Analyse statique des interactions entre structures élémentaires d'une grammaire

Guy Perrier
LORIA, Université de Lorraine,
équipe Sémagramme, bât. C,
Campus Scientifique
BP 239
54506 Vandøeuvre-lès-Nancy, cedex,France
guy.perrier@loria.fr

RÉSUMÉ

Nous nous intéressons ici à la construction semi-automatique de grammaires computationnelles et à leur utilisation pour l'analyse syntaxique. Nous considérons des grammaires lexicalisées dont les structures élémentaires sont des arbres, sous-spécifiés ou pas. Nous présentons un algorithme qui vise à prévoir l'ensemble des arbres élémentaires attachés aux mots qui peuvent s'intercaler entre deux mots donnés d'une phrase, dont on sait que les arbres élémentaires associées sont des compagnons, c'est-à-dire qu'ils interagiront nécessairement dans la composition syntaxique de la phrase.

ABSTRACT

Static Analysis of Interactions between Elementary Structures of a Grammar

We are interested in the semi-automatic construction of computational grammars and in their use for parsing. We consider lexicalized grammars with elementary structures which are trees, underspecified or not. We present an algorithm that aims at foreseeing all elementary trees attached at words which can come between two given words of a sentence, whose associated elementary trees are companions, that is, they will necessarily interact in the syntactic composition of the sentence.

MOTS-CLÉS : grammaire lexicalisée, grammaire d'interaction, construction de grammaires.

KEYWORDS: Lexicalized Grammar, Interaction Grammar, Grammar Construction.

1 Introduction

Nous poursuivons ici un travail commencé depuis plus de dix ans autour de la construction semi-automatique de grammaires computationnelles. Dans le cadre du formalisme des Grammaires d'Interaction (GI)(Guillaume et Perrier, 2009), nous avons développé FRIGRAM [1], une grammaire du français, et LEOPAR [2], un analyseur syntaxique pour les GI, permet d'appliquer cette grammaire à l'analyse de textes en français.

Notre ambition est d'obtenir une grammaire à large couverture pour analyser des corpus tout

1. http://wikilligramme.loria.fr/doku.php?id=frigram:frigram
2. http://leopar.loria.fr

venant. Même si nous sommes ouverts à intégrer des méthodes probabilistes dans notre approche, nous souhaitons conserver une base symbolique pour être en mesure de produire des analyses suffisamment riches pour que l'on puisse calculer à partir d'elles des représentations sémantiques complètes.

Nous devons faire face à un premier défi, celui de maintenir la cohérence d'une grammaire qui est nécessairement de taille importante. Certes, l'organisation d'une telle grammaire sous forme d'une hiérarchie de modules à l'aide d'une relation d'héritage facilite la tâche, mais cela ne résoud pas tout. Par ailleurs dans l'analyse syntaxique, nous sommes confrontés à un second défi, celui de l'explosion du nombre de structures syntaxiques candidates pour l'analyse d'une phrase.

Pour répondre à ces deux défis, nous pensons qu'il est utile d'analyser la grammaire de façon systématique pour prévoir les interactions entre les structures élémentaires qui la définissent. Un travail a commencé à être mené sur FRIGRAM mais il peut s'étendre aux grammaires construites dans d'autres formalismes, pour peu que ces grammaires soient lexicalisées.

Lorsque l'on analyse une phrase avec une grammaire lexicalisée, la première étape consiste à assigner à chaque mot de la phrase une structure syntaxique élémentaire de la grammaire. On obtient ce qu'on appelle une *sélection lexicale*. Le nombre de sélections lexicales possibles est exponentiel par rapport à la longueur de la phrase.

Pour filtrer les sélections lexicales, (Bonfante *et al.*, 2009) ont introduit la notion de *compagnon*. Un compagnon d'une structure syntaxique élémentaire est une structure syntaxique élémentaire qui peut se combiner avec la première dans la composition syntaxique d'une phrase. Le principe de filtrage est ensuite le suivant : si dans une sélection lexicale, il existe une structure syntaxique élémentaire qui ne trouve ni compagnon à gauche ni compagnon à droite, la sélection peut être éliminée. L'application de ce principe permet de réduire drastiquement le nombre de sélections lexicales. Les compagnons de chaque structure syntaxique élémentaire peuvent être pré-calculés sur la grammaire et pour réduire le nombre de calculs, ceux-ci sont effectués sur les structures syntaxiques élémentaires avant ancrage par des mots particuliers.

La faiblesse du principe de filtrage fondé sur les compagnons est qu'il est totalement indifférent aux contraintes de localité. Ainsi, si un mot du début d'une longue phrase trouve le compagnon de la structure syntaxique qu'il ancre auprès d'un mot qui est en fin de phrase, quelle que soit la longueur de la phrase, le principe est respecté.

C'est pour pallier cet inconvénient que nous proposons d'aller plus loin dans l'analyse statique des interactions entre structures syntaxiques élémentaires de la grammaire. Considérant un couple particulier de compagnons, nous proposons un algorithme qui permet de prévoir uniquement d'après la grammaire les structures syntaxiques élémentaires qui peuvent s'intercaler entre ces compagnons dans la composition syntaxique d'une phrase. Ce calcul devrait nous permettre d'aller plus loin dans le filtrage des sélections lexicales par application du principe suivant : si dans une sélection lexicale, nous sommes sûrs que deux mots ont leurs structures syntaxiques qui sont compagnons, nous devons vérifier que tous les structures syntaxiques ancrant les mots intermédiaires sont dans l'ensemble pré-calculé selon notre algorithme.

Dans la section 2, nous préciserons le concept de *compagnon*. Dans la section 3, nous décrirons l'algorithme de calcul des structures syntaxiques élémentaires s'intercalant entre deux compagnons et dans la section 4, nous déroulerons l'algorithme sur un exemple.

2 Les compagnons d'une structure syntaxique élémentaire

Nous nous situons dans le cadre de formalismes grammaticaux où les objets manipulés sont des structures syntaxiques notées *SSynt*. Parmi, celles-ci, nous distinguons les structures finales qui sont celles représentant la syntaxe complète des phrases. Une opération de composition binaire que nous noterons COMP permet de combiner les *SSynt* [3]. Les grammaires y sont définies comme des ensembles finis de *SSynt*, que nous appellerons *structures syntaxiques élémentaires* et que nous noterons *SSyntE*. Dans l'utilisation que nous faisons de la notion de compagnon, il est nécessaire que les grammaires soient lexicalisées : les *SSyntE* doivent être ancrées par des mots de la langue. Pour simplifier l'exposé, on considérera même que chaque *SSyntE* a une ancre unique.

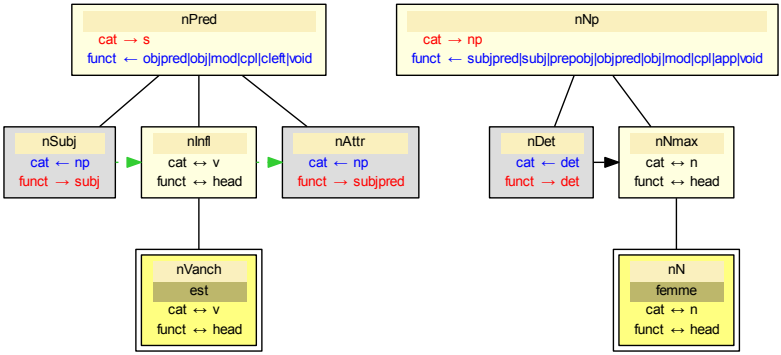

FIGURE 1 – Une *SSyntE* ancrant *femme* compagnon à droite d'une *SSyntE* ancrant *est*

Dans ces conditions, on appelle *compagnon à droite* (*compagnon à gauche*) d'une *SSyntE* S_1 toute *SSyntE* S_2 telle que COMP(S_1, S_2) soit définie et soit compatible avec le fait que l'ancre de S_1 précède (suit) celle de S_2 dans l'ordre linéaire de la phrase.

Appliquons cette notion au formalisme des GI où les *SSynt* sont des forêts d'arbres ordonnés sous-spécifiés. Les nœuds représentent des syntagmes et leurs propriétés morpho-syntaxiques sont représentées par des traits qui présentent la particularité d'être polarisés. Le système de polarités permet d'exprimer l'état de saturation des *SSynt* et leur aptitude à interagir entre elles. Les structures finales sont des arbres saturés. L'opération de composition syntaxique COMP entre deux *SSynt* S_1 et S_2 consiste à fusionner un nœud de S_1 avec un nœud de S_2 de façon à saturer un trait polarisé de S_1 qui ne l'était pas initialement [4]. Pour une description exhaustive du formalisme des GI, le lecteur peut se reporter à (Guillaume et Perrier, 2009). La grammaire à laquelle nous allons appliquer nos idées est la grammaire d'interaction du français FRIGRAM.

La figure 1 montre la *SSyntE* S_{est} ancrant le verbe *est* quand il prend un syntagme nominal comme

3. L'opération COMP n'est pas nécessairement déterministe et il peut y avoir plusieurs façons de composer deux *SSynt*.
4. Le résultat de l'opération COMP doit être un arbre sous-spécifié donc on peut en tenir compte pour résoudre un certain nombre de contraintes comme le fait qu'un nœud doit avoir un père unique.

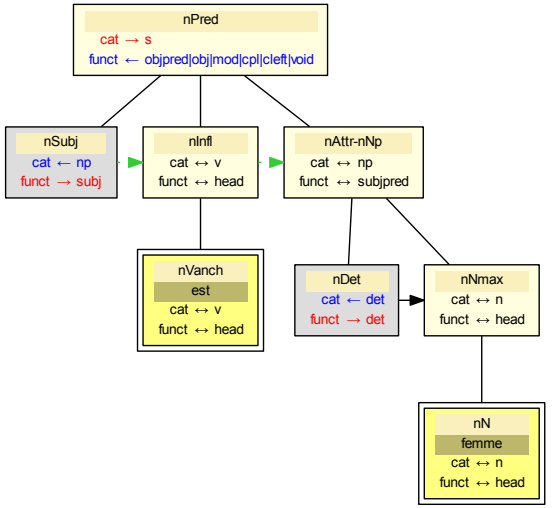

FIGURE 2 – La *SSynt* résultant de la composition syntaxique de la *SSyntE* ancrant *femme* avec la *SSyntE* ancrant *est*

attribut du sujet et un de ses compagnons à droite, la *SSyntE* S_{femme} ancrant le nom *femme* quand il est tête d'un syntagme nominal[5]. La figure 2 en fournit la justification en montrant COMP(S_{est},S_{femme}) obtenu en fusionnant le nœud *nAttr* de S_{est} avec le nœud *nNp* de S_{femme} de façon à saturer les deux traits polarisés du premier. Le résultat de la fusion des deux nœuds est le nœud *nAttr-nNp*. L'ordre des nœuds dans l'arbre montre bien que le compagnon est à droite.

On peut calculer de façon systématique tous les compagnons à droite et à gauche des *SSyntE* d'une grammaire mais pour éviter d'avoir un nombre trop important de calculs à faire, on le fait sur les *SSyntE* non ancrées. Ainsi par exemple si on considère la *SSyntE* non ancrée correspondant à S_{est}, on trouve dans FRIGRAM 129 compagnons permettant de saturer les traits du nœud *nAttr* : 58 à gauche seulement, 59 à droite seulement et 12 qui sont à la fois à gauche et à droite. Pour avoir l'ensemble des compagnons de S_{est}, il faut ajouter ceux qui permettent de saturer des traits polarisés de *nSubj* et de *nPred*.

Dans une phrase donnée, le nombre de compagnons possibles pour une *SSyntE* ancrant un mot est réduit et on utilise cette information pour filtrer les sélections lexicales. (Bonfante *et al.*, 2009) ont montré que le principe que toute *SSyntE* d'une sélection lexicale doit y trouver au moins un compagnon permet de filtrer efficacement les sélections lexicales. Ainsi pour la phrase "*Marie est considérée comme une femme intelligente.*", la grammaire FRIGRAM offre 13 047 840 sélections lexicales possibles et le filtrage fondé sur les compagnons permet de réduire ce nombre à 354.

5. Pour plus de lisibilité, tous les traits associés à chaque nœud n'ont pas été marqués. N'apparaissent que les traits *cat* et *funct*.

Le but du travail présenté ici est de montrer qu'il est encore possible d'aller plus loin pour pallier une faiblesse du principe : il est indifférent à la distance entre une *SSyntE* et ses compagnons. Dans notre exemple, comme c'est indiqué plus haut, S_{est} doit aller jusqu'au mot *une* pour trouver son premier compagnon. L'idée de l'algorithme présenté à la section suivante est de prévoir à partir de la grammaire les *SSyntE* qui peuvent être situées dans une sélection lexicale entre une *SSyntE* donnée et ses compagnons, calcul qui servira de base à un nouveau principe de filtrage.

3 L'algorithme de détection des structures syntaxiques élémentaires s'intercalant entre deux compagnons

L'algorithme va être appliqué au formalisme des GI mais cette application peut être étendu à tout formalisme manipulant des forêts d'arbres ordonnés avec des structures finales qui sont des arbres et une opération de composition qui est une forme de superposition d'arbres. Il part de l'observation que la plupart du temps, dans FRIGRAM, la composition d'une *SSyntE* avec un de ses compagnons produit une *SSynt* qui définit une zone triangulaire dont la base est délimitée par les deux ancres issues des *SSyntE* qui ont été composées et dont le sommet est le premier ancêtre commun. Désormais, nous appelleront une telle *SSynt* une *structure bi-ancrée*.

Formellement, une structure bi-ancrée S est une *SSynt* qui a deux ancres distinguées Ag et Ad, la première, l'ancre gauche, se situant avant la seconde, l'ancre droite, dans l'ordre linéaire de la phrase. En plus, il existe dans S deux suites de nœuds $R, N_1, \ldots, Ag$ et $R, M_1, \ldots, Ad$ ayant un début commun, le nœud R, et telles que chaque nœud de la suite est fils de celui qui le précède. La figure 2 montre un exemple de structure bi-ancrée. Les deux suites de nœuds formant les côtés du triangle sont *nPred*, *nInfl*, *nVanch* et *nPred*, *nAttr-nNp*, *nNmax*, *nN*.

Ces deux suites permettent de définir une partition sur les nœuds de S entre ceux qui se situent à l'intérieur du triangle défini par les deux chemins et ceux qui se situent à l'extérieur. Un nœud est *interne* s'il se situe après l'ancre gauche et avant l'ancre droite selon l'ordre défini sur la structure bi-ancrée [6]. Un nœud qui n'est pas interne, est un nœud *frontière* s'il fait partie d'une des deux listes de nœuds distinguées, sinon il est *externe*.

Le principe de l'algorithme s'appuie sur la forme particulière d'une structure bi-ancrée qui a la conséquence suivante : toute *SSyntE* dont l'ancre s'insère entre les deux ancres distinguées doit être reliée à un nœud interne ou frontière. Elle peut l'être de façon directe par composition avec la structure bi-ancrée mais elle peut l'être de façon indirecte via une chaîne d'autres *SSyntE*. Ces *SSyntE* doivent toutes avoir la propriété d'étendre vers le bas la structure bi-ancrée avec un nouveau nœud interne. C'est cela qui va être utilisé par l'algorithme qui se présente ainsi :

fonction CALCULER_GRAPHE (S,Ag,Ad,$noeuds$)
    initialiser G au graphe vide
    tantque $noeuds$ est non vide
        choisir un noeud N de $noeuds$ et le retirer de cet ensemble
        Mf = CREER_MOTIF (N, S)
        pourchaque SSyntE S_i de la grammaire
            si SUBSUMER(Mf , S_i)

6. Si l'ordre est sous-spécifié, un nœud est interne si, en ajoutant la contrainte de le placer après l'ancre gauche et avant l'ancre droite, on ne crée aucune incohérence dans l'ordre entre les nœuds de l'arbre.

$(S'_i, Ag_i, Ad_i, A_i, noeuds_i) =$ SUPERPOSER (S_i, S, Ag, Ad, Mf)
$G_i =$ CALCULER_GRAPHE $(S'_i, Ag_i, Ad_i, noeuds_i)$
si INTERNE$(A_i, S'_i, Ag_i, Ad_i))$
$G = G \cup$ COMPLETER_GRAPHE$(G_i, S_i))$
sinon $G = G \cup G_i$

retourner G

La fonction CALCULER_GRAPHE prend en entrée une structure bi-ancrée S avec ses deux ancres distinguées gauche et droite Ag et Ad ainsi qu'un ensemble *noeuds* de S qui vont être le point de départ de l'expansion vers le bas de S. Au départ, *noeuds* est initialisés aux nœuds internes et frontière de S à l'exception des ancres Ag et Ad.

En sortie, la fonction CALCULER_GRAPHE retourne un graphe dont les nœuds sont étiquetés par des *SSyntE* de la grammaire. Il s'agit en fait d'une forêt d'arbres dont la sémantique est la suivante :

> *Si une phrase est analysée avec succès par la grammaire à partir de la SSynt S et si w_1 et w_2 sont les deux mots de la phrase attachés aux ancres distinguées de S, pour tout mot w situé entre w_1 et w_2 qui contribue à l'analyse avec la SSyntE S_1 qu'il ancre, il existe une occurrence de S_1 dans le graphe dont tous ses prédécesseurs dans le graphe participe à l'analyse en ancrant des mots situés entre w_1 et w_2.*

Expliquons maintenant l'algorithme. Au départ on choisit un nœud N de l'ensemble *noeuds* que l'on retire de l'ensemble. Ce nœud va servir de point de départ à l'expansion vers le bas de S. A l'aide de la fonction CREER_MOTIF , on crée un motif Mf qui va permettre de filtrer les *SSyntE* de la grammaire pertinentes pour cette expansion. Mf est formé du nœud N ainsi que de tous ses ancêtres et tous ses frères dans S. On ajoute en plus un fils N' de N qui est laissé complètement sous-spécifié quant aux traits dont il est porteur. Il est seulement ordonné par rapport à ses frères éventuels qui sont sur la frontière. Ce nœud est capital car c'est lui qui va permettre l'expansion [7].

Ensuite, on passe en revue toutes les *SSyntE* de la grammaire à l'aide du filtre Mf. La fonction booléenne SUBSUMER teste si Mf subsume une *SSyntE* S_i quelconque de la grammaire. Cela veut dire que tout nœud de Mf s'interprète dans S_i et que cette interprétation conserve les relations père-fils ainsi que celles de précédence. En plus, les traits attachés à chaque nœud de Mf doivent aussi s'interpréter par des traits attachés à son nœud image dans S_i en respectant un certain nombre de propriétés qui sont spécifiques au formalisme grammatical utilisé. Par exemple, pour les GI, la polarité du trait image doit être compatible avec celle du trait antécédent.

Ensuite, si le test est positif, à l'aide la fonction SUPERPOSER, on compose la *SSyntE* S_i avec S en suivant le motif Mf et en utilisant l'opération COMP de composition syntaxique propre au formalisme. On obtient une *SSyntE* S'_i et on distingue dans celle-ci les ancres gauche et droite Ag_i et Ad_i qui sont la transposition dans S'_i des ancres Ag et Ad de S. En plus, on repère l'ancre A_i apportée par S_i car sa position va jouer un rôle décisif pour la suite. La variable $noeuds_i$ représente l'ensemble des nœuds de S'_i qui vont servir de point de départ aux expansions futures. Ce sont les nœuds internes de S'_i qui n'étaient présents au départ dans *noeuds*.

L'étape suivante consiste à appliquer récursivement la fonction CALCULER_GRAPHE. Elle va permettre de récupérer un graphe G_i et c'est là que l'ancre A_i va jouer un rôle important par le biais de la fonction booléenne INTERNE. Cette fonction teste si l'ancre A_i est un nœud interne à la structure bi-ancrée S'_i. Si nous reprenons notre exemple avec la phrase à analyser "*où Marie est-elle considérée comme une femme intelligente ?*", les *SSyntE* associées aux mots *où* et *une*

7. Bien entendu, ce nœud est un minimum et l'expansion peut se faire à l'aide de plusieurs nœuds.

vérifient toutes les deux la condition exprimée par la fonction SUBSUMER. Pour *où*, cela provient du fait que la *SSyntE* modélise une extraction. Pourtant, seule la seconde vérifie la condition exprimée par la fonction INTERNE, le mot *une* se situant entre *est* et *femme*. Dans ce cas, il va falloir ajouter S_i au graphe G_i. On l'ajoute comme nouvelle racine en le reliant par un arc à toutes les anciennes racines de G_i. C'est le rôle de la fonction COMPLÉTER_GRAPHE. Il ne reste plus qu'à faire l'union du graphe obtenu avec G, dans l'état où il est après utilisation d'un certain nombre de nœuds de *noeuds*. Si A_i est un nœud externe, on se contente de faire l'union de G_i avec G.

4 Application à un exemple

Appliquons l'algorithme à la structure bi-ancrée S de la figure 2. La valeur initiale de *noeuds* est l'ensemble {*nInfl*, *nPred*, *nAttr-nNp*, *nDet*, *nNmax*}. On choisit ensuite un nœud N dans cet ensemble, par exemple *nInfl*. On crée le motif Mf correspondant à l'aide de la fonction CRÉER_MOTIF. C'est le sous arbre de la structure bi-ancrée formé des trois nœuds *nPred*, *nInfl* et *nAttr-nNp*. On y ajoute un nouveau fils N' de *nInfl*.

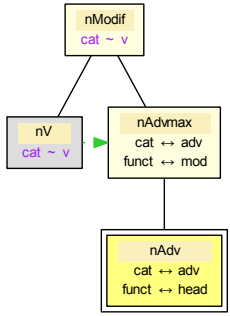

FIGURE 3 – *SSyntE* ancrant les adverbes modificateurs de verbes situés après ces verbes

Ensuite, on essaie de faire coïncider le motif Mf avec un sous-arbre de chaque *SSyntE* de FRIGRAM. Prenons un cas où l'appariement réussit, celui de la *SSyntE* ancrant les adverbes modificateurs de verbes et situés après ces verbes, nommée $ADVmod\_V$ et représentée sur la figure 3. La condition SUBSUMER(Mf,$ADVmod\_V$) est vraie et on superpose alors $ADVmod\_V$ avec S en suivant le motif Mf. Cela revient à étendre S en ajoutant comme frère droit de *nVanch* le nœud *nAdvmax* de $ADVmod\_V$ avec son fils *nAdv*.

On relance la fonction principale CALCULER_GRAPHE sur cette nouvelle structure bi-ancrée S'_i avec comme valeur pour $noeuds_i$ le singleton {*nAdvmax*}. Nous passerons sur le détail de son exécution en en donnant seulement le graphe G_i qu'elle retourne. Ce graphe est formé de deux nœuds isolés $ADVmod\_ADV1$ et $ADVmod\_ADV2$ ancrant les adverbes modificateurs d'adverbes.

Comme l'ancre de $ADVmod\_V$ est un nœud interne, la condition INTERNE est vraie et on complète

le graphe G qui est initialement vide à l'aide de la fonction COMPLÉTER_GRAPHE. On obtient un graphe de trois nœuds avec comme racine $ADVmod\_V$ et ses deux successeurs immédiats $ADVmod\_ADV1$ et $ADVmod\_ADV2$.

L'algorithme se poursuit par la sélection d'autre nœud de l'ensemble *noeuds*, *nPred* par exemple. Va s'ensuivre une extension de S vers le bas à partir de ce nœud. Il serait trop long de la décrire en détail mais il est important de noter que cette extension va entraîner la création d'un nœud qui représente un syntagme propositionnel. Ce syntagme peut représenter une proposition relative telle que "*qu'elle a*" dans la phrase "*Marie est avec l'expérience qu'elle a une femme intelligente.*". Compte tenu de la récursivité de la langue liée aux propositions qui peuvent s'imbriquer les unes dans les autres à l'infini, l'exécution de l'algorithme entre ici dans une boucle infinie. Pour éviter la non terminaison de l'algorithme, il suffit de couper l'extension vers le bas de la structure bi-ancrée quand on produit des nœuds source de bouclage ou si l'on atteint une certaine taille [8].

En définitive, nous obtiendrons un graphe G acyclique qui n'est pas forcément complet. Il est éventuellement amputé vers la "fin" mais ce qui est important c'est que toutes les racines peuvent être calculées. Dans notre exemple, le graphe aura quelques dizaines de racines qui sont des *SSyntE* ancrant des adverbes modificateurs de verbes ou de phrases, des adverbes entrant dans des constructions consécutives ou comparatives, des prépositions introduisant des compléments modificateurs de phrases, des pronoms comme *tous* ou *chacun*, des conjonctions de subordination introduisant des propositions circonstancielles, des déterminants et des adjectifs épithètes gauche.

Si S_{est} a un comme compagnon unique S_{femme} [9] dans une sélection lexicale qui produit une analyse, selon la sémantique du graphe exposée plus haut (même si ce graphe est incomplet), pour toute *SSyntE* S_k s'intercalant entre les deux compagnons dans la sélection, il existe un chemin dans le graphe commençant à une racine et terminant à un nœud qui n'a pas de successeur ou est une occurrence de S_k.

5 Conclusion

Si le calcul des compagnons est implémenté, ce n'est pas le cas pour l'algorithme de détection des *SSyntE* pouvant s'insérer entre deux compagnons. Seule son implémentation permettra de dire dans quelle mesure cet algorithme est utile pour accroître l'efficacité du filtrage.

Références

BONFANTE, G., GUILLAUME, B. et MOREY, M. (2009). Polarization and abstraction of grammatical formalisms as methods for lexical disambiguation. *In 11th International Conference on Parsing Technology, IWPT'09*, Paris, France.

GUILLAUME, B. et PERRIER, G. (2009). Interaction Grammars. *Research on Language and Computation*, 7:171–208.

8. Chaque appel récursif de la fonction CALCULER_GRAPHE entraîne une augmentation de la taille de la structure bi-ancrée.

9. Les deux *SSyntE* ne sont pas exactement S_{est} et S_{femme} mais les *SSyntE* non ancrées dont elles sont issues.