1 A_1 \ \setminus \ A)}$$

$$where \ \begin{cases} u_{i_k} = u' t_{p+1} \ldots t_{q-1} = u'_1 t_p \ldots t_{q-1} \\ A = C(t_{p+1} \ldots t_{q-1} t_q) \\ 1 \leq p+1 \leq q \end{cases}$$

— If $A_1 = \#$, by construction $\# \setminus A \in SubTp(G_{1D})$ is an assigned type and by (5) we have $\Delta = A_1 \setminus A$, therefore :

$$\Gamma, \Delta = \#, \# \setminus A$$

which is a particular (degenerate) case of (6) where $u' = \epsilon$ and $A = u_{i_1} t_q$ (by construction) for some u_{i_1} in the D instance and some primitive t_q.
· subcase $A_1 \notin \tau_{1D}(\Sigma_D)$, we have already $A_1 \in (Pr_D - \{t\})$ and $A_1 \setminus A \in SubTp(G_{1D})$ then A_1 is a number, and by construction $A_1 \setminus A \in \tau_{1D}(\Sigma_D)$ with shape $C(i u_i j)$ where $i \in \{1 \ldots n\}, j \in \{1 \ldots n\} \cup \{t\}$ and u_i from the given PCP-instance D ; by (4) we then get (since $\Delta = \tau_{1D}(w_2)$ for some suffix w_2 of w) :

$$\Delta = A_1 \setminus A$$

On the other hand the induction hypothesis applied to A_1 gives, where we use Δ_k as in previous case :

$$\exists k > 0 \;\; \exists i_1, \ldots i_k \in \{1, \ldots, n\} \;\; \exists u'_1 \;\; (possibly \;\; empty) \; :$$
$$\Gamma = \tilde{u_1}' \Delta_k C(y_k u'_1 \; A_1)$$
$$where \;\; \exists t'_{p'} \ldots t'_{q'} \in Pr_D \; : \begin{cases} u_{i_k} = u'_1 t'_{p'} \ldots t'_{q'-1} \\ A_1 = C(t'_{p'} \ldots t'_{q'-1} t'_{q'}) \\ 1 \leq p' \leq q' \end{cases}$$
$$y_1 = \# \;\; , \;\; and \;\; (\forall i \in \{2, \ldots k\} : y_j = i_j)$$

A_1 being a number, we first observe that $A_1 = t'_1$, $p' = q' = 1$ and $u_{i_k} = u'_1$; then by adjoining $\Delta = A_1 \setminus A$, let us write $i_{k+1} = y_{k+1} = A_1$, $u' = \epsilon$ (empty), and let $t_1 \ldots t_q \in Pr_D$ be such that $A = C(t_1 \ldots t_q)$ (possible and unique by construction) and let $u_{i_{k+1}} = u_i = t_1 \ldots t_{q-1}$, we then get the desired result (involving $k+1$ instead of k) as follows :

$$\Gamma, \Delta = \underbrace{\tilde{u}'}_{=\epsilon} \tilde{u}_{i_k} \Delta_k \underbrace{C(y_k u_{i_k} y_{k+1})}_{=C(y_k u'_1 A_1)} \underbrace{C(y_{k+1} u' A)}_{=C(A_1 A)}$$
$$where \;\; \exists t_p, \ldots t_q \in Pr_D \; : \begin{cases} u_{i_{k+1}} = u' t_1 \ldots t_{q-1} \\ A = C(t_1 \ldots t_{q-1} t_q) \\ 1 = p \leq q \end{cases}$$
$$y_1 = \# \;\; , \;\; and \;\; (\forall i \in \{2, \ldots k, k+1\} : y_j = i_j)$$

□

Proof of proposition 3.10. On the one hand all such strings w are in $L(G_{1D})$ (ie $\tau_{1D}(w) \vdash_{AB} t$) : by property (2) above where $A_{k+1} = t$; $A_1 = \#; A_2 = i_2; \ldots ; A_k = i_k$ and $\Gamma_j = u_{i_j}$ for $1 \leq j \leq k$.

Conversely, suppose w is a string in $L(G_{1D})$ that is we have a deduction for $\tau_{1D}(w) \vdash t$. The result is obtained by property (6) above where $A = t \notin \tau_{1D}(\Sigma_D)$ (and $p = q$ with $u_{i_k} = u'$) □

Proof of proposition 3.11. For ease of presentation, for any finite sequence of indices $s = i_1, \ldots i_p$, we write

$$c_{<s>} = d_{i_1,i_2} c_{i_2,i_3} \ldots c_{i_l,i_{l+1}} \ldots c_{i_{(p-1)},i_p} c_{i_p,t}$$

We may describe the languages equivalently as follows :

$L(G_{1D}) = \{\tilde{w_0} \; c_\# c_{<i_1,\ldots i_f>} : i_1, \ldots i_f \in \{1 \ldots n\} \; with \; \tau_{1D}(w_0) = u_{i_1} u_{i_2} .. u_{i_f} \in X^+\}$
$L(G_{2D}) = \{\tilde{w'_0} \; c_\# c_{<i'_1,\ldots i'_{f'}>} : i'_1, \ldots i'_{f'} \in \{1 \ldots n\}, \tau_{2D}(w'_0) = v_{i'_1} v_{i'_2} .. v_{i'_{f'}} \in X^+\}$

- If $w \in L(G_{1D}) \cap L(G_{2D})$, then there exists $i_1, \ldots i_f, i'_1, \ldots i'_{f'} \in \{1 \ldots n\}$ such that

$$w = \tilde{w_0} \; c_\# c_{<i_1,\ldots i_f>} = \tilde{w'_0} \; c_\# c_{<i'_1,\ldots i'_{f'}>}$$

  where $\tau_{1D}(w_0) = u_{i_1} u_{i_2} .. u_{i_f}$ and $\tau_{2D}(w'_0) = v_{i'_1} v_{i'_2} .. v_{i'_{f'}}$

  which gives $c_{<i_1,\ldots i_f>} = c_{<i'_1,\ldots i'_{f'}>}$, that is the two sequences of indices

are equal, and $w_0 = w'_0$ with :

$$\tau_{1D}(w_0) = u_{i_1}u_{i_2}..u_{i_f} = \tau_{2D}(w'_0) = v_{i_1}v_{i_2}..v_{i_f}$$

hence D is positive instance of PCP.

- Conversely, let us suppose there exists $i_1, ...i_f \in \{1 \dots n\}$ such that $u_{i_1}u_{i_2}..u_{i_f} = v_{i_1}v_{i_2}\dots v_{i_f}$ then let w_0 be the word on alphabet $\{c_a, c_b\}$ such that $\tau_{1D}(w_0) = u_{i_1}u_{i_2}..u_{i_f}$ then clearly $\tau_{1D}(w_0) = \tau_{2D}(w_0)$, hence :

$$\tilde{w_0}\ c_{\#}c_{<i_1,...i_f>} \in L(G_{1D}) \cap L(G_{2D})$$

$\square$

Proof of proposition 4.1 by induction. Clearly if $\tau(w) \vdash_{AB} t_0$ then $\tau(w) \vdash_L t_0$.

We show the following generalized converse :
if $\Gamma_0 \vdash_L t_0$ where Γ_0 consists in types of $SubTp(G)$ only then $\Gamma_0 \vdash_{AB} t_0$.
We proceed easily by induction on the length of deduction.

- axiom case : $\Gamma_0 = t_0$, it is also an axiom for $\vdash_{AB}$.
- rule $\backslash right$ is impossible since $t_0 \in Pr$
- rules $/left$ and $/right$ are never possible here due to the *subformula property* of Lambek calculus and since / does not occurr in SubTp(G) of a PCP-grammar.
- rule $\backslash left$ with conclusion

$$\underbrace{\Delta, \Gamma, A \setminus B, \Delta'}_{=\Gamma_0} \vdash t_0$$

and antecedents

$$\Gamma \vdash A \;\; and \;\; \Delta, B, \Delta' \vdash t_0$$

where $\Delta, \Gamma, A \setminus B, \Delta' = \Gamma_0$.
Clearly $A \in Pr$ since $\Gamma_0 \subseteq SubTp(G)^*$ is assumed , with in particular $A \setminus B \in SubTp(G)$.
We may then apply the induction hypothesis to both antecedents, where A and t_0 are primitive :

$$\Gamma \vdash_{AB} A \;\; and \;\; \Delta, B, \Delta' \vdash_{AB} t_0$$

From which we get the result by $\backslash left$ for $\vdash_{AB}$

$\square$