*Electronic Notes in Theoretical Computer Science 53 (2002)*

*URL:* [*http://www.elsevier.nl/locate/entcs/volume53.html*](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 gram- mars. We prove that the emptiness of intersection of two langages is an undecid- able problem for the following classes : k-valued classical categorial grammars, and k-valued Lambek categorial grammars, for each positive k.

## 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 gram- mars are lexicalized grammars that assign types (or categories) to the lexi- con; 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 *G1* and *G2*, is the intersec- tion of *L(G1)* and *L(G2)* 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*](mailto:foret@irisa.fr)

c 2002 Published by Elsevier Science B. V. *Open access under* [*CC BY-NC-ND license.*](http://creativecommons.org/licenses/by-nc-nd/3.0/)

undecidable for k-valued grammars, for any *k 1* in particular when re- stricted to rigid grammars, that is for *k = 1*. This result indicates in partic- ular 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.

## Background

* 1. *Categorial Grammars*

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

Letbe a fixed alphabet.

**Types.** *Types* are constructed from *Pr* (set of *primitive types*) and two bi- nary connectives *=* and *n* . *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* overis a finite relation *G* betweenand *T p*. If *< c; A >2 G*, we say that *G assigns A* to *c*, and we write *G : c 7! A*. We write *SubT p(G)* the set of subformulas of types that are assigned by *G* to some symbol in.

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

**Derivation** *` AB* **.** The relation *`AB* is the smallest relation *`* between *T p+*

and *T p*, such that for all *; 2 T p+* and for all *A; B 2 Tp* :

*A ` A*

if *` A* and *` A n B* then *; ` B* (Backward application) if *` B = A* and *` A* then *; ` B* (Forward application)

We consider Lambek calculus restricted to the two binary connectives

*n* and *=* .

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

**Lambek Derivation** *` L***.** The relation *`L* is the smallest relation *`* between

*T p+* and *T p*, such that for all *2 T p+; ; 0 2 T p* and for all *A; B 2 Tp*

(is non-empty) :

*A ` A*

if *A; ` B* then *` A n B* ( *nright* ) if *;A ` B* then *` B = A* ( *=right* )

if *` A* and *; B; 0 ` C* then *; ;A n B; 0 ` C* ( *nlef t* ) if *` A* and *; B; 0 ` C* then *;B = A; ; 0 ` C* ( *=lef t* )

We recall that the cut rule is satisfied by both *`AB* and *`L*.

**Language.** Let *G* be a classical categorial grammar over. *G generates* a

*1*

*n*

*1*

*n*

*i*

string *c*

*::: c*

*2 +* iff there are types *A ;::: ;A*

*2 Tp* such that : *G : c 7!*

*Ai (1 i n)* and *A1;::: ; An ` 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 *` AB* with *` 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 *1 = fJohn; Mary; likesg* and let *Pr = ft; ng* for sen- tences and nouns respectively.

Let *G1 = fJohn 7! n; Mary 7! n; likes 7! n n (t= n)g*

We get *(John likes Mary 2 L(G1))* since *(n; n n (t= n); n `AB t) G1* is a rigid (or 1-valued) grammar.

* 1. *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 =< (u1; v1); :::; (uk ; vk ) >* of pairs of non-empty strings on *X*, whether there exists a finite non-empty sequence of indices *i1; :::im* among *f1;::: ; kg* (with *m > 0*) such that:

*ui1 ui2 ::: uim = vi1 vi2 ::: vim*

**Theorem 2.2** *Post’s correspondence problemis undecidable.*

## Encoding PCP into classical rigid categorial grammars

Given an instance *< (u1; v1); :::; (uk ; vk ) >* of PCP, we construct two simi- lar grammars : for the *ui*’s and for the *vi*’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 *ui*’s, and to encode it as a sequence of types with two parts *1; 1* such that *1* encodes the entire word, *1* encodes the decomposition using a succession of indices and corresponding *ui*’s and such that *1; 1 `AB t*.

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

* 1. *A specific class of grammars and some properties*

Let *w~* denote the miror image of word *w* and let *~* denote the sequence of types ofin reverse order.

**Rigid injective grammars.** When the grammar *G* is rigid, let *G* denote its type assignment on; we extendin a natural way onby :

*G*

*G(x1x2 ::: xq) = G(x1); G(x2);::: ; G(xq) where xi 2*

We also write for a set *X* of words : *G(X) = f G(x) : x 2 Xg*

For a rigid grammar, let us call the *grammar injective*, when the type assignment onis injective.

**Definition 3.1** [PCP-grammars] Let us call a *PCP-grammar*, a classical cat- egorial grammar over an alphabet, (with primitives types *P r*, and a dis- tinguished type *t*), that assigns types *A* (to symbols in) only of the fol- lowing shape :

*A = t1 n (t2 n (::: tq 1 n tq)) where (q 1) and (8i : ti 2 P r)*

where *tq* is called the *right-most type* of *A* and *(tr n (::: tq 1 n tq)* are its

*right-subformulas* (for *1 r q*).

We define *Lambek-PCP-grammars* similarly.

**Definition 3.2** [Code-type] Given a non-empty sequenceof types *Ai* in *Tp* (not necessarily primitive), we associate to it a type written *C( )* called its *Code-type* defined as follows :

*C(A1; A2;::: ; Aq 1; Aq) = A1 n (A2 n (::: Aq 1 n Aq))*

(with *C(A1) = A1)*

**Example 3.3** [Using code-types] Let *Pr = fa; b; 1; 2; tg; u1 = ab; u2 = abb*, then *C(1u12) = 1 n (a n (b n 2))*. Note that using *u1* and *u~1* :

*u~11C(1u12) `AB* *2*

and that if we iterate using *u2* and *u~2* we get :

*u~2u~11C(1u12)C(2u2t) `AB u~22C(2u2t) `AB t*

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

**Proposition 3.4 (Code-types)** *Let G be a categorial grammar between*

*and Tp :*

* + 1. *for 2 T p ; 0 2 T p+ sequences of types ( 0 non-empty) :*

*~; C( ; 0 ) ` C( 0 )*

*AB*

*|{z}*

* + 1. *for k 1, 1 j k, 1 i k + 1, 2 T p (possibly empty) and Ai 2 Tp :*

*j*

*~k; ~k 1;::: ; ~1; A1;C(A1; 1; A2);::: ;C(Ak 1; k 1; Ak);C(Ak; k; Ak+1) `AB Ak+1*

*| {z } | {z }*

**Notation.** We use underbraces for ease of presentation only.

**Proposition 3.5 (Rigid PCP-grammars)** *Let G be a rigid categorial gram- mar between and T p, then:*

* + 1. *if G is a rigid PCP-grammar then for w 2 + and A 2 Tp :*

*G(w) `AB A implies A 2 SubT p(G)*

* + 1. *if G is a rigid PCP-grammar and if A is not a strict right-subformula in SubT p(G) then for w 2 + :*

*G(w) `AB A implies G(w) = A*

Proofs are given in Appendix.

* 1. *Construction of the grammars encoding a PCP-instance*

Let *D =< (u1; v1); :::; (un; vn) >* be an instance of PCP over a fixed alphabet

*X = fa; bg*. Let *X0 = Pr = X [ f1;::: ; ng[ ft; #g* (numbers and *#* are

*D*

intended as special marks). We associate to *D*, two grammars *G1D* and

*G2D* over an alphabet *D* as follows :

*D = fca; cb; c#g [ fci;j : i 2 f1;::: ; ng; j 2 f1;::: ; ng[ ftgg*

*[ fdi;j : i 2 f1;::: ; ng; j 2 f1;::: ; ng[ ftgg*

**Definition 3.6** We define *G1D* as the following assignments, (where *ui* *2*

*fa; bg )* :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *ca cb*  *c#* | *7!*  *7!*  *7!* | *a b*  *#* | *ci;j di;j* | *7!*  *7!* | *C(iuij) C(#uij)* | *:*  *:* | *f f* | *or*  *or* | *i i* | *2*  *2* | *f1;:::*  *f1;:::* | *;*  *;* | *ng;*  *ng;* | *j j* | *2*  *2* | *f1;:::*  *f1;:::* | *;*  *;* | *ng[*  *ng[* | *ftg ftg* |

We define *G2D* similarly, by exchanging the roles of all *ui* and *vi*. **Proposition 3.7** *G1D and G2D are both rigid injective PCP-grammars.* **Example 3.8** Let *D1 =< (ab; abbb); (bb; b) >* we get *P rD1 = fa; b; 1; 2; t; #g*

and *G1D1* as follows :

|  |  |  |
| --- | --- | --- |
|  | *c1;1 7! C(1ab1)* | *d1;1 7! C(#ab1)* |
| *ca 7! a*  *cb 7! b c# 7! #* | *c1;2 7! C(1ab2)*  *c1;t 7! C(1abt)*  *c2;1 7! C(2bb1)*  *c2;2 7! C(2bb2)* | *d1;2 7! C(#ab2)*  *d1;t 7! C(#abt)*  *d2;1 7! C(#bb1)*  *d2;2 7! C(#bb2)* |
|  | *c2;t 7! C(2bbt)* | *d2;t 7! 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(G1D1 )* is illustrated by the following derivation :

*w = cbcb cbcb cbca c# d1;2 c2;2 c2;t 2 L(G1D1 )*

*G1D (w) = bb bb ba # C(#ab2) C(2bb2) C(2bbt)*

*1*

*G1D (w) ` bb bb 2 C(2bb2) C(2bbt)*

*1*

*G1D (w) ` bb 2 C(2bbt)*

*1*

*G1D (w) ` t*

*1*

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

**Proposition 3.9 (Type descriptions)** *Let G1D be associated to D with type assignment 1D :*

1. *if A 2 1D( D) (ie A is an assigned type) then for w 2 + :*

*D*

*1D(w) `AB A implies 1D(w) = A*

1. *if A 62 1D( D) (ie A is not an assigned type) then for w 2 + : 2*

*D*

*1D(w) `AB A implies*

*9k > 0 9i1;::: ik 2 f1;::: ; ng 9u (possibly empty) :*

*0*

*0 0*

*1D(w) = u~ u~ik 1*

*::: u~i1 # C(#ui1 i2) ::: C(ik 1ui*

*k 1*

*ik) C(yku A)*

# *| {z }*

*:*

*such that 9tp ::: tq 2 P rD (1 p q) :*

*| {z 8<u*

*= u0 t} :::t*

*A = C(tp ::: tq 1tq)*

*ik*

*p*

*q 1*

*where y1 = # and if k > 1 : yk = ik*

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

* 1. *The correspondence*

We now describe *3* the languages of *G1D* and *G2D* (with type-assignment

*1D* and *2D*) associated to a PCP-instance *D =< (u1; v1); :::; (un; vn) >*.

**Proposition 3.10 (Language description)** *The language L(G1D) = fw :*

*1D(w) `AB tg associated to G1D can be described as follows ( L(G2D) can be described similarly) : 4*

*L(G1D) = fw : 1D(w) = u~ik u~ik*

*1 ::: u~i1 # C(#ui1 i2) ::: C(ik 1uik*

*1 ik) C(ykuik t);*

*| {z }*

*| {z }*

*| {z } | {z }*

*and i1;::: ; ik 2 f1;::: ; ng; y1 = # and if k > 1 : yk = ikg*

*2* in the degenerate case when *k = 1*, *(w)* is as follows : *(w) = u~0 #C(#u0 A)*

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

*1D 1D*

*1D*

*i1*

*i1*

*4* in the degenerate case when *k = 1*,

*1D*

*(w)* is as follows :

*(w) = u~ #C(#u t)*

Note that *1D (w)* consists in two main different parts separated by a *#* whose left part has no *n* 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(G1D ) \ L(G2D ) 6= ; 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 1 (in particular for rigid injective PCP-grammars).*

## 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, 8t0 2 Pr (primitive) :*

*G(w) `AB t0 iff G(w) `L t0*

**Corollary 4.2** *For a PCP-grammar, L(G) with respect to `AB and LL(G)*

*with respect to `L coincide.*

**Corollary 4.3 (Main)** *The emptiness of intersection problem for k-valued Lambek categorial grammars is undecidable for any k 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 *`L* with or without product). A similar argument also holds for *L* [9,5] the Lambek calculus extended by a pair of residu- ation modalities (*L* also enjoys the sub-formula property and Cut elimi- nation).

## Conclusion

This paper has answered a decidability question concerning each class of k-valued classical categorial grammars, and each class of k-valued Lam- bek grammars : the emptiness of intersection of two langages is an unde- cidable problem for each class. The proof relies on a specific class intro- duced 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 com- plexity 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.

1. W. Buszkowski. Mathematical linguistics and proof theory. In van Benthem and ter Meulen [14], chapter 12, pages 683–736.
2. 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.
3. John E. Hopcroft and Jeffrey Ullman. *Introduction to automata theory, languages and computation*. Addison Wesley, 1979.
4. Gerhard Jager. On the generative capacity of multimodal categorial grammars. *To appear in Journal of Language and Computation*, 2001.
5. Makoto Kanazawa. *Learnable classes of categorial grammars*. Studies in Logic, Language and Information. FoLLI & CSLI, 1998. distributed by Cambridge University Press.
6. Joachim Lambek. The mathematics of sentence structure. *American mathematical monthly*, 65:154–169, 1958.
7. Harry R. Lewis and Christos H. Papadimitriou. *Elements of the theory of computation*. Prentice Hall, 1981.
8. Michael Moortgat. Multimodal linguistic inference. *Journal of Logic, Language and Information, vol. 5, no 3/4, pp 349-385*, 1996.
9. Michael Moortgat. Categorial type logic. In van Benthem and ter Meulen [14], chapter 2, pages 93–177.
10. 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*.](http://www.inria.fr/RRRT/publications-eng.html)
11. Mati Pentus. Lambek grammars are context-free. In *Logic in Computer Science*. IEEE Computer Society Press, 1993.
12. Christian Retore´. Calcul de lambek et logique line´aire. *Traitement Automatique des Langues*, 37(2):39–70, 1996.
13. 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 *j j 0* of sequence

case *j j = 0* is *C( 0) ` C( 0)* that is an axiom

*AB*

case *j j > 0*, let *= 1; A1* where *A1* is a type of *Tp* :

*~; C( ; 0) = A ; ~ ; C( ;A ; )*

*0*

*1 1 1 1*

By induction applied on *j j*, where *j 0j > 0* :

*1*

*~ ; C( ;A ; ) `*

*0*

*C(A ; )*

*1 1 1*

*AB* *1*

*| {z }*

*0*

*= A1 n C( 0 )*

From which, by backward application together with axiom *(A1 `AB A1)*

:

*A ; ~ ; C( ;A ; ) ` C( )*

*0 0*

*1 1 1 1*

which is the desired result

*| {z }*

*`AB A1 n C( 0 )*

*2*

**Proof of (2)** by induction on the number *k* of sequences *j*. For ease of presentation, let us write :

*k = ~k; ~k 1;::: ; ~1; A1;C(A1; 1; A2);::: ;C(Ak 1; k 1; Ak);C(Ak; k; Ak+1)*

then (2) also rewrites to *k `AB Ak+1*.

case *k = 1* is a subcase of (1) with *A2 = C(A2)* : *~1; A1; C(A1; 1; A2) ` A2*

case *k > 1*, by induction for *k 1* : *k 1 `AB Ak*, that is:

*~k 1;::: ; ~1; A1; C(A1; 1; A2);::: ; C(Ak 1; k 1; Ak) `AB Ak*

*| {z } | {z }*

by backward application with axiom *Ak `AB Ak* : *5*

*k 1 ; C(Ak; k; Ak+1) `AB C( k; Ak+1)*

*| {z }*

*`AB Ak*

*| {z }*

*= Ak n C( k ;Ak+1)*

by (1) where *C(Ak+1) = Ak+1* :

*~k; C( k; Ak+1) `AB Ak+1*

*| {z }*

then by CUT on *C( k; Ak+1)* :

*~k; k 1; C(Ak; k; Ak+1) `AB Ak+1*

*| {z }*

*`AB C( k ;Ak+1)*

which is a writing of the desired result *k `AB Ak+1* *6*

*2*

*5* where *C(Ak ; k ; Ak+1 ) = Ak n C( k ; Ak+1 )*

*6* when *k* is empty *C( k ; Ak+1)* is *C(Ak+1)*

**Proof of (3)** ( *G* is written as) by easy induction on the length *jDj* of a derivation *D* ending in *(w) `AB A*

case *jDj = 0*, it is an axiom with *(w) = A* and clearly *w 2* therefore

*A 2 SubT p(G)*

case *jDj > 0*, if the last rule is forward application : then the induction hypothesis would lead to a type with *=* in *SubT p(G)* which is not possi- ble for PCP-grammars.

case *jDj > 0*, if the last rule is backward application : the antecedents

of *D* are of the form, where *(w) = ;* and *9w ;w 2 + : (w ) =*

*; (w2) =* :

*1 2 1*

*` A1* and *` A1 n A*

by induction hypothesis, *A1 n A 2 SubT p(G)* which implies *A 2 SubT p(G)*

by definition of SubTp

*2*

**Proof of (4)** ( *G* is written as) by induction on the length *jDj* of a deriva- tion *D* ending in *(w) `AB A*

case *jDj = 0*, it is an axiom, and clearly *(w) = A*

case *jDj > 0*, the last rule of *D* is backward application (as in (3) , forward application is not possible) the antecedents of *D* are of the form :

*` A1* and *` A1 n A*

whereandare non-empty and *; = (w)* ; but in this case *9w1 :*

*(w1) =* and by (3) : *A1 n A 2 SubT p(G)* hence *A* would be a strict right-subformula in *SubT p(G)*, which is not possible by assumption

*2*

**Proof of (5)** this is a particular case of (4) specialized to grammars *G1D* , such that by construction : if *A 2 1D ( D )* (*fa; b; #g* if primitive) then *A* is not a strict right-subformula in *SubT p(G1D )* *2*

**Proof of (6)** by induction on the length *jDj* of a derivation *D* of *1D (w) ` A* ; suppose *A 62 1D ( D )*

case *jDj = 0*, it is an axiom : *1D (w) = A*, which implies *w 2 D* but this is impossible since *A* is not an assigned type.

case *jDj > 0*, the last rule of *D* is backward application, (as in (3) , for- ward application is not possible) the antecedents of *D* are of the form :

*` A1* and *` A1 n A* with *; = 1D (w)*

by (3) *A1 n A 2 SubT p(G1D )* and by construction of *G1D* : *A1 2 (P rD ftg) = fa; bg[ f1;::: ; ng[ f#g*.

We now discuss according to whether *A1 2 1D ( D )* or not.

subcase *A1 2 1D ( D )* (it is also primitive), then *A1 2 X [ f#g =*

*fa; b; #g* and by (4) we get (since *= 1D(w1)* for some prefix *w1* of *w*)

*= A1*

—On the other hand, if *A1 6= #* by induction hypothesis (6) applied to

*` A1 n A* we get :

*9k > 0 9i1;::: ik 2 f1;::: ; ng 9u1 (possibly empty) :*

*0*

*0 0*

*= u~1 u~ik 1 ::: u~i1 y1 C(y1ui1 y2) ::: C(yk 1uik 1 yk) C(yku1 A1 n A)*

# *| {z }*

*| 8>u*

*= u0{tz ::: t }*

*ik 1 p*

*q 1*

*where 9tp ::: tq 2 P rD : <A1 n A = C(tp ::: tq 1tq)*

*>*

*>>1 p q*

*:*

*y1 = # ; and (8i 2 f2;::: kg : yj = ij)*

For ease of presentation, let us write :

*k = u~ik*

*1 ::: u~i1 y1 C(y1ui1 y2) ::: C(yk 1uik*

*1 yk)* (with *1 = y1*)

we the*|*n re*{*w*z*rite*}*:

# *| {z }*

*0 0*

*= u~1 kC(yku1 A1 n A)*

we first observe that *A1 = tp*, and *A = C(tp+1 ::: tq 1tq)* with *1 p +1 q*;

then by adjoining *= A* , if we let *u0 = u0 A*

*= u0 t*

we get the desired

*1*

result as follows :

*1 1 1 p*

*1D(w) = ; = u~*

*0*

*0*

*k C(yku A)*

*|={Az1}*

*k*

*1*

*1*

*k*

*1*

*1*

*|A{zu~0}*

*1*

*1*

# *| {z }*

*=C(y u0 A A)=C(y u0 A*

*n A)*

*8>uik*

*1 p*

*= u0 t*

*p+1*

*::: t*

*q 1*

*= u0 t*

*::: t*

*q 1*

*where*

*<A = C(tp+1 ::: tq 1tq)*

*>>:1 p +1 q*

— If *A1 = #*, by construction *# n A 2 SubT p(G1D)* is an assigned type and by (5) we have *= A1 n A*, therefore :

*; = #; # n A*

which is a particular (degenerate) case of (6) where *u0 =* and *A = u t*

*i1 q*

(by construction) for some *ui1* in the *D* instance and some primitive *tq*.

subcase *A1 62 1D( D)*, we have already *A1 2 (P rD ftg)* and *A1 n A 2*

*SubT p(G1D)* then *A1* is a number, and by construction *A1 n A 2 1D( D)*

with shape *C(iuij)* where *i 2 f1 ::: ng;j 2 f1 ::: ng [ ftg* and *ui* from the given PCP-instance *D* ; by (4) we then get (since *= 1D(w2)* for some suffix *w2* of *w*) :

*= A1 n A*

On the other hand the induction hypothesis applied to *A1* gives, where we use *k* as in previous case :

*9k > 0 9i1;::: ik 2 f1;::: ; ng 9u1 (possibly empty) :*

*0*

*0 0*

*= u~1 kC(yku1 A1)*

*8>uik*

*0 0*

*= u t*

*1 p0*

*0*

*q0 1*

*::: t*

*where 9t*

*::: t*

*2 Pr : <*

*0 0 0*

*p0 q0 D A1 = C(tp0 ::: tq0 1tq0 )*

*0*

*0*

*>>:1 p0 q0*

*y1 = # ; and (8i 2 f2;::: kg : yj = ij)*

*A1* being a number, we first observe that *A1 = t* , *p = q = 1* and

*0 0 0*

*1*

*1*

*uik*

*= u0* ; then by adjoining *= A*

*n A*, let us write *i*

*k+1*

*= yk+1*

*= A1*,

*u0 =* (empty), and let *t*

*1*

*::: tq*

*2 P rD*

be such that *A = C(t1*

*::: tq)*

(possible and unique by construction) and let *uik+1 = ui = t1 ::: tq 1*, we then get the desired result (involving *k +1* instead of *k*) as follows :

*1*

*; = u~ u~ik k C(ykuik yk+1) C(yk+1u A)*

*0*

*0*

*|{=z }*

*| =C(y{zu0 A1) }|*

*=C{(Az1 A) }*

*k* *1*

# *>>8*

*uik+1*

*= u0t*

*::: tq 1*

*where 9tp;::: tq 2 P rD : <A = C(t1 ::: tq 1tq )*

*1*

*:>>*

*1 = p q*

*y1 = # ; and (8i 2 f2;::: k;k + 1g : yj = ij)*

*2*

**Proof of proposition 3.10.** On the one hand all such strings *w* are in *L(G1D)*

(ie *1D(w) `AB t*) : by property (2) above where *Ak+1 = t* ; *A1 = #; A2 = i2; ::: ; Ak = ik* and *j = uij* for *1 j k*.

Conversely, suppose *w* is a string in *L(G1D)* that is we have a deduction for *1D(w) ` t*. The result is obtained by property (6) above where *A = t 62*

*( )* (and *p = q* with *u = u0* ) *2*

*1D D ik*

**Proof of proposition 3.11.** For ease of presentation, for any finite se- quence of indices *s = i1; :::ip*, we write

*c<s> = di1 ;i2 ci2 ;i3 ::: cil;il+1 ::: ci(p 1);ip cip;t*

We may describe the languages equivalently as follows :

*L(G1D ) = fw~0 c#c<i ;:::i > : i1; :::if 2 f1 ::: ng with 1D(w0) = ui ui ::ui 2 X+g*

*1 f 1 2 f*

*~0 0 0 0 +*

*L(G2D ) = fw0 c#c<i0 ;:::i0 > : i 1; :::i f0 2 f1 ::: ng; 2D(w0) = vi0 1 vi0 2 ::vi0 0 2 X g*

*1 f 0 f*

If *w 2 L(G ) \ L(G*

*)*, then there exists *i ; :::i ; i0 ; :::i0*

*2 f1 ::: ng* such

*1D 2D*

that

*1 f 1 f 0*

*~0*

*w = w~0 c#c<i1 ;:::i > = w0 c#c<i0 ;:::i0 >*

where

*f 1 f 0*

*(w ) = u u ::u* and *(w0 ) = v v ::v*

*0 0*

*0*

*1D* *0*

*i1 i2 if*

*2D* *0*

*i 1 i 2*

*i f 0*

which gives *c<i1;:::i > = c<i0 ;:::i0 >*, that is the two sequences of indices

*f 1 f 0*

are equal, and *w0 = w* with :

*0*

*0*

*1D(w0) = ui1 ui2 ::uif = 2D(w0) = vi1 vi2 ::vif*

hence *D* is positive instance of PCP.

*0*

Conversely, let us suppose there exists *i1; :::if 2 f1 ::: ng* such that *ui ui ::ui =*

*1 2 f*

*vi1 vi2 ::: vif* then let *w0* be the word on alphabet *fca; cbg* such that *1D(w0) =*

*ui1 ui2 ::uif*

then clearly *1D(w0) = 2D(w0)*, hence :

*w~0 c#c<i1 ;:::if > 2 L(G1D ) \ L(G2D )*

*2*

**Proof of proposition 4.1** by induction. Clearly if *(w) `AB t0* then *(w) `L t0*.

We show the following generalized converse :

if *0 `L t0* where *0* consists in types of *SubT p(G)* only then *0 `AB t0*. We proceed easily by induction on the length of deduction.

axiom case : *0 = t0*, it is also an axiom for *`AB*.

rule *nright* is impossible since *t0 2 Pr*

rules *=lef t* 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 *nlef t* with conclusion

and antecedents

*; ;A n B; ` t0*

*| {z }*

*0*

*= 0*

*` A and ; B; ` t0*

*0*

where *; ;A n B; 0 =* .

*0*

Clearly *A 2 Pr* since *SubT p(G)* is assumed , with in particular

*0*

*A n B 2 SubT p(G)*.

We may then apply the induction hypothesis to both antecedents, where

*A* and *t0* are primitive :

*`AB A and ; B; `AB t0*

*0*

From which we get the result by *nlef t* for *`AB*

*2*