Réseaux de preuve et génération pour les grammaires de types logiques

Sylvain Pogodalla

Institut National
Polytechnique de Lorraine

Département de formation doctorale en informatique

École doctorale IAEM Lorraine

Réseaux de preuve et génération pour les grammaires de types logiques

THÈSE

présentée et soutenue publiquement le 27 septembre 2001

pour l'obtention du

Doctorat de l'Institut National Polytechnique de Lorraine
(spécialité informatique)

par

Sylvain Pogodalla

Composition du jury

Rapporteurs :	Alexandre Dikovsky Gérard Huet Karl Tombre
Examinateurs :	Marc Dymetman Alain Lecomte (directeur de thèse) Christian Retoré
Invité :	Glyn Morrill

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Institut National
Polytechnique de Lorraine

Département de formation doctorale en informatique

École doctorale IAEM Lorraine

Réseaux de preuve et génération pour les grammaires de types logiques

THÈSE

présentée et soutenue publiquement le 27 septembre 2001

pour l'obtention du

Doctorat de l'Institut National Polytechnique de Lorraine
(spécialité informatique)

par

Composition du jury

Rapporteurs :	Alexandre Dikovsky
	Gérard Huet
	Karl Tombre
Examinateurs :	Marc Dymetman
	Alain Lecomte (directeur de thèse)
	Christian Retoré
Invité :	Glyn Morrill

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Réseaux de preuve et génération pour les grammaires de types logiques

Remerciements

La part personnelle de ce travail ne saurait cacher combien il doit à de nombreuses personnes. En premier lieu, mes remerciements vont à Alain Lecomte, qui a accepté d'encadrer cette thèse et qui a su ménager à la fois direction et autonomie scientifique, à Marc Dymetman, qui a nourri ma réflexion grâce à ses remarques toujours très constructives et pertinentes, et à Christian Retoré, qui a marqué ce travail de son enthousiasme communicatif et de son inventivité.

L'équipe Calligramme, pour son soutien et son intérêt, mérite également ma gratitude, de même que le Centre de Recherche de Xerox à Meylan[1], et tout particulièrement l'équipe MLTT qui m'a accueilli. Riche de la diversité de ses membres, elle a été le cadre de nombreux et fructueux échanges.

Dans sa phase finale, ce travail a également bénéficié des lectures attentives et des remarques pertinentes des rapporteurs Gérard Huet, Alexandre Dikovsky et Karl Tombre que je remercie. De même, je remercie Philippe de Groote, Jean-Marc Andreoli, Aarne Ranta, Jean-Yves Vion-Dury et Guy Perrier : leur intérêt pour cette recherche et les discussions que nous avons eues aux différents stades d'avancement m'ont permis d'affiner ma réflexion.

Au quotidien, ou presque, de nombreuses difficultés pratiques me furent épargnées grâce à l'intervention efficace de Christine Marcel, Lætitia Lemoine et, à Nancy, de Guillaume Bonfante, ce dont je leur suis des plus reconnaissants. Pour leur soutien et pour les discussions sur les sujets les plus variés qu'ils m'ont offertes, je remercie également Laurent Griot, Caroline Brun, Mathieu Mangeot, Laurent Plagne et, pour sa patience et son attention, Veronika Lux.

Enfin, pour leurs contributions à la tâche si ingrate de relecture et de correction, je remercie Laurent Vaucher, Anne, François, Simone et Édouard, d'autant plus vivement que mes modifications ultérieures, riches en fautes de tous genres, en rendent bien mal compte.

[1] Ce travail, rendu possible par le soutien de Xerox et de l'ANRT, a été réalisé dans le cadre d'une convention CIFRE.

« En chantant cette chanson »
(G. Brassens)

À mes parents.

Sommaire

Introduction

L'un des objets de la construction d'une grammaire, d'un point de vue linguistique, est de définir un ensemble de structures linguistiques bien formées. Ces structures peuvent posséder plusieurs niveaux : phonologiques (chaînes de caractère dans le cas de l'écrit), morphologiques, syntagmatiques, syntaxiques ou sémantiques, et la grammaire se doit également de décrire les articulations entre ces différents niveaux. Du point de vue informatique, il s'agit de fournir des modèles permettant la description des différents niveaux linguistiques et l'obtention d'une implémentation. Cela se traduit par une recherche de bonnes propriétés calculatoires pour les grammaires et une représentation de haut niveau des propriétés qu'elles décrivent.

Le présent travail s'attache aux relations qui existent entre la syntaxe et la sémantique, en particulier lors de l'analyse — qui consiste à retrouver un contenu sémantique à partir d'un contenu syntaxique — et lors de la génération — qui consiste à retrouver un contenu syntaxique à partir d'un contenu sémantique. Les propriétés qui caractérisent ces relations dépendent du choix du modèle grammatical et dans le cas présent, ce dernier s'inscrit dans la famille des grammaires catégorielles [Ajd35, Moo97], plus particulièrement vue sous le côté logique. Ainsi le calcul central de ce travail, le calcul de Lambek [Lam58], admet-il une étude duale : logique et grammaticale. Cela signifie que les réponses aux questions de linguistique informatique s'obtiennent parfois grâce à l'éclairage de la théorie de la démonstration.

Le principe sous-jacent aux grammaires catégorielles réside dans la caractérisation des propriétés de combinaison des mots les uns avec les autres et privilégie la description fonctionnelle du langage. Ainsi, la similarité de construction des phrases suivantes :

Philippe dort
Philippe maugrée
Corinne dort
Corinne maugrée

laisse penser qu'un certain nombre d'expressions, comme *Philippe* ou *Corinne,* possèdent les mêmes propriétés d'interaction et peuvent se regrouper sous la même classe désignée par le type *np*, par exemple. De même pour les expressions *dort* et *maugrée.* Si l'on souhaite de plus exprimer pour ces dernières qu'elles sont les foncteurs capables de prendre un argument pour obtenir une phrase correcte caractérisée par un type *S*, leur type sera composé du type de leur argument et du type du résultat. On l'écrit $np\backslash S$ indiquant par là que l'argument est attendu à gauche du foncteur.

Si l'on considère d'autres phrases, comme *Philippe contrarie Corinne*, l'expression incomplète *contrarie Corinne,* qui peut se substituer à *dort* ou *maugrée,* reçoit le même type, soit $np\backslash S$. L'étape supplémentaire de décomposition de *contrarie Corinne* permet de considérer que *contrarie* se comporte comme un foncteur sur deux arguments de type *np*. Le premier, qui doit se placer à droite, permet de lui attribuer finalement le type $(np\backslash S)/np$. La correction syntaxique d'une expression se calcule ensuite à partir des types associés à chacune des expressions élémentaires et à partir des règles

de réécriture de l'application :

$$X\ X\backslash Y \rightarrow Y \qquad Y/X\ X \rightarrow Y$$

La correction des expressions ci-dessus s'exprime simplement par le fait que les types des sous-expressions qui les composent permettent une réécriture vers le type S :

$$np\ np\backslash S \rightarrow S$$

et

$$np\ ((np\backslash S)/np\ np) \rightarrow np\ np\backslash S \rightarrow S$$

Cela nous amène à énoncer quelques principes de base des grammaires catégorielles : premièrement, elles s'articulent autour de règles fixes, indépendantes en particulier de la langue choisie, et d'un lexique associant des types aux mots de la langue. Le lexique porte toute l'information syntaxique par l'intermédiaire des types. Deuxièmement, la définition du lexique et des types utilisés restent des paramètres de la modélisation soumis à un ajustement complètement libre. En particulier, on peut utiliser les types de base *np*, *S* ou *n*, mais aussi tout autre type de base jugé utile, et toute formule obtenue à partir des types de base et des constructeurs $/$ et $\backslash$; et un mot peut se voir attribuer un ou plusieurs types.

Au-delà de la vision algébrique, ou bien axiomatique à la Hilbert, des règles de réduction ci-dessus correspondant aux grammaires AB [BHGS64], une étape importante intervient en plongeant ces règles dans un système logique similaire à la déduction naturelle, ce que fait le calcul de Lambek proprement dit. Les règles générales décrivant la combinaison des items lexicaux appartiennent alors à un système déductif, et l'analyse d'expressions revient à la construction de preuves dans ce système. Cependant, là où la logique apporte un modèle pour la grammaire, cette dernière demande des propriétés inhabituelles pour la logique. En effet, les propriétés requises s'expriment essentiellement en terme d'interactions, d'échanges et de ressources. Le modèle doit pouvoir exprimer la disponibilité de ces dernières et compter avec leur multiplicité, ce qui n'a guère de sens pour la notion habituelle de vérité qui accompagne la logique. Précurseur, le calcul de Lambek rejoint une classe de logiques capables de ces distinctions et que l'on nomme *sensibles aux ressources*. L'une des plus complètes est la logique linéaire [Gir87a], qui apporte des outils d'analyse fine, comme les réseaux de preuves, et qui occupe une place centrale dans ce travail.

Les grammaires de types logiques synthétisent des fonctionnements mis en valeur dans différents formalismes grammaticaux. Le premier tient à la coexistence d'un lexique et d'un système de calcul très général pour obtenir les représentations. Elles rejoignent en cela les thèses soutenues par Chomsky, outre celle de grammaire universelle, avec le programme minimaliste [Pol97]. La mise en avant dans ce programme de deux opérations de déplacement (*move*) et de fusion (*merge*) lors du passage de la D-structure à la S-structure, à partir des items lexicaux et en respectant des principes d'économie, conduit à des propositions de formalisations comme celle de [Sta97] qui décrit un système dans lequel le langage reconnu correspond à la clôture d'un lexique fini sous des règles de construction structurelle. Le lien avec les grammaires de types logiques intervient de deux manières : tout d'abord dans le côté dérivationnel de l'approche, mais également dans la construction de la dérivation. En effet, les items lexicaux s'accompagnent de *traits* de sélection faibles ou forts qui déclenchent les opérations de mouvement et guident la construction de la dérivation. On retrouve en cela les notions de ressources centrales dans les grammaires de types logiques. Le support théorique des logiques de ressources agit comme intermédiaire entre le minimalisme, dont il peut éclairer la compréhension formelle, fournir les outils de démonstration automatique pour un support algorithmique de l'analyse, préciser les liens vers une sémantique formelle et les grammaires de types logiques qui peuvent bénéficier des idées

linguistiques nouvelles de ce programme ou des problèmes formels auquel il est confronté. Différents travaux poursuivent cette voie [RS99, LR99, LR01].

La notion centrale de ressource des grammaires de types logiques se retrouve dans le programme minimaliste, mais aussi dans des formalismes grammaticaux appartenant aux grammaires d'unification, telles que la grammaire lexicalisée fonctionnelle LFG [KB82]. Cette dernière adopte en effet un critère de bonne formation des analyses syntaxiques qui comporte un principe d'unicité et un principe de complétude assurant l'exacte correspondance entre les fonctions sous-catégorisées et celles requises par la fonction principale : des principes de gestion de ressources radicalisés et centraux dans les grammaires de types logiques.

Le deuxième fonctionnement mis en valeur repose sur la description lexicale. Celle-ci donne pour chaque mot les moyens dont il dispose pour se combiner avec les autres mots. Cette notion de dépendance [Tes88] se traduit effectivement dans la représentation des preuves de bonne formation, qui permet à son tour, sous forme de réseaux de preuve, de retrouver les représentations issues des grammaires de dépendances [Lec92, Lec96].

Les grammaires de types logiques permettent ainsi de prendre un point de vue radicale sur les notions habituelles de dérivation et de ressource en linguistique en les plongeant dans un cadre théorique ayant pour objet même l'étude des dérivations, vues comme des preuves : la théorie de la démonstration.

En considérant les deux règles des grammaires AB comme deux utilisations orientées du *modus ponens* (de A et $A \Rightarrow B$ on peut déduire B), [Lam58] donne le premier éclairage logique aux grammaires catégorielles, proche de la logique intuitionniste. L'analyse d'une expression se décompose alors en deux phases : premièrement, le choix d'une des catégories attribuées par le lexique aux mots de l'expression, deuxièmement la vérification que ces catégories permettent bien de prouver le type S des phrases correctes, c'est-à-dire d'en construire une preuve.

Ce système, motivé par des raisons linguistiques, est resté isolé du fait de sa sensibilité aux ressources, des notions étrangères à celle de vérité plus habituelle. La logique linéaire, motivée par des raisons tout à fait différentes, a permis de voir le calcul de Lambek comme élément d'une plus grande famille, un fragment multiplicatif non commutatif de la logique linéaire intuitionniste. Elle a aussi apporté de nouveaux outils aux grammaires de types logiques, en particulier les réseaux de preuve [Gir87a, Roo91, LR96a, Ret96a]. La preuve joue un rôle déterminant dans l'analyse syntaxique puisque son existence détermine la grammaticalité d'une expression. Le nombre de preuves lui-même joue un rôle, car il indique le nombre de dérivations possibles, c'est-à-dire l'ambiguïté d'une expression. Il importe donc que le système n'ajoute pas au nombre des preuves possibles. La syntaxe particulière des réseaux de preuve, en représentant de manière unique des preuves essentiellement semblables, le permet. Nous utilisons la présentation basée sur les R&B-graphes de [Ret99] que nous augmentons pour pouvoir tenir compte dans le même formalisme des connecteurs exponentiels.

Pour toute classe de grammaires, une question importante se pose : quel langage exactement peut-elle reconnaître ? Pour les grammaires de Lambek, l'approche logique a permis à [Pen93] de résoudre ce problème difficile grâce à un résultat d'interpolation. Il a montré que la famille des langages engendrés était la même que celle des langages engendrés par les grammaires hors-contexte. Ce résultat n'établit l'équivalence que pour le langage, et pas pour les dérivations. Ces dernières, étapes vers la représentation sémantique, restent très différentes, maintenant l'intérêt des grammaires de Lambek.

Une autre raison de considérer l'approche logique tient également à l'expressivité des grammaires. Si elles rendent compte efficacement de certains phénomènes linguistiques comme les dépendances non bornées, la conjonction de non-constituants ou la portée de quantificateur, les grammaires de Lambek restent fortement contraintes par l'emploi de la seule opération de concaténation sur les chaînes. La modélisation des constituants discontinus pose problème, de même que celle des ordres de mots moins stricts. Pour augmenter l'acuité de ces grammaires, l'étude du modèle logique conduit à

diverses solutions. On trouve par exemple des calculs directement issus de la logique linéaire, comme le calcul ordonné [Ret93] ou la logique non commutative [dG96, Rue97]. Ceux-ci permettent de faire cohabiter à la fois des connecteurs commutatifs et des connecteurs non commutatifs, et des travaux comme [LR95, LR96b] en tirent parti. Nous montrons qu'ils permettent d'obtenir au moins la même couverture que les grammaires d'arbres adjoints. Un autre courant très important étudie les logiques sous-structurelles. À partir de [Lam61] qui présente un calcul non commutatif et non associatif, différents degrés peuvent être atteints [Mor94], en particulier avec les logiques multimodales [Moo96] introduisant des opérateurs unaires permettant une gestion fine des postulats structuraux. Ces approches conservent la présentation des preuves avec les réseaux [MP99].

Le point de vue logique des grammaires catégorielles pour la représentation syntaxique se révèle également fructueux dans l'articulation avec la représentation sémantique. Il établit un lien de choix avec une représentation à la Montague [Mon74, DWP81]. Cette dernière se rapproche des buts de la sémantique linguistique considérés par Frege ou Russell et concerne les conditions sous lesquelles un énoncé se vérifie. Elle utilise pour cela le λ-calcul typé et des modèles. L'approche fonctionnelle permet en outre d'exprimer fidèlement le principe de composition indiquant que le sens d'une expression s'obtient par combinaison du sens de ses sous-expressions. Si cette vision utilisant également opérateurs booléens et formules du calcul des prédicats reste controversée, elle permet d'expliquer un certain nombre de phénomènes concernant entre autres la quantification ou les coréférences [Car97].

Comme l'indique [vB83], le calcul de la forme sémantique d'une expression, exprimée par un λ-terme, peut s'obtenir directement depuis l'analyse syntaxique de cette expression par l'isomorphisme de Curry-Howard [How80a, GLT88] puisque cette analyse n'est rien d'autre qu'une preuve intuitionniste. Ainsi obtient-on très directement une articulation entre la syntaxe et la sémantique. Notons que nous utilisons ici la représentation sémantique comme une représentation indépendante de la langue plutôt que dans sa dimension vériconditionnelle.

L'éclairage théorique sur la syntaxe, la sémantique et leur articulation permet de faire bénéficier les uns et les autres des outils mis en œuvre. Un élément central de notre travail consiste, selon la proposition de [dGR96], à transposer la représentation sémantique des λ-termes en réseaux de preuve, toujours via l'isomorphisme de Curry-Howard. Cela nous permet de donner un éclairage nouveau à la génération et de proposer des solutions algorithmiques aux problèmes posés, et en particulier de répondre favorablement à la question de la réversibilité des grammaires de types logiques.

Le cadre fixé pour notre étude du problème de la génération établit un certain nombre d'hypothèses. La première consiste à ne considérer que des énoncés sous forme d'expressions indépendantes, c'est-à-dire qu'on ne tient pas compte du discours environnant et, par exemple, du traitement des coréférences, malgré son importance sur la qualité linguistique des énoncés engendrés. La deuxième hypothèse porte sur la manière de définir le sens que doivent transmettre les expressions engendrées. Traditionnellement, le processus de génération se divise en deux sous-tâches : la première, conceptuelle, s'attache à donner une représentation sémantique du message que l'on veut exprimer. La seconde, grammaticale, cherche à en construire une expression dans un langage donné. Seule la seconde partie nous intéresse. Nous considérons donc le problème, étant donné une représentation sémantique, de formuler une expression ayant cette représentation.

Dans leur objectif abstrait de mettre en relation représentation syntaxique et représentation sémantique, les grammaires ne privilégient *a priori* pas un sens de l'un vers l'autre, et l'exploitation d'une grammaire doit pouvoir se faire aussi bien en analyse qu'en génération. Cette propriété de *réversibilité* traduit différentes considérations : au point de vue méthodologique, pour l'écriture des grammaires, cela évite la spécialisation et les constructions *ad hoc* pour l'un ou l'autre des algorithmes. Au point de vue pratique, cela permet de s'assurer, notamment pour un système en interaction avec un utilisateur, de la cohérence entre le langage engendré et le langage reconnu. Ainsi, par mimétisme, offre-t-on un moyen de repérer les limites du système.

Cependant, cette propriété de réversibilité est plus qu'un simple problème d'implémentation utilisant le même programme pour l'analyse et la génération. En effet, pour une grammaire donnée, les capacités d'analyse et de génération ne sont pas en général liées : admettons une grammaire G dont le vocabulaire est l'ensemble des nombres premiers et dont les expressions sont des suites de nombres premiers auxquelles elle attache comme représentation sémantique le produit de ces nombres. L'analyse consiste simplement à effectuer une ou plusieurs multiplications. La génération par contre consiste à décomposer un nombre en ses facteurs premiers, une opération qui peut être très ardue. Construisant des grammaires dont l'analyse se révèle décidable alors que la génération se révèle indécidable, ou bien le contraire, [Dym92] introduit la notion de *réversibilité intrinsèque* qui caractérise les propriétés calculatoires d'analyse et de génération d'une grammaire donnée.

L'étude de ces propriétés sur divers formalismes, notamment sur les grammaires d'unification, a apporté des résultats contrastés sur cette propriété de réversibilité intrinsèque. Des contraintes sur la sémantique [Shi88, Wed95] ou sur la forme des règles [vN93, GH90] permettent d'assurer la décidabilité de la génération. Pour les grammaires de types logiques, si de nombreux travaux se consacrent à l'analyse, très peu en revanche se consacrent à la génération, en particulier en exploitant le lien fort entre syntaxe et sémantique tel qu'il y est exprimé. La raison tient sans doute à la difficulté évoquée par [Als92, vN93] consistant à inverser le processus de β-réduction. Cette réticence se trouve d'ailleurs confortée par la méthode de réalisation syntaxique proposée par [MM97] qui se base sur l'unification des λ-termes, indécidable dès le deuxième ordre [Gol81, Dow01]. Pourtant, des considérations de recherche de preuve semblent montrer que le problème devrait rester décidable.

Nous montrons comment l'utilisation des réseaux de preuve permet de formuler le problème sans utiliser le λ-calcul, et de proposer un algorithme restant au plus près de la complexité du problème. Cet algorithme se base sur un procédé de démonstration automatique original fondé sur l'interprétation algébrique de la logique linéaire [Gir89], qui s'aide d'un réseau cible en forme normale. Il aboutit à la résolution d'une équation matricielle et permet de montrer que le problème de la génération pour les grammaires de types logiques, dont les formes sémantiques des items lexicaux sont linéaires et pourvues d'au moins une constante, est décidable. En outre, nous caractérisons les items lexicaux qui permettent une réalisation syntaxique en temps quadratique. Ainsi nous obtenons un cadre uniforme pour l'analyse et la génération, en tant que recherche de preuve sur des réseaux. Nous avons réalisé un prototype d'analyse et de génération, basé sur un démonstrateur automatique de la logique linéaire avec les réseaux de preuve comme des R&B-graphes, qui illustre la propriété de la décidabilité intrinsèque établie théoriquement. Les bonnes performances de la partie de réalisation syntaxique dans le cas linéaire suggèrent également qu'un éclairage par les réseaux sur des problèmes de filtrage ou d'unification de λ-termes permettrait peut-être d'y apporter une efficacité algorithmique.

Compte tenu de l'importance de la logique dans cette approche de la linguistique informatique, la première partie présente les éléments de logique et de théorie de la démonstration employés dans le traitement grammatical. La deuxième partie s'attache à la modélisation des phénomènes linguistiques telle qu'elle apparaît dans les grammaires de types logiques. Le chapitre 1 donne les outils utilisés tout au long de cette thèse. Consacré tout d'abord à la notation du calcul des séquents pour la logique classique, il met en évidence, en décrivant ses principes, les propriétés requises par la logique pour une utilisation de modélisation de la langue. Un regard sur ces propriétés montre l'influence des règles structurelles et les conséquences de leur abandon, ou du moins de leur contrôle précis. La présentation de la logique linéaire [Gir87a] qui suit synthétise les caractéristiques et les intuitions liées au nouveau système. Les propriétés fondamentales des systèmes logiques telles que l'élimination des coupures, qui se retrouve tout au long de cette thèse, sont développées dans la version du calcul des séquents, de même que leurs conséquences sur la recherche automatique de preuve qui fonde notre approche informatique de la linguistique.

Le deuxième outil transversal à notre travail et à nos résultats, aussi bien sur le plan de la théorie

de la preuve pure que sur son intérêt pour l'étude des propriétés de réversibilités des grammaires de types logiques apparaît au chapitre 2 : les réseaux de preuve. Après l'évocation des principes relatifs à cette syntaxe des preuves, nous définissons la représentation choisie des R&B-graphes et nous donnons les résultats principaux qui justifient cette syntaxe : à partir de chaque R&B-graphe vérifiant une certaine condition, on peut construire une preuve du calcul des séquents, et réciproquement [Ret99]. Cette propriété fondamentale a le nom de théorème de séquentialisation. Elle permet de ne plus avoir à revenir au calcul des séquents et de ne considérer que les réseaux, d'autant plus que ceux-ci bénéficient également de la propriété d'élimination des coupures qui prend la forme d'une réécriture de graphe. Il s'agit là encore d'un point crucial dans notre approche.

La logique linéaire permet une gestion fine des ressources, mais également de les considérer comme infiniment disponibles, c'est-à-dire retrouvant leur statut de pérennité de la logique classique. Cela se fait grâce à des connecteurs unaires : les exponentiels. Pour ces connecteurs existe également une présentation sous forme de réseaux, mais cette dernière ne s'adapte pas directement à la présentation comme R&B-graphe. Nous définissons en section 2.4 une nouvelle présentation de ces connecteurs qui reste dans les R&B-graphes, avec un critère de correction purement graphique. Enfin, la section 2.5 présente les liens entre l'élimination des coupures et le calcul matriciel via la géométrie de l'interaction [Gir89].

Après l'évocation des différents points de vue sur la logique linéaire, le chapitre 3 décrit ses liens avec le λ-calcul. La définition de ce dernier, dans sa version typée, et les modèles telles que les considère la sémantique de Montague permet d'introduire l'isomorphisme de Curry-Howard pour la déduction naturelle, le calcul des séquents intuitionnistes et les réseaux de preuve. Dès lors, nous disposons des outils formels aussi bien pour la syntaxe que la sémantique des grammaires de types logiques.

Toutefois, avant d'en tirer pleinement parti, le chapitre 4 définit les grammaires AB, donne quelques exemples de modélisation et finit par évoquer leur pouvoir génératif. Nous reprenons alors des résultats importants, dus notamment à [Kan98], concernant l'apprentissage sous forme d'acquisition du lexique. Il s'agit dans un premier temps d'un apprentissage à partir d'expressions munies de leur dérivation. Cela a conduit [Tel98a] à proposer un apprentissage à partir d'expressions munies cette fois de leur représentation sémantique. Nous montrons à cette occasion les liens que les problèmes de génération entretiennent avec cette approche, indiquant une possibilité d'extension de nos travaux dans cette direction.

L'aspect logique des grammaires catégorielles constitue le cœur du chapitre 5, en décrivant précisément comment fonctionne l'analyse syntaxique et sémantique dans ce cadre. En particulier, nous voyons précisément comment l'élimination des coupures rentre en jeu lorsque les réseaux de preuve servent à modéliser aussi bien la partie syntaxique que la partie sémantique des entrées lexicales. La modélisation de quelques phénomènes linguistiques montre l'application de ces techniques, mais également les limites de cette modélisation et justifie les propositions d'extensions. Nous évoquons principalement les systèmes multimodaux [Moo96] et ceux basés sur le calcul ordonné [LR95]. Cela nous permet de prouver qu'un certain fragment du calcul ordonné est équivalent aux grammaires d'arbres adjoints.

Enfin, le chapitre 6 définit précisément le problème de la génération et celui de la réversibilité intrinsèque. En se référant à différents travaux sur les grammaires d'unification [Shi88, Dym92, vN93, Wed95], il montre l'intérêt de ce problème et les questions qu'il pose. Concernant les grammaires de types logiques, la difficulté d'inverser le processus de β-réduction sur les λ-termes donne les limites de l'approche de [Mor94]. Tous les outils définis dans les chapitres précédents nous permettent de formuler alors une nouvelle proposition, en donnant une méthode de recherche originale guidée par un réseau en forme normale. Cette méthode fait le lien entre l'élimination des coupures et son expression sous forme de relation matricielle. Elle nous permet d'énoncer des conditions suffisantes à

la réversibilité intrinsèque des grammaires de types logiques. Toutes les étapes définies théoriquement dans ce chapitre apparaissent dans le prototype implémenté et dont l'annexe B donne des exemples d'utilisations.

Première partie

Éléments de logique et de théorie de la démonstration

Chapitre 1

La logique linéaire

1.1 Généralités

1.1.1 Calcul des séquents

Dans un premier temps, nous présentons la logique linéaire uniquement d'un point de vue syntaxique, sans évoquer les aspects sémantiques (phases, espaces cohérents) pour lesquels nous renvoyons le lecteur à [Gir87a]. L'aspect sémantique n'intervient que par l'intermédiaire de la *géométrie de l'interaction,* à la section 2.5.

Cette présentation de la syntaxe permettra de mettre en avant les différences qui existent avec la logique dite classique, la plus familière. Nous pourrons ainsi donner des intuitions du fonctionnement de la logique linéaire et de sa pertinence, en particulier pour le cas qui nous préoccupe : la modélisation du fonctionnement grammatical des langues.

Pour la description des systèmes logiques, la logique classique comme la logique linéaire, nous utiliserons la présentation due à Gerhard Gentzen (1935). Elle propose un système de règles, basées purement sur la syntaxe, pour étudier les lois de la logique. On utilise des *séquents,* qui sont des expressions $\Gamma \vdash \Delta$ où Γ et Δ sont des suites finies de formules (par exemple $A_1, \ldots, A_n$ et $B_1, \ldots, B_m$) et qui, en première approximation, se lisent : « les hypothèses A_1 et …et A_n impliquent B_1 ou …ou B_m ». Les « et», « ou» et « impliquent» ont un sens à définir précisément car ils peuvent varier avec les différents systèmes considérés. Chaque règle prend la forme

$$\frac{\text{Prémisse}_1 \quad \cdots \quad \text{Prémisse}_n}{\text{Conclusion}}$$

et permet de construire de nouveaux théorèmes (Conclusion) dans le système à partir d'autres théorèmes (Prémisses$_i$).

Les formules se construisent à partir de $\mathcal{P}$ l'ensemble des variables propositionnelles suivant la grammaire suivante[2] :

$$\mathcal{F}_{\mathsf{LK}} ::= \mathcal{P} | \neg \mathcal{F}_{\mathsf{LK}} | \mathcal{F}_{\mathsf{LK}} \wedge \mathcal{F}_{\mathsf{LK}} | \mathcal{F}_{\mathsf{LK}} \vee \mathcal{F}_{\mathsf{LK}} | \mathcal{F}_{\mathsf{LK}} \Rightarrow \mathcal{F}_{\mathsf{LK}}$$

Dans le calcul des séquents, les règles sont réparties en trois catégories : les identités, les règles structurelles et les règles logiques. Pour la logique classique **LK**, ce sont celles décrites au tableau 1.1.

La règle axiome (Id), la seule sans prémisse, permet d'initier toute démonstration. La règle de coupure permet d'utiliser, pour démontrer un théorème, un résultat intermédiaire (un lemme) A.

[2] Nous ne considérerons que des systèmes propositionnels.

Identités : (*axiome* et *coupure*)	$\dfrac{}{A \vdash_{LK} A}\ \mathsf{Id}$	$\dfrac{\Gamma \vdash_{LK} A, \Delta \quad \Gamma', A \vdash_{LK} \Delta'}{\Gamma, \Gamma' \vdash_{LK} \Delta, \Delta'}\ \mathsf{Cut}$
Règles structurelles : (*échange*)	$\dfrac{\Gamma, A, B, \Gamma' \vdash_{LK} \Delta}{\Gamma, B, A, \Gamma' \vdash_{LK} \Delta}\ \mathsf{LX}$	$\dfrac{\Gamma \vdash_{LK} \Delta, A, B, \Delta'}{\Gamma \vdash_{LK} \Delta, B, A, \Delta'}\ \mathsf{RX}$
(*affaiblissement*)	$\dfrac{\Gamma \vdash_{LK} \Delta}{\Gamma, A \vdash_{LK} \Delta}\ \mathsf{LW}$	$\dfrac{\Gamma \vdash_{LK} \Delta}{\Gamma \vdash_{LK} A, \Delta}\ \mathsf{RW}$
(*contraction*)	$\dfrac{\Gamma, A, A \vdash_{LK} \Delta}{\Gamma, A \vdash_{LK} \Delta}\ \mathsf{LC}$	$\dfrac{\Gamma \vdash_{LK} A, A, \Delta}{\Gamma \vdash_{LK} A, \Delta}\ \mathsf{RC}$
Règles logiques :	$\dfrac{\Gamma, A, B \vdash_{LK} \Delta}{\Gamma, A \wedge B \vdash_{LK} \Delta}\ \mathsf{L}\wedge$	$\dfrac{\Gamma \vdash_{LK} A, \Delta \quad \Gamma' \vdash_{LK} B, \Delta'}{\Gamma, \Gamma' \vdash_{LK} A \wedge B, \Delta, \Delta'}\ \mathsf{R}\wedge$
	$\dfrac{\Gamma, A \vdash_{LK} \Delta \quad \Gamma', B \vdash_{LK} \Delta'}{\Gamma, \Gamma', A \vee B \vdash_{LK} \Delta, \Delta'}\ \mathsf{L}\vee$	$\dfrac{\Gamma \vdash_{LK} A, B, \Delta}{\Gamma \vdash_{LK} A \vee B, \Delta}\ \mathsf{R}\vee$
	$\dfrac{\Gamma \vdash_{LK} A, \Delta}{\Gamma, \neg A \vdash_{LK} \Delta}\ \mathsf{L}\neg$	$\dfrac{\Gamma, A \vdash_{LK} \Delta}{\Gamma, \vdash_{LK} \neg A, \Delta}\ \mathsf{R}\neg$
	$\dfrac{\Gamma \vdash_{LK} A, \Delta \quad \Gamma', B \vdash_{LK} \Delta'}{\Gamma, \Gamma', A \Rightarrow B \vdash_{LK} \Delta, \Delta'}\ \mathsf{L}\Rightarrow$	$\dfrac{\Gamma, A \vdash_{LK} B, \Delta}{\Gamma \vdash_{LK} A \Rightarrow B, \Delta}\ \mathsf{R}\Rightarrow$

TAB. 1.1 – Règles de la logique classique

Les règles logiques indiquent les manières autorisées d'introduire dans les formules les connecteurs de conjonction, disjonction, négation et implication. Ceci aussi bien pour les formules à gauche que les formules à droite du séquent.

Les règles structurelles[3] indiquent la manière de manipuler les contextes. Comme nous le verrons par la suite, elles précisent le sens des « et », « ou » et « implique ». [Gir95a] considère : « It is not too excessive to say that a logic is essentially a set of structural rules ! ».

Pour montrer cette influence des règles structurelles, étudions la règle suivante :

$$\frac{\Gamma \vdash_{\mathsf{LK}} A, \Delta \quad \Gamma \vdash_{\mathsf{LK}} B, \Delta}{\Gamma \vdash_{\mathsf{LK}} A \wedge B, \Delta}\ \mathsf{R}\wedge'$$

Par rapport à la règle $\mathsf{R}\wedge$, elle demande que les contextes des formules A et B soient exactement les mêmes. On peut alors se demander en quoi un système avec la règle $\mathsf{R}\wedge'$ différerait de celui défini avec la règle $\mathsf{R}\wedge$: les deux systèmes permettraient de prouver exactement les mêmes théorèmes. En effet, chacune des règles permet de simuler l'autre[4], comme le montrent les figures 1.1(a) et 1.1(b).

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash_{\mathsf{LK}} A, \Delta}{\Gamma, \Gamma' \vdash_{\mathsf{LK}} A, \Delta}\ \mathsf{LW}}{\Gamma, \Gamma' \vdash_{\mathsf{LK}} A, \Delta, \Delta'}\ \mathsf{RW} \qquad \dfrac{\dfrac{\Gamma' \vdash_{\mathsf{LK}} B, \Delta'}{\Gamma, \Gamma' \vdash_{\mathsf{LK}} B, \Delta'}\ \mathsf{LW}}{\Gamma, \Gamma' \vdash_{\mathsf{LK}} B, \Delta, \Delta'}\ \mathsf{RW}}{\Gamma, \Gamma' \vdash_{\mathsf{LK}} A \wedge B, \Delta, \Delta'}\ \mathsf{R}\wedge' \qquad \dfrac{\dfrac{\Gamma \vdash_{\mathsf{LK}} A, \Delta \quad \Gamma \vdash_{\mathsf{LK}} B, \Delta}{\Gamma, \Gamma \vdash_{\mathsf{LK}} A \wedge B, \Delta, \Delta}\ \mathsf{R}\wedge}{\Gamma \vdash_{\mathsf{LK}} A \wedge B, \Delta, \Delta}\ \mathsf{LC}}{\Gamma \vdash_{\mathsf{LK}} A \wedge B, \Delta}\ \mathsf{RC}$$

(a) Simulation de $\mathsf{R}\wedge$ (b) Simulation de $\mathsf{R}\wedge'$

FIG. 1.1 – Règles équivalentes

Autrement dit, grâce aux règles structurelles de contraction et d'affaiblissement, pour démontrer la conjonction de deux formules, peu importe que les contextes établissant chacune de ces formules diffèrent. En particulier, le remplacement des règles logiques décrites dans le tableau 1.1 par les règles du tableau 1.2 permet de prouver exactement les mêmes théorèmes.

$$\frac{\Gamma, A \vdash_{\mathsf{LK}'} \Delta}{\Gamma, A \wedge B \vdash_{\mathsf{LK}'} \Delta}\ \mathsf{L}\wedge_1 \qquad \frac{\Gamma, B \vdash_{\mathsf{LK}'} \Delta}{\Gamma, A \wedge B \vdash_{\mathsf{LK}'} \Delta}\ \mathsf{L}\wedge_2 \qquad \frac{\Gamma \vdash_{\mathsf{LK}'} A, \Delta \quad \Gamma \vdash_{\mathsf{LK}'} B, \Delta}{\Gamma \vdash_{\mathsf{LK}'} A \wedge B, \Delta}\ \mathsf{R}\wedge'$$

$$\frac{\Gamma, A \vdash_{\mathsf{LK}'} \Delta \quad \Gamma, B \vdash_{\mathsf{LK}'} \Delta}{\Gamma, A \vee B \vdash_{\mathsf{LK}'} \Delta}\ \mathsf{L}\vee' \qquad \frac{\Gamma \vdash_{\mathsf{LK}'} A, \Delta}{\Gamma \vdash_{\mathsf{LK}'} A \vee B, \Delta}\ \mathsf{R}\vee_1 \qquad \frac{\Gamma \vdash_{\mathsf{LK}'} B, \Delta}{\Gamma \vdash_{\mathsf{LK}'} A \vee B, \Delta}\ \mathsf{R}\vee_2$$

$$\frac{\Gamma \vdash_{\mathsf{LK}'} A, \Delta \quad \Gamma, B \vdash_{\mathsf{LK}'} \Delta}{\Gamma, A \Rightarrow B \vdash_{\mathsf{LK}'} \Delta}\ \mathsf{L}\Rightarrow \qquad \frac{\Gamma, A \vdash_{\mathsf{LK}'} B, \Delta}{\Gamma \vdash_{\mathsf{LK}'} A \Rightarrow B, \Delta}\ \mathsf{R}\Rightarrow$$

TAB. 1.2 – Règles alternatives d'introduction des connecteurs

Nous allons montrer dans la suite quels rôles jouent ces règles structurelles dans la modélisation des grammaires et pourquoi il semble naturel de les amender, voire de les abandonner, ce qui nous amènera à utiliser le système de la logique linéaire.

[3] En anglais, *eXchange*, *Contraction* et *Weakening*. Dans toutes les règles, le L et le R indiquent que la règle s'applique à gauche ou à droite.

[4] Avec des règles structurelles.

Notons qu'un fragment intéressant du calcul des séquents est celui dit *intuitionniste.* Il n'autorise qu'une seule formule du côté droit du signe $\vdash$. On s'aperçoit que les règles structurelles droites disparaissent, de même que la négation. C'est un moyen détourné de ne plus utiliser ces règles. Alors, si le séquent intuitionniste $\vdash_{\mathsf{LK}} A$ est prouvable (sans coupure), alors la dernière règle est une règle logique droite. En particulier, si A est la disjonction $A_1 \vee A_2$, cela signifie que $\vdash_{\mathsf{LK}} A_1$ ou $\vdash_{\mathsf{LK}} A_2$ est prouvable. Ceci n'est bien sûr pas vrai de **LK** puisque le tiers exclus $A \vee \neg A$ est prouvable même pour A variable propositionnelle.

1.1.2 Principe des grammaires de types logiques

Les grammaires de types logiques ont leurs racines dans les grammaires catégorielles. Ces dernières furent tout d'abord développées par [Ajd35] pour donner une structure syntaxique précise aux formules de la logique. Son application aux langues naturelles apparut avec les travaux [BH50] puis [Lam58].

Un des intérêts majeurs des grammaires de ce genre réside dans ses possibilités de traiter parallèlement les phénomènes syntaxiques et sémantiques, comme nous le montrerons au chapitre 5. Nous allons pour l'instant nous contenter de donner quelques principes suffisant à l'illustration par de simples exemples des propriétés requises par les logiques sous-jacentes.

Une propriété caractéristique de ces grammaires est d'être lexicalisées, c'est-à-dire que le lexique porte l'essentiel des connaissances syntaxiques et sémantiques de la langue. L'utilisation de ces connaissances intervient par l'intermédiaire des règles d'inférences, indépendamment donc de la langue. Dans le cas présent, le lexique associe à chaque mot une ou plusieurs formules de la logique (on appelle ces formules *type*, ou *type syntaxique*). De plus, une variable propositionnelle (en général le type *S*) permet de reconnaître les phrases syntaxiquement correctes. Le tableau 1.3 propose un exemple de lexique.

L'analyse d'une phrase, qui revient à se demander si la phrase considérée est syntaxiquement correcte, consiste à vérifier que le type S se dérive à partir des types associés à chacun des mots de la phrase. Plus formellement, étant donné un lexique fini qui, pour tout $i \in [1, n]$, associe au mot m_i les types $(T_1^{(i)}, \ldots, T_{l_i}^{(i)})$, la phrase $m_{a_1} m_{a_2} \cdots m_{a_p}$ est correcte si pour tout $i \in [1, p]$ il existe $j_i \in [1, l_{a_i}]$ tel que

$$T_{j_1}^{a_1}, T_{j_2}^{a_2}, \ldots, T_{j_p}^{a_p} \vdash_{\mathsf{LK}} S$$

Le langage reconnu est l'ensemble des phrases correctes.

Remarque 1. Dès que l'on parlera de phrase syntaxiquement correcte, on pensera « preuve ». Établir la correction d'une phrase reviendra à établir l'existence d'une preuve d'un séquent et construire une phrase correcte reviendra à construire une preuve de ce séquent. D'où l'importance accordée par la suite aux traitements automatiques de preuve.

Les grammaires logiques [AD89] mettent en œuvre ce principe de représentation de l'analyse par un procédé déductif à l'aide de grammaires décrivant faits et règles d'inférences en programmation logique (essentiellement avec Prolog). Demander si une phrase appartient bien au langage revient à laisser Prolog prouver cette hypothèse.

Toutefois, l'approche des grammaires logiques se distingue de celle des grammaires de types logiques, car si elles partagent la notion de prouvabilité, le moteur de Prolog n'utilise qu'une seule règle : celle de coupure (ou résolution, étendue au premier ordre) [CL93a, Boi88]. De plus, la résolution permet l'implémentation des règles des grammaires logiques, qui ne sont pas nécessairement des règles logiques. Par exemple, on peut dans ces grammaires décrire des phrases par la règle suivante [AD89] :

$$\text{statement} \rightarrow \text{name}, \text{verb}(L), \text{comps}(L)$$

L indique la liste attendue des compléments du verbe. Les règles suivantes complètent cette grammaire :

$$
\begin{aligned}
\text{name} &\rightarrow [\text{Ann}].\\
\text{name} &\rightarrow [\text{Tom}].\\
\text{name} &\rightarrow [\text{Paris}].\\
&\vdots\\
\text{comps}([]) &\rightarrow [].\\
\text{comps}([\text{arg}(P)|L]) &\rightarrow \text{comps}(L), \text{preposition}(P), \text{name}.\\
\text{preposition}(\text{direct}) &\rightarrow [].\\
\text{preposition}(\text{in}) &\rightarrow [\text{in}].\\
&\vdots\\
\text{verb}([]) &\rightarrow [\text{laughs}].\\
\text{verb}([\text{arg}(\text{in})]) &\rightarrow [\text{lives}].\\
&\vdots
\end{aligned}
$$

Au contraire, les règles des grammaires de types logiques sont les règles logiques, appliquées à des formules (les types), ce qui justifient leur nom de « de types logiques» car les preuves se construisent avec les règles logiques et les formules qui typent les entrées lexicales. On est plus proche du principe *parsing as deduction* de [PW83].

Mot	**Type**
Pierre	np
mange	$np \Rightarrow (np \Rightarrow S)$
une	$n \Rightarrow np$
pomme	n

TAB. 1.3 – Exemple de lexique

Ainsi on peut essayer de vérifier si la phrase *Pierre mange une pomme* est correcte, étant donné le lexique du tableau 1.3.

La preuve

$$
\dfrac{np \vdash_{\mathsf{LK}} np \qquad \dfrac{\dfrac{\dfrac{n \vdash_{\mathsf{LK}} n \qquad \dfrac{\dfrac{np \vdash_{\mathsf{LK}} np \quad S \vdash_{\mathsf{LK}} S}{np, np \Rightarrow S \vdash_{\mathsf{LK}} S}\mathsf{L}\Rightarrow}{np \Rightarrow S, np \vdash_{\mathsf{LK}} S}\mathsf{LX}}{n, np \Rightarrow S, n \Rightarrow np \vdash_{\mathsf{LK}} S}\mathsf{L}\Rightarrow}{np \Rightarrow S, n, n \Rightarrow np \vdash_{\mathsf{LK}} S}\mathsf{LX}}{np \Rightarrow S, n \Rightarrow np, n \vdash_{\mathsf{LK}} S}\mathsf{LX}}{\underset{\textit{Pierre}}{np}, \quad \underset{\textit{mange}}{np \Rightarrow (np \Rightarrow S)}, \quad \underset{\textit{une}}{n \Rightarrow np}, \quad \underset{\textit{pomme}}{n} \quad \vdash_{\mathsf{LK}} S}\mathsf{L}\Rightarrow
$$

nous assure la correction syntaxique[5]. Le lexique que nous proposons, avec le calcul de la logique classique, semble donc avoir une certaine capacité de modélisation. Pour aller plus loin, nous voudrions aussi nous assurer qu'une phrase reconnue incorrecte en français ne soit pas capable de dériver le type *S*.

Malheureusement, on s'aperçoit très vite que des phrases comme **Pierre mange une pomme pomme*[6] apparaissent correctes dans ce système :

$$\dfrac{np, np \Rightarrow (np \Rightarrow S), n \Rightarrow np, n \vdash_{\mathsf{LK}} S}{\underset{\textit{Pierre}}{np}, \quad \underset{\textit{mange}}{np \Rightarrow (np \Rightarrow S)}, \quad \underset{\textit{une}}{n \Rightarrow np}, \quad \underset{\textit{pomme}}{n}, \quad \underset{\textit{pomme}}{n} \quad \vdash_{\mathsf{LK}} S}\ \mathsf{LW}$$

montre que $np, np \Rightarrow (np \Rightarrow S), n \Rightarrow np, np \vdash_{\mathsf{LK}} S$ est un théorème car on a précédemment montré que $np, np \Rightarrow (np \Rightarrow S), n \Rightarrow np, np \vdash_{\mathsf{LK}} S$ en est un. De même, une utilisation judicieuse de LX nous permettrait de montrer que la phrase **Pierre une mange pomme* apparaît correcte.

Les règles structurelles se montrent les fautrices principales de ce laxisme syntaxique, et il devient naturel de considérer un calcul soumettant leur utilisation à plus de contrôle. La logique linéaire nous fournit ce cadre formel.

1.2 La logique linéaire

Quelles conséquences ont la suppression des règles d'affaiblissement et de contraction ? Nous allons les décrire d'abord en terme d'intuition du sens donné aux formules, puis plus formellement, avec les propriétés importantes du système qui en découle.

1.2.1 Action et situation

En logique classique, si A est vraie et $A \Rightarrow B$ l'est aussi, alors B est vraie et A *reste vraie*. Le fait d'avoir utilisé la formule A n'a rien changé à sa disponibilité pour d'autres démonstrations. En particulier, le séquent $A, A \Rightarrow B \vdash_{\mathsf{LK}} A \wedge B$ est prouvable :

$$\dfrac{\dfrac{\dfrac{}{A \vdash_{\mathsf{LK}} A}\,\mathsf{Id} \qquad \dfrac{\dfrac{}{A \vdash_{\mathsf{LK}} A}\,\mathsf{Id} \qquad \dfrac{}{B \vdash_{\mathsf{LK}} B}\,\mathsf{Id}}{A, A \Rightarrow B \vdash_{\mathsf{LK}} B}\,\mathsf{L}\Rightarrow}{A, A, A \Rightarrow B \vdash_{\mathsf{LK}} A \wedge B}\,\mathsf{R}\wedge}{A, A \Rightarrow B \vdash_{\mathsf{LK}} A \wedge B}\,\mathsf{LC}$$

La suppression des règles d'affaiblissement et de contraction empêche de telles dérivations, et la notion habituelle de vérité, notion pérenne, ne convient plus comme sémantique informelle (intuitive) du nouveau système : la logique linéaire. Par contre, la notion de consommation (dans la preuve de B, A et $A \Rightarrow B$ ont été consommées. Il est donc normal que A ne soit plus disponible alors que B l'est), et donc de ressources, donne une représentation intuitive des formules de la logique linéaire.

Cette notion de consommation et de ressources pourrait paraître artificielle, pourtant il traduit la notion d'échange très présente dans le monde matériel. Ainsi (pour reprendre l'exemple classique de [Gir95b]), le caractère causal de l'implication empêche (dans le monde réel) sa réitération, puisque les prémisses sont modifiées. Par exemple, si A exprime une dépense de 20F, et B exprime l'obtention

[5] Dans la suite il nous arrivera souvent d'omettre les règles d'échange dans les preuves.

[6] Comme d'habitude, une phrase précédée du signe * est une phrase considérée incorrecte en français.

d'un paquet de cigarettes, dans l'action d'obtenir un paquet de cigarettes en dépensant 20F, on perd effectivement 20F qui ne peuvent pas être réutilisés.

La section 1.2.3 illustre cette notion, après que les connecteurs de la logique linéaire ont été introduits.

1.2.2 Calcul des séquents

Nous allons maintenant donner, comme nous l'avions fait pour la logique classique, les règles du calcul des séquents linéaires.

Puisque la simulation de R$\wedge$ par R$\wedge'$ nécessite les règles LC et LW, leur suppression annule cette équivalence. C'est pourquoi apparaissent *deux* connecteurs pour la coordination : l'un qui demande deux contextes différents (noté $\otimes$, le *tenseur*), et l'autre qui demande le même contexte (noté $\&$, *avec*). De même pour les disjonctions (respectivement notées $\parr$, le *par* et $\oplus$, *plus*).

Les connecteurs demandant le même contexte sont dits *additifs*, les autres sont dits *multiplicatifs*. Les autres connecteurs sont :

- la négation $A^\perp$;
- l'implication linéaire (multiplicative) $\multimap$ qui peut être définie comme $A \multimap B = A^\perp \parr B$;
- les connecteurs *exponentiels* (notés !, *bien sûr* et ?, *pourquoi pas*), duaux.

Ils ont les propriétés suivantes :

- La négation est involutive ($A = A^{\perp\perp}$) ;
- les connecteurs duaux ($\parr$ et $\otimes$, $\oplus$ et $\&$, ! et ?) vérifient les lois de De Morgan ;
- $\otimes$, $\parr$, $\oplus$ et $\&$ sont commutatifs ;
- $\otimes$ se distribue sur $\oplus$ (et $\parr$ sur $\&$).

À partir de l'ensemble des variables propositionnelles désigné par $\mathcal{P}$, l'ensemble $\mathcal{F}_{\text{LL}}$ des formules se construit selon la grammaire :

$$\mathcal{F}_{\text{LL}} ::= \mathcal{P} | \mathcal{F}_{\text{LL}}^\perp | \mathcal{F}_{\text{LL}} \otimes \mathcal{F}_{\text{LL}} | \mathcal{F}_{\text{LL}} \parr \mathcal{F}_{\text{LL}} | \mathcal{F}_{\text{LL}} \multimap \mathcal{F}_{\text{LL}} | \mathcal{F}_{\text{LL}} \& \mathcal{F}_{\text{LL}} | \mathcal{F}_{\text{LL}} \oplus \mathcal{F}_{\text{LL}} | !\mathcal{F}_{\text{LL}} | ?\mathcal{F}_{\text{LL}}$$

Les règles sont présentées dans le tableau 1.4[7]. Par rapport aux séquents classiques (bilatères), le calcul des séquents linéaires se présente habituellement sous forme unilatère. Cela offre une présentation plus condensée, sachant que l'on peut toujours convertir un séquent bilatère en un séquent unilatère grâce à la négation : de $A_1, \ldots, A_n \vdash B_1, \ldots, B_n$ à $\vdash A_1^\perp, \ldots, A_n^\perp, B_1, \ldots, B_n$, et réciproquement.

Signalons tout d'abord que la virgule du séquent (le « ou ») que l'on considère est la disjonction multiplicative. Dans le cas de séquents bilatères, la virgule du côté gauche est la conjonction multiplicative. Ensuite, les règles d'affaiblissement et de contraction ne réapparaissent que contrôlées par les modalités ! et ?. Ces connecteurs unaires permettent en quelque sorte de redonner à une formule le statut qu'elle avait en logique classique, c'est-à-dire d'être vraie, ou plutôt d'être une ressource dont la disponibilité est infinie. Ainsi toute formule F de la logique intuitionniste peut-elle être modélisée par F^* [Gir87a] :

- si $F = A$ et A atomique, alors $F^* = !A$ (pour qu'une formule de la logique linéaire retrouve son statut classique, il faut lui donner une disponibilité infinie) ;
- si $F = \neg A$ alors $F^* = ?(A^*)^\perp = (!A^*)^\perp$;
- si $F = A \wedge B$ alors $F^* = A^* \& B^*$;
- si $F = A \vee B$ alors $F^* = !A^* \oplus !B^*$;
- si $F = A \Rightarrow B$ alors $F^* = (!A^*) \multimap B^* = ?(A^*)^\perp \parr B^*$.

[DJS94] détaille différentes modélisations de la logique classique et de la logique intuitionniste.

[7] On notera $?\Gamma$ une suite de formules $?A_1, \ldots, ?A_n$.

Identités :
$$\frac{}{\vdash A, A^{\perp}}\,\mathsf{Id} \qquad \frac{\vdash \Gamma, A \quad \vdash A^{\perp}, \Delta}{\vdash \Gamma, \Delta}\,\mathsf{Cut}$$

Règles structurelles :
$$\frac{\vdash \Gamma, A, B, \Delta}{\vdash \Gamma, B, A, \Delta}\,\mathsf{X}$$

Règles logiques :
$$\frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}\otimes \qquad \frac{\vdash A, B, \Gamma}{\vdash A \parr B, \Gamma}\parr$$

$$\frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \& B}\,\& \qquad \frac{\vdash A, \Gamma}{\vdash A \oplus B, \Gamma}\oplus_1 \qquad \frac{\vdash B, \Gamma}{\vdash A \oplus B, \Gamma}\oplus_2$$

$$\frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,! \qquad \frac{\vdash A, \Gamma}{\vdash ?A, \Gamma}\,\mathsf{d?} \qquad \frac{\vdash \Gamma}{\vdash ?A, \Gamma}\,\mathsf{w?} \qquad \frac{\vdash ?A, ?A, \Gamma}{\vdash ?A, \Gamma}\,\mathsf{c?}$$

TAB. 1.4 – Règles de la logique linéaire

Notation. Nous noterons $\vdash [\Delta]\Gamma$, où Γ et Δ sont comme d'habitude une suite de formules, la conclusion d'une preuve de $\vdash \Gamma$ contenant des Cut sur les formules de Δ. La preuve sera notée $\pi(\vdash [\Delta]\Gamma)$.

Suivant les règles logiques considérées, on peut définir différents fragments. On parlera du fragment multiplicatif lorsqu'on ne considère que le $\parr$ et le $\otimes$ (**MLL**, pour *multiplicative linear logic*). Ce fragment peut être étendu avec le fragment additif lorsqu'on considère en plus le $\&$ et le $\oplus$ (**MALL**, pour *multiplicative additive linear logic*), ou étendu avec le fragment exponentiel lorsqu'on considère en plus ! et ? (**MELL**, pour *multiplicative exponential linear logic*).

Enfin notons que nous avons conservé une règle structurelle : la règle d'échange. Pourtant, l'exemple de *Pierre une mange pomme* indique que cette règle aussi pose problème. Néanmoins nous traiterons les problèmes de commutativité et de non-commutativité à part : différentes solutions peuvent être envisagées ([Yet90, Abr91, Ret93, Rue97] pour les problèmes théoriques de commutativité dans la logique linéraire, ou bien [Moo96, LR95] pour les aspects de non-commutativité dans la modélisation linguistique).

1.2.3 Interprétation informelle des connecteurs

Deux familles de connecteurs, multiplicatifs et additifs, apparaissent là où en logique classique une seule était présente. Quel sens donner alors à cette conjonction et cette disjonction multiplicatives, à cette conjonction et cette disjonction additives ?

Les deux conjonctions $\otimes$ et $\&$ correspondent à deux usages différents. Dans le cas de $B \otimes C$, les deux actions sont possibles et peuvent être faites (les contextes différents montrent que l'on a suffisamment de ressources pour effectuer l'un *et* l'autre). Dans le cas de $B \& C$, les deux actions également sont possibles dans le contexte, mais une seule peut être effectivement réalisée grâce aux ressources présentes (le contexte permet de réaliser chacune des deux actions, mais pas simultanément). En reprenant l'exemple des paquets de cigarettes, si l'on a :

- A : dépenser 20F ;
- B : obtenir un paquet de Gauloises ;
- C : obtenir un paquet de Gitanes,

on peut exprimer qu'une action de type $A \multimap B$ permet d'obtenir un paquet de Gauloises en dépensant 20F. Supposons donc que l'on ait $A \multimap B$ et $A \multimap C$ (actions possibles chez un marchand de tabac par exemple), si, avec 20F, les deux actions d'obtenir un paquet de Gauloises et un paquet de Gitanes sont possibles, elles ne le sont pas simultanément : on ne peut pas former $A \multimap B \otimes C$ (avec 20F, on n'obtient pas à la fois un paquet de Gitanes et un paquet de Gauloises. Cependant, il peut y avoir l'action $A \otimes A \vdash B \otimes C$ signifiant que si l'on double les ressources, les deux actions B et C deviennent possibles simultanément). Par contre, l'action $A \multimap B \& C$ est possible, indiquant que la ressource A nous permet de réaliser l'action B et de réaliser l'action C, mais pas simultanément.

La négation peut se comprendre également en terme de ressource : A est une offre de ressource, tandis que $A^{\perp}$ est une demande de la même ressource.

Dans notre cas, si l'on exprime que les verbes transitifs attendent deux syntagmes nominaux pour être transformés en phrases (*mange* : *np* $\multimap$ (*np* $\multimap$ *S*)), chacune des expressions *Pierre* (*np*) et *une pomme* (*np*) apportant exactement une ressource, elles vont pouvoir être utilisées une et une seule fois dans la formation d'une phrase (Pierre mange une pomme). Et des expressions comme **Pierre mange une pomme pomme* sont rejetées.

La vision de ressources et de nécessité de les gérer se retrouve dans d'autres formalismes grammaticaux, par exemple avec les *principes d'unicité et de complétude* en LFG (*Lexicalized Functionnal Grammar* [KB82, Abe93]). Ici, le formalisme utilisé pour la modélisation est lui-même sensible aux ressources. Notons que cet aspect est utilisé dans d'autres modélisations en informatique telles que les réseaux de Petri [MOM89], les calculs de processus [Asp91] ou l'ordonnancement [MTV93].

1.2.4 L'élimination des coupures

Dans ce paragraphe, nous allons évoquer des propriétés de la logique linéaire qui se révèlent à la fois importantes sur le plan même de la théorie et sur les applications que nous voulons en faire à la linguistique.

La première de ces propriétés est l'élimination des coupures. Du point de vue de la recherche de preuve (c'est-à-dire chercher un arbre dont les nœuds sont les règles du calcul considéré, dont les feuilles sont des règles Id et dont la racine est le séquent à prouver), on utilise les règles du bas vers le haut. On s'aperçoit alors que pour chaque règle (c'est vrai pour la logique linéaire comme pour la logique classique), à l'exception de la règle Cut, toute formule des prémisses est présente comme formule ou sous-formule dans la conclusion, ce qui permet de savoir exactement quelles règles au plus permettent de prouver le séquent dont on cherche à établir la correction.

Une preuve sans règle Cut vérifie la *propriété de la sous-formule* : elle ne fait apparaître que des sous-formules des formules de la conclusion.

Cette propriété permet d'établir, en l'absence de Cut, la décidabilité du calcul propositionnel classique. Ce résultat est faux pour la logique linéaire avec modalités [LMSS92], mais est vrai pour une variante (dite *affine*) où la règle d'affaiblissement n'est plus contrôlée par une modalité [Kop95] et vrai aussi pour le fragment multiplicatif. D'où son importance dans le cadre de la *recherche automatique de preuve.*

Pour la logique classique **LK**, la démonstration utilise un système équivalent. Définissons le système **LK″** de la manière suivante : un séquent Γ est désormais un *ensemble* de formules, et $\Gamma = \{A_1, \ldots, A_n\}$, qu'on écrira toujours $A_1, \ldots, A_n$. (Ceci va nous permettre de ne plus exprimer explicitement les règles structurelles.) Ses règles d'inférences sont celles exprimées dans le tableau 1.5.

Théorème 1. *Un séquent* $\Gamma \vdash_{LK} \Delta$ *est prouvable sans coupure dans* **LK** *si et seulement si* $\Gamma \vdash_{LK''} \Delta$ *est prouvable dans* **LK″**.

$$\frac{}{\Gamma, A \vdash_{\mathsf{LK}''} A, \Delta}\ \mathsf{Ax}''$$

$$\frac{\Gamma, B, A \vdash_{\mathsf{LK}''} \Delta}{\Gamma, A \wedge B \vdash_{\mathsf{LK}''} \Delta}\ \mathsf{L}\wedge_1''
\qquad
\frac{\Gamma, A, B \vdash_{\mathsf{LK}''} \Delta}{\Gamma, A \wedge B \vdash_{\mathsf{LK}''} \Delta}\ \mathsf{L}\wedge_2''
\qquad
\frac{\Gamma \vdash_{\mathsf{LK}''} A, \Delta \quad \Gamma \vdash_{\mathsf{LK}''} B, \Delta}{\Gamma \vdash_{\mathsf{LK}''} A \wedge B, \Delta}\ \mathsf{R}\wedge''$$

$$\frac{\Gamma, A \vdash_{\mathsf{LK}''} \Delta \quad \Gamma, B \vdash_{\mathsf{LK}''} \Delta}{\Gamma, A \vee B \vdash_{\mathsf{LK}''} \Delta}\ \mathsf{L}\vee''
\qquad
\frac{\Gamma \vdash_{\mathsf{LK}''} A, B, \Delta}{\Gamma \vdash_{\mathsf{LK}''} A \vee B, \Delta}\ \mathsf{R}\vee_1''
\qquad
\frac{\Gamma \vdash_{\mathsf{LK}''} B, A, \Delta}{\Gamma \vdash_{\mathsf{LK}''} A \vee B, \Delta}\ \mathsf{R}\vee_2''$$

TAB. 1.5 – Règles de **LK″**

À partir de ce théorème, puisqu'il n'y a qu'un nombre fini de sous-formules pour une formule, et qu'un séquent de **LK″** possède un nombre fini de formules, il n'y a qu'un nombre fini de séquents de sous-formules pour un séquent donné de **LK″**. Les arbres de preuve (sans coupure) non redondants (tels que si un même séquent apparaît deux fois dans la même branche, la partie qu'il y a entre les deux est supprimée) sont alors en nombre fini et il suffit de regarder si l'un d'entre eux est une preuve pour décider si un séquent est prouvable dans **LK″** [Gir87b]. Et donc, il est également décidable de dire si $\Gamma \vdash_{\mathsf{LK}} \Delta$ est un théorème de **LK** *sans coupure*.

La règle Cut présente donc un sérieux désavantage, qui demande l'invention *ex nihilo* d'une formule A. Mais que devient le système sans cette règle ? Peut-on produire plus ou moins de théorèmes ?

La réponse à cette question consiste en une des propriétés essentielles d'une logique : *l'élimination des coupures*. Dans un calcul pourvu de cette propriété, toute preuve d'un séquent utilisant la règle Cut peut se réécrire en une preuve *sans* règle Cut (*cut-free*).

Théorème 2 (Gentzen, 1935). *La règle Cut est redondante dans le calcul des séquents classiques.*

Différentes preuves existent. On peut en trouver par exemple dans [Gen69, GLT88, Joi93] pour des approches variées.

D'après ce théorème, s'il existe une preuve d'un séquent $\Gamma \vdash_{\mathsf{LK}} \Delta$, il en existe une preuve sans coupure. Or il est décidable de dire s'il existe une preuve sans coupure, donc **LK** est décidable.

Théorème 3 (Girard). *Il y a un algorithme transformant toute preuve d'un séquent* $\vdash \Gamma$ *de la logique linéaire en une preuve sans Cut du même séquent.*

On trouvera une preuve de ce théorème par exemple dans [Gir95b].

Outre son importance pour la recherche automatique de preuve, on verra par la suite (section 3.2) que dans l'isomorphisme de Curry-Howard [How80b] entre preuve et λ-calcul, l'élimination des coupures joue le rôle important de contrepartie à la β-réduction.

Néanmoins nous ne développerons pas les preuves d'élimination des coupures pour le calcul des séquents, préférant les développer dans le cas d'une syntaxe de la logique linéaire non plus sous forme de séquents, mais sous forme de réseaux. Cette syntaxe, que nous présentons maintenant, est celle que nous utiliserons pour nos travaux de modélisation.

1.3 Recherche de preuve et complexité

Lorsque l'on parle de recherche de preuve, il faut bien distinguer la vérification de la recherche de preuve. Dans le premier cas, cela peut être très rapide (il suffit de vérifier que l'arbre de démonstration a bien des règles valides, ou bien dans le cas des réseaux de vérifier un critère de correction. Dans ce dernier cas, cela peut être quadratique, voire linéaire [Gue99]).

La recherche proprement dite, elle, requiert en plus la construction de l'arbre (ou du réseau) et est donc en général plus difficile. Dans le cas multiplicatif, par exemple, le problème est NP-complet. Le tableau 1.6 (tiré de [LMSS92]) résume la complexité pour différents fragments.

connecteurs du fragment	complexité du fragment
$\otimes \parr \quad \& \oplus \quad !?$	indécidable [LMSS92]
$\otimes \parr \quad \& \oplus$	PSpace-complet [LMSS92]
$\otimes \parr \quad\quad !?$	inconnu
$\otimes \parr$	NP-complet [Kan91]

TAB. 1.6 – Complexités des fragments de la logique linéaire

À noter que pour les fragments intuitionnistes implicatifs cycliques [Yet90] respectivement associatifs et non associatifs de la logique linéaire, qui correspondent au calcul de Lambek associatif et non associatif, la complexité est respectivement inconnue et PTime [dG99].

Chapitre 2

Les réseaux de preuve de la logique linéaire

Le rôle important de la preuve et de la recherche de preuve dans l'utilisation des grammaires de types logiques nous amène à considérer ce qui peut différencier deux preuves. En effet, comme le montre la section 3.2, il existe une manière de calculer la sémantique[8] d'une phrase à partir de la preuve de sa correction. Du nombre de preuves dépend donc le nombre de sens possibles de la phrase, c'est-à-dire son ambiguïté.

Le calcul de la sémantique se base sur la structure de la preuve, mais deux différences structurelles, si elles sont inessentielles, peuvent conduire à une seule et même sémantique. Ainsi du séquent $\vdash ((A \otimes B) \otimes C) \otimes D, A^{\perp} \parr C^{\perp}, B^{\perp} \parr D^{\perp}$ peut-on avoir les preuves suivantes :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash A, A^{\perp} \quad \vdash B, B^{\perp}}{\vdash A \otimes B, A^{\perp}, B^{\perp}}\otimes \qquad \vdash C, C^{\perp}}{\vdash (A \otimes B) \otimes C, A^{\perp}, C^{\perp}, B^{\perp}}\otimes \qquad \vdash D, D^{\perp}}{\vdash \boldsymbol{((A \otimes B) \otimes C) \otimes D}, A^{\perp}, C^{\perp}, B^{\perp}, D^{\perp}}\otimes}{\vdash ((A \otimes B) \otimes C) \otimes D, \boldsymbol{A^{\perp} \parr C^{\perp}}, B^{\perp}, D^{\perp}}\parr}{\vdash ((A \otimes B) \otimes C) \otimes D, A^{\perp} \parr C^{\perp}, \boldsymbol{B^{\perp} \parr D^{\perp}}}\parr \tag{2.1}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash A, A^{\perp} \quad \vdash B, B^{\perp}}{\vdash A \otimes B, A^{\perp}, B^{\perp}}\otimes \qquad \vdash C, C^{\perp}}{\vdash (A \otimes B) \otimes C, A^{\perp}, C^{\perp}, B^{\perp}}\otimes \qquad \vdash D, D^{\perp}}{\vdash \boldsymbol{((A \otimes B) \otimes C) \otimes D}, A^{\perp}, C^{\perp}, B^{\perp}, D^{\perp}}\otimes}{\vdash ((A \otimes B) \otimes C) \otimes D, A^{\perp}, C^{\perp}, \boldsymbol{B^{\perp} \parr D^{\perp}}}\parr}{\vdash ((A \otimes B) \otimes C) \otimes D, \boldsymbol{A^{\perp} \parr C^{\perp}}, B^{\perp} \parr D^{\perp}}\parr \tag{2.2}$$

[8] On différencie bien entendu la sémantique (le modèle) associée au calcul de la sémantique d'une phrase de la langue naturelle. Nous parlons ici de cette dernière.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash A, A^\perp \quad \vdash B, B^\perp}{\vdash A \otimes B, A^\perp, B^\perp}\otimes \quad \vdash C, C^\perp}{\vdash (A \otimes B) \otimes C, A^\perp, C^\perp, B^\perp}\otimes}{\vdash (A \otimes B) \otimes C, \boldsymbol{A^\perp \parr C^\perp}, B^\perp}\parr \quad \vdash D, D^\perp}{\vdash \boldsymbol{((A \otimes B) \otimes C) \otimes D}, A^\perp \parr C^\perp, B^\perp, D^\perp}\otimes}{\vdash ((A \otimes B) \otimes C) \otimes D, A^\perp \parr C^\perp, \boldsymbol{B^\perp \parr D^\perp}}\parr \tag{2.3}$$

On remarque que ces trois preuves ne diffèrent que par l'ordre d'introduction des connecteurs et que seule la linéarité de la syntaxe choisie contraint ce dernier. Si les preuves représentent, comme on l'a vu au chapitre précédent, des analyses d'expressions, cela signifie qu'une expression admet plusieurs analyses. Cela peut être justifié, notamment dans le cas où l'expression est ambiguë. Si les différences entre les preuves sont inessentielles, elles donnent naissance à des ambiguïtés dites *fallacieuses* (ou *spurious ambiguities*). En outre ces ambiguïtés augmentent énormément et inutilement la complexité du problème de recherche de preuve.

Par contre, un même séquent $\vdash A^\perp \otimes B, B \otimes B^\perp, (B^\perp \otimes B) \parr (B^\perp \parr B)$ peut avoir deux preuves :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash A, A^\perp \quad \vdash B_1, B_1^\perp}{\vdash A, A^\perp \otimes B_1, B_1^\perp}\otimes \quad \vdash B_2, B_2^\perp}{\vdash A, A^\perp \otimes B_1, B_1^\perp \otimes B_2, B_2^\perp}\otimes \quad \vdash B_3, B_3^\perp}{\vdash A, A^\perp \otimes B_1, B_1^\perp \otimes B_2, B_2^\perp \otimes B_3, B_3^\perp}\otimes}{\vdash A^\perp \otimes B_1, B_1^\perp \otimes B_2, B_2^\perp \otimes B_3, B_3^\perp \parr A}\parr}{\vdash A^\perp \otimes B_1, B_1^\perp \otimes B_2, (B_2^\perp \otimes B_3) \parr (B_3^\perp \parr A)}\parr \tag{2.4}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash B_1, B_1^\perp \quad \vdash B_2, B_2^\perp}{\vdash B_1, B_1^\perp \otimes B_2, B_2^\perp}\otimes \quad \vdash A, A^\perp}{\vdash A, A^\perp \otimes B_1, B_1^\perp \otimes B_2, B_2^\perp}\otimes \quad \vdash B_3, B_3^\perp}{\vdash A, A^\perp \otimes B_1, B_1^\perp \otimes B_2, B_2^\perp \otimes B_3, B_3^\perp}\otimes}{\vdash A^\perp \otimes B_1, B_2^\perp \otimes B_3, B_1^\perp \otimes B_2, B_3^\perp \parr A}\parr}{\vdash A^\perp \otimes B_1, B_2^\perp \otimes B_3, (B_1^\perp \otimes B_2) \parr (B_3^\perp \parr A)}\parr \tag{2.5}$$

qui diffèrent de manière plus essentielle : dans l'une des preuves, B_3 et $B_3^\perp$ appartiennent à des sous-formules d'une même formule, mais pas dans l'autre preuve. La deuxième preuve déconnecte ces deux conclusions de l'axiome.

Pour obvier à ce problème, [Gir87a] propose une autre syntaxe pour la logique linéaire. Cette syntaxe, née de la volonté d'avoir une « déduction naturelle » pour la logique linéaire, conserve toutefois d'importantes propriétés de symétrie — contrairement à la déduction naturelle (voir section 3.2.1) qui n'autorise qu'une seule formule à droite du séquent.

Deux éléments guident cette syntaxe : le premier est de marquer les atomes duaux, reliés entre eux par un lien axiome. Le deuxième est d'isoler, pour chaque règle utilisée dans une preuve, la nouvelle formule. Les trois premières preuves du même séquent

$$\vdash ((A \otimes B) \otimes C) \otimes D, A^\perp \parr C^\perp, B^\perp \parr D^\perp$$

montrent que s'il est nécessaire d'avoir déjà construit la formule $(A \otimes B) \otimes C$ pour construire $((A \otimes B) \otimes C) \otimes D$ ou $A^\perp \parr C^\perp$, ces deux formules peuvent être construites indépendamment l'une de l'autre, parallèlement. On extrait en quelque sorte des preuves séquentielles un graphe de dépendance entre les formules (qui revient finalement à l'arbre syntaxique de la formule même), auquel on ajoute les liens axiomes.

Ainsi, les trois premières preuves permettent de construire le graphe de la figure 2.1, alors que les deux suivantes amènent à la construction des deux réseaux de la figure 2.2.

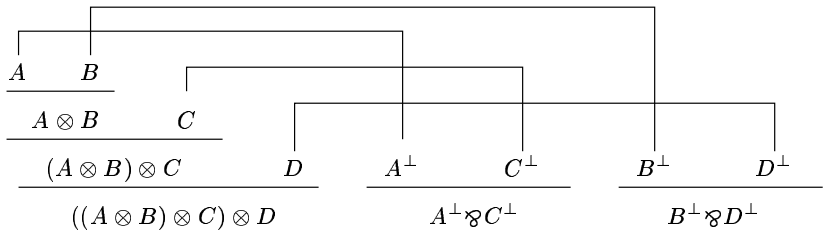


FIG. 2.1 – Un réseau pour trois preuves de $\vdash ((A \otimes B) \otimes C) \otimes D, A^\perp \parr C^\perp, B^\perp \parr D^\perp$

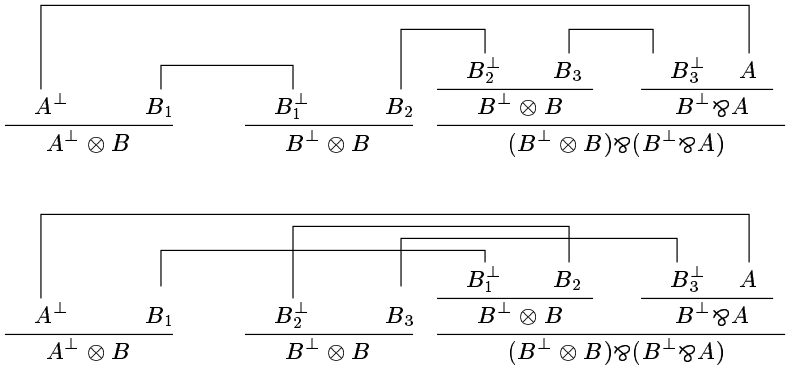


FIG. 2.2 – Deux réseaux pour deux preuves de $\vdash A^\perp \otimes B, B \otimes B^\perp, (B^\perp \otimes B) \parr (B^\perp \parr B)$

2.1 La syntaxe

Différentes syntaxes ont évolué à partir de la première, définie par [Gir87a]. On notera en particulier celles de [DR89, Ret93, Ret96b, Gue99]. Parce qu'elle permet de traiter plus facilement les cas non commutatifs, en particulier celui correspondant au calcul de Lambek (voir [AM98]) ou s'étend à *pomset logic* [Ret93, Ret97b], nous avons choisi la présentation de [Ret96b, Ret99]. Même si des réseaux ont été proposés pour les connecteurs additifs [Gir87a], nous nous limiterons dans la suite aux connecteurs multiplicatifs et nous travaillerons dans le fragment multiplicatif de la logique linéaire (MLL), quelquefois augmenté des exponentiels.

Toutes ces syntaxes ont en commun d'être graphiques. C'est-à-dire que chaque preuve ne se représente plus comme dans le calcul des séquents par un arbre mais par une autre forme graphique (appelée généralement *structure de preuve* ou *préréseau*). Comme pour les arbres de preuve dont seuls quelques-uns, parmi tous les arbres possibles dont les noeuds sont des règles, correspondent à des preuves, parmi toutes les structures de preuve possibles, seules quelques-unes correspondent à des

preuves : les *réseaux de preuve*. Elles se caractérisent par une propriété particulière (dépendante de la représentation choisie) appelée *critère de correction.* Cette propriété permet de traiter des réseaux de manière intrinsèque, sans revenir au calcul des séquents. Néanmoins, pour s'assurer que l'on continue à parler de la même chose, chaque représentation possible doit établir deux propriétés particulièrement importantes : la première, facile en général, est de montrer que chaque preuve dans le calcul des séquents peut se traduire sous forme de réseau de preuve, c'est-à-dire de structure de preuve vérifiant le critère de correction. La seconde, appelée *séquentialisation* montre à l'inverse qu'à tout réseau de preuve correspond une preuve du calcul des séquents.

Nous allons donc, après avoir défini notre choix pour la représentation graphique et le critère de correction, montrer chacune de ces propriétés. En particulier, nous commençons par présenter le formalisme de représentation des réseaux : les R&B-graphes.

2.1.1 Graphes

Nous suivons la définition de [Ret99] des réseaux comme R&B-graphes, en l'adaptant toutefois pour tenir compte des exponentiels. Une conséquence est que les propriétés montrées sur les R&B-graphes et les réseaux pour le fragment multiplicatif sont reprises de [Ret99] et légèrement modifiées. Par contre, à partir du moment où l'on considère les axiomes propres, l'utilisation des boîtes dans les R&B-graphes ou les exponentiels sans boîtes, les démonstrations sont nôtres. Nous considérons le calcul sans mix, c'est-à-dire que nous ne considérons que des graphes connexes. Donnons maintenant la terminologie précise.

Définition 1. Un *graphe* $G = (V, E)$ est défini par :
- V un ensemble fini de *sommets* ;
- $E \subset V \times V$ un multi-ensemble d'arêtes. Une arête se note xy, avec un indice s'il y a plusieurs arêtes xy, et les paires ne sont pas ordonnées ($\forall (x, y) \in V^2, xy = yx$). G est *simple* si E est un ensemble.

Si $xy \in E$ on dit que l'arête xy *joint* les sommets x et y et qu'elle est *incidente* à ces sommets. Ceux-ci sont dits *adjacents.* De même, deux arêtes partageant un même sommet sont dites *adjacentes,* et *indépendantes* dans le cas contraire.

Le *degré* d'un sommet est le nombre d'arêtes qui lui sont incidentes. On dit qu'un sommet est *pendant* si son degré est 1.

Soient G et H deux graphes et $f : G \longrightarrow H$ une application bijective sur H. f est un *isomorphisme* si elle conserve le nombre d'arêtes entre chaque paire de sommets. Deux graphes sont dits *isomorphes* s'il existe une telle bijection.

Remarque. Cette définition autorise les arêtes réflexives $xx \in V \times V$, contrairement aux définitions classiques pour les graphes series-parallèles [Möh89].

Notation. Pour un ensemble fini E, $|E|$ est le nombre de ses éléments. L'union disjointe de deux ensembles E_1 et E_2 se note $E_1 \uplus E_2$, l'union de multi-ensembles se note $E_1 \oplus E_2$.

De plus, pour un graphe $G = (V, E)$ et $V' \subset V$, $G|_{V'} = (V', E \cap (V' \times V'))$.

Définition 2. Un *chemin* c de *longueur* n est une suite $c = x_1 x_2 \cdots x_{n+1}$ de sommets (les x_i) et d'arêtes (les $x_i x_{i+1}$) telles que $\forall i \in [1, n], x_i x_{i+1} \in E$. On note :
- ϵ le chemin vide ;
- $\overline{c} = \cup_1^{n+1} \{x_i\}$;
- $c \downarrow_{x_i} = x_1 \cdots x_i$;
- $c \uparrow_{x_i} = x_i \cdots x_{n+1}$;
- $c^{-1} = x_{n+1} \cdots x_1$;

– $\odot$ la concaténation de chemins : si $c_1 = x_1 \cdots x_n$ et $c_2 = x_n \cdots x_{n+m}$ sont deux chemins, alors $c_1 \odot c_2 = x_1 \cdots x_n x_{n+1} \cdots x_{n+m}$ est un chemin.

Un *sous-chemin* c' de c est un chemin tel qu'il existe c_1 et c_2 chemins avec $c = c_1 \odot c' \odot c_2$ (c_1 et c_2 sont aussi des sous-chemins).

Si les sommets sont deux à deux distincts, le chemin est *élémentaire*. Il est simple s'il n'existe pas $i \neq j$ tels que $x_i = x_j$ et $x_{i+1} = x_{j+1}$. Par contre, on peut avoir $x_i = x_{j+1}$ et $x_{i+1} = x_j$.

Un *cycle* est un chemin dont la longueur est au moins 2 et dont le premier et le dernier sommet sont les mêmes. Si tous les sommets sont deux à deux distincts, exceptés le premier et le dernier, le cycle est *élémentaire*.

Définition 3. Soit $G = (V, E)$ et $H = (V', E')$ deux graphes. H est un sous-graphe de G si $V' \subset V$ et $E' \subset E$.

Si H est un sous-graphe de G tel que $\forall (x, y) \in V'^2, xy \in E \Rightarrow xy \in E'$ alors H est un sous-graphe *induit* de G.

S'il existe un chemin entre toute paire de sommets, le graphe est *connexe*. Les sous-graphes connexes maximaux induits de G sont appelés ses *composantes connexes*. Une arête xy est un *isthme* si $G' = (V, E \backslash \{xy\})$ a plus de composantes connexes que G. Un ensemble d'arêtes $A = \{a_1, \ldots, a_n\}$ est un *ensemble scindant* si $G' = (V, E \backslash A)$ a plus de composantes connexes que G.

Exemple. – x y est sous-graphe de x—y mais n'est pas sous-graphe induit.

– x—y est sous-graphe induit de x—y z

Définition 4. Soit $G = (V, E)$ un graphe et $B \subset E$ tel que les arêtes de B ne soient pas deux à deux adjacentes. B est appelé *couplage*. Il est *parfait* si tout sommet de G est incident à une arête de B.

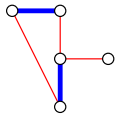
Dans un graphe $G = (V, E)$ et un couplage B, un chemin p est *alternant* si les arêtes de p sont alternativement dans B et dans $E \backslash B$.

On note :

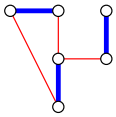
– *æ-chemin* un chemin alternant élémentaire ;
– *æ-boucle* un cycle alternant élémentaire de longueur impaire (la première arête et la dernière sont toutes les deux dans $E \backslash B$ puisque B est un couplage) ;
– *æ-cycle* un cycle alternant élémentaire de longueur paire (la première et la dernière arête n'appartiennent pas toutes les deux à $E \backslash B$).

Aucun de ces chemins ou cycles ne peut emprunter une arête réflexive car ils sont élémentaires.

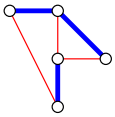
Exemple 1. Dans ces exemples, en anticipant sur ce qui suit, les arêtes de B sont dessinées en bleu ou en trait épais, les arêtes de R en rouge ou en trait fin, et les arêtes rouges et bleues $R \cup B$ représentent toutes les arêtes E des graphes.



(a) B est un couplage, mais il n'est pas parfait



(b) B est un couplage parfait



(c) B n'est pas est un couplage

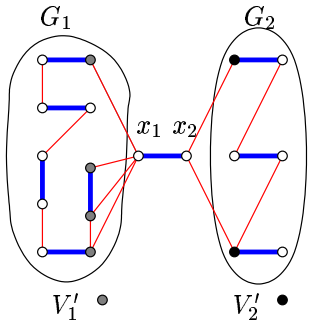
FIG. 2.3 – Exemples de graphes et de couplages

2.1.2 R&B-graphes

Définition 5. Un *R&B-graphe* $G = (V; B, R)$ se définit par la donnée de deux graphes simples $G_R = (V, R)$ et $G_B = (V, B)$ tels que B est un couplage parfait du graphe sous-jacent $\underline{G} = (V, B \oplus R)$ et R ne contient aucune arête réflexive. Il est *simple* ou *connexe* si $\underline{G}$ l'est. On parle de R et de B comme des *couleurs* (**r**ouge et **b**leu) du graphe.

Définition 6 (Jonction). Soient $G_1 = (V_1; B_1, R_1)$ et $G_2 = (V_2; B_2, R_2)$ deux R&B-graphes disjoints, x_1 et x_2 deux nouveaux sommets, V_1' un sous-ensemble non vide de V_1 et V_2' un sous-ensemble non vide de V_2 (sauf si $G_2 = G_\emptyset$). L'opération de *jonction* entre G_1 et G_2 permet d'obtenir un nouveau R&B-graphe G défini par :

$$G = (V_1 \cup \{x_1\} \cup V_2 \cup \{x_2\}; B_1 \cup B_2 \cup \{x_1x_2\}, R_1 \cup \{v_1x_1 | v_1 \in V_1'\} \cup R_2 \cup \{v_2x_2 | v_2 \in V_2'\})$$



Définition 7. L'ensemble R&B$^+$ est le plus petit ensemble de R&B-graphes qui contient le graphe vide $G_\emptyset = (\emptyset; \emptyset, \emptyset)$ et tous les R&B-graphes à une arête ($G = (\{x, y\}; xy, \emptyset)$) clos sous l'opération de jonction.

L'intérêt des R&B-graphes de cette classe est de pouvoir être partagés en deux en coupant une arête bleue, un isthme pour ces graphes. Cette propriété est utilisée dans les preuves de séquentialisation pour réduire la taille des graphes considérés et procéder par récurrence.

Définition 8. Soient $G = (V; B, R)$ un R&B-graphe simple et l une æ-boucle d'origine $x = x_0 = x_{2n+1}$. *Contracter* la boucle $l = xx_1x_2 \cdots x_{2n}x$, c'est identifier tous ses sommets à x et supprimer les éventuels arcs xx pour obtenir le graphe $G' = (V'; B', R')$ tel que :

- $V' = V \setminus \cup_1^{2n} \{x_i\}$;
- $B' = B \setminus \cup_1^{n} \{x_{2i-1}x_{2i}\}$;
- $R' = R|_{V'} \cup \{xy | y \neq x \text{ and } \exists i \in [1, 2n]\, x_iy \in R\}$.

Lemme 4 ([Ret99]). *Soit $G' = (V'; B', R')$ obtenu à partir du R&B-graphe $G = (V; B, R)$ en contractant une æ-boucle sur x. Alors :*

1. *G' est un R&B-graphe (et B' est un couplage parfait) ;*
2. *si G' n'est pas simple alors G admet un æ-cycle passant par x ;*
3. *s'il existe un æ-chemin entre deux sommets de G' alors il en existe un dans G avec des arêtes de mêmes couleurs aux extrémités ;*
4. *si G ne contient pas de æ-cycle, G' non plus ;*
5. *si une arête v_1v_2 de B est un isthme de G', alors v_1v_2 est un isthme de G.*

Théorème 5. *Soient $G = (V; B, R)$ un R&B-graphe simple et $x \in V$. Il existe un æ-chemin de x à un æ-cycle, à un isthme dans B ou à une arête réflexive de B.*

Démonstration. La démonstration s'adapte directement de [Ret99] en ajoutant comme condition d'arrêt à la construction du chemin le fait d'utiliser une arête réflexive. □

Corollaire 6. *Si G n'a ni æ-cycle ni B-arête réflexive, alors il existe un isthme dans B.*

Théorème 7 ([Ret99]). *Pour un R&B-graphe $G = (V; B, R)$ simple sans B-arête réflexive, les propriétés suivantes sont équivalentes :*

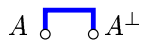
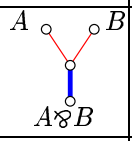
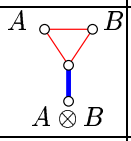
1. *$G \in$ R&B$^+$;*
2. *il n'y a pas d'æ-cycle dans G ;*
3. *B est l'unique couplage parfait du graphe sous-jacent $\underline{G}$.*

2.2 Structures de preuve et réseaux

Maintenant que nous avons défini les R&B-graphes, nous pouvons les utiliser dans la définition des structures de preuve. La grammaire des formules, à partir des variables propositionnelles de $\mathcal{P}$, est :

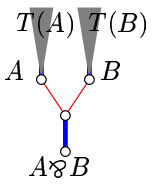
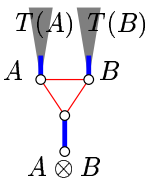
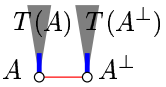
$$\mathcal{F}_{\text{Rés}} ::= \mathcal{P} | \mathcal{P}^\perp | \mathcal{F}_{\text{Rés}} \parr \mathcal{F}_{\text{Rés}} | \mathcal{F}_{\text{Rés}} \otimes \mathcal{F}_{\text{Rés}}$$

Définition 9. Nous appelons *liens* les graphes (aux sommets étiquetés) définis dans le tableau 2.1.

Lien	*axiome*	*par*	*tenseur*	*cut* ou *coupure*
Prémisses	Aucune	A et B	A et B	A et $A^\perp$
Graphes				
Conclusions	A et $A^\perp$ ($A \in \mathcal{P}$)	$A \parr B$	$A \otimes B$	Aucune

TAB. 2.1 – Les *liens*

Définition 10. Le *R&B-arbre* $T(C)$ d'une formule C, ou d'une coupure entre les formules C et $C^\perp$, se définit inductivement comme indiqué dans le tableau 2.2.

Formule C	$p \in \mathcal{P} \cup \mathcal{P}^\perp$	$A \parr B$	$A \otimes B$	coupure entre A et $A^\perp$
R&B-arbre $T(C)$	a ∘			
Hauteur $h(T(C))$	0	$\max(h(T(A)), h(T(B))) + 1$	$\max(h(T(A)), h(T(B))) + 1$	$h(T(A)) + 1$

TAB. 2.2 – R&B-arbres

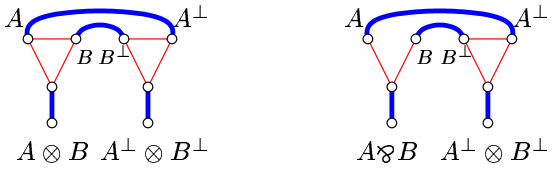
Définition 11. Une *structure de preuve*, ou *préréseau*, est un R&B-graphe tel qu'il existe une partition des arêtes dont chacune des parties (avec ses sommets incidents) est isomorphe à un lien du tableau 2.1, les étiquettes des sommets s'unifiant.

Les sommets de degré 1 sont les *conclusions* du préréseau.

On peut définir de manière équivalente les préréseaux comme des R&B-arbres de formules C_1 à C_n avec des arêtes de B entre chaque couple d'atomes duaux.

Définition 12. Les *réseaux de preuve* sont les prérésaux qui ne contiennent pas d'æ-cycle et pour lesquels, pour tout couple de sommets, il existe un æ-chemin entre ces sommets.

La figure 2.4 donne deux exemples de prérésaux dont seul le second est un réseau. En effet, la première figure possède un æ-cycle $A, A^\perp, B^\perp, B, A$.



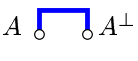
(a) Prérésaux qui n'est pas un réseau

(b) Prérésaux qui est un réseau

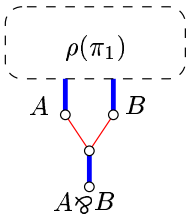
FIG. 2.4 – Exemples de prérésaux

2.2.1 Des séquents aux réseaux

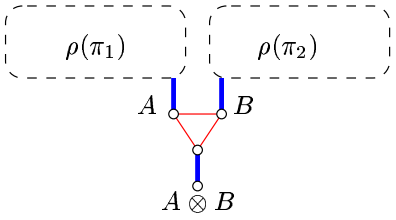
Soit une preuve $\pi(\vdash [\Delta]\Gamma)$ du séquent Γ de MLL. On construit inductivement le prérésaux Π de la manière suivante :

Si $\pi(\vdash [], A, A^\perp) = \dfrac{}{\vdash A, A^\perp}\,\mathsf{Ax}$, alors $\rho(\pi) = \Pi =$ 

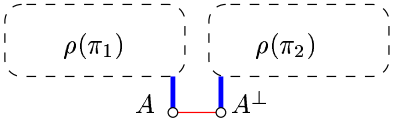
Si $\pi(\vdash [\Delta]\Gamma, A⅋B, \Gamma') = \dfrac{\pi_1(\vdash [\Delta]\Gamma, A, B, \Gamma')}{\vdash [\Delta]\Gamma, A⅋B, \Gamma'}\,⅋$ alors

$\rho(\pi) = \Pi =$ 

Si $\pi(\vdash [\Delta_1, \Delta_2]\Gamma, A \otimes B, \Gamma') = \dfrac{\pi_1(\vdash [\Delta_1]\Gamma, A) \quad \pi_2(\vdash [\Delta_2]B, \Gamma')}{\vdash [\Delta_1, \Delta_2]\Gamma, A \otimes B, \Gamma'}\,\otimes$ alors

$\rho(\pi) = \Pi =$ 

Si $\pi(\vdash [\Delta_1, \Delta_2, A]\Gamma_1, \Gamma_2) = \dfrac{\pi_1(\vdash [\Delta_1]\Gamma_1, A) \quad \pi_2(\vdash [\Delta_2]A^\perp, \Gamma_2)}{\vdash [\Delta_1, \Delta_2, A]\Gamma_1, \Gamma_2}\,\otimes$ alors

$\rho(\pi) = \Pi =$ 

Proposition 8 ([Ret99]). *Pour toute preuve π, le prérésaux $\rho(\pi)$ construit inductivement est un réseau.*

Nous constatons que la preuve

$$\dfrac{\dfrac{\dfrac{}{\vdash A, A^{\perp}}\,\mathsf{Ax} \quad \dfrac{}{\vdash B, B^{\perp}}\,\mathsf{Ax}}{\vdash A, A^{\perp} \otimes B^{\perp}, B}\,\otimes}{\vdash A \parr B, A^{\perp} \otimes B^{\perp}}\,\parr$$

permet la construction du réseau de la figure 2.4(b), par contre, aucune preuve ne correspond au préréseau de la figure 2.4(a) : d'une part d'après la propriété 8 il ne contiendrait pas d'æ-cycle, d'autre part $\vdash A \otimes B, A^{\perp} \otimes B^{\perp}$ n'est effectivement pas prouvable dans le calcul des séquents.

2.2.2 Des réseaux aux séquents

Nous avons montré comment passer des séquents aux réseaux. La question se pose alors de savoir si tout travail sur les réseaux correspond à un travail sur les preuves. En d'autres termes, tous les réseaux correspondent-ils à des preuves ou bien y en a-t-il qui n'ont pas de sens dans le calcul des séquents. Le théorème 9 apporte la réponse.

Théorème 9 (Séquentialisation [Ret99]). *Soit Π un réseau de preuve avec la conclusion Γ et des coupures sur les formules de Δ. Alors il existe une preuve $\rho^*(\Pi) = \pi(\vdash [\Delta]\Gamma)$ dans le calcul des séquents de MLL.*

Ainsi, à partir d'un préréseau, le critère de correction nous permet (suivant s'il est vérifié ou non) d'affirmer de manière intrinsèque que l'on manipule des conclusions prouvables ou non dans le calcul des séquents, sans avoir besoin de se référer encore à ce calcul.

2.3 L'élimination des coupures

Puisque nous avons désormais un moyen de manipuler des preuves sans se préoccuper de leur représentation sous forme de séquent, il faut s'assurer que cette représentation possède toujours, et de manière intrinsèque, la propriété d'élimination des coupures.

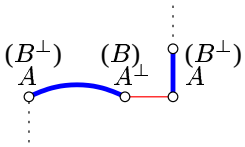
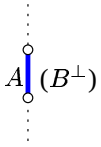
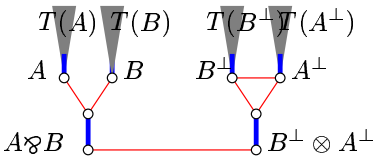
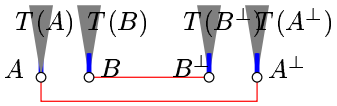
En effet, en se plaçant dans l'optique de recherche de preuve (qui devient une construction de réseau), le lien cut reste un problème : comment identifier les formules auxquelles l'appliquer ? Nous allons voir que les réseaux possèdent également la propriété d'élimination des coupures et que celle-ci se présente comme une simple réécriture de graphe, qui plus est, locale. On distingue deux cas : la coupure se fait entre deux liens axiomes ou entre un tenseur et un par. Les réécritures sont illustrées dans le tableau 2.3.

Proposition 10. *Chacune des étapes élémentaires AX/? et Par/TS conserve l'absence d'æ-cycle.*

Définition 13 (Confluence et forte normalisation). Une opération de réduction, dans notre cas l'élimination des coupures, est dite *confluente* si chaque fois qu'un élément, ici un réseau, Π se réduit en deux éléments Π_1 et Π_2 (par exemple s'il y a plusieurs coupures dans Π), il existe un élément Π' tel que Π_1 et Π_2 se réduisent en Π'.

Une opération de réduction est dite *fortement normalisable* si toute suite de réduction à partir d'un élément est finie.

Théorème 11 ([Ret93]). *L'élimination des coupures sur les réseaux est confluente et fortement normalisable.*

	Redex	Réduit
Avec un axiome (AX/ ?)		
Entre par et tenseur (Par/Tenseur)		

TAB. 2.3 – Élimination des coupures

2.4 Les modalités

Nous n'avons pour l'instant évoqué les réseaux de preuve que pour un type de connecteurs, les multiplicatifs. Outre ceux-là, nous utilisons dans notre modélisation les exponentiels (afin de retrouver la puissance non seulement du λ-calcul linéaire mais aussi tout le λ-calcul typé, voir chapitre 3.2). Conserver les outils des réseaux et leurs propriétés nécessite alors de les étendre à ces connecteurs ; cela fait l'objet de cette partie.

Avant de montrer comment ajouter ces connecteurs à la syntaxe des réseaux, rappelons leurs règles d'introduction dans le calcul des séquents :

$$\frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,! \qquad \frac{\vdash A, \Gamma}{\vdash ?A, \Gamma}\,\mathsf{d?} \qquad \frac{\vdash \Gamma}{\vdash ?A, \Gamma}\,\mathsf{w?} \qquad \frac{\vdash ?A, ?A, \Gamma}{\vdash ?A, \Gamma}\,\mathsf{c?}$$

On remarque que, parmi ces règles, deux donnent un rôle particulier au contexte : la règle !, qui demande que chaque formule du contexte commence par ?, et la règle d'affaiblissement, qui lie l'introduction de la formule A au contexte Γ.

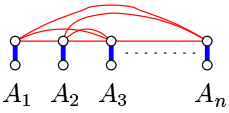
La difficulté apparaît dans la manière d'exprimer cette dépendance entre formule et contexte dans une syntaxe — les réseaux — qui jusqu'alors n'exprimait que des dépendances locales. Nous allons voir différents moyens de lever cette difficulté. En particulier, nous proposons ici une extension originale de la syntaxe choisie, les R&B-graphes, pour qu'elle tienne désormais compte des exponentiels en gardant un critère local de correction.

2.4.1 Les axiomes propres dans les R&B-graphes

Pour l'instant, nous n'avons considéré que les axiomes de la forme $\dfrac{}{\vdash A, A^\perp}\,\mathsf{Ax}$. En fait, les résultats indiqués jusqu´à présent restent vrais si l'on autorise des *axiomes propres* de la forme $\dfrac{}{\vdash A_1, \ldots, A_n}\,\mathsf{Ax}$.

Avant d'examiner les propriétés sur les réseaux, il nous faut donner une syntaxe pour les liens axiomes propres et vérifier que le critère de correction a toujours un sens.

Nous rajoutons donc les liens du tableau 2.4. On remarque que l'axiome habituel peut aussi se définir ainsi : avec deux arêtes bleues et une arête rouge. Du point de vue des æ-cycles, rien ne change. Par contre, seule la configuration avec deux conclusions permet de remplacer chaque chemin rouge-bleu-rouge par une arête bleue tout en gardant B couplage parfait.

Lien	*axiome propre* ($n = 1$)	*axiome propre* ($n \geq 2$)
Prémisses	Aucune	Aucune
Graphe		
Conclusions	A_1	$A_1, \ldots, A_n$

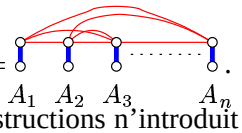
TAB. 2.4 – Liens pour les axiomes propres

Les définitions des préréseaux et des réseaux ne changent pas. De plus, on a les propriétés suivantes :

Proposition 12. *Pour toute preuve π du calcul des séquents, le préréseau $\rho(\pi)$ construit inductivement est un réseau.*

Démonstration. Par rapport à la proposition 8, on ajoute à la définition de construction inductive :

– si $\pi(\vdash [] A_1) = \dfrac{}{\vdash A_1}\,\mathsf{Ax}$, alors $\rho(\pi) =$ 

– si $\pi(\vdash [] A_1, \ldots A_n) = \dfrac{}{\vdash A_1, \ldots, A_n}\,\mathsf{Ax}$, alors $\rho(\pi) =$  .

Par induction, $\rho(\pi)$ est bien un préréseau et aucune des constructions n'introduit d'æ-cycle dans le nouveau préréseau. De plus, entre deux sommets, il existe toujours un æ-chemin. □

Lemme 13. *Dans un réseau, l'une au moins des deux composantes connexes d'arêtes de R situées à chacun des sommets d'un B-isthme xy qui n'est pas pendant est un lien tenseur ou un lien cut. De plus, si l'isthme est un lien axiome, il est prémisse de deux liens qui ne sont pas des par.*

Démonstration. Soit Π le réseau. Aucune des deux composantes connexes rouges n'est vide (sinon l'isthme serait pendant). Supposons alors que les deux soient des liens par. Alors l'isthme est prémisse d'au moins un des liens. On appelle alors x_0 ce sommet (sans restreindre la généralité, on suppose que $x = x_0$), z celui correspondant à l'autre prémisse, x_1 le sommet tel que $x_0x_1 \in R$ et x_2 celui de la conclusion (donc $x_1x_2 \in B$). Par hypothèse (définition d'un réseau), il existe un æ-chemin p de x_0 à z. Il passe nécessairement par x_1 et x_2 puisque xy est un isthme et x_0x_1z est un chemin de x à z. S'il en existait un passant par y, cela contredirait le fait que xy est un isthme. Et $p = (x_0x_1) \odot (x_1x_2 \cdots x_nz)$. Alors $x_1 \cdots x_nzx_1$ est un æ-cycle, ce qui contredit que Π est un réseau. Par l'absurde, l'un des deux liens n'est pas un par.

Si l'isthme est un lien axiome, ses deux sommets sont prémisses et le raisonnement ci-dessus s'applique pour chacune des composantes connexes. □

Lemme 14. *Soit $\Pi = (V; B, R)$ un réseau de preuve et $\Pi' = (V'; B', R')$ où si $S = \{x \in V | xx \in B\}$, on a :* $\begin{cases} V' = V \backslash S \\ B' = B \backslash \{xx | x \in S\} \\ R' = R \backslash \{xy | x \in S \text{ et } xy \in R\} \end{cases}$ *(Π' est Π à qui l'on a supprimé tous les sommets incidents à une arête réflexive). Alors tout B-isthme de Π', R&B-graphe, est également B-isthme de Π.*

Démonstration. Soient $z_0z_1 \in B$ isthme de Π' et $x \in S$. Pour montrer que z_0z_1 est aussi un isthme de Π, il suffit de montrer qu'il n'existe pas de chemin élémentaire de z_0 à z_1 passant par x dans Π (et donc utilisant deux arêtes rouges adjacentes en x) .

Par définition des réseaux, $xx \in B$ est un axiome propre à une conclusion, et ne peut qu'être prémisse d'un lien par, tenseur ou cut :

- si c'est un lien cut ou un lien par, alors il existe un unique $y \in V$ tel que $xy \in R$, et donc il n'existe pas de chemin élémentaire de z_0 à z_1 passant par x dans Π ;
- si c'est un lien tenseur, alors il existe y et y', $y \neq y'$ et $xy, xy' \in R$. Or $yy' \in R$ et $y, y' \in V'$. Supposons donc qu'il existe un chemin élémentaire c de z_0 à z_1 passant par x. Puisque les seules arêtes de R incidentes à x sont xy et xy', alors $c = \alpha_0 \odot (yxy') \odot \alpha_1 z_1$ (ou bien avec y et y' dans l'autre sens). Alors $\alpha_0 \odot (yy') \odot \alpha'_1$ est également un chemin élémentaire de z_0 à z_1, ce qui contredit que $z_0 z_1$ est un isthme de Π'.

Donc il n'existe pas de chemin élémentaire de z_0 à z_1 passant par x et $z_0 z_1$ également isthme de Π. Ceci est vrai pour tout isthme de Π'. □

Bien entendu, la propriété importante là encore est celle énoncée par le théorème suivant :

Théorème 15 (Séquentialisation). *Soit Π un réseau de preuve avec la conclusion Γ et des coupures sur les formules de Δ. Alors il existe une preuve $\rho^*(\Pi) = \pi(\vdash [\Delta]\Gamma)$ dans le calcul des séquents de MLL enrichi des axiomes propres.*

Démonstration. On montre ce théorème par récurrence sur le nombre d'arêtes de B. S'il n'y en a qu'une, c'est un lien axiome et $\rho^*(\Pi) = \dfrac{}{\vdash A, A^\perp}\,\mathsf{Ax}$, ou bien, dans le cas d'un axiome propre, $\rho^*(\Pi) = \dfrac{}{\vdash A}\,\mathsf{Ax}$.

Supposons maintenant un nombre quelconque d'arêtes de B. Si l'une des conclusions est un lien $A \parr B$, en supprimant ce lien, on obtient un réseau Π' avec moins d'arêtes bleues que Π. On peut donc appliquer l'hypothèse de récurrence à Π' pour trouver $\rho(\Pi')$ et alors $\rho^*(\Pi(\vdash [\Delta]\Gamma, A \parr B)) = \dfrac{\rho^*(\Pi'(\vdash [\Delta]\Gamma, A, B))}{\vdash [\Delta]\Gamma, A \parr B}\,\parr$

Supposons maintenant qu'aucune des conclusions ne soit un lien $\parr$. On construit Π^- à partir de Π en supprimant tous les sommets auxquels des arêtes pendantes (nécessairement de B) sont incidentes. Π^- est connexe.

Si Π^- est vide, comme toute conclusion d'axiome propre (avec $n \geq 2$) qui n'est pas pendante n'est pas supprimée et que puisque Π est connexe, une B-arête réflexive n'est jamais pendante, cela signifie que l'on a : un seul axiome propre, sans autre lien, ou des axiomes logiques dont une conclusion est pendante. Dans ce dernier cas, tout $\mathsf{R\&B}$-arbre doit être de hauteur maximale 1 pour que chacun de ses sommets soit supprimé. Donc Π, connexe, est constitué de deux liens axiomes logiques reliés par un lien tenseur ou un lien coupure.

Si c'est un lien tenseur alors $\rho^*(\Pi(\vdash []\Gamma)) = \dfrac{\dfrac{}{\vdash A, A^\perp}\,\mathsf{Ax} \qquad \dfrac{}{\vdash B, B^\perp}\,\mathsf{Ax}}{\vdash []\Gamma}\,\otimes$, avec éventuellement $A = B$.

Si c'est un lien coupure alors $\rho^*(\Pi(\vdash [A]A, A^\perp)) = \dfrac{\dfrac{}{\vdash A, A^\perp}\,\mathsf{Ax} \qquad \dfrac{}{\vdash A, A^\perp}\,\mathsf{Ax}}{\vdash [A]A, A^\perp}\,\mathsf{Cut}$.

Si Π^- n'est pas vide, il n'a plus d'arêtes de B pendantes mais il n'a toujours pas d'æ-cycle. On définit Π^{--} en supprimant tous les sommets x tels que $xx \in B$. Si Π^{--} est vide, il n'y avait que des axiomes propres à une conclusion et la séquentialisation est claire. Si Π^{--} n'est pas vide, le corollaire 6 du théorème 5 s'applique toujours et il existe une arête de B isthme de Π^{--}, et donc isthme de Π^- (d'après le lemme 14), et donc isthme de Π.

On a alors plusieurs cas :

1. cette arête est celle d'un lien axiome $A, A^\perp$. Ce lien ne peut pas être pendant (toutes les arêtes pendantes ont été supprimées) et la première règle qui lui est appliquée ne peut pas être un $\parr$ d'après le lemme 13. Soient alors V_1 et V_2, contenant respectivement A et $A^\perp$, les ensembles des sommets des deux graphes obtenus en supprimant l'isthme de Π et $\Pi_1 = (V_1 \cup \{A_0^\perp\}; B|_{V_1} \cup \{A, A_0^\perp\}, R|_{V_1})$ et $\Pi_2 = (V_2 \cup \{A_0\}; B|_{V_2} \cup \{A^\perp, A_0\}, R|_{V_2})$. Ce sont bien deux réseaux de preuve de conclusions respectives $\vdash [\Delta_1]\Gamma_1, A_0^\perp$ et $\vdash [\Delta_2]\Gamma_2, A_0$ (voir les figures 2.5). Or chacun a strictement moins d'arêtes de B que Π. En appliquant l'hypothèse de récurrence, on obtient deux preuves $\pi_1 = \rho^*(\Pi_1)$ et $\pi_2 = \rho^*(\Pi_2)$. En particulier, π_2 est de la forme :

$$\dfrac{\dfrac{\vdots}{\vdash [\Delta_2']\Gamma_2' \quad \vdots} \qquad \dfrac{}{\vdash A_0, A^\perp}\,\mathsf{Ax}}{\vdash [\Delta_2'']\Gamma_2'', A_0}\,\circ_1 \qquad \vdots \qquad \vdots \qquad \vdots}{\vdash [\Delta_2]\Gamma_2, A_0}\,\circ_2$$

avec, si $\circ_1 = \otimes$, $\Delta_2' = \Delta_2''$ et $\Gamma_2'' = A_1, \ldots, A_n, B \otimes A^\perp$ avec $\Gamma_2' = A_1, \ldots, A_n, B$. Si $\circ_1 = \mathsf{Cut}$, $\Delta_2'' = \Delta_2', A$ et $\Gamma_2'' = A_1, \ldots, A_n$ avec $\Gamma_2' = A_1, \ldots, A_n, A$. En remplaçant $\vdash A_0, A^\perp$ dans cette preuve par $\pi_1(\vdash [\Delta_1]\Gamma_1, A_0^\perp)$ (le $A_0^\perp$ prend la place de $A^\perp$ et $[\Delta_1]\Gamma_1$ celle de A_0), on obtient :

$$\dfrac{\dfrac{\vdots}{\vdash [\Delta_2']\Gamma_2' \quad \vdots} \qquad \pi_1(\vdash [\Delta_1]\Gamma_1, A_0^\perp)}{\vdash [\Delta_1, \Delta_2'']\Gamma_2'', \Gamma_1}\,\circ_1 \qquad \vdots \qquad \vdots \qquad \vdots}{\vdash [\Delta_1, \Delta_2]\Gamma_2, \Gamma_1}\,\circ_2$$

qui est bien une preuve de $\vdash [\Delta]\Gamma$ correspondant à $\Pi(\vdash [\Delta]\Gamma)$.

2. cette arête est celle d'un axiome propre $A_1, \ldots A_n$ (et $n > 1$). Soient alors V_1 et V_2, contenant respectivement $A_1, \ldots, A_{n-1}, A_n^+$ etA_0, les ensembles des sommets des deux graphes obtenus en supprimant l'isthme de Π (avec A_n^+ l'unique sommet incident à A_n grâce à une arête bleue), en rajoutant l'axiome propre $A_1', \ldots, A_{n-1}', A_n$ et $\Pi_1 = (V_1 \cup \{A_0\}; B|_{V_1} \cup \{A_n^+, A_0\}, R|_{V_1})$ et $\Pi_2 = (V_2 \cup \{A_1', \ldots, A_{n-1}', A_n, \}; B|_{V_2} \cup \{A_i'^+ A_i' | 1 \leq i \leq n-1\} \cup \{A_n'^+ A_n\}, R|_{V_2} \cup \{A_i'^+ A_j'^+ | 1 \leq i, j \leq n\})$. Ce sont bien deux réseaux de preuve de conclusions respectives $\vdash [\Delta_1]\Gamma_1, A_0$ et $\vdash [\Delta_2]\Gamma_2, A_1', \ldots, A_{n-1}'$ (voir les figures 2.6). Or chacun a strictement moins d'arêtes de B que Π. En appliquant l'hypothèse de récurrence, on obtient deux preuves $\pi_1 = \rho^*(\Pi_1)$ et $\pi_2 = \rho^*(\Pi_2)$. En particulier, π_2 est de la forme :

$$\dfrac{\dfrac{\vdots}{\vdash [\Delta_2']\Gamma_2' \quad \vdots} \qquad \dfrac{}{\vdash A_1', \ldots, A_{n-1}', A_n}\,\mathsf{Ax}}{\vdash [\Delta_2'']\Gamma_2'', A_1', \ldots, A_{n-1}'}\,\circ_1 \qquad \vdots \qquad \vdots \qquad \vdots}{\vdash [\Delta_2]\Gamma_2, A_1', \ldots, A_{n-1}'}\,\circ_2$$

avec, si $\circ_1 = \otimes$, $\Delta_2' = \Delta_2''$ et $\Gamma_2'' = C_1, \ldots, C_m, B \otimes A_n$ avec $\Gamma_2' = C_1, \ldots, C_n, B$. Si $\circ_1 = \mathsf{Cut}$, $\Delta_2'' = \Delta_2', A_n$ et $\Gamma_2'' = C_1, \ldots, C_n$ avec $\Gamma_2' = C_1, \ldots, C_n, A_n^\perp$. En remplaçant

$\vdash A'_1, \ldots, A'_{n-1}, A_n$ dans cette preuve par $\pi_1(\vdash [\Delta_1]\Gamma_1, A_0)$ (le A_0 prend la place de A_n et $[\Delta_1]\Gamma_1$ celle des A'_i), on obtient :

$$\dfrac{\dfrac{\vdots}{\vdash [\Delta'_2]\Gamma'_2} \qquad \pi_1(\vdash [\Delta_1]\Gamma_1, A_0)}{\vdash [\Delta_1, \Delta''_2]\Gamma''_2, \Gamma_1}\,\circ_1 \qquad \vdots}{\vdash [\Delta_1, \Delta_2]\Gamma_2, \Gamma_1}\,\circ_2$$

qui est bien une preuve de $\vdash [\Delta]\Gamma$ correspondant à $\Pi(\vdash [\Delta]\Gamma)$.

3. cette arête est celle d'un lien par. D'après le lemme 13, elle est prémisse d'un lien qui ne peut être que tenseur ou cut (toujours à cause de l'existence d'un æ-chemin) et on applique ce qui suit.

4. cette arête est celle d'un lien tenseur ou cut dont un sommet est étiqueté par la formule A (soit x_A l'autre sommet de l'arête). On définit de même que tout à l'heure V_1 et V_2 les ensembles des sommets des deux graphes obtenus en supprimant l'isthme de Π (avec cette fois-ci A dans V_2) puis les réseaux $\Pi_1(\vdash [\Delta_1]\Gamma_1, A_0) = (V_1 \cup \{A_0\}, B|_{V_1} \cup \{x_A A_0\}, R|V_1)$ et $\Pi_2(\vdash [\Delta_2]\Gamma_2, A_0^\perp) = (V_1 \cup \{A_0^\perp\}, B|_{V_2} \cup \{A A_0^\perp\}, R|V_2)$ (voir les figures 2.7). De même que précédemment, on obtient deux preuves $\pi_1 = \rho^*(\Pi_1)$ et $\pi_2 = \rho^*(\Pi_2)$ et dans π_2 une feuille $\vdash A_0^\perp, A$. On remplace alors cette feuille par π_1 pour obtenir $\pi = \rho^*(\Pi)$. π_2 est de la forme :

$$\dfrac{\dfrac{\vdots}{\vdash [\Delta'_2]\Gamma'_2} \qquad \dfrac{}{\vdash A, A_0^\perp}\,\mathsf{Ax}}{\vdash [\Delta''_2]\Gamma''_2, A_0^\perp}\,\circ_1 \qquad \vdots}{\vdash [\Delta_2]\Gamma_2, A_0^\perp}\,\circ_2$$

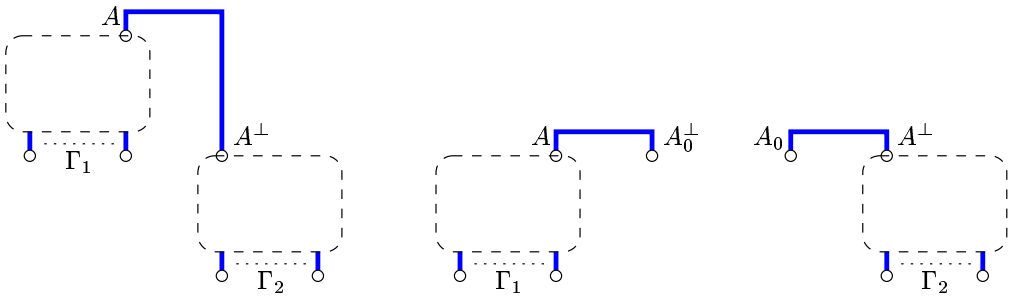
En remplaçant $\vdash A, A_0^\perp$ dans cette preuve par $\pi_1(\vdash [\Delta_1]\Gamma_1, A_0)$ (et le A_0 prend la place du A), on obtient :

$$\dfrac{\dfrac{\vdots}{\vdash [\Delta'_2]\Gamma'_2} \qquad \pi_1(\vdash [\Delta_1]\Gamma_1, A_0)}{\vdash [\Delta_1, \Delta''_2]\Gamma''_2, \Gamma_1}\,\circ_1 \qquad \vdots}{\vdash [\Delta_1, \Delta_2]\Gamma_2, \Gamma_1}\,\circ_2$$

la preuve π.

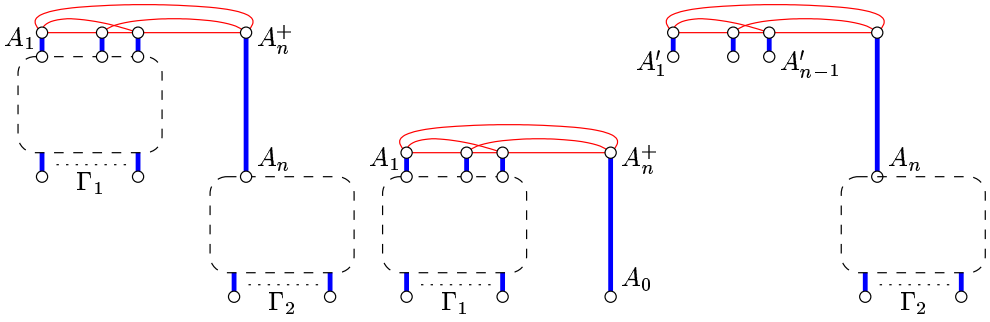
□

Ce théorème va nous permettre d'introduire les exponentiels dans la syntaxe des réseaux, que ce soit à la manière de [Gir87a] avec les boîtes dans la prochaine section ou sans les boîtes dans la section suivante.



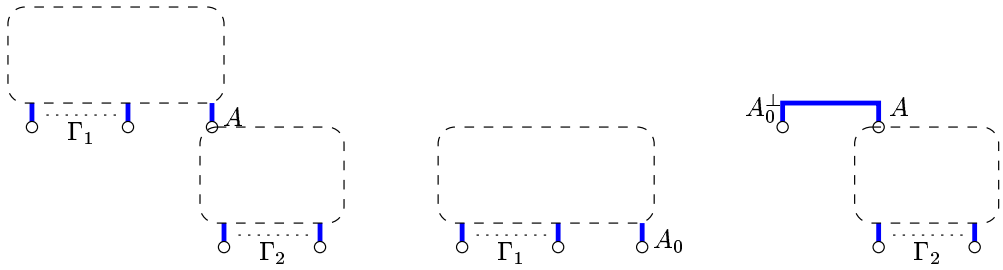
(a) Le réseau Π (b) Le réseau Π_1 (c) Le réseau Π_2

FIG. 2.5 – Séquentialisation si l'isthme est un lien axiome



(a) Le réseau Π (b) Le réseau Π_1 (c) Le réseau Π_2

FIG. 2.6 – Séquentialisation si l'isthme est un lien axiome propre



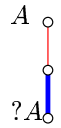
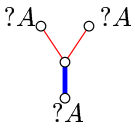
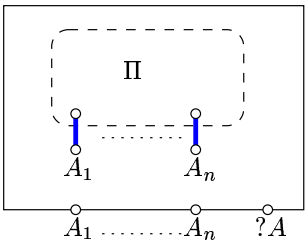
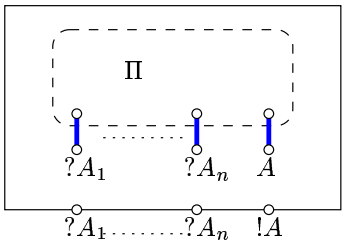
(a) Le réseau Π (b) Le réseau Π_1 (c) Le réseau Π_2

FIG. 2.7 – Séquentialisation si l'isthme est une arête d'un lien tenseur ou coupure

2.4.2 Les boîtes

Nous présentons ici, aménagé de [Gir87a] pour tenir compte des R&B-graphes, l'utilisation de boîtes pour le calcul des réseaux. Le nom de boîte [Gir87a] vient de l'idée de boîtes noires utilisées dans le calcul. Celles-ci n'interagissent avec l'extérieur que grâce à leur interface, et le contenu reste imperméable à l'environnement.

Nous ajoutons donc à la syntaxe les liens du tableau 2.5.

Lien	*déréliction*	*contraction*	*affaiblissement*	*promotion*
Prémisses	A	$?A$ et $?A$	Aucune	Aucune
Graphe				
Conclusions	$?A$	$?A$	$A_1, \ldots, A_n, ?A$	$?A_1, \ldots, ?A_n, !A$

TAB. 2.5 – Liens pour les exponentiels

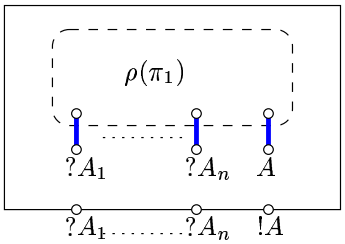
Définition 14. On définit par récurrence sur le nombre d'emboîtement les *préréseaux* :
- chaque boîte contient un préréseau ;
- en remplaçant chaque boîte par un axiome propre de mêmes conclusions on a un préréseau.

Définition 15. On définit de même par récurrence les *réseaux* :
- chaque boîte contient un réseau ;
- en remplaçant chaque boîte par un axiome propre de mêmes conclusions on a un réseau de plus que de boîtes (c'est-à-dire des préréseaux ne contenant pas d'æ-cycle et pour lesquels, entre deux sommets quelconques, il existe toujours un æ-chemin).

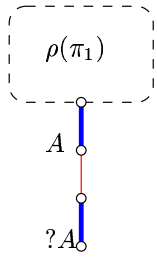
Proposition 16. *Pour toute preuve π du calcul des séquents de MELL, le préréseau $\rho(\pi)$ construit inductivement est un réseau.*

Démonstration. C'est la même que pour la proposition 8, en ajoutant à la définition de construction inductive :

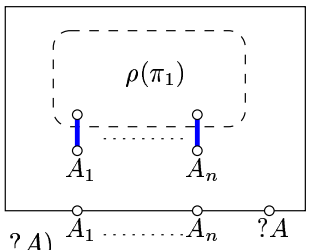
- si $\pi(\vdash [\Delta]?\Gamma, !A) = \dfrac{\pi_1(\vdash [\Delta]?\Gamma, A)}{[\Delta]?\Gamma, !A}\,!$, alors

$\rho(\pi) =$ 

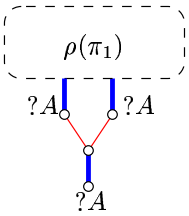
- si $\pi(\vdash [\Delta]\Gamma, ?A) = \dfrac{\pi_1(\vdash [\Delta]\Gamma, A)}{[\Delta]\Gamma, ?A}\,\mathsf{d}?$, alors

$\rho(\pi) =$ 

– si $\pi(\vdash [\Delta]\Gamma, ?A) = \dfrac{\pi_1(\vdash [\Delta]\Gamma)}{[\Delta]\Gamma, ?A}$ w?, alors

$\rho(\pi) =$ 

– si $\pi(\vdash [\Delta]\Gamma, ?A, ?A) = \dfrac{\pi_1(\vdash [\Delta]\Gamma, ?A, ?A)}{[\Delta]\Gamma, ?A}$!, alors

$\rho(\pi) =$ 

□

Théorème 17 (Séquentialisation). *Soit* Π *un réseau de preuve avec la conclusion* Γ *et des coupures sur les formules de* Δ*. Alors il existe une preuve* $\rho^*(\Pi) = \pi(\vdash [\Delta]\Gamma)$ *dans le calcul des séquents de MELL.*

Démonstration. La preuve de ce théorème s'obtient aisément à l'aide des théorèmes 9 et 15. □

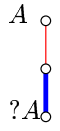
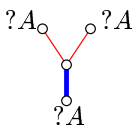
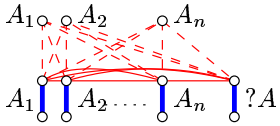
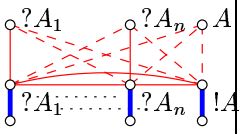
2.4.3 Exponentiels sans boîte dans les R&B-graphes

La définition même des réseaux avec les boîtes facilite la démonstration du théorème 17. Elle fait intervenir, à la différence du cas purement multiplicatif, l'induction, et en fait internalise la séquentialisation. En effet, la définition même indique comment chaque réseau est construit à partir de réseaux plus petits, et donc comment se partage la preuve sous forme de séquents, et on peut alors considérer que les réseaux n'ont plus une définition intrinsèque. C'est pourquoi différents travaux proposent de supprimer ces boîtes (tout au moins de les coder avec un critère de correction ne requérant plus d'induction), tel [Lam94].

Néanmoins, ces derniers se placent dans le cas intuitionniste, avec une représentation des réseaux très différente de celle que nous avons adoptée. Aussi proposons-nous dans cette section une version des réseaux qui étend celle basée sur les R&B-graphes à la logique linéaire multiplicative avec les exponentiels sans recours aux boîtes. Nous définissons une notion de parenthésage correspondant à celle d'emboîtement des liens promotion et affaiblissement.

Les liens

Par rapport aux liens du tableau 2.1 (cas purement multiplicatif), on ajoute les liens du tableau 2.6. Ceux-ci font apparaître des arêtes en pointillé. L'ensemble de ces arêtes est noté O (pour **o**range), et $O \subset R$.

Lien	*déréliction*	*contraction*	*affaiblissement* ($n \geq 1$)	*promotion* ($n \geq 0$)
Prémisses	A	$?A$ et $?A$	$A_1, \dots, A_n$	$?A_1, \dots, ?A_n, A$
Graphe				
Conclusions	$?A$	$?A$	$A_1, \dots, A_n, ?A$	$?A_1, \dots, ?A_n, !A$

TAB. 2.6 – Liens pour les exponentiels

Définition 16. Pour un réseau $\Pi = (V; B, R)$, on appelle $\mathcal{P} = \{xy \in R | (\exists z \in V xz \in O) \wedge (\nexists z \in V (xz \in O) \wedge (yz \in O))\}$ l'ensemble des *parenthèses* de Π. (Ce sont les liens rouges ou orange qui relient prémisses et conclusions des liens affaiblissement ou promotion.)

Pour un lien L et sa composante connexe C_L d'arêtes rouges, $\mathcal{P}_L = C_L \cap \mathcal{P}$ est l'ensemble des *parenthèses de* L.

Sur un chemin $c = c_1 \odot (xy) \odot c_2$, $xy \in \mathcal{P}$ est une *parenthèse ouvrante* si y est prémisse du lien et *parenthèse fermante* si x est prémisse du lien.

Remarque. La définition d'une parenthèse est propre au réseau. Sa qualification d'ouvrante et fermante est relative à la donnée d'un chemin qui l'utilise et au sens dans lequel elle est utilisée.

Définition 17. Un chemin c est *bien parenthésé* si :
- c est le chemin vide ;
- $c = (xy) \odot c'$ avec $xy \in B \cup (R \backslash \mathcal{P})$ et c' est bien parenthésé ;
- $c = (xy) \odot c' \odot (zw) \odot c''$ et il existe un lien L tel que $xy, zw \in \mathcal{P}_L$ et xy et zw de sens opposés (ouvrante-fermante ou fermante-ouvrante), et c' et c'' sont bien parenthésés.

Si de plus xy parenthèse ouvrante et zw parenthèse fermante, c est *très bien parenthésé.*

Définition 18. Un chemin $c = x_1 \dots x_n$ *simplement bouclé en* x_j est un chemin simple alternant tel qu'il existe $i < j < k < l \in [1, n]^4$ avec :
- $x_1 \cdots x_i$ est un æ-chemin (éventuellement vide) ;
- $x_l \dots x_n$ est un æ-chemin (éventuellement vide) ;
- $x_j \cdots x_k$ est une æ-boucle en $x_j = x_k$;
- $x_i \cdots x_j = x_l \cdots x_k$ (nécessairement non vide).

Dans un R&B-graphe, on a nécessairement $j + 3 \leq k$

Un chemin $c = c_1 \odot \cdots \odot c_n$ est *bouclé* si chacun des c_i est simplement bouclé et pour tout i, j, c_i et c_j n'ont aucun sommet commun, sauf si $j = i + 1$ auquel cas ils ont de commun le dernier sommet de c_i et le premier sommet de c_{i+1}. On dit que c a n boucles.

Définition 19 (Chemin simple alternant (semi)-maximal). Soit $c = x_1 \cdots x_n$ un chemin simple alternant. Si x_1 et x_n sont pendants, alors c est un chemin simple alternant maximal (sam-chemin). Si $x_1 = x_n$ (même non pendants), alors c est un chemin simple alternant semi-maximal. Un chemin peut être à la fois maximal et semi-maximal si $x_1 = x_n$ est pendant.

Définition 20. Une *structure de preuve*, ou *préréseau*, est un R&B-graphe tel qu'il existe une partition des arêtes dont chacune des parties (avec ses sommets incidents) est isomorphe à un lien des tableaux 2.1 et 2.6, les labels des sommets s'unifiant.

Définition 21. Les *réseaux de preuve* sont les préréseaux qui ne contiennent pas d'æ-cycle et pour lesquels, pour tout couple de sommets, il existe un æ-chemin entre ces sommets et dont les chemins simples alternants semi-maximaux sont bien parenthésés et les chemins simples alternants maximaux sont très bien parenthésés.

Proposition 18. *Dans un réseau, sur un æ-chemin c qui utilise au moins deux arêtes de $\mathcal{P}_L$, deux telles arêtes apparaissant consécutivement dans c sont respectivement ouvrante et fermante.*

Démonstration. Dans chacun des autres cas (parenthèses ouvrante puis ouvrante, parenthèses fermante puis fermante, ou parenthèses fermante puis ouvrante), on peut construire un æ-cycle. □

Corollaire 19. *Dans un réseau, étant donné un lien L, un æ-chemin emprunte au plus deux arêtes de $\mathcal{P}_L$. De plus, elles sont dans l'ordre ouvrante et fermante.*

Lemme 20 (La croisée des chemins). *Soit $c = x_1 \cdots x_n$ ($n \geq 3$) un æ-chemin dans un préréseau tel que $x_n x_1 \in B \cup R$. Alors $b = x_1 \cdots x_n x_1$ est soit un æ-cycle soit une æ-boucle en x_1.*

Démonstration. Procédons par exhaustion des cas : tout d'abord, $(x_1x_2, x_{n-1}x_1) \notin B \times B$ sinon B n'est pas un couplage.

1. $x_1x_2 \in R$ et $x_nx_1 \in R$ (voir figure 2.8(a)). Alors on a une æ-boucle en x_1 ;
2. $x_1x_2 \in R$ et $x_nx_1 \in B$ (voir figure 2.8(b)). Alors on a un æ-cycle ;
3. $x_1x_2 \in B$ et $x_nx_1 \in R$. Idem.

□

(a) Deux arêtes rouges adjacentes

(b) Une arête rouge et une arête bleue adjacentes

FIG. 2.8 – Recroquevillage d'un æ-chemin

Corollaire 21. *Dans un réseau, puisqu'il n'y a pas d'æ-cycle, tout æ-chemin qui se prolonge pour ne plus être élémentaire donne naissance à une æ-boucle bien parenthésée.*

Définition 22. Soit $c = x_1 \cdots x_m$ un æ-chemin. On apelle *extension maximale* de c le æ-chemin

$$x_1 \cdots x_m x_{m+1} \cdots x_n$$

tel que l'on ait l'un ou l'autre des cas :

- x_n est pendant ;
- il existe $i \in [1, n-1]$ tel que $x_{n-1}x_n \in B$ et $x_nx_i \in R$. x_i est le *sommet de confluence* de cette extension.

Une extension maximale existe toujours, mais n'est en général pas unique.

Lemme 22. *Dans un réseau, soit $c = x_1 \cdots x_n$ un æ-chemin tel que x_1 soit pendant et x_n conclusion d'un lien affaiblissement ou promotion L. Alors c n'emprunte aucune ou exactement deux arêtes de $\mathcal{P}_L$. Dans ce dernier cas, on peut en extraire un æ-chemin qui n'emprunte aucune arête de $\mathcal{P}_L$.*

Démonstration. On appelle x_{n+1} l'unique sommet tel que $x_n x_{n+1} \in B$. c emprunte au plus deux arêtes de $\mathcal{P}_L$ en vertu du corollaire 19. De plus, il existe alors x_i tel que $x_1 \cdots x_i x_{n+1} x_n$ soit un æ-chemin n'empruntant aucune arête de $\mathcal{P}_L$ (en prenant x_i le premier sommet de la première arête de $\mathcal{P}_L$ qui est ouvrante).

Supposons qu'il n'en emprunte qu'une, $x_i x_{i+1}$. Si elle est ouvrante, alors

$$x_1 \cdots x_n x_{n+1} x_i x_{i-1} \cdots x_1$$

est un chemin simple alternant maximal mal parenthésé (une seule parenthèse). Si elle est fermante, cela signifie que $x_{i+1} x_{n+1} \in R$ et alors $x_{i+1} \cdots x_n x_{n+1} x_{i+1}$ est un æ-cycle. □

Lemme 23. *Soit un lien L affaiblissement ou promotion d'un réseau. Il n'existe pas de chemin* élémentaire *$c = x_1 \cdots x_n$ et $i \in]1, n[$ tels que :*

- *x_1 soit prémisse et x_n conclusion de L ;*
- *aucune arête $x_j x_{j+1}$ de c ne soit une parenthèse (ouvrante ou fermante) de L ;*
- *$x_1 \cdots x_i$ et $x_i \cdots x_n$ soient des æ-chemins ;*
- *$x_{i-1} x_i \in R$, $x_i x_{i+1} \in R$ mais $x_{i-1} x_{i+1} \notin R$.*

Démonstration. Supposons l'existence d'un tel chemin. On peut alors supposer sans restreindre la généralité que x_n est la seule conclusion de L dans c. Soit alors x_{n+1} l'unique sommet tel que $x_n x_{n+1} \in B$ et alors $x_{n+1} x_1 \in \mathcal{P}_L$. Donc $x_i x_{i+1} \cdots x_{n+1} x_1 \cdots x_i$ est un chemin semi-maximal mal parenthésé (voir figure 2.9). □

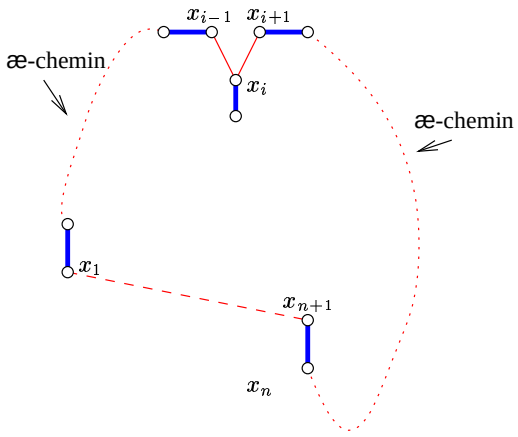


FIG. 2.9 – $m = 1$ et $r = 0$

Lemme 24. *Soit un lien L affaiblissement ou promotion d'un réseau. Il n'existe pas de chemin* simple *$c = x_1 \cdots x_n$ et $i \in]1, n[$ tels que :*

- *x_1 soit prémisse et x_n conclusion de L ;*
- *aucune arête $x_j x_{j+1}$ de c ne soit une parenthèse (ouvrante ou fermante) de L ;*
- *$x_1 \cdots x_i$ soit un chemin bouclé et $x_i \cdots x_n$ soit un æ-chemin ;*
- *$x_{i-1} x_i \in R$, $x_i x_{i+1} \in R$ mais $x_{i-1} x_{i+1} \notin R$.*

– *pour tout* $(k,l) \in [1, i-1] \times [i+1, n]$, $x_k \neq x_l$.

Démonstration. La preuve est la même que pour le lemme 23. □

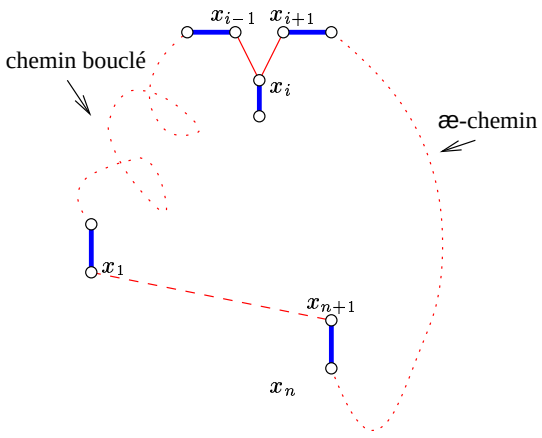


FIG. 2.10 – $m = 1$ et $r > 0$

Définition 23. Soit un lien L affaiblissement ou promotion d'un réseau. Pour tout $M, R \in \mathbb{N}^2$, on définit la propriété $\mathsf{Prop}_L(M, R)$ par : pour tout $(m, r) \in [0, M] \times [0, r]$, il n'existe pas de chemin $c = x_1 \cdots x_n$ et $(i_j) \in]1, n[^{[1,m]}$ tels que :

– x_1 soit prémisse et x_n conclusion de L ;
– aucune arête $x_j x_{j+1}$ de c ne soit une parenthèse (ouvrante ou fermante) de L ;
– $c_0 = x_1 \cdots x_{i_1}$ soit un æ-chemin ($r = 0$) ou un chemin bouclé (à r boucles), $c_m = x_{i_m} \cdots x_n$ soit un æ-chemin, et pour tout $j \in [1, m-1]$, $c_j = x_{i_j} \cdots x_{i_{j+1}}$ est un æ-chemin ;
– pour tout $j \in [1, m-1]$, $x_{i_j - 1} x_{i_j} \in R$, $x_{i_j} x_{i_j + 1} \in R$ mais $x_{i_j - 1} x_{i_j + 1} \notin R$;
– pour tout $(k,l) \in [1, i_1 - 1] \times [i_1 + 1, n]$, $x_k \neq x_l$.

On notera souvent, en l'absence d'ambiguïté, Prop au lieu de Prop_L.

Lemme 25. *Soit un lien L affaiblissement ou promotion d'un réseau. Pour tout $M, R \in \mathbb{N}^2$, la propriété $\mathsf{Prop}(M, R)$ est vraie.*

Avant de démontrer ce lemme, nous allons établir quelques propriétés intermédiaires. En premier lieu, $\mathsf{Prop}(0, r)$ est vraie par définition du réseau.

Lemme 26. *Pour tout r, $\mathsf{Prop}(1, r)$ est vraie.*

Démonstration. C'est exactement ce que disent les lemmes 23 et 24. □

Lemme 27. *À m fixé, si pour tout r $\mathsf{Prop}(m-1, r)$ est vraie, alors $\mathsf{Prop}(m, 0)$ est vraie.*

Démonstration. Supposons l'existence d'un tel chemin avec $x_{i_1}, \ldots, x_{i_m}$. On peut alors supposer sans restreindre la généralité que x_n est la seule conclusion de L dans c. Soit alors x_{n+1} l'unique sommet tel que $x_n x_{n+1} \in B$ et alors $x_{n+1} x_1 \in \mathcal{P}_L$.

Il existe un unique y_2 tel que $x_{i_1} y_2 \in B$. Soit alors $c' = y_1 y_2 \ldots y_m$ une extension maximale de $x_{i_1} y_2$. Il n'existe pas j (que l'on choisirait alors minimum[9]) tel que $y_j = x_k$ et $k < i_1$, sinon :

[9]Cela permet de s'assurer que $y_{j-1} y_j \in R$. En effet, dans le cas contraire, puisqu'il rejoint un chemin alternant, cela signifierait que $x_{k-1} = y_{j-1}$ ou $x_{k+1} = y_{j-1}$, contredisant la minimalité de j. Nous utiliserons fréquemment cette propriété.

- s'il existe $y_{j'}y_{j'+1} \in \mathcal{P}_L$ avec $j' \geq j$ (j' minimum) ou si un tel j' n'existe pas : c_0 est un æ-chemin donc $c'_0 = x_1 \cdots x_k y_{j-1} \cdots y_1 x_{i_1+1} \cdots x_{i_2}$ est un æ-chemin et $c'_0, c_2, \ldots, c_m$ contrediraient la propriété $\mathsf{Prop}(m-1, 0)$ (voir figure 2.11(a)) ;
- s'il existe $y_{j'}y_{j'+1} \in \mathcal{P}_L$ avec $j' < j$ et j' minimum, alors si $y_{j'+1}$ prémisse alors $y_{j'}x_1 \in \mathcal{P}_L$ et

$$x_1 \cdots x_{i_1} y_2 \cdots y_{j'} x_1$$

est un chemin simple alternant semi-maximal mal parenthésé, et si $y_{j'}$ prémisse, en prenant

$$c'_0 = y_{j'} y_{j'-1} \cdots y_1 x_{i_1+1} \cdots x_{i_2}$$

et les c_p pour $p \geq 2$, on contredit la propriété $\mathsf{Prop}(m-1, 0)$.

De la même manière, il n'existe pas j (que l'on choisirait alors minimum) et $k > i_1$ tels que $y_j = x_k$, sinon :

- s'il existe $y_{j'}y_{j'+1} \in \mathcal{P}_L$ avec $j' \geq j$ (j' minimum) ou si un tel j' n'existe pas :
  - si $x_k x_{k+1} \in B$, alors :
    - si $k > i_m$, on a un æ-cycle $x_1 \cdots x_{i_1} y_2 \cdots y_j x_{k+1} \cdots x_{n+1} x_1$;
    - si $i_l < k < i_{l+1}$, alors $c'_0 = x_1 \cdots x_{i_1} y_2 \cdots y_j x_{k+1} \cdots x_{i_{l+1}}$, et les c_p pour $l+1 \leq p \leq m$ contredisent les hypothèses de la propriété $\mathsf{Prop}(m-1, 0)$ (voir figure 2.11(b)) ;
    - $k = i_l$ est impossible (car $x_{i_l} x_{i_l+1} \in R$) ;
  - si $x_k x_{k+1} \in R$, alors d'une part $k \geq i_2$ (sinon $x_{k-1} \cdots x_{i_1} \cdots y_j x_{k-1}$ est un æ-cycle). D'autre part, si $y_{j-1}x_{k+1} \in R$, on a les mêmes cas que précédemment en remplaçant y_j par x_{k+1}. Si $y_{j-1}x_{k+1} \notin R$:
    - si $k > i_m$, $c'_0 = x_1 \cdots x_{i_1} y_j$ et $c'_1 = x_k \cdots x_n$ contredisent les hypothèses de la propriété $\mathsf{Prop}(1, 0)$;
    - si $i_l < k < i_{l+1}$, alors $c'_0 = x_1 \cdots x_{i_1} y_2 \cdots y_j$, $c'_1 = x_k \cdots x_{i_{l+1}}$, et les c_p pour $l+1 \leq p \leq m$ contredisent les hypothèses de la propriété $\mathsf{Prop}(m-l+1, 0)$ (et $m-l+1 < m$ car $k \geq i_2 \Rightarrow l \geq 2$), voir figure 2.11(c) ;
    - si $k = i_l$ ($l \geq 2$) :
      - s'il existe $y_{j-1}x_{i_l+1} \in R$, alors avec $c'_0 = x_1 \cdots x_{i_1} y_2 \cdots y_{j-1} x_{i_l+1} \cdots x_{i_{l+1}}$ et les c_p pour $p \geq l+1$, cela contredit les hypothèses de la propriété $\mathsf{Prop}(m-l, 0)$;
      - s'il n'existe pas $y_{j-1}x_{i_l+1} \in R$, alors avec $c'_0 = x_1 \cdots x_{i_1} y_2 \cdots y_j$ et les c_p pour $l \leq p \leq m$, cela contredit les hypothèses de la propriété $\mathsf{Prop}(m-l+1, 0)$ ($m-l+1 < m$) ;
- s'il existe $y_{j'}y_{j'+1} \in \mathcal{P}_L$ avec $j' < j$ (j' minimum) ou si un tel j' n'existe pas, on arrive à une contradiction comme dans le cas où l'on a supposé $k < i_1$.

Finalement, on a montré que c' et $\cup_0^m c_i$ n'ont que $x_{i_1} = y_{i_1}$ comme sommet commun. Montrons maintenant que c' n'emprunte aucune arête de $\mathcal{P}_L$.

En effet, s'il en empruntait une $y_{j'}y_{j'+1}$, on pourrait appliquer le même raisonnement que ci-dessus pour $j' < j$, donc il n'existe pas de tel j' et c' n'emprunte aucune arête de $\mathcal{P}_L$.

Soit $z_1 z_2 \cdots z_p$ une extension maximale de $x_2 x_1$. On a $z_2 z_3 \in \mathcal{P}_L$ (et donc aucune autre arête $z_q z_{q+1}$ n'est dans $\mathcal{P}_L$ d'après le corollaire 19) et donc $y_m \cdots y_1 x_{i_1-1} \cdots x_1$ et $z_2 \cdots z_p$ n'ont que x_1 en commun (sinon cela contredit le lemme 23 ou bien on a un æ-cycle). Si y_m est une conclusion, et si de plus z_p est une conclusion, on a de y_m à z_p un chemin simple alternant maximal mal parenthésé. Si y_m est une conclusion et si z_p n'est pas une conclusion, on a également un chemin simple alternant maximal bien parenthésé mais pas très bien parenthésé avec une boucle (voir figure 2.11(d)).

Si y_m n'est pas une conclusion, il existe y_j sommet de confluence et avec

$$c'_0 = x_1 \cdots x_{i_1} y_2 \cdots y_m y_j y_{j-1} y_1 x_{i_1+1} \cdots x_{i_2}$$

et les c_p pour $p \geq 2$, on contredit la propriété $\mathsf{Prop}(m-1, 1)$.

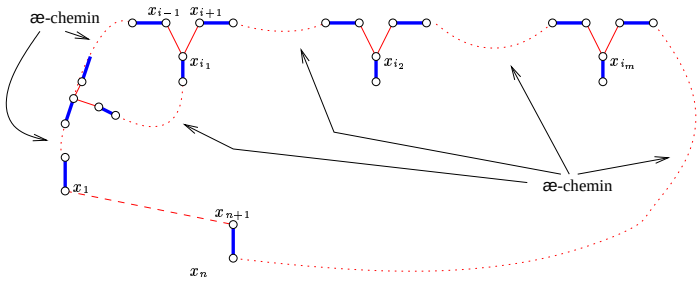
Par l'absurde, on a montré qu'un tel chemin c n'existe pas et donc $\mathsf{Prop}(m,0)$ est vraie. □

Lemme 28. *À m et r fixés, si $\mathsf{Prop}(m-1,q)$ est vraie pour tout q et que $\mathsf{Prop}(m,r)$ est vraie alors $\mathsf{Prop}(m,r+1)$ est vraie.*

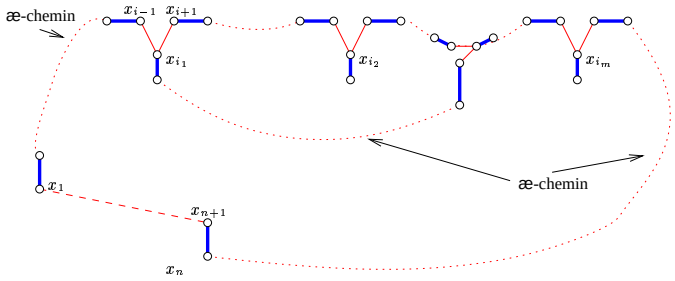
Démonstration. Supposons alors l'existence d'un tel chemin c avec $x_{i_1},\dots,x_{i_m}$ et $r+1$ boucles. On peut alors supposer sans restreindre la généralité que x_n est la seule conclusion de L dans c. Soit alors x_{n+1} l'unique sommet tel que $x_nx_{n+1}\in B$ et alors $x_{n+1}x_1\in\mathcal{P}_L$.

Il existe un unique y_2 tel que $x_{i_1}y_2\in B$. Soit alors $c'=y_1y_2\dots y_m$ une extension maximale de $x_{i_1}y_2$. Il n'existe pas j (que l'on choisirait alors minimum) tel que $y_j=x_k$ et $k<i_1$, sinon :

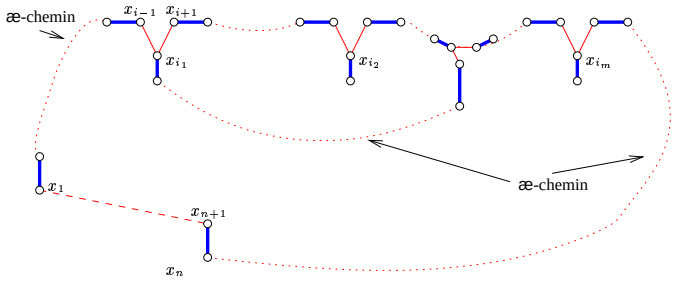
- s'il existe $y_{j'}y_{j'+1}\in\mathcal{P}_L$ avec $j'\geq j$ (j' minimum) ou si un tel j' n'existe pas : c_0 est bouclé (et sa dernière boucle est en $x_p=x_q$), alors, en prenant k maximum (sur certaines parties du chemin, en effet, il y a deux valeurs possibles pour k) $k\notin[p+1,q-1]$. En effet, dans le cas contraire, soit $x_kx_{k+1}\in B$ et $x_kx_{k+1}\cdots x_{i_1}y_2y_{j-1}x_k$ est un æ-cycle, soit $x_{k-1}x_k\in B$ (et alors $k\neq p+1$) et
$$(x_kx_{k-1}\cdots x_p)\odot(x_qx_{q+1}x_{i_1}y_2\cdots y_{j-1}x_k)$$
est un æ-cycle. Si $k=p$, on construit de même un æ-cycle ;
Soit alors $1<p'<p$ tel que p' soit le plus petit entier tel qu'il existe $q'>q$ avec $x_{p'}=x_{q'}$. Alors :
  - si $q+1\leq k\leq q'$, alors $x_kx_{k+1}\in R$ et il existe $k'\in[p',p-1]$ avec $x_k=x_{k'}$, et $c'_0=x_1\cdots x_{k'}$ est un chemin bouclé,avec r boucles,
$$c'_1=y_jy_{j-1}\cdots y_1x_{i_1+1}\cdots x_{i_2}$$
est un æ-chemin, alors $c'_0,c'_1,c_2,\dots,c_m$ contrediraient la propriété $\mathsf{Prop}(m,r)$;
  - si $k\leq p'-1$, alors si $x_kx_{k+1}\in R$, $c'_0=x_1\cdots x_{k-1}x_ky_{j-1}\cdots y_1x_{i_1+1}\cdots x_{i_2}$ et $c_2,\dots c_m$ contredisent la propriété $\mathsf{Prop}(m-1,r')$; et si $x_kx_{k+1}\in B$, $k>1$ (sinon $y_{j-1}y_j\in\mathcal{P}_L$) et alors $x_{k-1}x_k\in R$ et avec
$$c'_0=x_1\cdots x_k$$
$$c'_1=y_jy_{j-1}\cdots y_1x_{i_1+1}\cdots x_{i_2}$$
et les $c_2,\dots,c_m$, cela contredit la propriété $\mathsf{Prop}(m,r')$ avec $r'\leq r$ (voir figure 2.12(a)) ;
  - si $k=q'$ et $x_{p'-1}y_{j-1}\notin R$, avec $c'_0=x_1\cdots x_{p'}$ (chemin avec r boucles) et
$$c'_1=y_j\cdots y_1x_{i_1+1}\cdots y_{i_2}$$
et les $c_2,\dots,c_m$ on contredit $\mathsf{Prop}(m,r)$; si $x_{p'-1}y_{j-1}\in R$, avec
$$c'_0=x_1\cdots x_{p'-1}y_{j-1}\cdots y_1x_{i_1+1}\cdots y_{i_2}$$
et les $c_2,\dots,c_m$, on contredit $\mathsf{Prop}(m-1,r)$;
  - si $k>q'$ alors $x_kx_{k+1}\in R$ et avec $c'_0=x_1\cdots x_ky_{j-1}\cdots y_1x_{i_1+1}\cdots y_{i_2}$ et les $c_2,\dots,c_m$, on contredit $\mathsf{Prop}(m-1,r+1)$ (voir figure 2.12(b)) ;
- s'il existe $y_{j'}y_{j'+1}\in\mathcal{P}_L$ avec $j'<j$ et j' minimum, alors si $y_{j'+1}$ prémisse alors $y_{j'}x_1\in\mathcal{P}_L$ et
$$x_1\cdots x_{i_1}y_2\cdots y_{j'}x_1$$
est un chemin simple alternant semi-maximal mal parenthésé, et si $y_{j'}$ prémisse, en prenant
$$c'_0=y_{j'}y_{j'-1}\cdots y_1x_{i_1+1}\cdots x_{i_2}$$
et les c_p pour $p\geq 2$, on contredit la propriété $\mathsf{Prop}(m-1,0)$.



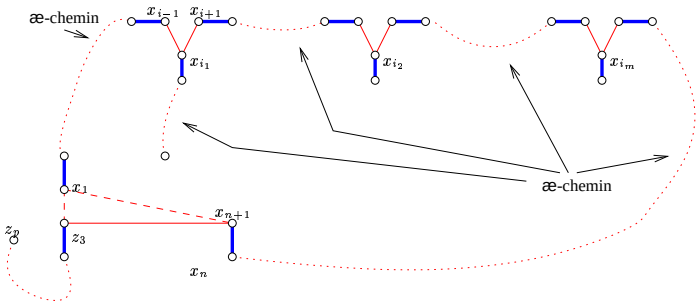
(a) $k < i_1$



(b) $k > i_l$ — premier cas



(c) $k > i_l$ — deuxième cas



(d) $c' \cap \cup_0^m c_j = \emptyset$

FIG. 2.11 – $m > 1$ et $r = 0$

De la même manière, il n'existe pas j (que l'on choisirait alors minimum) et $k > i_1$ tels que $y_j = x_k$, sinon :

- s'il existe $y_{j'}y_{j'+1} \in \mathcal{P}_L$ avec $j' \geq j$ (j' minimum) ou si un tel j' n'existe pas :
  - si $x_k x_{k+1} \in B$, alors :
    - si $k > i_m$, $x_1 \cdots x_{i_1} y_e \ldots y_j x_{k+1} x_{n+1} x_1$ est un chemin simple alternant semi-maximal mal parenthésé ;
    - si $i_l < k < i_{l+1}$, alors $c'_0 = x_1 \cdots x_{i_1} y_2 \cdots y_j x_{k+1} \cdots x_{i_{l+1}}$, et les c_p pour $l+1 \leq p \leq m$ contredisent la propriété $\mathsf{Prop}(m-1, r+1)$ (voir figure 2.12(c)) ;
    - $k = i_l$ est impossible (car $x_{i_l} x_{i_l+1} \in R$) ;
  - si $x_k x_{k+1} \in R$, alors d'une part $k \geq i_2$ (sinon $x_{k-1} \cdots x_1 \cdots y_j x_{k-1}$ est un æ-cycle). D'autre part, si $y_{j-1}x_{k+1} \in R$, on a les mêmes cas que précédemment en remplaçant y_j par x_{k+1}. Si $y_{j-1}x_{k+1} \notin R$:
    - si $k > i_m$, $c'_0 = x_1 \cdots x_{i_1} y_j$ et $c'_1 = x_k \cdots x_n$ contredisent les hypothèses de la propriété $\mathsf{Prop}(1, r+1)$;
    - si $i_l < k < i_{l+1}$, alors $c'_0 = x_1 \cdots x_{i_1} y_2 \cdots y_j$, $c'_1 = x_k \cdots x_{i_{l+1}}$, et les c_p pour $l+1 \leq p \leq m$ contredisent les hypothèses de la propriété $\mathsf{Prop}(m-l+1, r+1)$ (or $m-l+1 < m$ car $k \geq i_2 \Rightarrow l \geq 2$), voir figure 2.12(d) ;
    - si $k = i_l$ ($l \geq 2$) :
      - s'il existe $y_{j-1}x_{i_l+1} \in R$, alors avec $c'_0 = x_1 \cdots x_{i_1} y_2 \cdots y_{j-1} x_{i_l+1} \cdots x_{i_{l+1}}$ et les c_p pour $p \geq l+1$, cela contredit les hypothèses de la propriété $\mathsf{Prop}(m-l, r+1)$;
      - s'il n'existe pas $y_{j-1}x_{i_l+1} \in R$, alors avec $c'_0 = x_1 \cdots x_{i_1} y_2 \cdots y_j$ et les c_p pour $l \leq p \leq m$, cela contredit les hypothèses de la propriété $\mathsf{Prop}(m-l+1, r+1)$ et $m-l+1 < m$;
- s'il existe $y_{j'}y_{j'+1} \in \mathcal{P}_L$ avec $j' < j$ (j' minimum) ou si un tel j' n'existe pas, on arrive à une contradiction comme dans le cas où l'on a supposé $k < i_1$.

Finalement, on a montré que c' et $\cup_0^m c_i$ n'ont que $x_{i_1} = y_{i_1}$ comme sommet commun. Montrons maintenant que c' n'emprunte aucune arête de $\mathcal{P}_L$.

En effet, s'il en empruntait une $y_{j'}y_{j'+1}$, on pourrait appliquer le même raisonnement que ci-dessus pour $j' < j$, donc il n'existe pas de tel j'.

Soit $z_1 z_2 \cdots z_p$ une extension maximale de $x_2 x_1$. On a $z_2 z_3 \in \mathcal{P}_L$ (et donc aucune autre arête $z_q z_{q+1}$ n'est dans $\mathcal{P}_L$ d'après le corollaire 19) et donc $y_m \cdots y_1 x_{i_1-1} \cdots x_1$ et $z_2 \cdots z_p$ n'ont que x_1 en commun (sinon cela contredit la propriété $\mathsf{Prop}(1, r+1)$ ou bien on a un chemin semi-maximal mal parenthésé). Si y_m est une conclusion, et si de plus z_p est une conclusion, on a de y_m à z_p un chemin simple alternant maximal mal parenthésé. Si y_m est une conclusion, soit c'' un æ-chemin (qui existe bien puisque l'on a un réseau) vers x_n. D'après le lemme 22, on peut trouver c'' n'empruntant pas d'arête de $\mathcal{P}_L$. Puisque y_m est une conclusion et c'' un æ-chemin, il existe un sommet w_2 de c'' qui est le premier qui soit conclusion de L (éventuellement x_n) et w_1 un sommet de $d = y_m \cdots y_1 x_{i_1-1} \cdots x_1$ qui est le dernier sommet de c'' avant w_2 commun à d et c'' (éventuellement w_2).

Alors $d \downarrow_{w_2}$ est un chemin bouclé ($r' \leq r+1$ boucles). Avec $c'' \uparrow_{w_2}$, ils contredisent $\mathsf{Prop}(1, r+1)$.

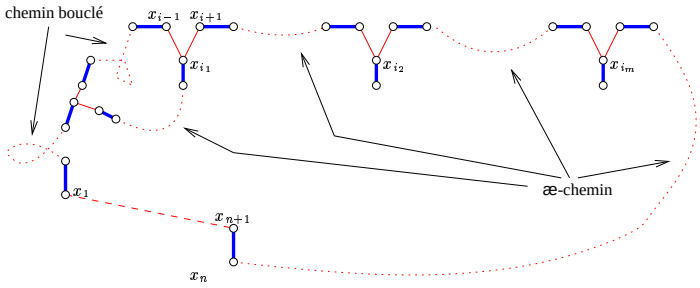
Si y_m n'est pas une conclusion, il existe y_j sommet de confluence et avec

$$c'_0 = x_1 \cdots x_{i_1} y_2 \cdots y_m y_j y_{j-1} y_1 x_{i_1+1} \cdots x_{i_2}$$

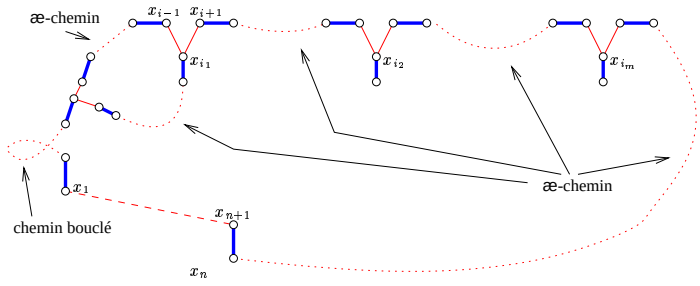
et les c_p pour $p \geq 2$, on contredit la propriété $\mathsf{Prop}(m-1, r+2)$.

Par l'absurde, on a montré qu'un tel chemin n'existe pas et donc $\mathsf{Prop}(m, r+1)$ est vraie. □

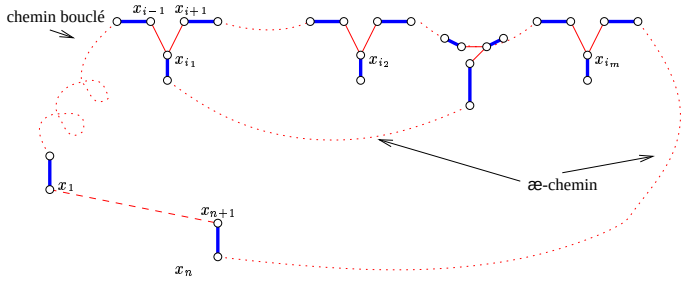
Preuve du lemme 25. Par récurrence sur m et r, en utilisant judicieusement tous les lemmes intermédiaires, la propriété est prouvée. □



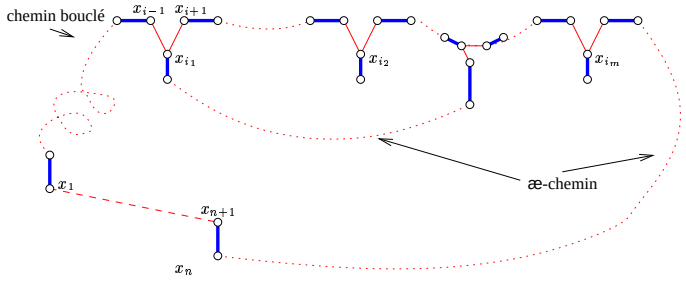
(a) $k < p'$



(b) $k > q'$



(c) $k > i_l$ — premier cas



(d) $k > i_l$ — deuxième cas

FIG. 2.12 – $m > 1$ et $r > 0$

Corollaire 29. *Entre une prémisse (x_1) et une conclusion (x_2) d'un lien affaiblissement ou promotion, il n'existe pas de chemin élémentaire $c = x_1 \cdots x_m$ avec $x_1x_2 \in B$ qui ait au plus deux arêtes rouges consécutives, et tel que pour tout x_{i-1}, x_i et x_{i+1} avec $x_{i-1}x_i, x_ix_{i+1} \in R$, $x_{i-1}x_{i+1} \notin B \cup R$ et qui n'utilise aucune parenthèse du lien.*

Démonstration. D'après le lemme 25. □

Théorème 30. *Dans un réseau, étant donné un lien affaiblissement ou promotion L, $\mathcal{P}_L$ est un ensemble d'arêtes scindantes.*

Démonstration. Ce théorème découle directement du lemme précédent. En effet, si l'on appelle $\Pi = (V; B, R)$ le réseau, soit L un lien du réseau, l le nombre de prémisses et n le nombre de conclusions. $l = n$ pour un lien promotion, et $l = n - 1$ pour un lien affaiblissement. Les prémisses sont les sommets y_i pour $1 \leq i \leq l$ et les conclusions sont les sommets y_i'' pour $i \leq i \leq n$ de telle façon que ces prémisses et conclusions correspondent. Dans le cas d'un lien affaiblissement, on définit de plus y_n' l'unique sommet joint à y_n'' par un lien bleu.

$\Pi' = (V; B, R \backslash \cup_1^l y_iy_i')$ le R&B-graphe après qu'on a supprimé les arêtes de $\mathcal{P}$ du lien considéré.

Raisonnons par l'absurde et supposons que ces liens ne soient pas un ensemble scindant. Alors il existe $c = x_0 \cdots x_m$ chemin élémentaire dans Π^- de y_1 vers y_k'' ($x_0 = y_1$ et $x_m = y_k''$). Le chemin n'est pas alternant sinon on a un æ-cycle, donc il existe $i \in [1, m]$ tel que $x_{i-1}x_i$ et $x_ix_{i+1} \in R$.

L'argument important est que l'on peut toujours se ramener à ce que $x_{i-1}x_{i+1} \notin R$ et qu'au plus deux arêtes rouges se suivent. En effet, si $x_{i-1}x_{i+1} \in R$, on peut considérer que le chemin est en fait $x_0 \cdots x_{i-1}x_{i+1} \cdots x_m$ et recommencer jusqu'à ce que sur le chemin considéré, aucune suite de deux arêtes rouges ne puisse être court-circuitée par une autre arête rouge. L'examen des différents liens nous montre qu'alors une telle configuration impose que x_{i-1} et x_{i+1} soient des prémisses et alors $x_{i-1}x_{i+1} \notin R$. C'est dû à ce que les liens affaiblissement et promotion sont des graphes séries-parallèles ($(\parr_i y_i) \otimes (\otimes_j y_j')$) et que l'absence de configuration P_4 (un chemin sans corde entre quatre sommets distincts) [Möh89, théorème 2.6] caractérise cette classe de graphes.

Or, cela contredit le lemme 25. Un tel chemin n'existe donc pas et les parenthèses de L constituent bien un ensemble d'arêtes scindantes. C'est vrai pour tout lien. □

Définition 24. Pour un lien affaiblissement ou promotion d'un réseau, on appelle *intérieur* du lien les sommets de la composante connexe du graphe sous-jacent, obtenue en supprimant les parenthèses du lien, et à laquelle les prémisses appartiennent.

Les sommets *extérieurs* sont ceux qui ne sont pas intérieurs. Tout chemin entre un sommet intérieur et un sommet extérieur à L emprunte une arête de $\mathcal{P}_L$.

Proposition 31. *Dans un réseau, un chemin c simple alternant n'emprunte pas successivement deux arêtes fermantes (ou ouvrantes) d'un même lien affaiblissement ou promotion.*

Démonstration. Sinon, on construit sans peine un chemin semi-maximal mal parenthésé entre l'extrémité finale de la première arête et l'extrémité initiale de la deuxième (si deux parenthèses ouvrantes se suivent). Si ce sont deux parenthèses fermantes, on considère c^{-1}. □

Corollaire 32. *Dans un réseau, si x est intérieur à un lien affaiblissement ou promotion L, et y extérieur à ce lien, pour tout chemin simple alternant c entre x et y, si c emprunte une arête ouvrante de L alors c emprunte nécessairement encore au moins une arête de L et la suivante est fermante.*

Démonstration. Si la dernière parenthèse de L empruntée par c est ouvrante et que z est sa prémisse, z est intérieur à L et y extérieur et $c \uparrow_z$ n'emprunte aucune parenthèse de L, cela contredit que $\mathcal{P}_L$ est un ensemble d'arêtes scindantes.

Et la parenthèse suivante est fermante d'après la proposition 31. □

Corollaire 33. *Dans un réseau, si c_1 et c_2 sont deux chemins simples alternants de mêmes extrémités x et y, alors :*
- *soit c_1 et c_2 sont bien parenthésés ;*
- *soit c_1 arrive en y avec une parenthèse ouverte (alors x extérieur et y intérieur) et de même pour c_2 ;*
- *soit c_1 arrive en y avec une parenthèse fermée (alors x intérieur et y extérieur) et de même pour c_2.*

Proposition 34. *Dans un réseau, si une conclusion x d'un lien affaiblissement ou promotion L_1 est intérieur à un lien L_2, toutes les conclusions de L_1 sont intérieures à L_2. De même pour les prémisses.*

Démonstration. Supposons qu'il y ait une conclusion y de L_1 extérieure à L_2. Par définition de L_1, il existe un æ-chemin de x à y (bleu-rouge-bleu) qui n'utilise aucune parenthèse d'aucun lien. Ce chemin n'est donc pas modifié par la suppression des arêtes de $\mathcal{P}_{L_2}$, ce qui contredit que y est extérieur.

C'est vrai également pour les prémisses puisque relativement à un lien L_2 une conclusion et une prémisse de L_1 appartiennent simultanément à l'intérieur ou à l'extérieur de L_2. □

Corollaire 35. *Dans un réseau, pour deux liens affaiblissement ou promotion L_1 et L_2, soit l'intérieur de l'un est inclus dans l'intérieur de l'autre, soit les deux intérieurs sont disjoints.*

Proposition 36. *Pour toute preuve π du calcul des séquents de MELL, le préréseau construit inductivement en ajoutant les définitions du tableau 2.7 est un réseau.*

Démonstration. On procède toujours par induction. Premièrement, aucun des liens (y compris affaiblissement et contraction) ne crée d'æ-cycle. Deuxièmement, entre deux sommets, il existe toujours un æ-chemin. Ces deux propriétés se vérifient aisément. Il reste à montrer que tous les chemins simples alternants maximaux et semi-maximaux sont bien parenthésés.

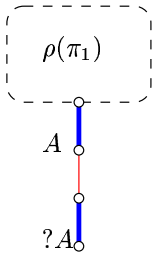
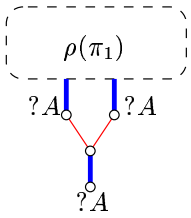
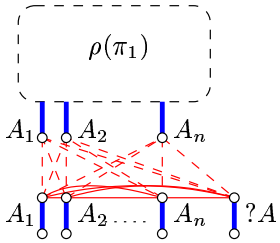
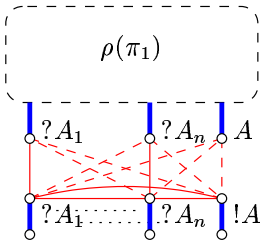
Soit c un chemin (semi-)maximal. Montrons qu'il est bien parenthésé. S'il n'utilise aucune arête du nouveau lien, par hypothèse d'induction il est bien parenthésé. Pour les liens par, déréliction, contraction, affaiblissement et promotion, s'il utilise une arête du nouveau lien, c'est qu'il passe par au plus deux prémisses de ce lien. Par hypothèse d'induction, la partie en dehors du nouveau lien est très bien parenthésée, et alors c est également très bien parenthésé.

Pour les liens coupures et tenseur, c'est vrai aussi :
- s'il passe par la conclusion (ou par l'unique sommet adjacent à la conclusion), alors il utilise un ou deux chemins maximaux (dans ce dernier cas, ils sont mis bout à bout et restent donc très bien parenthésés) ;
- s'il ne passe pas par la conclusion, alors il fait une boucle dans l'une des prémisses, qui est donc très bien parenthésée par hypothèse d'induction. Dans l'autre, il est venu par un chemin simple c_1 et repart par un chemin simple c_2 tel que $c = c_1 \alpha c_2$, et c_1 et c_2^{-1} sont de mêmes extrémités (l'origine de c et la deuxième prémisse du lien tenseur ou coupure). Donc, d'après le corollaire 33, c_1 et c_2^{-1} ont même parenthésage et c est bien parenthésé. D'après le lemme 22, si l'origine de c est pendante, il est extérieur à tout lien affaiblissement et promotion, donc c_1 (et c_2, et donc c) est très bien parenthésé.

□

Proposition 37. *Dans un réseau, l'intérieur d'un lien affaiblissement ou promotion est un réseau.*

Démonstration. Il suffit de vérifier que les chemins simples alternants maximaux c de l'intérieur du lien sont très bien parenthésés. Ils sont bien parenthésés : il suffit dans le réseau complet de considérer le chemin semi-maximal obtenu en rajoutant deux arêtes adjacentes de $\mathcal{P}_L$ et dont l'autre sommet est une des extrémité de c.

$\pi(\vdash [\Delta]\Gamma, ?A) = \dfrac{\pi_1(\vdash [\Delta]\Gamma, A)}{\vdash [\Delta]\Gamma, ?A}\,\mathsf{d?}$	
$\pi(\vdash [\Delta]\Gamma, ?A, ?A) = \dfrac{\pi_1(\vdash [\Delta]\Gamma, ?A, ?A)}{\vdash [\Delta]\Gamma, ?A}\,!$	
$\pi(\vdash [\Delta]A_1, \ldots, A_n, ?A) = \dfrac{\pi_1(\vdash [\Delta]A_1, \ldots, A_n)}{\vdash [\Delta]A_1, \ldots, A_n, ?A}\,\mathsf{w?}$	
$\pi(\vdash [\Delta]?A_1, \ldots, ?A_n, !A) = \dfrac{\pi_1(\vdash [\Delta]?A_1, \ldots, ?A_n, A)}{\vdash [\Delta]?A_1, \ldots, ?A_n, !A}\,!$	

TAB. 2.7 – Construction d'un réseau à partir d'un séquent

De plus, le corollaire 35 assure que si c traverse un lien affaiblissement ou promotion L', c'est toujours dans le sens entrée-sortie, soit avec des parenthèses ouvrantes puis fermantes. Donc c est très bien parenthésé. □

Théorème 38 (Séquentialisation). *Soit Π un réseau de preuve avec la conclusion Γ et des coupures sur les formules de Δ. Alors il existe une preuve $\rho^*(\Pi) = \pi(\vdash [\Delta]\Gamma)$ dans le calcul des séquents de MELL.*

Démonstration. La preuve de ce théorème s'obtient aisément comme le théorème 17 grâce à la proposition 37. □

2.5 Géométrie de l'interaction et interprétation graphique de l'élimination des coupures

La géométrie de l'interaction cherche à donner une sémantique du calcul [Gir89, Gir95b], c'est-à-dire à modéliser des algorithmes et leur exécution dans le modèle choisi. Dans ce processus, la logique linéaire intervient comme lien entre un calcul, le système $\mathbb{F}$ [GLT88] duquel elle est traduite, et son interprétation en terme d'algèbre. Nous n'utilisons dans le cadre de la génération (voir section 6.4) qu'une infime partie de ce programme qui se situe au niveau de l'interprétation des preuves de la logique linéaire.

Dans cette approche, l'élimination des coupures est cruciale. En effet, dans le paradigme de preuve comme programme (*proofs-as-programs*) que le chapitre 3 développe, la coupure traduit l'utilisation de résultats intermédiaires, comme des lemmes, dans un programme plus grand par la simple donnée de leur spécification. L'utilisation effective d'un résultat intermédiaire, c'est-à-dire son calcul au sein du programme plus grand, se traduit par l'élimination de la coupure. Cette dynamique de la preuve traduit la dynamique du calcul, et la géométrie de l'interaction permet d'établir la relation entre la preuve avec coupure et la preuve sans coupure comme une relation entre deux opérateurs.

Ainsi, en associant un opérateur U à une preuve, avec une isométrie partielle σ qui décrit la partie dynamique, les coupures (pour reprendre la description de [Gir89], si $\sigma = 0$, U est une donnée, c'est-à-dire un algorithme sans dynamique), l'exécution consiste à transformer U et la partie dynamique σ en une donnée. Elle s'exprime par :

$$\mathrm{Res}(U, \sigma) = (1 - \sigma^2)U(1 - \sigma U)^{-1}(1 - \sigma^2)$$

où $\mathrm{Res}(U, \sigma)$ correspond donc à une preuve sans coupures. Autrement dit, et c'est ce dont nous tirons parti pour la génération, cette formule exprime le processus d'élimination des coupures.

Nous n'allons pas détailler cette formule dans le cadre général de [GLT88, Gir95a], mais la relier au processus d'élimination des coupures pour les réseaux de preuve. Pour la génération, il suffit de considérer des réseaux de preuve dont les seuls liens sont des liens axiomes et des liens coupures. À partir d'un tel réseau $\overline{\Pi}$, nous considérons le sous-graphe des liens axiomes et le sous-graphe des liens coupures, ils correspondent respectivement aux sous-graphes bleus et rouges.

Observons uniquement du point de vue des graphes l'élimination d'une coupure entre deux axiomes, comme à la figure 2.13[10]. Dans le nouveau graphe, il y a une arête bleue e_1e_4 si et seulement si il existe dans l'ancien graphe e_2 et e_3 (tous les sommets sont distincts) tels que $e_1e_2 \in B$, $e_2e_3 \in R$ et $e_3e_4 \in B$, c'est-à-dire s'il existe un chemin bleu-rouge-bleu de e_1 vers e_4. Il est classique d'exprimer les graphes par des matrices [FGS93]. Dans le cas présent, prenons U matrice d'incidence

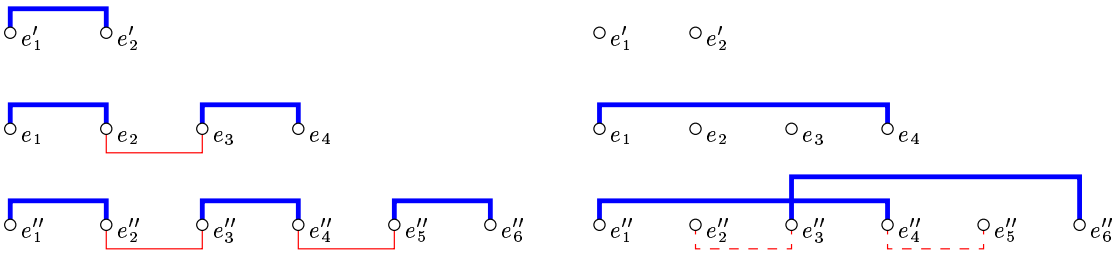
[10] Soient x et y deux sommets pendants, donc qui ne sont pas prémisses d'un lien coupure, joints par un æ-chemin. On appelle *niveau* de chacun des liens coupure du chemin le nombre d'arêtes rouges du chemin. Sur cette figure, le niveau de la coupure est 1.

correspondant à B et σ la matrice d'incidence correspondant à R. Ces matrices sont symétriques puisque les graphes sont non orientés. S'il y a plusieurs coupures dans le réseau $\overline{\Pi}$, faire une première étape d'élimination revient à chercher tous les e_1 et e_4 comme précédemment. Or la matrice mettant en relation tous les éléments qui sont extrémités d'un chemin bleu-rouge-bleu s'exprime simplement comme le produit $U\sigma U$. Cette matrice comprend tous les liens axiomes issus de l'élimination des coupures de niveau 1, mais elle peut également comprendre d'autre relations si le réseau initial comporte des coupures de niveau 2, ainsi que le montre la figure 2.14(b).

(a) Coupure de niveau 1

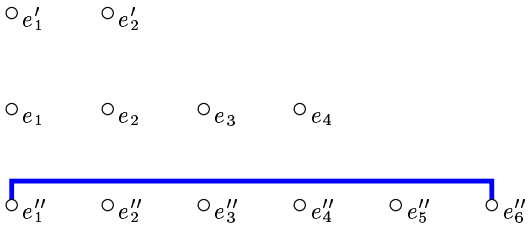
(b) Élimination de cette coupure

FIG. 2.13 – Réécriture de graphe pour l'élimination des coupures



(a) Coupures de niveaux 0, 1 et 2

(b) Élimination des coupures de niveau 1



(c) Élimination des coupures de niveau 2

FIG. 2.14 – Tous les chemins bleu-rouge-bleu-…-bleu du graphe

Pour ne garder que les liens issus d'une élimination de coupure de niveau 1, il suffit de ne prendre parmi toutes les relations exprimées par $U\sigma U$ celles dont les sommets incidents ne sont pas prémisses d'un lien coupure et qui n'ont pas d'arêtes de σ incidentes. Et donc, ce sont les sommets entre lesquels il n'y a pas de chemin rouge-rouge-bleu-rouge-bleu (les deux premières arêtes rouge-rouge font qu'un tel chemin ne peut commencer que sur un sommet prémisse d'un lien coupure : c'est un aller-retour sur un lien coupure car il n'y a pas deux liens rouges adjacents) ou bleu-rouge-bleu-rouge-rouge. Il faut donc supprimer les relations exprimées par $\sigma\sigma U\sigma U$ et par $U\sigma U\sigma\sigma$. Attention, si un chemin bleu-rouge-bleu est entre des sommets tous les deux prémisses de liens coupure (caracté-

risé par $\sigma\sigma U\sigma U\sigma\sigma$), ce chemin aura été supprimé deux fois. Autrement dit les liens axiomes issus de l'élimination des coupures de niveau 1 sont ceux décrits par la matrice

$$U\sigma U - \sigma^2 U\sigma U - U\sigma U\sigma^2 + \sigma^2 U\sigma U\sigma^2 = (1-\sigma^2)U\sigma U(1-\sigma^2)$$

Il faut maintenant tenir compte des coupures de niveau deux. Là encore, on ne va conserver que les liens entre sommets reliés par un chemin bleu-rouge-bleu-rouge-bleu (c'est-à-dire une relation décrite dans $U\sigma U\sigma U$) et qui ne sont pas prémisses d'un lien coupure. Au final : $(1-\sigma^2)U\sigma U\sigma U(1-\sigma^2)$. De même pour les coupures de niveau 3, 4, etc. L'absence d'æ-cycle assure que pour n suffisament grand, $U(\sigma U)^n = 0$ (σU est nilpotente), c'est-à-dire qu'il n'y a pas d'æ-chemin de longueur infinie.

Pour calculer le graphe issu de l'élimination de toutes les coupures, il suffit de calculer la matrice somme de chacune des matrices d'élimination des coupures de niveau n, sans oublier le niveau 0 pour les liens axiomes dont aucune conclusion n'est prémisse d'un lien coupure. Donc le graphe résultant de l'élimination des coupures σ sur U est donné par la matrice :

$$\begin{aligned}
\mathrm{Res}(U,\sigma) &= (1-\sigma^2)U(1-\sigma^2) \\
&\quad + (1-\sigma^2)U\sigma U(1-\sigma^2) \\
&\quad + (1-\sigma^2)U\sigma U\sigma U(1-\sigma^2) + \dots \\
&= (1-\sigma^2)(U + U\sigma U + U\sigma U\sigma U + \dots)(1-\sigma^2) \\
&= (1-\sigma^2)U(1+\sum_{k=1}^{\infty}(\sigma U)^k)(1-\sigma^2)
\end{aligned}$$

La dernière étape est valide car σU est nilpotente.

Or si (σU) est nilpotente

$$\begin{aligned}
(1-\sigma U)(1+\sum_{k=1}^{\infty}(\sigma U)^k) &= 1 + \sum_{k=1}^{\infty}(\sigma U)^k - \sigma U - \sum_{k=1}^{\infty}(\sigma U)^{k+1} \\
&= 1 + \sum_{k=1}^{\infty}(\sigma U)^k - \sigma U - \sum_{k=2}^{\infty}(\sigma U)^k \\
&= 1 + \sum_{k=1}^{\infty}(\sigma U)^k - \sum_{k=1}^{\infty}(\sigma U)^k \\
&= 1 \\
&= (1+\sum_{k=1}^{\infty}(\sigma U)^k)(1-\sigma U)
\end{aligned}$$

D'où la formule exprimant le graphe résultat de l'élimination des coupures à partir des matrices codant les liens axiomes et les liens coupures :

$$\mathrm{Res}(U,\sigma) = (1-\sigma^2)U(1-\sigma U)^{-1}(1-\sigma^2)$$

Cette formule nous sert au chapitre 6 pour exprimer le problème de la génération comme une relation entre une preuve avec coupures et une preuve sans coupure.

Chapitre 3

Déduction naturelle, λ-calcul typé et modèles

Ce chapitre permet de définir comment les grammaires de types logiques articulent la syntaxe et la sémantique. Cette dernière, comme le montre la section 3.1.3, s'exprime dans le langage du λ-calcul typé que définit la première section. L'essence du lien entre la syntaxe et la sémantique relève, vu les choix de modélisation, de la correspondance existant entre la logique (syntaxe) et le λ-calcul (sémantique). Elle s'exprime à travers l'isomorphisme de Curry-Howard. Nous la présentons d'une part pour le calcul des séquents et d'autre part pour les réseaux à la dernière section.

3.1 λ-calcul typé

Le λ-calcul fut introduit par Alonzo Church, pour fournir un langage de description de fonctions et de combinaison de fonctions. La nécessité de définir deux fonctions f de x et g de y différentes :

$$f : \quad x \rightarrowtail x - y \qquad g : y \rightarrowtail x - y$$

sans avoir besoin de spécifier le nom de la fonction a amené à utiliser un symbole pour « lier » les variables (comme les quantificateurs existentiels et universels en logique du premier ordre) : le λ.

On écrit ainsi, respectivement au lieu de f et g :

$$\lambda x.x - y \qquad \lambda y.x - y$$

Et les valeurs $f(0)$ et $f(1)$ se notent :

$$(\lambda x.x - y)0 = 0 - y \qquad (\lambda x.x - y)1 = 1 - y$$

Mais cette notation permet surtout de manipuler facilement des fonctions d'ordre supérieur (des fonctions de fonctions).

Cette section présente le fragment typé du λ-calcul. Il s'agit d'ajouter à chaque λ-terme une formule, son type, construite selon certaines règles et exprimant une propriété du λ-terme. On obtient ainsi une classe restreinte de λ-termes possédant certaines propriétés intéressantes telles la normalisation forte. [Kri90] donne une présentation détaillée, aussi bien du λ-calcul en général que de la partie typée et des modèles. Bien que nous en inspirant, nous donnons tout de suite une vision certes restreinte du λ-calcul, mais plus près de l'utilisation qui en est faite en linguistique pour la représentation sémantique.

En particulier, traditionnellement on représente le sens d'une phrase *Pierre aime Marie* comme une formule où un prédicat **aime** est appliqué à deux variables **p** et **m** : $\mathbf{aime}(\mathbf{p}, \mathbf{m})$. Pour un verbe intransitif, comme dans la phrase *Jean court*, un prédicat unaire représente le sens : $\mathbf{court}(\mathbf{j})$. La représentation d'une expression comme *aime Marie*, un verbe transitif avec son complément, par $\mathbf{aime}(x, \mathbf{m})$ fait apparaître une variable libre. Or son type $np\backslash S$ est le même que celui d'un verbe intransitif. La relation qui existe au niveau syntaxique (tous deux $np\backslash S$) entre un verbe intransitif et un verbe transitif avec son complément disparaît au niveau sémantique puisqu'un prédicat unaire représente l'un tandis qu'un prédicat binaire représente l'autre. Le λ-calcul permet d'unifier fonctionnellement ces deux expressions.

3.1.1 Langage : types et termes

Définition 25 (Types). Les *types* sont définis à partir de l'ensemble fini des types de base $\mathcal{T}_0$ selon la grammaire suivante :

$$\mathcal{T} ::= \mathcal{T}_0 | \mathcal{T} \to \mathcal{T} | \mathcal{T} \times \mathcal{T}$$

Si T est un type qui ne fait pas intervenir le produit $\times$, on définit l'*ordre de* T par :
- $o(T) = 1$ si T est atomique ;
- $o(T_1 \to T_2) = \max\{1 + o(T_1), o(T_2)\}$.

Définition 26 (Termes). Les termes sont construits de la manière suivante : pour chaque type $\tau \in \mathcal{T}$, soit $\mathcal{V}_\tau$ un ensemble dénombrable de *variables* de type τ et $\mathcal{C}_\tau$ un ensemble fini de *constantes* de type τ ($\mathcal{V} = \cup_\tau \mathcal{V}_\tau$ et $\mathcal{C} = \cup_\tau \mathcal{C}_\tau$).

On a alors :
- les éléments de $\mathcal{V}_\tau$ et de $\mathcal{C}_\tau$ sont des termes de type τ ;
- si u et v sont des termes de types respectifs U et V, alors $\langle u, v\rangle$ est un terme de type $U \times V$;
- si α est un terme de type $U \times V$, alors $\pi^1\alpha$ et $\pi^2\alpha$ sont des termes de types respectifs U et V ;
- si v est un terme de type V et x^U est une variable de type U alors $\lambda x^U.v$ est un terme de type $U \to V$;
- si α et u sont deux termes de types respectifs $U \to V$ et U, alors $(\alpha), u$ est un terme de type V.

Un terme de la forme $t\,u$ est une application (de t à u), et un terme de la forme $\lambda x.\alpha$ est une abstraction fonctionnelle, ou λ-abstraction. Ainsi, si **m** et **p** sont de type e, et **aime** de type $e \to e \to t$, $(\mathbf{aime}\,\mathbf{m})\mathbf{p}$ est un terme de type t : l'application de **aime** à **m** et **p**.

Définition 27 (Occurrences libres de variable). Pour un terme α et une variable x, les *occurrences libres* de x dans α se définissent par :
- si α est une variable x, l'occurrence de x dans α est libre ;
- si $\alpha = (u)v$, les occurrences libres de x dans α sont celles de x dans u et v ;
- si $\alpha = \lambda y.u$, les occurrences libres de x dans α sont celles de x dans u sauf si $x = y$. Dans ce cas, x n'a pas d'occurrences libres dans α.

Une *variable libre* de α est une variable ayant au moins une occurrence libre dans α. Un terme sans variable libre est appelé un *terme clos*.

Une *variable liée* de α est une variable apparaissant dans α derrière le symbole λ.

Définition 28 (Substitution simple). Soient $\alpha, t_1, \ldots, t_k$ des termes et $x_1, \ldots, x_k$ des variables distinctes. On définit le terme $\alpha\langle t_1/x_1, \ldots, t_k/x_k\rangle$ comme le résultat de la substitution de t_i à chaque occurence libre de x_i dans α, obtenu par induction :
- si $\alpha = x_i$, $\alpha\langle t_1/x_1, \ldots, t_k/x_k\rangle = t_i$;
- si α est une variable différente de chaque x_i, $\alpha\langle t_1/x_1, \ldots, t_k/x_k\rangle = \alpha$;
- si $\alpha = (w)v$, $\alpha\langle t_1/x_1, \ldots, t_k/x_k\rangle = (w\langle t_1/x_1, \ldots, t_k/x_k\rangle)v\langle t_1/x_1, \ldots, t_k/x_k\rangle$;

- si $\alpha = \langle w, v\rangle$, $\alpha\langle t_1/x_1, \dots, t_k/x_k\rangle = \langle w\langle t_1/x_1, \dots, t_k/x_k\rangle, v\langle t_1/x_1, \dots, t_k/x_k\rangle\rangle$;
- si $\alpha = \lambda x_i.v$ alors

$$\alpha\langle t_1/x_1, \dots, t_k/x_k\rangle = \lambda x_i.v\langle t_1/x_1, \dots, t_{i-1}/x_{i-1}, t_{i+1}/x_{i+1}, \dots, t_k/x_k\rangle$$

  ;
- si $\alpha = \lambda y.v$ avec $y \neq x_1, \dots, x_k$ alors $\alpha\langle t_1/x_1, \dots, t_k/x_k\rangle = \lambda y.v\langle t_1/x_1, \dots, t_k/x_k\rangle$.

Définition 29 (α-equivalence). On définit la relation binaire $u =_\alpha u'$ par :
- si u est une variable, $u =_\alpha u'$ si et seulement si $u = u'$;
- si $u = (w)v$, $u =_\alpha u'$ si et seulement si $u' = (w')v'$ avec $v =_\alpha v'$ et $w =_\alpha w'$;
- si $u = \langle w, v\rangle$, $u =_\alpha u'$ si et seulement si $u' = \langle w', v'\rangle$ avec $v =_\alpha v'$ et $w =_\alpha w'$;
- si $u = \lambda x.v$, $u =_\alpha u'$ si et seulement si $u' = \lambda y.v'$ avec $v\langle z/x\rangle =_\alpha v'\langle z/y\rangle$ pour toute variable z sauf un nombre fini.

Cette relation est une relation d'équivalence sur l'ensemble des termes.

Deux termes $u =_\alpha u'$ ont même structure et mêmes variables libres. Par la suite, on identifie les termes α-équivalents (qui diffèrent juste à cause des occurences liées) en considérant l'ensemble des termes quotienté par la relation $=_\alpha$.

Définition 30 ([Kri90]). On définit pour un terme u le terme $u[t_1/x_1, \dots, t_k/x_k]$ le résultat de la substitution de t_i aux occurences libres de x_i dans u commme étant $u'\langle t_1/x_1, \dots, t_k/x_k\rangle$ pour un terme u' tel que :
- $u =_\alpha u'$;
- aucune variable liée de u' ne soit libre dans l'un des t_i.

Il existe bien un tel terme u', et la classe d'équivalence de $u'\langle t_1/x_1, \dots, t_k/x_k\rangle$ ne dépend pas du choix de u'.

Cette nouvelle substitution, différente de la substitution simple, permet d'éviter de transformer la structure des termes. Ainsi, pour $u = \lambda x.y$:
- $u\langle x/y\rangle = \lambda x.x$, c'est-à-dire l'identité ;
- $u[x/y] = u'\langle x/y\rangle$ avec $u' = \lambda z.y$ par exemple, soit $u[x/y] = \lambda z.x$, qui n'est pas l'identité.

Ainsi, lors de la substitution, si l'on introduit une variable qui se trouve être liée dans le terme par un λ, on procède tout d'abord à un changement de nom de la variable liée, afin de ne pas modifier le comportement opérationnel du terme.

Définition 31. Un terme est *normal* si aucun de ses sous-termes n'est de la forme :

$$\pi^1\langle u, v\rangle \text{ ou } \pi^2\langle u, v\rangle \text{ ou } (\lambda x^U.v)u$$

On définit la relation binaire $\alpha \to_\beta \alpha'$ (α se *convertit* en α') par :
- si α est une variable, $\alpha \to_\beta \alpha'$ est faux pour tout α' ;
- si $\alpha = \pi^1\langle u, v\rangle$ alors $\alpha \to_\beta \alpha'$ si et seulement si $\alpha' = u$;
- si $\alpha = \pi^2\langle u, v\rangle$ alors $\alpha \to_\beta \alpha'$ si et seulement si $\alpha' = v$;
- si $\alpha = \lambda x.u$ alors $\alpha \to_\beta \alpha'$ si et seulement si $\alpha' = \lambda x.u'$ avec $u \to_\beta u'$;
- si $\alpha = (v)u$ alors alors $\alpha \to_\beta \alpha'$ si et seulement si :
  - ou bien $t' = (v)u'$ avec $u \to_\beta u'$;
  - ou bien $t' = (v')u$ avec $v \to_\beta v'$;
  - ou bien $v = (\lambda x.w)$ et $\alpha' = w[u/x]$.

α est appelé *redex* et α' son *contracté*.

On définit la relation binaire $\alpha \to_\beta^* \alpha'$ (α se *réduit* ou se β-réduit en un terme α') comme la fermeture reflexive et transitive de $\to_\beta$.

Exemple 2. Voici quelques réductions :

– $(\lambda x.(x)\, y)\, (\lambda z.z) \rightarrow^*_\beta (x)y[(\lambda z.z)/x] = (\lambda z.z)y \rightarrow^*_\beta y$;

– soit $\alpha = \lambda n.\lambda f.\lambda x.((n)f)(f)x$. On a ensuite :

$$\begin{aligned}\gamma &= (\alpha)\lambda f.\lambda x.fx \\ &= (\lambda n.\lambda f.\lambda x.((n)f)(f)x)(\lambda f.\lambda x.(f)x)\end{aligned}$$

Puis

$$\begin{aligned}\gamma &\rightarrow^*_\beta \lambda f.\lambda x.((\lambda f.\lambda x.(f)x)f)(f)x \\ &\rightarrow^*_\beta \lambda f.\lambda x.(\lambda x.(f)x)(f)x \\ &\rightarrow^*_\beta \lambda f.\lambda x.(f)(f)x\end{aligned}$$

– soit alors $(\alpha)(\alpha)\lambda f.\lambda x.(f)x$. Ce terme se réduit en :

$$(\alpha)(\lambda f.\lambda x.(f)(f)x) \rightarrow^*_\beta \lambda f.\lambda x.((\lambda f.\lambda x.(f)(f)x)f)(f)x = \gamma_2$$

Puis $\gamma_2 \rightarrow^*_\beta \lambda f.\lambda x.(\lambda x.(f)(f)x)(f)x \rightarrow^*_\beta \lambda f.\lambda x.(f)(f)(f)x$.

Remarquons que dans le dernier exemple, nous avons réduit en premier le terme

$$\gamma = (\alpha)\lambda f.\lambda x.(f)x$$

Or, pour réduire $(\alpha)\,\gamma = (\lambda n.\lambda f.\lambda x((n)f)(f)x)\gamma$, on aurait pu commencer par réduire le terme le plus extérieur, soit $(\alpha)\,\gamma \rightarrow^*_\beta \lambda f.\lambda x((\gamma)f)(f)x$ puis seulement $(\gamma)f$. On s'aperçoit que dans tous les cas, la réduction finit et converge vers le même terme. Mais est-ce vrai pour tous les λ-termes ? La réponse est donnée par les théorèmes 39 et 40.

Théorème 39. *$\rightarrow^*_\beta$ est confluente (si u se réduit en u_1 et u_2, il existe un terme v tel que u_1 se réduit en v et u_2 se réduit en v). Et donc un terme α a au plus une forme normale.*

Définition 32. Un terme α est *fortement normalisable* s'il n'admet aucune suite infinie de réduction.

Théorème 40. *Tous les termes du λ-calcul simplement typé sont fortement normalisables.*

Pour les différentes preuves, nous renvoyons à [GLT88].

Ce théorème montre que les termes de l'exemple se réduisent bien car ils sont bien typés. Par contre, dans le λ-calcul non typé, on peut trouver des termes admettant des réductions infinies (tels $\omega = (\lambda x.x\, x)\lambda x.x\, x$ qui n'est pas normalisable, et donc $(\lambda x.y)\omega$ qui n'est pas fortement normalisable). La section 3.1.3 montre comment cette propriété permet d'assurer systématiquement le calcul du sens pour une phrase syntaxiquement correcte.

Enfin, notons l'importante propriété que les relations $\rightarrow_\beta$ et $\rightarrow^*_\beta$ préservent les types.

Définition 33. Une *équation* est une paire de termes α, β. Un *problème d'unification* est un exemble fini d'équations. Une *solution*, ou un *unificateur* d'un tel problème est une substitution θ telle que pour toute paire α, β du problème, les termes $\alpha[\theta]$ et $\beta[\theta]$ ont la même forme normale. Si β est un terme sans variable libre, on parle d'un problème de *filtrage* (*matching*).

Une substitution θ_1 est *plus générale* que θ_2 s'il existe une substitution η telle que $\theta_2 = \eta \circ \theta_1$.

Une autre transformation comparable peut-être ajoutée, celle de $v = \lambda x.(ux)$ en u quand x n'apparaît pas dans u. On l'appelle η-conversion et permet d'identifier les termes v et u. Le passage de u à v s'appelle η-expansion. Les théorèmes de confluence et normalisation sont encore vérifiés grâce au typage.

Maintenant que nous avons défini les termes, nous allons définir les modèles du λ-calcul et en particulier ceux dans lesquels exprimer la vérité de certaines phrases du langage. Pour le λ-calcul simplement typé, il existe bien d'autres modèles encore.

3.1.2 Modèle

Définition 34. Soit une famille $(\mathcal{D}_\tau)_{\tau\in\mathcal{T}_0}$ d'ensembles. On définit ensuite la *structure* $S = \cup_{\tau\in\mathcal{T}}\mathcal{D}_\tau$ telle que :

- si $\tau = \tau_1 \times \tau_2$, $\mathcal{D}_\tau = \mathcal{D}_{\tau_1} \times \mathcal{D}_{\tau_2}$ (produit cartésien) ;
- si $\tau = \tau_1 \to \tau_2$, $\mathcal{D}_\tau = {\mathcal{D}_{\tau_2}}^{\mathcal{D}_{\tau_1}}$ (ensemble des fonctions de $\mathcal{D}_{\tau_1}$ dans $\mathcal{D}_{\tau_2}$).

Un *modèle* est une paire $\mathcal{M} = (S, [\![\cdot]\!])$ où S est une structure et $[\![\cdot]\!] \in S^{\mathcal{C}}$ une fonction d'interprétation respectant le typage telle que $[\![\alpha]\!] \in \mathcal{D}_\tau$ si $\alpha \in \mathcal{C}_\tau$.

Une *valuation* est une fonction $\theta : \mathcal{V} \to S$ telle que $\theta(x) \in \mathcal{D}_\tau$ si $x \in \mathcal{V}_\tau$.

La *dénotation* $[\![\alpha]\!]^\theta_{\mathcal{M}}$ d'un terme α suivant le modèle $\mathcal{M} = (S, [\![\cdot]\!])$ et la valuation θ est définie par :

- $[\![x]\!]^\theta_{\mathcal{M}} = \theta(x)$ si $x \in \mathcal{V}$;
- $[\![c]\!]^\theta_{\mathcal{M}} = [\![c]\!]$ si $c \in \mathcal{C}$;
- $[\![\alpha\beta]\!]^\theta_{\mathcal{M}} = [\![\alpha]\!]^\theta_{\mathcal{M}}([\![\beta]\!]^\theta_{\mathcal{M}})$;
- $[\![\lambda x.\alpha]\!]^\theta_{\mathcal{M}} = f \in S^S$ telle que pour tout $a \in S$, $f(a) = [\![\alpha]\!]^{\theta'}_{\mathcal{M}}$ où $\theta'(x) = a$ et pour tout $y{\neq}x$, $\theta'(y) = \theta(y)$;
- $[\![\langle\alpha, \beta\rangle]\!]^\theta_{\mathcal{M}} = ([\![\alpha]\!]^\theta_{\mathcal{M}}, [\![\beta]\!]^\theta_{\mathcal{M}})$;
- $[\![\pi^1\alpha]\!]^\theta_{\mathcal{M}} = u$ si $[\![\alpha]\!]^\theta_{\mathcal{M}} = (u, v)$;
- $[\![\pi^2\alpha]\!]^\theta_{\mathcal{M}} = v$ si $[\![\alpha]\!]^\theta_{\mathcal{M}} = (u, v)$.

Théorème 41. *Si α est un terme de type τ, alors $[\![\alpha]\!]^\theta_{\mathcal{M}} \in \mathcal{D}_\tau$ pour tous modèles $\mathcal{M}$ et valuations θ.*

Démonstration. La preuve se fait par induction sur les termes. C'est vrai pour les variables et les constantes. Si $\alpha\beta$ est de type V, alors par définition il existe $U \in \mathcal{T}$ tel que α soit de type $U \to V$ et β soit de type U. Par hypothèse d'induction, $[\![\alpha]\!]^\theta_{\mathcal{M}} \in \mathcal{D}_{U\to V} = {\mathcal{D}_V}^{\mathcal{D}_U}$ et donc $[\![\alpha\beta]\!]^\theta_{\mathcal{M}} = [\![\alpha]\!]^\theta_{\mathcal{M}}([\![\beta]\!]^\theta_{\mathcal{M}}) \in \mathcal{D}_V$.

Si le terme est $\lambda x.\alpha$ de type $U \to V$, α de type V, et pour tout $a \in \mathcal{D}_U$, $f(a) = [\![\alpha]\!]^{\theta'}_{\mathcal{M}}$ et par hypothèse d'induction, $[\![\alpha]\!]^{\theta'}_{\mathcal{M}} \in \mathcal{D}_V$ d'où $[\![\lambda x.\alpha]\!]^\theta_{\mathcal{M}} = f \in {\mathcal{D}_V}^{\mathcal{D}_U} = \mathcal{D}_{U\to V}$.

De même si le terme est $\langle\alpha, \beta\rangle$ de type $U \times V$, par définition $[\![\langle\alpha, \beta\rangle]\!]^\theta_{\mathcal{M}} = ([\![\alpha]\!]^\theta_{\mathcal{M}}, [\![\beta]\!]^\theta_{\mathcal{M}})$ et $([\![\alpha]\!]^\theta_{\mathcal{M}}, [\![\beta]\!]^\theta_{\mathcal{M}}) \in \mathcal{D}_U \times \mathcal{D}_V$ par hypothèse d'induction, c'est-à-dire aussi $\mathcal{D}_{U\times V}$.

Enfin, si le terme est $\pi^1\alpha$ de type U, α est de type $U \times V$ et $[\![\alpha]\!]^\theta_{\mathcal{M}} = \langle u, v\rangle \in \mathcal{D}_{U\times V}$ par hypothèse d'induction. D'où $[\![\pi^1\alpha]\!]^\theta_{\mathcal{M}} = u \in \mathcal{D}_U$. De même pour π^2. □

3.1.3 Sémantique de Montague

L'approche de Montague [Mon74, DWP81, Mor94] pour la modélisation de la sémantique de la langue s'appuie sur deux éléments principaux. Le premier consiste en une architecture basée sur une association règle (syntaxique) à règle (sémantique) qui reprend les notions de compositionalité de Frege. Le second consiste en l'utilisation, comme langage de description sémantique, du λ-calcul typé.

La logique intensionnelle définie par Montague utilise la référence à différents mondes possibles, ainsi qu'au temps, et utilise pour cela un type de base particulier. Nous nous limitons dans la suite à un seul monde et un seul temps, car outre une simplification des notations et des modèles, notre utilisation de la sémantique n'a pas pour objet de vérifier si une phrase exprime un fait vrai ou faux, mais plutôt celui d'être une représentation abstraite d'une phrase. Cette indépendance vis-à-vis de la langue permet d'utiliser cette représentation comme *interlangue* pour passer d'expressions d'une langue à une autre langue.

La sémantique utilisée par Montague se base sur un modèle particulier de ceux définis à la section précédente. On a, étant donné un ensemble non vide A d'entités :

- $\mathcal{T}_0 = \{e, t\}$;

– $\mathcal{D}_t = \{0, 1\}$;
– $\mathcal{D}_e = A$.

Si $A = \{\text{Othello}, \text{Desdémone}\}$, le modèle $\mathcal{M} = (S, [\![\cdot]\!])$ où :

– $[\![\mathbf{o}]\!] = \text{Othello}$;
– $[\![\mathbf{d}]\!] = \text{Desdémone}$;
– $[\![\mathbf{meurt}]\!] = g$ telle que

$$g : \begin{array}{ll} \text{Othello} & \to 1 \\ \text{Desdémone} & \to 1 \end{array}$$

– $[\![\mathbf{aime}]\!] = f$ telle que

$$f : \begin{array}{ll} \text{Othello} & \to \left\{ \begin{array}{ll} \text{Othello} & \to 0 \\ \text{Desdémone} & \to 1 \end{array} \right. \\ \text{Desdémone} & \to \left\{ \begin{array}{ll} \text{Othello} & \to 1 \\ \text{Desdémone} & \to 0 \end{array} \right. \end{array}$$

permet de donner une interprétation aux λ-termes. On remarque par exemple que **meurt** peut s'interpréter comme un prédicat unaire. C'est pourquoi on parle de cette catégorie de termes comme exprimant une propriété des entités. On peut interpréter aussi par exemple les λ-termes $(\mathbf{aime}\,\mathbf{d})\mathbf{o}$ et $(\mathbf{aime}\,\mathbf{o})\mathbf{d}$.

Parmi les constantes typées utilisées fréquemment, notons :

– $\forall : (e \to t) \to t$;
– $\exists : (e \to t) \to t$;
– $\Rightarrow : t \to (t \to t)$;
– $\wedge : t \to (t \to t)$;

avec leur interprétation respective intuitive :

– $[\![\forall]\!] = h$ telle que pour tout $P \in \{0,1\}^A$, $h(P) = 1$ si et seulement si pour tout $x \in A$, $P(x) = 1$;
– $[\![\exists]\!] = h'$ telle que pour tout $P \in \{0,1\}^A$, $h(P) = 1$ si et seulement s'il existe $x \in A$ avec $P(x) = 1$;
– $[\![\Rightarrow]\!] = k$ telle que

$$k : \begin{array}{ll} 0 & \to \left\{ \begin{array}{ll} 0 & \to 1 \\ 1 & \to 1 \end{array} \right. \\ 1 & \to \left\{ \begin{array}{ll} 0 & \to 0 \\ 1 & \to 1 \end{array} \right. \end{array}$$

  Si α et β sont deux termes de type t, $(\Rightarrow\ \alpha)\beta$ est de type t. Mais on s'autorise à utiliser également la notation infixée $\alpha \Rightarrow \beta$;
– $[\![\wedge]\!] = k'$ telle que

$$k' : \begin{array}{ll} 0 & \to \left\{ \begin{array}{ll} 0 & \to 0 \\ 1 & \to 0 \end{array} \right. \\ 1 & \to \left\{ \begin{array}{ll} 0 & \to 0 \\ 1 & \to 1 \end{array} \right. \end{array}$$

  Si α et β sont deux termes de type t, $(\wedge\,\alpha)\beta$ est de type t. Mais on s'autorise à utiliser également la notation infixée $\alpha \wedge \beta$.

On utilisera de même les symboles habituels $=, \Leftrightarrow \ldots$

Reprenons les deux termes : $(\mathbf{aime}\,\mathbf{d})\mathbf{o}$ et $(\mathbf{aime}\,\mathbf{o})\mathbf{d}$. Si l'on sait comment relier les phrases *Othello aime Desdémone* et *Desdémone aime Othello* à ces deux termes, on pourra dire si dans le modèle choisi, ces deux énoncés sont vrais ou faux. On voudrait d'ailleurs que les phrases *Othello*

loves Desdemona et *Desdemona loves Othello* aient les même interprétations que leur traduction en français. On voit alors que pour traduire d'une langue vers une autre ou bien pour engendrer dans plusieurs langues des phrases ayant le même sens, le λ-terme suffit et ne demande pas nécessairement un modèle explicite. Aussi pouvons-nous laisser aux critiques littéraires et aux psychanalystes le soin de débattre sur le fait qu'Othello aime ou non Desdémone (c'est-à-dire savoir si dans $\mathcal{M}$, $[\![(\textbf{aime d})\textbf{o}]\!]$ vaut 0 ou 1) pour nous consacrer à l'association des règles de composition syntaxique et sémantique.

Pour cela, après avoir donné le langage formel permettant la description de la sémantique, définissons très rapidement celui qui permet à Montague de décrire la syntaxe. Elle se base sur des principes similaires à ceux des grammaires catégorielles (voir chapitre 4, ou encore ceux sommairement décrits dans la section 1.1.2) : aux mots sont associées des catégories syntaxiques telles que les expressions de catégorie A/B et les expressions de catégorie B se combinent pour donner des expressions de catégorie A.

Sans entrer dans la totalité des définitions (que l'on peut retrouver dans [DWP81]), le tableau 3.1 résume les principales catéories syntaxiques retenues par Montague.

Nom de la catégorie	Définition de la catégorie	Traduction de la catégorie en logique intensionnelle
e	e	e
t	t	t
IV	t/e	$e \to t$
TV	$(t/e)/e$	$e \to (e \to t)$
T	t/IV	$(e \to t) \to t$
CN	t/e	$e \to t$
etc.	etc.	etc.

TAB. 3.1 – Définition des catégories syntaxiques de base

Puis, à partir des ensembles d'expressions simples (non composées) des différentes catégories (B_A est l'ensemble des expressions simples de catégorie A) et des ensembles P_A des expressions composées de catégorie A, on donne les règles de formations syntaxiques. Là encore, nous ne donnons ici que quelques-unes de ces règles (suivant la numérotation de Montague, les règles concernant la formation syntaxique sont numérotées S_n. Les règles concernant la formation sémantique sont notées T_n).

Règle (S_1). $B_A \subset P_A$ pour toute catégorie A.

Règle (S_4). Si $a \in P_T$ et $b \in P_{IV}$, alors $F_4(a, b) = ab' \in P_t$ où b' résulte du remplacement du premier verbe dans b par sa forme à la première personne du singulier au présent de l'indicatif.

F_4 est une opération syntaxique (de manière générale, les F_n sont des opérations syntaxiques) qui décrit comment les éléments de base sont modifés et concaténés. Une opération est définie la première fois qu'elle est utilisée dans une règle, mais peut-être utilisée dans plusieurs règles. Ainsi, l'opération F_4 pourrait très bien être utilisée dans une règle S_{15} pour en décrire le résultat.

Par exemple, si l'on a *Othello* $\in B_T$ et *mourir* $\in B_{IV}$, puisqu'alors *Othello* $\in P_T$ et *mourir* $\in P_{IV}$ grâce à la règle S_1, la règle S_4 nous permet de construire l'expression $F_4(\textit{Othello}, \textit{mourir}) =$ *Othello meurt* $\in P_t$.

Le principe de compositionalité amène à définir la régle de formation sémantique T_4 correspondant à S_4 :

Règle (T_4). Si $a \in P_T$ et $b \in P_{IV}$ se traduisent en logique intensionnelle respectivement par α et β, alors $F_4(a, b)$ se traduit par $\alpha\,\beta$.

En reprenant l'exemple précédent, et en supposant que le lexique associe respectivement à *Othello* et *mourir* les constantes (bien typées sémantiquement relativement à leur type syntaxique) $\lambda x^{e\to t}.x\,\mathbf{o}$ (on définit l'individu *Othello* en intension comme l'ensemble des propriétés que vérifie *Othello*) et **meurt**, l'expression *Othello meurt* $= F_4(\textit{Othello}, \textit{mourir})$ se traduit par le terme $(\lambda x.x\,\mathbf{o})\mathbf{meurt}$, équivalent à $\mathbf{meurt}\,\mathbf{o}$.

Donnons encore un exemple de règle, celle des constructions déterminant-nom. Elle nous permettra d'illustrer à la fois la pertinence de cette représentation sémantique pour le traitement des quantificateurs, ainsi que l'ambiguïté de la langue naturelle.

Règle (S_2). Si $a \in P_{T/CN}$ et $b \in P_{CN}$, alors $F_2(a,b) = a'b \in P_T$ où a' est :
- *la* si a est *le* et b est féminin ;
- *une* si a est *un* et b est féminin ;
- *toute* si a est *tout* et b est féminin.

Cette règle permet la formation d'expressions comme *un homme* et *une femme*.

Règle (T_2). Si $a \in P_{T/CN}$ et $b \in P_{CN}$ se traduisent respectivement en α et β, alors $F_2(a,b)$ se traduit en $\alpha\,\beta$.

Si l'on utilise comme traductions des termes de base :
- *tout* : $\lambda P.\lambda Q.\forall(\lambda x.(Px \Rightarrow Qx))$;
- *le* : $\lambda P.\lambda Q.\exists(\lambda y.(\forall(\lambda x.(Px \Leftrightarrow (x = y)))) \wedge (Q\,y))$;
- *un* : $\lambda P.\lambda Q.\exists(\lambda x.(Px) \wedge (Qx))$,

on peut former *tout homme meurt* $= F_4(F_2(\textit{tout}, \textit{homme}), \textit{mourir})$ dont le sens est donné par :

$$\begin{aligned}((\lambda P.\lambda Q.\forall(\lambda x.(Px \Rightarrow Qx)))\mathbf{homme})\mathbf{meurt} &\to^*_\beta (\lambda Q.\forall(\lambda x.(\mathbf{homme}\,x \Rightarrow Q\,x)))\mathbf{meurt}\\ &\to^*_\beta \forall(\lambda x.(\mathbf{homme}x \Rightarrow \mathbf{meurt}x))\end{aligned} \tag{3.1}$$

Le théorème 40 nous assure que cette forme normale est toujours calculable.

Nous verrons par la suite d'autres exemples qui illustrent les capacités de cette formalisation, notamment en ce qui concerne les quantificateurs. En particulier nous donnerons quelques exemples d'ambiguïtés dues au choix de la portée des quantificateurs. Nous verrons aussi comment on peut obtenir des paraphrases en choisissant judicieusement les traductions sémantiques des expressions, de façon à les définir non pas avec de nouvelles constantes typées mais avec des constantes apparaissant déjà pour d'autres expressions.

En regardant les règles définies, du point de vue sémantique une opération apparaît : l'application d'un terme à un autre. De même, aux modifications morpho-syntaxiques près (pour tenir compte du genre et des variations qu'il entraîne), l'opération de base de la combinaison syntaxique est la concaténation des éléments syntaxiques, à condition qu'ils correspondent à un certain typage, ce dernier se traduisant directement pour les λ-termes exprimant la sémantique.

Comme le montre le chapitre 5, en s'appuyant sur la section suivante, oter la prise en compte des variations morphologiques des opérations de combinaisons syntaxiques permet d'unifier les règles de combinaison et surtout de correspondance entre la formation syntaxique et la formation sémantique. Nous allons maintenant décrire comment cette liaison syntaxe-sémantique s'exprime de façon très naturelle entre les types considérés comme des formules logiques et les λ-termes.

3.2 Isomorphisme de Curry-Howard

Comme l'indique le chapitre 1, la logique classique s'interprète habituellement en terme de vérité. Ainsi le principe du tiers exclu ($p \vee \neg p$ est toujours vrai) exprime-t-il qu'une proposition doit être

vraie ou fausse. Cela conduit à des démonstrations non *constructives*, permettant de démontrer des propositions comme :

Il existe deux nombres irrationnels x et y tels que x^y soit un nombre rationnel.

(Preuve : si $\sqrt{2}^{\sqrt{2}}$ est un nombre rationnel, alors on prend $x = y = \sqrt{2}$, sinon on prend $x = \sqrt{2}^{\sqrt{2}}$ et $y = \sqrt{2}$.) Par contre, la démonstration ne nous donne pas d'algorithme de calcul de x et de y.

Ces considérations ont donné naissance à la logique intuitionniste. Pour celle-ci, l'interprétation des connecteurs permet l'interprétation d'une preuve comme un algorithme. Ainsi, une preuve de $A \wedge B$ c'est la donnée d'une preuve de A et d'une preuve de B. Une preuve de $A \vee B$ c'est un nombre $i \in \{1, 2\}$ avec une preuve de A si $i = 1$ et une preuve de B si $i = 2$. Une preuve de $A \to B$ est une méthode qui transforme toute preuve de A en une preuve de B, etc.

Ce principe s'illustre bien dans le système de la *déduction naturelle*. L'intérêt ne réside plus dans la valeur de vérité mais dans l'objet preuve lui-même, et les opérateurs logiques s'interprètent alors comme des constructeurs de preuves, et apportent une notion *procédurale* à la prouvabilité.

Cette propriété se traduit syntaxiquement dans l'isomorphisme de Curry-Howard, qui établit le lien entre la notion de preuve et celle de calculabilité, exprimée par le λ-calcul. Cette section présente cet isomorphisme sous trois formes : la première, avec la déduction naturelle, donne un nouveau système logique. À la fois, ce système permet le mieux d'illustrer le principe sous-jacent de l'isomorphisme, et, dans le chapitre 5, il permet de donner de façon plus naturelle les analyses syntaxiques vues comme des déductions. Le deuxième n'est qu'un fragment de la logique classique, équivalent à celui de la déduction naturelle. Mais il offre de faire facilement le lien avec la troisième présentation : celle avec les réseaux de preuve.

3.2.1 Déduction naturelle

Les règles d'inférence (tableau 3.2) de la déduction naturelle traduisent directement les propriétés constructivistes attendues. Une *déduction* est la dérivation d'une proposition à partir d'un nombre fini d'hypothèses en utilisant ces règles d'inférence. Durant la déduction, différentes hypothèses peuvent être utilisées, et une déduction dont toutes les hypothèses ont été utilisées est une *preuve*.

La présentation de ces règles utilise la notation des séquents. On peut lire $\Gamma \vdash_{\mathsf{DN}} A$ comme « sous les hypothèses de Γ, on déduit A ». Nous reprenons la notation de [Gal95], dans laquelle Γ est un ensemble fini de propositions étiquetées tel que $x_1 : A_1, \ldots x_n : A_n$ où les x_i sont deux à deux distincts, mais pas forcément les A_i. Et alors, la notation $\Gamma, x : A$ n'est bien définie que lorsque $x \neq x_i$ pour tout $i \in [1, n]$.

$$\frac{}{\Gamma, x : A \vdash_{\mathsf{DN}} A}\ \text{Axiome}$$

$$\frac{\Gamma \vdash_{\mathsf{DN}} A \quad \Gamma \vdash_{\mathsf{DN}} B}{\Gamma \vdash_{\mathsf{DN}} A \wedge B}\ \wedge_i \qquad \frac{\Gamma \vdash_{\mathsf{DN}} A \wedge B}{\Gamma \vdash_{\mathsf{DN}} A}\ \wedge_e^1 \qquad \frac{\Gamma \vdash_{\mathsf{DN}} A \wedge B}{\Gamma \vdash_{\mathsf{DN}} B}\ \wedge_e^2$$

$$\frac{\Gamma, x : A \vdash_{\mathsf{DN}} B}{\Gamma \vdash_{\mathsf{DN}} A \Rightarrow B}\ \Rightarrow_i \qquad \frac{\Gamma \vdash_{\mathsf{DN}} A \quad \Gamma \vdash_{\mathsf{DN}} A \Rightarrow B}{\Gamma \vdash_{\mathsf{DN}} B}\ \Rightarrow_e$$

TAB. 3.2 – Règle de la déduction naturelle

Remarque 2. Il existe d'autres présentations, notamment celle de Prawitz, utilisée dans [GLT88]. Quelle que soit la présentation, c'est un système *différent* de celui du calcul des séquents intuitionnistes présenté à la section suivante, mais qui permet de prouver les mêmes théorèmes. Ses propriétés sont donc différentes, notamment en ce qui concerne la normalisation (puisqu'il n'y a pas de règle de coupure).

Exemple 3. Voici deux preuves dans ce système ;

1. Si $\Gamma = x : A \Rightarrow B, y : A \Rightarrow C, z : A$, on a la preuve

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash_{\mathsf{DN}} A}\,\mathsf{Ax} \quad \dfrac{}{\Gamma \vdash_{\mathsf{DN}} A \Rightarrow B}\,\mathsf{Ax}}{\Gamma \vdash_{\mathsf{DN}} B}\Rightarrow_e \quad \dfrac{\dfrac{}{\Gamma \vdash_{\mathsf{DN}} A}\,\mathsf{Ax} \quad \dfrac{}{\Gamma \vdash_{\mathsf{DN}} A \Rightarrow C}\,\mathsf{Ax}}{\Gamma \vdash_{\mathsf{DN}} C}\Rightarrow_e}{x : A \Rightarrow B, y : A \Rightarrow C, z : A \vdash_{\mathsf{DN}} B \wedge C}\wedge_i}{x : A \Rightarrow B, y : A \Rightarrow C \vdash_{\mathsf{DN}} A \Rightarrow (B \wedge C)}\Rightarrow_i}{y : A \Rightarrow C \vdash_{\mathsf{DN}} (A \Rightarrow B) \Rightarrow (A \Rightarrow (B \wedge C)}\Rightarrow_i}{\vdash_{\mathsf{DN}} (A \Rightarrow C) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow (B \wedge C))}\Rightarrow_i$$

2. Si $\Gamma = x : A \Rightarrow B, y : A \Rightarrow C, z_1 : A, z_2 : A$, on a la preuve

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash_{\mathsf{DN}} A}\,\mathsf{Ax} \quad \dfrac{}{\Gamma \vdash_{\mathsf{DN}} A \Rightarrow B}\,\mathsf{Ax}}{\Gamma \vdash_{\mathsf{DN}} B}\Rightarrow_e \quad \dfrac{\dfrac{}{\Gamma \vdash_{\mathsf{DN}} A}\,\mathsf{Ax} \quad \dfrac{}{\Gamma \vdash_{\mathsf{DN}} A \Rightarrow C}\,\mathsf{Ax}}{\Gamma \vdash_{\mathsf{DN}} C}\Rightarrow_e}{x : A \Rightarrow B, y : A \Rightarrow C, z_1 : A, z_2 : A \vdash_{\mathsf{DN}} B \wedge C}\wedge_i}{x : A \Rightarrow B, y : A \Rightarrow C, z_2 : A \vdash_{\mathsf{DN}} A \Rightarrow (B \wedge C)}\Rightarrow_i}{y : A \Rightarrow C, z_2 : A \vdash_{\mathsf{DN}} (A \Rightarrow B) \Rightarrow (A \Rightarrow (B \wedge C)}\Rightarrow_i}{z_2 : A \vdash_{\mathsf{DN}} (A \Rightarrow C) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow (B \wedge C))}\Rightarrow_i}{\vdash_{\mathsf{DN}} A \Rightarrow ((A \Rightarrow C) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow (B \wedge C))))}\Rightarrow_i$$

On voit là comment les étiquettes permettent de contrôler le nombre de décharges d'hypothèses. C'est en quelque sorte le pendant des règles structurelles dans le calcul des séquents.

La symétrie des règles n'apparaît plus dans l'introduction à gauche et à droite du signe $\vdash_{\mathsf{DN}}$, mais dans les règles d'*introduction* et d'*élimination* des connecteurs. Ainsi la règle $\Rightarrow_i$ signifie que si, en faisant l'hypothèse A on déduit B, alors sans faire l'hypothèse de A, on peut déduire $A \Rightarrow B$.

Réciproquement, si l'on a réussi à déduire A sous les hypothèses Γ, et que l'on a une preuve de $A \Rightarrow B$, alors on peut construire directement une preuve de B. La règle $\Rightarrow_e$ est celle du *modus ponens.*

Remarque 3. Si l'on a une déduction D_1 de $\Gamma, x : A \vdash_{\mathsf{DN}} B$ et une déduction D_2 de $\Gamma \vdash_{\mathsf{DN}} A$, alors on peut construire la déduction :

$$\dfrac{\dfrac{\vdots\, D_2}{\Gamma \vdash_{\mathsf{DN}} A} \quad \dfrac{\dfrac{\vdots\, D_1}{\Gamma, x : A \vdash_{\mathsf{DN}} B}}{\Gamma \vdash_{\mathsf{DN}} A \Rightarrow B}\Rightarrow_i}{\Gamma \vdash_{\mathsf{DN}} B}\Rightarrow_e$$

Mais n'est-il pas possible d'obtenir directement B en « utilisant» D_2 dans D_1 ? Toutes les feuilles de D_1 (les axiomes) sont de la forme $\Gamma, x_1 : A_1, \ldots x_n : A_n, x : A \vdash_{\mathsf{DN}} A_i$ et $\Gamma, x_1 : A_1, \ldots, x_n : A_n, x : A \vdash_{\mathsf{DN}} A$ (pour tout i, $x_i \neq x$).

Maintenant, rajoutons à chacun des séquents de la dérivation D_2 les hypothèses $x_1 : A_1$, ..., $x_n : A_n$ (en renommant éventuellement les variables de D_2 afin qu'aucune ne soit égale à un x_i). On a alors une dérivation D_2' de $\Gamma, x_1 : A_1, \ldots, x_n : A_n \vdash_{\mathsf{DN}} A$ que l'on peut substituer à chacune des feuilles $\Gamma, x_1 : A_1, \ldots x_n : A_n, x : A \vdash_{\mathsf{DN}} A$ de D_1 et dans le reste de la dérivation, $x : A$ n'est pas utilisé (car $x : A$ apparaît en conclusion de D_1), on peut donc supprimer $x : A$ de tous les séquents pour avoir une preuve de B sans avoir eu besoin de l'hypothèse A. On le note $D_1[D_2/x]$.

Ce processus est appelé *étape de réduction* ou de *normalisation.* De même, si l'on a :

$$\dfrac{\dfrac{\begin{matrix}\vdots D_1\\ \Gamma \vdash_{\mathsf{DN}} A\end{matrix} \qquad \begin{matrix}\vdots D_2\\ \Gamma \vdash_{\mathsf{DN}} B\end{matrix}}{\Gamma \vdash_{\mathsf{DN}} A \wedge B}\wedge_i}{\Gamma \vdash_{\mathsf{DN}} A}\wedge_e^1$$

peut-on obtenir directement une preuve de A avec D_1. Et similairement de

$$\dfrac{\dfrac{\begin{matrix}\vdots D_1\\ \Gamma \vdash_{\mathsf{DN}} A\end{matrix} \qquad \begin{matrix}\vdots D_2\\ \Gamma \vdash_{\mathsf{DN}} B\end{matrix}}{\vdash_{\mathsf{DN}} A \wedge B}\wedge_i}{\vdash_{\mathsf{DN}} B}\wedge_e^2$$

peut-on obtenir directement une preuve de A avec D_2.

Intuitivement, on observe :

- une occurrence d'hypothèse A est représentée par x, variable élément de A ;
- si une déduction se finit par $\wedge_i$ à partir de deux autres déductions représentées par u et v, alors elle se représente par $\langle u, v\rangle$;
- si une déduction se finit par $\wedge_e^1$ à partir de la déduction représentée par t, alors elle se représente par $\pi^1 t$. De même, si une déduction se finit par $\wedge_e^2$ à partir de la déduction représentée par t, alors elle se représente par $\pi^2 t$;
- si une déduction se finit par $\Rightarrow_i$ à partir d'une déduction représentée par u et ayant utilisé une hypothèse A étiquetée par x, u dépend de x et la déduction est représentée par $\lambda x.u$;
- si une déduction se finit par $\Rightarrow_e$ à partir des déductions $A \Rightarrow B$ et A représentées par u et v, u est une fonction de A dans B et v est un élément de A, la déduction se représente par $u\,v$.

Le tableau 3.3 donne la correspondance exacte. Du point de vue de la formation des termes (on retrouve les règles de formations des termes du λ-calcul typé de la section 3.1.1), la conclusion d'un séquent prouve le bon typage du terme. Cette correspondance s'appelle l'*isomorphisme de Curry-Howard.*

Proposition 42 (Isomorphisme de Curry-Howard). *On a les propositions suivantes :*

1. *Si* $x_1 : A_1, \ldots x_n : A_n \vdash_{DN'} t : B$ *alors* $x_1 : A_1, \ldots x_n : A_n \vdash_{DN} B$ *;*
2. *si* $x_1 : A_1, \ldots x_n : A_n \vdash_{DN} B$ *alors il existe un* λ*-terme* t *tel que* $x_1 : A_1, \ldots x_n : A_n \vdash_{DN'} t : B$.

La preuve se fait par induction sur les dérivations (voir par exemple [SU98]).

Mais le point essentiel est que la normalisation des preuves correspond exactement à la normalisation des λ-termes.

3.2.2 Logique intuitionniste

Le chapitre 1 mentionne déjà la logique intuitionniste comme un fragment du calcul des séquents n'admettant qu'une seule formule à droite du signe $\vdash_{\mathsf{LK}}$ (comme conclusion). En fait ce système permet de démontrer exactement les mêmes formules que la déduction naturelle.

$$\frac{}{\Gamma, x : A \vdash_{\mathsf{DN}'} x : A}\ \mathsf{Axiome}$$

$$\frac{\Gamma, x : A \vdash_{\mathsf{DN}'} t : B}{\Gamma \vdash_{\mathsf{DN}'} \lambda x^A.t : A \to B} \to_i \qquad \frac{\Gamma \vdash_{\mathsf{DN}'} t : A \to B \quad \Gamma \vdash_{\mathsf{DN}'} u : A}{\Gamma \vdash_{\mathsf{DN}'} tu : B} \to_e$$

$$\frac{\Gamma \vdash_{\mathsf{DN}'} t : A \quad \Gamma \vdash_{\mathsf{DN}'} u : B}{\Gamma \vdash_{\mathsf{DN}'} \langle t, u\rangle : A \wedge B} \wedge_i \qquad \frac{\Gamma \vdash_{\mathsf{DN}'} t : A \wedge B}{\Gamma \vdash_{\mathsf{DN}'} \pi^1 t : A} \wedge_e^1 \quad \frac{\Gamma \vdash_{\mathsf{DN}'} t : A \wedge B}{\Gamma \vdash_{\mathsf{DN}'} \pi^2 t : B} \wedge_e^2$$

TAB. 3.3 – Isomorphisme de Curry-Howard

Autrement dit, si l'on considère le calcul des séquents classique, modifié de façon à tenir compte de la contrainte de n'avoir qu'une seule formule conclusion, on obtient un fragment de la logique classique, dit *logique intuitionniste* et :

Proposition 43. *Un séquent $\Gamma \vdash_{IL} A$ est démontrable en logique intuitionniste si et seulement si il est racine d'une déduction $\Gamma \vdash_{DN} A$.*

La preuve se fait par induction sur les dérivations (voir [SU98]). Naturellement, l'isomorphisme de Curry-Howard peut donc s'exprimer en étiquetant ces règles également. On obtient celles du tableau 3.4

$$\frac{}{x : A \vdash_{\mathsf{IL}} x : A}\ \mathsf{Axiome} \qquad \frac{\Gamma \vdash_{\mathsf{IL}} t : A \quad \Gamma', x : A \vdash_{\mathsf{IL}} u : B}{\Gamma, \Gamma' \vdash_{\mathsf{IL}} u[t/x] : B}\ \mathsf{Cut}$$

$$\frac{\Gamma, x : A, y : B, \Gamma' \vdash_{\mathsf{IL}} t : C}{\Gamma, y : B, x : A, \Gamma' \vdash_{\mathsf{IL}} t : C}\ \mathsf{Ex} \qquad \frac{\Gamma, x : A, x : A \vdash_{\mathsf{IL}} t : C}{\Gamma, x : A \vdash_{\mathsf{IL}} t : C}\ \mathsf{C} \qquad \frac{\Gamma \vdash_{\mathsf{IL}} t : B}{\Gamma, x : A \vdash_{\mathsf{IL}} t : B}\ \mathsf{W}$$

$$\frac{\Gamma, x : A, y : B \vdash_{\mathsf{IL}} t : C}{\Gamma, u : A \wedge B \vdash_{\mathsf{IL}} t[\pi^1 u/x, \pi^2 u/y] : C}\ \mathsf{L}\wedge^1 \qquad \frac{\Gamma \vdash_{\mathsf{IL}} t : A \quad \Gamma' \vdash_{\mathsf{IL}} u : B}{\Gamma, \Gamma' \vdash_{\mathsf{IL}} \langle u, b\rangle : A \wedge B}\ \mathsf{R}\wedge$$

$$\frac{\Gamma \vdash_{\mathsf{IL}} t : A \quad \Gamma', y : B \vdash_{\mathsf{IL}} u : C}{\Gamma, \Gamma', z : A \to B \vdash_{\mathsf{IL}} u[z\,t/y] : C}\ \mathsf{L}\to \qquad \frac{\Gamma, x : A \vdash_{\mathsf{IL}} t : B}{\Gamma \vdash_{\mathsf{IL}} \lambda x.t : A \to B}\ \mathsf{R}\to$$

TAB. 3.4 – Isomorphisme de Curry-Howard et logique intuitionniste

Bien entendu, se pose alors naturellement la question de savoir ce que devient cette correspondance dans le cas de la logique linéaire où, là aussi, on peut contraindre les séquents à n'avoir qu'une seule formule conclusion. C'est l'objet de la section suivante.

3.2.3 Réseaux de preuve intuitionnistes

La première étape, comme au chapitre 1, consiste à supprimer les règles de contraction et d'affaiblissement. Une dérivation ne peut plus alors identifier deux termes par une contraction, en particulier deux variables. Dans toute démonstration, chaque règle Axiome différencie donc les variables introduites (autre nom, index, etc.). Les λ-termes obtenus appartiennent à une classe particulière des

λ-termes. Si l'on ne prend que le fragment implicatif, ce sont les λ-termes *linéaires* (Λ_l), c'est-à-dire que chaque abstraction lie une et une seule variable. Pour le fragment avec l'implication et la conjonction, tout λ-terme t (de Λ_l^+) obtenu par l'isomorphisme est tel que pour tout sous-terme s de t, s *compte pour un* dans t (par définition, un sous-terme s de t compte pour un dans t si s est clos, s'il n'y a qu'une seule occurrence de s dans t, ou s'il y a un sous-terme u de t tel que s compte pour un dans u et $\pi^1 u$ et $\pi^2 u$ comptent chacun pour un dans t). Cela permet d'accepter des termes comme $(\pi^1 x)(\pi^2 x)$. Cette définition, ainsi qu'une preuve de la correspondance entre preuves de la logique linéaire intuitionniste (réseaux) et classe de λ-termes apparaît dans [Roo91].

Avant de pouvoir exprimer la correspondance, puisque nous avons adopté une présentation unilatère du calcul des séquents linéaires (adaptée aux réseaux), donnons le moyen de reconnaître les séquents intuitionnistes (qui ne peuvent plus être définis comme ayant une seule formule à droite).

Polarités

Lorsque le calcul est exprimé sous forme unilatère, ou avec les réseaux, un autre artifice doit permettre de reconnaître l'unique conclusion qui caractérise les séquents intuitionnistes de la logique linéaire (**ILL**). Cet artifice apparaît sous la forme des polarités dans différents travaux [Roo91, Reg92, Lam94, Ret96a].

Le principe est le suivant : le réseau doit rendre compte de la forme $A_1, \ldots, A_n \vdash A$ du séquent. Pour ceci, on *polarise* les formules du séquent.

Moyennant les lois de De Morgan et la décomposition de $A \multimap B$ en $A^\perp \parr B$, le fragment intuitionniste dans la représentation unilatère (et par la suite dans les réseaux) s'obtient en ajoutant les contraintes de polarisation :

- si $\alpha \in \mathcal{P}$, α est polarisée positivement et $\alpha^\perp$ est polarisée négativement ;
- $F = G \parr H$ est polarisée suivant la tableau 3.5 en fonction de la polarisation de G et de H (la polarisation n'est pas définie pour les cases vides) ;
- $F = G \otimes H$ est polarisée suivant la tableau 3.5 en fonction de la polarisation de G et de H (la polarisation n'est pas définie pour les cases vides).

Pour une conclusion positive, on parle d'*entrée* (*input*) et de *sortie* (*output*) pour une conclusion négative.

$\parr$	$+$	$-$
$+$		$+$
$-$	$+$	$-$

$\otimes$	$+$	$-$
$+$	$+$	$-$
$-$	$-$	

TAB. 3.5 – Définition des polarisations des formules

Définition 35. Une preuve du calcul des séquents est intuitionniste si les formules sont bien polarisées et qu'il a une et une seule conclusion positive.

Un préréseau est *intuitionniste* si ses formules sont bien polarisées et qu'il a une et une seule conclusion positive.

Un réseau intuitionniste est un réseau qui est un préréseau intuitionniste.

La correspondance entre preuve du calcul des séquents linéaires et λ-termes est donnée au tableau 3.6. Outre la caractérisation donnée par [Roo91], nous remarquons que pour tout λ-terme t tel qu'il existe s sous-terme de t et $\pi^1 s$ (resp $\pi^2 s$) sous-terme de t, alors $\pi^2 s$ (resp. $\pi^1 s$) est également sous-terme de t (en effet, en l'absence de la règle d'affaiblissement, dans la règle $\parr$, x et y ont bien chacun une occurrence dans t).

Notation. Lorsque l'on souhaitera préciser la polarité d'une formule, on ajoutera en exposant $+$ ou $-$.

$$\frac{}{\vdash_{\text{ILL}} x : A, x : A^{\perp}}\ \text{Axiome} \qquad \frac{\vdash_{\text{ILL}} \Gamma, t : A \quad \vdash_{\text{ILL}} \Delta, x : A^{\perp}, u : B^{+}}{\vdash_{\text{ILL}} \Gamma, \Delta, u[t/x] : B^{+}}\ \text{Cut}$$

$$\frac{\vdash_{\text{ILL}} \Gamma, t : A^{+} \quad \vdash_{\text{ILL}} \Delta, u : B^{+}}{\vdash_{\text{ILL}} \Gamma, \Delta, \langle u, b\rangle : A \otimes B}\ \otimes_1 \qquad \frac{\vdash_{\text{ILL}} \Gamma, x : A^{-}, y : B^{-}, t : C^{+}}{\vdash_{\text{ILL}} \Gamma, u : A \parr B, t[\pi^1 u/x, \pi^2 u/y] : C^{+}}\ \parr_1$$

$$\frac{\vdash_{\text{ILL}} \Gamma, t : A^{+} \quad \vdash_{\text{ILL}} \Delta, y : B^{-}, u : C^{+}}{\vdash_{\text{ILL}} \Gamma, \Delta, z : A \otimes B, u[z\ t/y] : C^{+}}\ \otimes_2 \qquad \frac{\vdash_{\text{ILL}} \Gamma, x : A^{-}, t : B^{+}}{\vdash_{\text{ILL}} \Gamma, \lambda x.t : A \parr B}\ \parr_2$$

TAB. 3.6 – Isomorphisme de Curry-Howard et logique intuitionniste

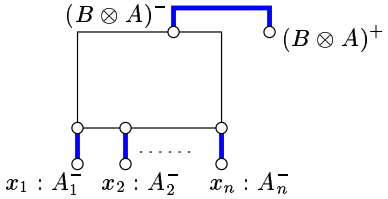
Encodage des λ-termes

Maintenant que l'on a exactement les réseaux correspondant à des preuves de la logique linéaire intuitionniste, donnons la correspondance entre réseaux et λ-termes. La définition que nous donnons ici ne demande pas d'unification, et étend la présentation de [dGR96] au produit. Attention, maintenant les formules sont données dans le fragment implicatif et conjonctif. C'est-à-dire que les formules sont toutes de la forme $A \multimap B$ ou $A \otimes B$ (en plus des variables propositionnelles). Pour retrouver la formule équivalente dans $\mathcal{F}_{\text{LL}}$, il suffit, pour les formules négatives, de passer à la négation.

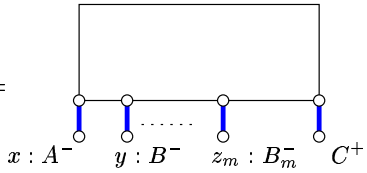
Tout λ-terme de type A avec les variables libres x_i de types respectifs A_i correspond à un réseau dont l'unique conclusion positive est de type A et dont les conclusions négatives de types A_i sont étiquetées par les variables x_i. De plus, si t ne s'écrit pas $\lambda x.u$ ou $\langle u, v\rangle$, l'unique conclusion positive est la conclusion d'un lien axiome. Soit $t \in \Lambda_l^+$. Voici comment construire le réseau $\pi_\lambda(t)$ correspondant à t.

Si t a comme sous-terme $\pi^1 s$ (respectivement $\pi^2 s$) Il a aussi comme sous-terme $\pi^2 s$ (respectivement $\pi^1 s$). Soit alors t' tel que $t = t'[\pi^1 s/x, \pi^2 s/y]$ (x et y variable libres de t') :

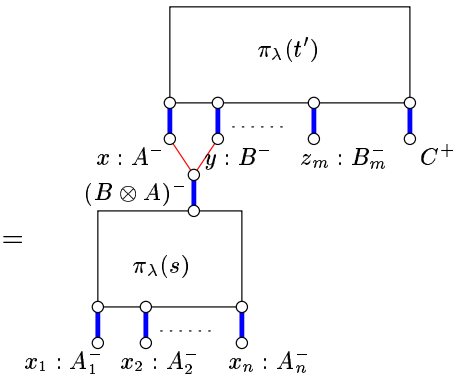
– si $s \neq \langle u, v\rangle$, alors $\pi_\lambda(s) =$



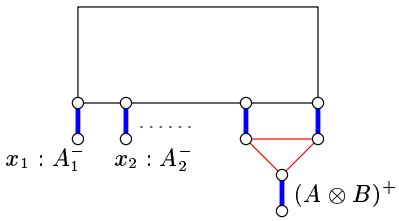
et $\pi_\lambda(t') =$



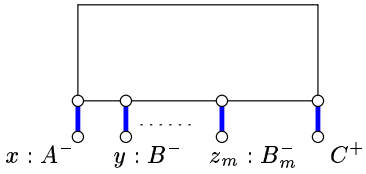
et alors $\pi_\lambda(t'[\pi^1 s/x, \pi^2 s/y]) =$



– si $s = \langle u, v \rangle$, alors $\pi_\lambda(s) =$

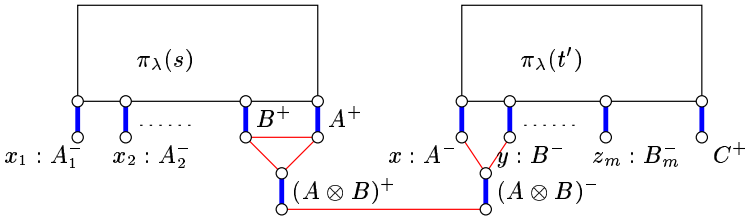


et $\pi_\lambda(t') =$

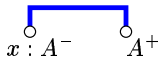


et alors

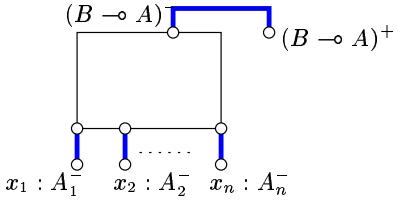
$\pi_\lambda(t'[\pi^1 s/x, \pi^2 s/y]) =$



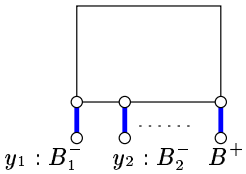
Si $t = x$ alors $\pi_\lambda(t) =$



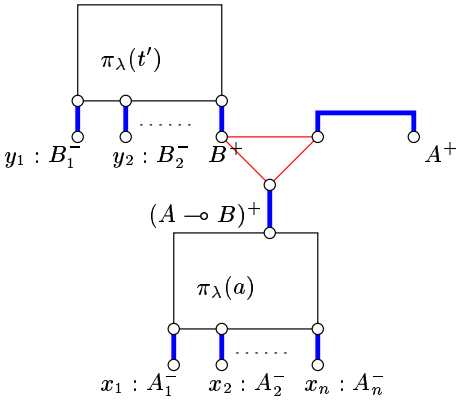
Si $t = at'$ avec $a \neq \lambda x.a'$ alors $\pi_\lambda(a) =$



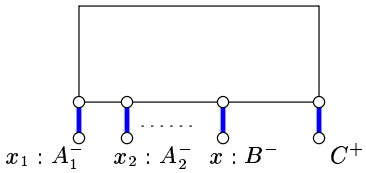
et $\pi_\lambda(t') =$



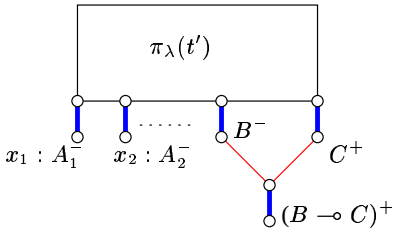
et alors $\pi_\lambda(at') =$



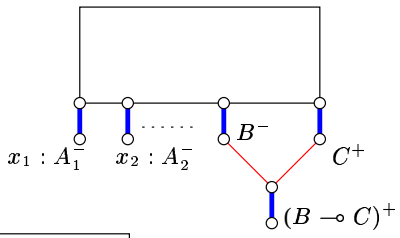
Si $t = \lambda x.t'$ avec $\pi_\lambda(t') =$



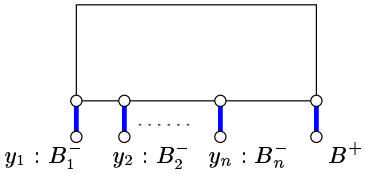
alors $\pi_\lambda(\lambda x.t') =$



Si $t = (\lambda x.a)t'$ avec $\pi_\lambda(\lambda x.a) =$

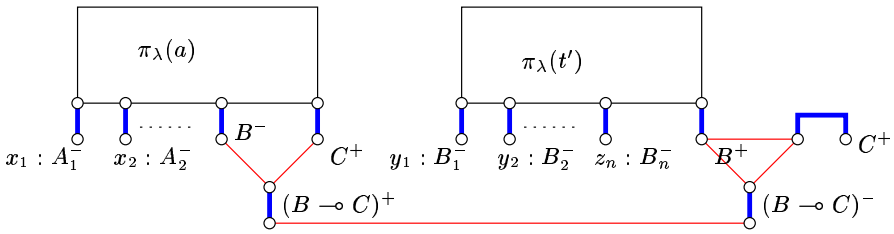


et $\pi_\lambda(t') =$

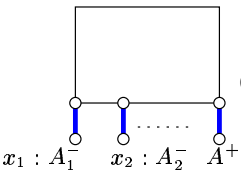


alors

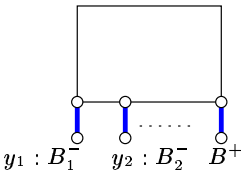
$\pi_\lambda((\lambda x.a)t') =$

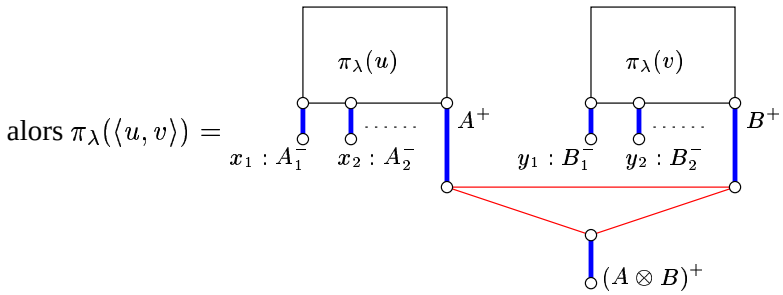


Si $t = \langle u, v \rangle$ avec $\pi_\lambda(u) =$



et $\pi_\lambda(v) =$





Exemple 4. Codons le λ-terme $\lambda x.(v(\pi^1(ux))(\pi^2(ux))$. On a $t = t'[\pi^1 s/y, \pi^2 s/z]$ où $s = ux$ et $t' = \lambda x.(vy)z$. D'où le codage de la figure 3.1.

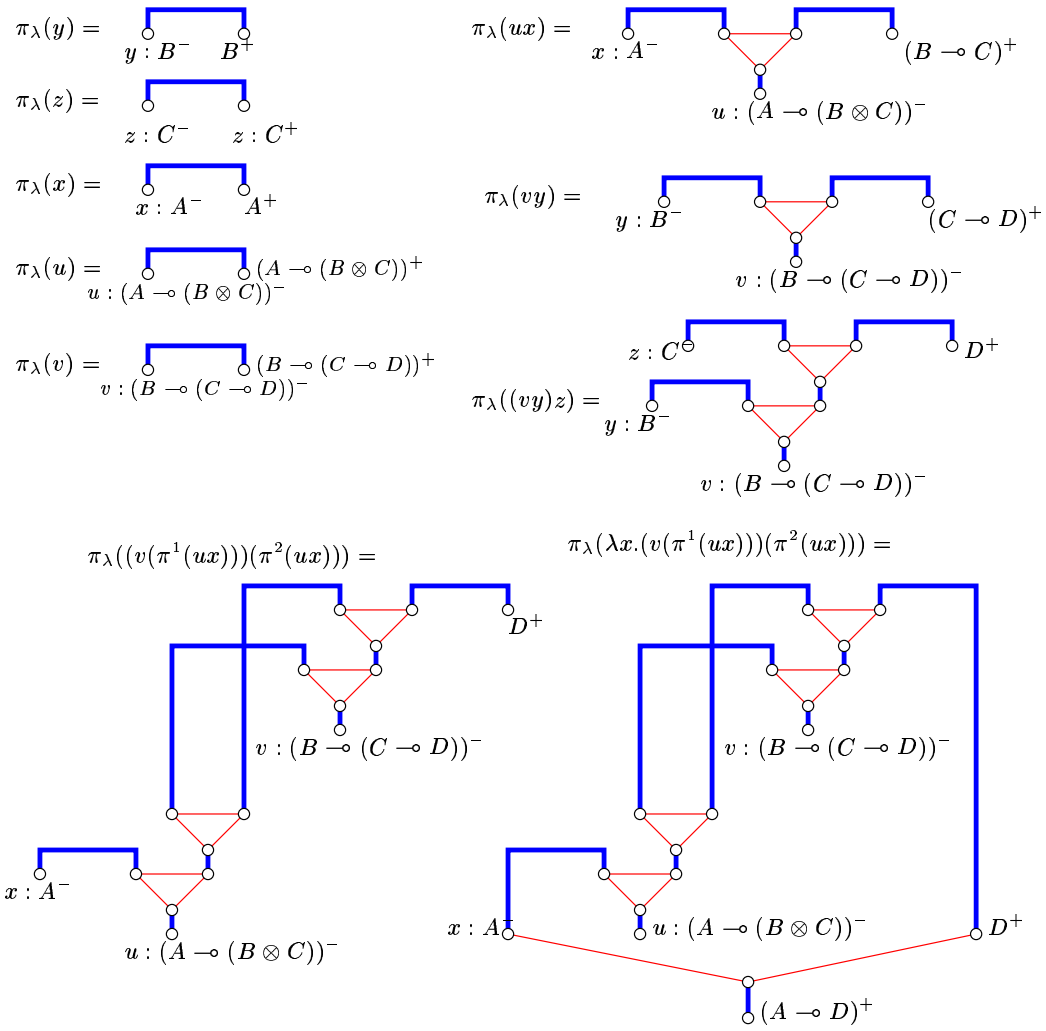


FIG. 3.1 – Codage de $\lambda x.(v(\pi^1(ux)))(\pi^2(ux))$

Lecture des réseaux

La lecture d'un réseau, pour obtenir le λ-terme qu'il code, se fait en étiquetant (comme dans [Roo91]) chaque lien d'un réseau par des λ-termes comme dans le tableau 3.7, et en unifiant les

termes étiquetant les conclusions d'un même lien axiome ou les prémisses d'un même lien cut (et comme les prémisses positives sont toujours étiquetées par une variable, on a en fait des valuations). [Roo91] montre que dans le cas d'un réseau, l'unification réussit, et le λ-terme associé au réseau est celui étiquetant l'unique conclusion positive après unification. La figure 3.2, reprenant le réseau de la figure 3.1 illustre cette approche (modulo l'α-conversion des variables libres).

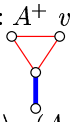
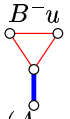
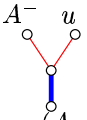
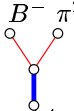
Après l'étiquetage des sommets, on a le problème défini par :

$$\begin{aligned} v_1 &= v_4 \\ u_4 &= \pi^2(tu_2) \\ u_3 &= \pi^1(tu_2) \\ u_2 &= u_1 \\ v_4 &= (t_2u_3)u_4 \end{aligned}$$

On a donc :

$$\begin{aligned} \lambda u_1.v_1 &= \lambda u_1.v_4 \\ &= \lambda u_1.(t_2u_3)u_4 \\ &= \lambda u_1.(t_2(\pi^1(tu_2)))(\pi^2(tu_2)) \\ &= \lambda u_1.(t_2(\pi^1(tu_1)))(\pi^2(tu_1)) \end{aligned}$$

ce qui est bien le résultat attendu au renommage des variables près.

Prémisses	A^+ et B^+	A^+ et B^-	A^- et B^+	A^- et B^-
Graphes	$u : A^+$ $v : B^+$  $\langle u, v\rangle : (A \otimes B)^+$	$tu : B^-$ $u : A^+$  $t : (A \multimap B)^-$	$v : A^-$ $u : B^+$  $\lambda u.v : (A \multimap B)^+$	$\pi^1 t : B^-$ $\pi^2 t : A^-$  $t : (A \otimes B)^-$
Nouvelles variables	u et v	u	u et v	

TAB. 3.7 – Les liens étiquetés par des λ-termes

3.3 Non-linéarité

On peut également considérer le fragment multiplicatif intuitionniste de la logique linéaire avec les exponentiels. Au niveau des règles du calcul des séquents, cela aujoute celles du tableau 3.8, que l'on peut utiliser pour les réseaux, comme l'a montré la section 2.4.

$$\frac{\vdash_{\text{ILL}} (!A_1)^-, \ldots, (!A_n)^-, A^+}{\vdash_{\text{ILL}} (!A_1)^-, \ldots, (!A_n)^-, (!A)^+}\,! \qquad \frac{\vdash_{\text{ILL}} \Gamma, A^-, B^+}{\vdash_{\text{ILL}} \Gamma, (!A)^-, B^+}\,?$$

$$\frac{\vdash_{\text{ILL}} \Gamma, C^+}{\vdash_{\text{ILL}} \Gamma, (!A)^-, C^+}\,\mathsf{W} \qquad \frac{\vdash_{\text{ILL}} \Gamma, (!A)^-, (!A)^-, C^+}{\vdash_{\text{ILL}} \Gamma, (!A)^-, C^+}\,\mathsf{C}$$

TAB. 3.8 – Règles exponentielles intuitionnistes

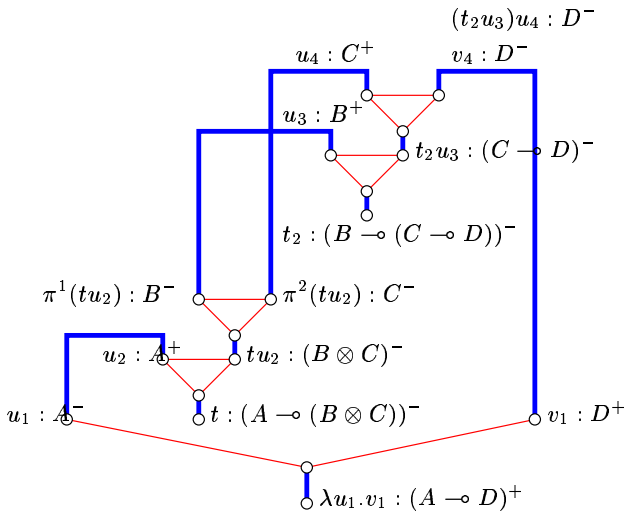
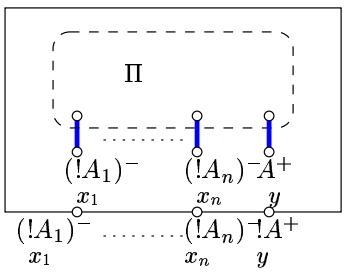
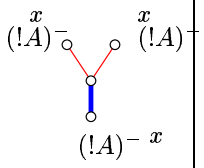
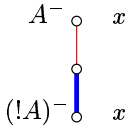


FIG. 3.2 – Exemple d'étiquetage d'un réseau par des λ-termes

Comme le montre le chapitre 6, la méthode adoptée pour la génération ne repose que sur des réseaux de preuve, et pas des λ-termes, lorsque la sémantique en particulier n'est pas exprimée par des λ-termes mais des réseaux. Il montre en particulier comment elle s'adapte pour tenir partiellement[11] compte des exponentiels. Sans chercher à donner une correspondance de Curry-Howard pour le fragment MEILL que nous ne connaissons pas, précisons toutefois que le fragment implicatif (juste avec $\multimap$, ? et !) permet lui aussi de coder les λ-termes (sans produit).

Ce codage correspond en fait à celui de la logique intuitionniste dans la logique linéaire intuitionniste, ou $A \to B$ est traduit par $!A^* \multimap B^*$ (où $A*$ est la traduction dans le fragment implicatif de la logique linéaire intuitionniste avec exponentiels de la formule A de la logique intuitionniste). En particulier, si on ne s'autorise pas l'affaiblissement, [Reg92] a montré que les liens peuvent s'étiqueter comme l'indique le tableau 3.9.

Prémisses	$?A^-, \ldots, ?A_n^-, A+$	$?A^-, ?A^-$	A^-
Graphes			

TAB. 3.9 – Étiquetage des liens exponentiels

Ainsi le réseau de la figure 3.3 code-t-il le λ-terme $\lambda P.\lambda Q.\forall(\lambda x.(Px) \Rightarrow (Qx))$, c'est-à-dire la forme sémantique habituellement associée à *tout*. Remarquons les types pour les constantes :

– $\forall : (!e) \multimap t) \multimap t$

– $\Rightarrow : t \multimap (t \multimap t)$

[11] Si celle-ci n'apparaissent pas dans l'unique conclusion positive.

L'introduction de $!e$ est rendue nécessaire par l'utilisation, par deux fois dans le terme, de la variable x dont il est le type, afin de pouvoir *contracter* $(!e)^-, (!e)^-$ en $(!e)^-$ (voir figure 3.3). Sinon, le terme n'est pas codable en logique linéaire multiplicative.

$\forall(\lambda x.(Px) \Rightarrow (Qx))$

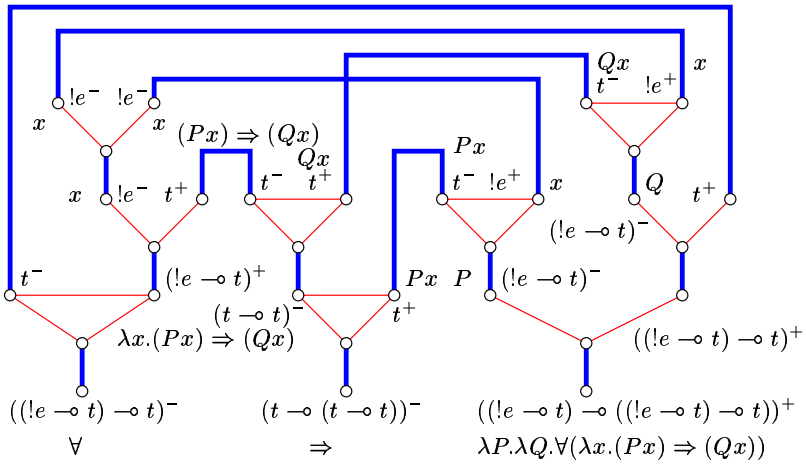


FIG. 3.3 – Codage de $\lambda P.\lambda Q.\forall(\lambda x.(Px) \Rightarrow (Qx))$

3.4 Conclusion

L'isomorphisme de Curry-Howard permet de faire le lien entre formule logique et λ-calcul. Son expression sous forme de déduction naturelle montre que l'association $A, A \rightarrow B$ correspond à l'application de λ-termes. Or, comme nous l'avons vu, si l'association est la concaténation des mots typés par A et $A \rightarrow B$, on retrouve l'expression de la correspondance règle à règle telle que la définit Montague.

En fait, il suffit de donner une transformation $\mathcal{H}$ entre les types syntaxiques et les types sémantiques qui conserve la preuve et l'association. Ainsi, d'une preuve sur les types syntaxiques A, $A \rightarrow B$, etc. et de sa transformée (par un homomorphisme $\mathcal{H}$ qui conserve la forme des termes) en une preuve sur les *types sémantiques* e, $e \rightarrow t$, etc. on peut extraire, en appliquant à cette dernière l'isomorphisme de Curry-Howard, le λ-terme qui traduit naturellement l'association A, $A \rightarrow B$, etc.

Le chapitre 5 exploite cette expression de la compositionalité comme interface entre la syntaxe et la sémantique lorsque la syntaxe se base sur les grammaires de types logiques et la sémantique sur celle de Montague.

Deuxième partie

Modélisation de phénomènes linguistiques et grammaires de types logiques

Chapitre 4

Grammaires catégorielles

En décrivant une certaine technique de modélisation de la langue, cette partie justifie les préliminaires théoriques sur la logique, la théorie de la démonstration et la sémantique que nous avons développés jusqu'à présent. Ils ont pu parfois paraître arides ou stériles, mais outre leur intérêt du simple point de vue du calcul (en tant que modèle), nous pensons que la familiarité acquise avec leur syntaxe et les modes de raisonnement associés permet de dégager plus facilement les éléments essentiels de la modélisation.

Nous donnons donc tout d'abord les aspects historiques (en se référant largement à [BHGS64, Cas88]), pour montrer à quelles nécessités ont répondu les divers modèles. Puis nous présentons le calcul de Lambek proprement dit, avec son pouvoir génératif et ses limitations. Naturellement, nous le considérons comme fragment de la logique linéaire, car cette dernière offre alors un cadre possible pour diverses extensions.

4.1 Catégories sémantiques

Jusqu'à présent, nous faisions référence aux catégories en évoquant la correction syntaxique (voir section 1.1.2). Pourtant, d'un point de vue historique, l'introduction des catégories se justifie sémantiquement. Selon [Tar90], repris par [Lam58], les principes de base de la théorie des catégories sémantiques reviennent à E. Husserl. Pour étudier la manière dont les expressions linguistiques de longueur et de complexité arbitraires se combinent, il considère qu'elles s'articulent autour d'éléments, tels que les phrases, les groupes nominaux, les prédicats, etc. ayant une signification propre (éléments *catégorématiques*), et autour d'éléments qui n'ont pas de signification propre (éléments *syncatégorématiques*) mais qui jouent toutefois un rôle important pour déterminer le sens global de l'expression, tels que les mots *et, ou, est,* etc. Les éléments syncatégorématiques se révèlent incomplets linguistiquement et doivent intégrer d'autres éléments du discours pour donner un tout signifiant. Par exemple la forme *cet* N *est* P peut s'instancier de différentes manières (*cet œillet est rose, cet éléphant est rose, cet entier est rose*), à condition que N et P soient respectivement des éléments catégorématiques nominaux et adjectivaux. Sans cela, les catégories ne répondent pas à celles demandées par *est* et l'unité de sens est perdue.

Le principe peut s'énoncer ainsi :

- toute expression linguistique doit appartenir à une catégorie sémantique ;
- toute expression signifiante résulte de l'intégration de ses parties et le mode d'intégration dépend des catégories syntaxiques auxquelles chaque partie appartient ;
- remplacer une partie d'expression signifiante par une partie appartenant à une catégorie sémantique différente enlève toujours tout sens à cette expression.

Ainsi l'exemple précédent, même si l'on a un *contresens* (c'est-à-dire dans ce cas l'absence d'un entier rose), il se distingue d'un *non-sens* (c'est-à-dire une expression composée d'éléments de catégories différentes de celles requises. Par exemple : *cet ou est éléphant*). Husserl considère cette grammaire purement logique comme un ensemble de règles analytiques communes à toutes les langues.

D'autre part, l'étude des paradoxes logiques tels celui de B. Russell (en considérant l'ensemble de tous les ensembles qui ne sont pas éléments d'eux-mêmes) a conduit à considérer des systèmes hiérarchisés de types. Dans le même esprit, S. Leśniewski a développé un système de catégories hiérarchisées (appelé grammaire des catégories sémantiques), dans lequel on trouve des *catégories fonctionnelles*, de degré croissant, construites sur des *catégories élémentaires*. Les catégories fonctionnelles, comme les catégories syncatégorématiques de Husserl supposent la saturation par d'autres catégories. Par exemple, pour une catégorie élémentaire t, on peut former la catégorie fonctionnelle $t \to t \to t$ la catégorie des expressions qui peuvent transformer deux expressions de type t en une expression de type t. Ainsi, dans le cas du calcul propositionnel, si l'on attribue aux propositions la catégorie t, on peut aussi attribuer aux connecteurs $\wedge$ et $\vee$ la catégorie $t \to t \to t$ qui transforme bien deux propositions en une nouvelle proposition.

[Ajd35] donne une notation algébrique pour la hiérarchie des catégories, se présentant sous forme de fraction. Cette écriture permet de vérifier très simplement si une expression appartient bien à une catégorie donnée. Elle se présente de la manière suivante :

- il y a des catégories élémentaires (par exemple t, s, n, etc.) ;
- les catégories fonctionnelles sont construites par $\frac{c_0}{c_1 \ldots c_n}$ où c_0 est la catégorie obtenue si l'expression se compose en plus des types (élémentaires ou pas) $c_1, \ldots, c_n$.

Pour le calcul propositionnel, on peut par exemple attribuer à $\wedge$ la catégorie $\frac{t}{tt}$. Et si p et q sont deux variables propositionnelles, et donc de catégorie t, l'expression $\wedge pq$ est aussi de catégorie t car $\frac{t}{tt}\, t\, t$ se simplifie en t.

Dans l'approche originale, pour passer de la notation infixée à la notation postfixée, cette dernière permettant la vérification de la bonne formation, Ajdukiewicz donne une condition supplémentaire sur les expressions qui doivent être « syntaxiquement connexes ». Nous ne décrirons pas cette condition, car [BH50] étend ce travail et donne des conditions plus simples et désormais connues dans la littérature sous le nom de *grammaires AB* (*AB grammars*), *grammaires catégorielles classiques* (*classical categorial grammars*), ou encore *grammaires catégorielles élémentaires* (*basic categorial grammars*).

4.2 Grammaires AB

4.2.1 Définition

Alors que [Ajd35] se préoccupait essentiellement des langages formels, en particulier la logique, [BH50] se consacre plus spécifiquement aux langues naturelles. Pour ceci, et pour tenir compte de l'ordre des mots qui devient une part très importante, il introduit désormais deux types de barres de fractions sensibles à l'ordre. Les catégories, ou *types* se définissent comme suit :

$$T ::= P | T\backslash T | T/T$$

où P est l'ensemble des types élémentaires (contenant d'une part *S* mais d'autres types arbitraires, par exemple *S*, *np*, etc.).

Définition 36. Soit un vocabulaire V. Une *grammaire AB* G se définit à l'aide d'un *lexique*, c'est-à-dire une fonction **Lex** qui associe un ensemble fini de types aux mots. (On a besoin d'un ensemble fini de types car un même mot peut avoir des règles de combinaisons, et donc des catégories, très

différentes. Par exemple *été* qui peut être le participe passé du verbe *être* ou le nom commun désignant la saison.)

Une expression est une suite de mots $m_1 \dots m_n$. Son type est u si pour tout $i \in [1, n]$, il existe $t_i \in \mathbf{Lex}(m_i)$ tel que $t_1 \dots t_n \to u$ ($t_1 \dots t_n$ se réduit en u) avec les règles de réduction suivantes :

$$\forall v, w \in T \qquad \begin{array}{ll} v\,(v\backslash w) \to w & (\backslash_e) \\ (v/w)\,w \to v & (/_e) \end{array}$$

Si pour tout $a \in V$, $|\mathbf{Lex}(a)| \leq k$, on dit que G est *k-valuée.* Si $k = 1$, on dit qu'elle est *rigide.*

Ces règles se lisent intuitivement ainsi : une expression e de type $A\backslash B$ attend une expression a de type A sur sa gauche pour donner une expression ae de type B. De même une expression e' de type A/B attend une expression b de type B sur sa droite pour donner une expression $e'b$ de type A[12].

Le langage engendré par la grammaire est l'ensemble des expressions de type S.

On voit qu'il y a eu glissement des catégories sémantiques vers les catégories syntaxiques. Néanmoins, comme nous le montre la section 3.1.3, les catégories sémantiques se redéfinissent en reprenant d'une part l'idée de fonction et d'argument pour saturer ces fonctions, et d'autre part en gardant un lien étroit avec leur catégorie syntaxique.

4.2.2 Quelques exemples de grammaire AB

Comme indiqué précédemment, il nous faut tout d'abord définir un lexique.

Soit le lexique $\mathbf{Lex}_1$ défini par le tableau 4.1.

$\mathbf{Lex}_1$	
a	$(S/B)/S$
b	B
ϵ	S

TAB. 4.1 – Définition de $\mathbf{Lex}_1$

Le langage $\mathcal{L}(\mathbf{Lex}_1)$ engendré par $\mathbf{Lex}_1$ est $a^n\epsilon b^n$. En effet, on a bien $\epsilon \in \mathcal{L}(\mathbf{Lex}_1)$ et pour $n \geq 1$, si $a^n\epsilon b^n \in \mathcal{L}(\mathbf{Lex}_1)$, alors le type de $a^n\epsilon b^n$ est S et avec l'arbre :

$$\dfrac{\dfrac{\begin{array}{cc} a & a^n\epsilon b^n \\ (S/B)/S & S \end{array}}{S/B}\,/_e \quad \begin{array}{c} b \\ B \end{array}}{S}\,/_e$$

$a^{n+1}\epsilon b^{n+1}$ est une expression de type S. Réciproquement, une expression e de $\mathcal{L}(\mathbf{Lex}_1)$ soit est ϵ soit commence nécessairement par a (dont seule la catégorie produit un S) et donc s'écrit $e = ae_1e_2$, avec e_1 de type S et e_2 de type B. Or seul b peut fournir un B. Donc $e = ae_1b$ et e_1 de type S. Par récurrence sur la longueur de l'expression, $\mathcal{L}(\mathbf{Lex}_1) = \{a^n\epsilon b^n | n \geq 0\}$.

Ce petit exemple permet de montrer que les langages engendrés par les grammaires catégorielles peuvent se comparer aux classes habituelles de langage (dans la hiérarchie de Chomsky) et de vérifier qu'ils sont non réguliers. La section 4.3 présente des résultats précis à ce sujet.

Donnons maintenant un exemple plus proche des langues naturelles, et utilisons $\mathbf{Lex}_2$, défini dans le tableau 4.2.

[12] Pour savoir de quel côté l'expression doit être complétée, on peut utiliser le moyen mnémotechnique suivant : orienter la flèche systématiquement vers le bas, comme ceci : $A \searrow B$ et $B \swarrow A$ et lire « A donne B », ou « A implique B ». Comme on le verra par la suite, d'un point de vue logique, on a effectivement une implication.

Lex$_2$	
Marie	np
livre	n
le	np/n
lit	$(np\backslash S)/np$

TAB. 4.2 – Définition de **Lex**$_2$

L'expression *Marie lit le livre* appartient-elle au langage ? Oui, car on peut construire l'arbre de dérivation suivant :

$$\dfrac{np \qquad \dfrac{(np\backslash S)/np \qquad \dfrac{np/n \quad n}{np}\,/_e}{np\backslash S}\,\backslash_e}{S}$$

montrant que l'expression $np\,(np\backslash S)/np\,np/n\,n$ se réduit bien à S.

Par rapport à l'exemple donné à la section 1.1.2, les règles tiennent cette fois bien compte de la non-commutativité, et il n'est pas possible de dériver S à partir de l'expression **Marie le lit livre*.

4.2.3 Limitations et extensions des grammaires catégorielles

Cherchons maintenant à étendre **Lex**$_2$ pour pouvoir analyser des expressions comme *le livre que Marie lit* (non pas comme une phrase, bien sûr, mais comme une expression de type *np*).

Quel type peut-on associer à *que* ? C'est un mot qui attend à sa droite une expression à qui il manque à droite un *np* pour être de type S, soit S/np, et à sa gauche une expression de type n pour redonner une expression de type n. Donc on peut typer *que* par $(n\backslash n)/(S/np)$ et considérer le lexique du tableau 4.3. (Bien entendu, la solution n'est pas en général unique et seule l'analyse de nombreux exemples permet de définir judicieusement le type d'une expression. C'est la technique d'apprentissage dont la section 4.4 parle plus longuement.)

Lex$_3$	
Marie	np
livre	n
le	np/n
lit	$(np\backslash S)/np$
que	$(n\backslash n)/(S/np)$

TAB. 4.3 – Définition de **Lex**$_3$

Cependant, l'expression *Marie lit* ne se réduit pas au type S/np. Il lui manque pourtant bien un *np* à droite pour se réduire au type S. On voudrait donc avoir

$$np\,(np\backslash S)/np \rightarrow S/np$$

Cela peut s'obtenir au moyen de deux règles de plus dans le système :

$$A/B\;B/C \rightarrow A/C \tag{4.1}$$

$$A \rightarrow B/(A\backslash B) \tag{4.2}$$

La règle (4.1) se comprend de la manière suivante : la combinaison $e_3 = e_1 e_2$ de deux expressions e_1 de type A/B et e_2 de type B/C se comporte comme une expression de type A/C. En effet, supposons que l'on ajoute c de type C à e_3, $e_1 e_2 c$ peut se réduire au type A. Donc $e_3 c$ peut se réduire au type A et e_3 au type A/C. On donne généralement à cette règle le nom de *composition* (on compose effectivement une fonction de C dans B avec une fonction de B dans A pour obtenir une fonction de C dans A).

La règle (4.2) dit qu'une expression $e_3 = e_1 e_2$ de type B avec e_1 de type A permet de considérer que e_2 attend une expression de type A à sa gauche pour donner un B ($A\backslash B$), et donc que e_1 peut aussi être considérée comme ayant un type permettant. à partir d'une expression de type $A\backslash B$ à sa droite, de donner le type B, soit $B/(A\backslash B)$. On appelle généralement cette règle *élévation de type*.

Ces deux règles ont leur pendant avec les concaténations dans l'autre sens :

$$A\backslash B \; B\backslash C \rightarrow A\backslash C \tag{4.3}$$

$$A \rightarrow (B/A)\backslash B \tag{4.4}$$

On a alors :

$$\dfrac{\dfrac{\textit{le}\quad \textit{livre}\quad \textit{que}\quad \textit{Marie}\quad \textit{lit}}{\dfrac{\textit{np/n}\quad \textit{n}}{\textit{np}}/_e \quad \dfrac{\textit{(n\backslash n)/(S/np)}\quad \dfrac{\dfrac{\textit{np}}{\textit{S/(np\backslash S)}}(4.2)\quad \textit{(np\backslash S)/np}}{\textit{S/np}}}{\textit{n\backslash n}}/_e}{\textit{n}}\backslash_e}{\textit{np}}/_e \; (4.1)$$

Pour pouvoir modéliser des phénomènes linguistiques un peu plus complexes, on s'aperçoit donc que les deux règles de base des grammaires catégorielles ne suffisent pas. Un certain nombre de propositions (telles les règles de composition, de division — règles de Geach — ou d'élévation de type [Gea72]) peut être ajouté. Nous ne développons pas plus ces règles car les systèmes logiques que nous considérons (voir chapitre 5) permettent de les obtenir directement.

4.3 Pouvoir génératif et grammaires hors-contexte

Considérant les grammaires catégorielles élémentaires, la question de la classe de langage qu'elles engendrent se pose naturellement. Cette section rappelle différentes équivalences avec les grammaires hors-contexte.

Nous prenons le temps de décrire plus précisément les grammaires hors contexte (en se basant essentiellement sur [HU79]) et leurs liens avec les grammaires catégorielles car, outre les résultats d'équivalence existant entre ces deux systèmes, les linguistes aussi bien que les informaticiens connaissent en général bien (et beaucoup mieux que les grammaires catégorielles) les grammaires hors-contexte.

Définition 37. Une *grammaire hors-contexte* $G = (V, T, P, S)$ est définie par :

- deux ensembles disjoints V et T de *variables* et de *terminaux* ;
- P un ensemble fini de *règles de production* de la forme $A \rightarrowtail \alpha$ où $A \in V$ et $\alpha \in (V \cup T)^*$;
- un symbole spécial S, le *symbole de départ.*

Définition 38. Soit la relation $\rightarrow$ définie sur $(V \cup T)^{*2}$ avec $\alpha A \gamma \rightarrow \alpha\beta\gamma$ si et seulement si $A \rightarrowtail \beta$ (on dit que $\alpha A\gamma$ se réécrit en $\alpha\beta\gamma$). On appelle $\twoheadrightarrow$ sa clôture réflexive et transitive.

Un arbre est un *arbre de dérivation* de A si :
- tout sommet est étiqueté par un symbole de $V \cup T \cup \{\epsilon\}$;
- la racine est étiquetée A ;
- si un sommet est intérieur, étiqueté par B, alors $B \in V$;
- si un sommet s est étiqueté par B et que les sommets $s_1, \ldots, s_n$ sont ses fils, ordonnés de gauche à droite, de labels respectifs $X_1 \cdots X_n$, alors $B \rightarrowtail X_1, \ldots, X_n \in P$;
- si un sommet est étiqueté ϵ, alors c'est une feuille et c'est l'unique fils de son père.

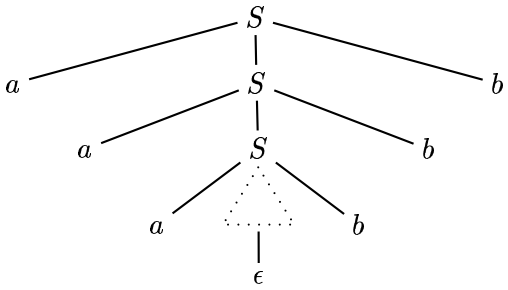
Définition 39. Soit G une grammaire hors-contexte. Le *langage* (hors-contexte) engendré par G est $\mathcal{L}(G) = \{w \in T^* | S \twoheadrightarrow w\}$.

Deux grammaires qui engendrent le même langage sont *faiblement équivalentes.* Si de plus les arbres de dérivations sont isomorphes, alors elles sont *fortement équivalentes.*

Exemple 5. La grammaire $G = (V, T, P, S)$ où :
- $V = \{S\}$;
- $T = \{a, b\}$;
- $P = \left\{ \begin{array}{ccc} S & \rightarrowtail & aSb \\ S & \rightarrowtail & \epsilon \end{array} \right\}$

engendre le langage $\mathcal{L}(G) = \{a^n b^n | n \geq 0\}$.

En effet, $S \rightarrow aSb \rightarrow a\,aSb\,b \rightarrow a\,a\,aSb\,b\,b \rightarrow \cdots \rightarrow a^n \epsilon b^n$. Avec l'arbre de dérivation



Théorème 44 (Forme normale de Chomsky). *Tout langage hors-contexte L sans ϵ ($\epsilon \notin L$) est engendré par une grammaire dont les règles de production sont de la forme $A \rightarrowtail BC$ ou $A \rightarrowtail a$ avec $(A, B, C, a) \in V^3 \times T$.*

Théorème 45 (Forme normale forte de Greibach). *Tout langage hors-contexte L sans ϵ est engendré par une grammaire dont les règles de production sont de la forme $A \rightarrowtail a\alpha$ ou $A \rightarrowtail a$ avec $(A, a, \alpha) \in V \times T \times (V^2)$.*

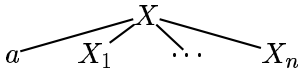
[BHGS64] établit les résultats suivants à propos des équivalences entre grammaires AB et grammaires hors-contextes :

Théorème 46. *Toute grammaire hors-contexte sans ϵ (le langage engendré est sans ϵ) sous forme normale de Greibach est fortement équivalente à une grammaire AB.*

Démonstration. Soit $G = (V, T, P, S)$ une telle grammaire hors contexte. On définit G' une grammaire catégorielle grâce au lexique **Lex** tel que $\forall a \in T$

$$\textbf{Lex}(a) = \{((\ldots((X/X_n)/X_{n-1})\ldots)/X_2)/X_1 | X \rightarrowtail aX_1 \cdots X_n \in P\}$$

Il suffit alors de constater que pour tout sous-arbre de réécriture



sous-arbre de l'arbre d'une réécriture dans G,

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{a}{((\dots((X/X_n)/X_{n-1})\dots)/X_2)/X_1} \quad X_1}{(((X/X_n)/X_{n-1}\cdots)/X_2}/_e \quad X_2}{(((X/X_n)/X_{n-1}\cdots)/X_3}}{\vdots}}{(X/X_n)/X_{n-1}} \quad X_{n-1}}{X/X_n}/_e \quad X_n}{X}/_e$$

est une réécriture correcte pour G'. □

Théorème 47. *Toute grammaire hors-contexte sans ϵ est faiblement équivalente à une grammaire AB contenant uniquement des types de la forme X, X/Y ou $(X/Y)/Z$.*

Démonstration. Conséquence directe des théorèmes 45 et 46. La démonstration de ce dernier donne les bons types si les règles sont en forme normale de Greibach. □

Théorème 48. *Toute grammaire AB G est fortement équivalente à une grammaire hors-contexte en forme normale de Chomsky.*

Démonstration. Définissons $G' = (V, T, P, S)$ telle que :

- T est l'ensemble des mots de G ;
- V est l'ensemble des sous-types des types du lexique (par exemple, si $(np\backslash S)/np)$ est un type du lexique, $(np\backslash S)/np)$, $np\backslash S$, np et S appartiennent à V.
- $P = \left\{ \begin{array}{c} \{X \rightarrowtail a | X \in \mathbf{Lex}(a)\} \\ \cup \\ \{X \rightarrowtail (X/Z)Z | X, Z, (X/Z) \in V\} \\ \cup \\ \{X \rightarrowtail Z(Z\backslash X) | X, Z, (Z\backslash X) \in V\} \end{array} \right.$

Les arbres de réécriture dans G' se déduisent alors directement des dérivations dans G. □

4.4 Apprentissage

Comme nous l'avons constaté, la modélisation à l'aide d'une grammaire catégorielle repose essentiellement sur la construction du lexique. La question se pose alors naturellement des moyens pour acquérir ce lexique. Au-delà des résultats généraux sur l'apprentissage de [Gol67], plus récemment certains travaux [Bus87, BP89, Kan98] ont apporté des perspectives nouvelles sur l'apprentissage des grammaires catégorielles, notamment en termes de complexité. Notons également l'approche très originale de [Adr92] qui veut considérer les classes de langages non plus traditionellement (dans la hiérarchie de Chomsky), mais suivant leur complexité (au sens de la théorie de l'information).

La présentation rapide des techniques les (désormais) plus classiques d'apprentissage faite dans cette section permettra à la fois :

- de donner un aperçu de ce qui nous semble un travail indispensable à l'utilisation des grammaires catégorielles en vraie grandeur ;
- de justifier l'affectation des types dans les exemples tout au long des chapitres ;

– surtout, de montrer que certains des problèmes auxquels nous sommes confrontés pour la génération se retrouvent en apprentissage quand celui-ci se base également sur la sémantique [Tel98a]. L'approche algorithmique que nous développons pour la génération devrait donc pouvoir se transposer pour apprentissage.

Bien que ces techniques fonctionnent pour des grammaires k-valuées, nous limitons cependant nos exemples au cas des grammaires rigides.

Le problème de l'apprentissage (apprentissage à la limite défini par [Gol67]) consiste à définir une fonction **Apprends** qui associe à un ensemble fini E d'exemples positifs (c'est-à-dire qui sont des expressions correctes du langage) une grammaire G telle que :

– G engendre tous les exemples de E ;
– pour une grammaire donnée G_0 et une suite (e_i) d'expressions engendrées par G_0, si l'on définit $E_i = \{e_k | k \leq i\}$, alors il existe n_0 tel que pour tout $n \geq n_0$

$$\textbf{Apprends}(E_n) = \textbf{Apprends}(E_{n_0}) = G_\infty \text{ et } \mathcal{L}(G_0) = \mathcal{L}(G_\infty)$$

Une classe de langage peut *s'apprendre* dans ce modèle s'il existe une fonction **Apprends** qui permet d'apprendre une grammaire G à partir d'une suite infinie d'exemples de tout langage L de cette classe. Dans ce cadre, les premiers résultats, négatifs, indiquent qu'aucune des classes de langage de la hiérarchie de Chomsky ne peut s'apprendre [Gol67]. Toutefois, des classes de langages non triviales peutvent être apprises [Ang80], en particulier les grammaires contextuelles dont le nombre de règles est borné [Shi91] et, dans le cas qui nous intéresse, les grammaires catégorielles k-valuées [Kan96].

4.4.1 Exemples structurés

Par rapport aux grammaires contextuelles ou hors-contexte dont il faut inférer les règles, dans le cas des grammaires catégorielles, ce sont en fait les lexiques qu'il faut découvrir, puisque les règles sont fixées. D'autre part, pour des raisons d'efficacité, les expressions ne sont pas une simple suite de mots mais des expressions *structurées*, c'est-à-dire pourvues de leur arbre de dérivation. Pour les grammaires rigides par exemple, l'algorithme décrit ci-dessous calcule l'affectation des types dans un temps linéaire en fonction du nombre d'exemples, si ceux-ci sont structurés. S'ils ne sont pas structurés, la recherche de structure introduit un élément exponentiel : si n_i est le nombre de mots de l'exemple i, il y a $2^{\sum(n_i-1)}$ structures possibles [Nic99].

La figure 4.1 donne les deux exemples positifs sur lesquels nos décrivons l'algorithme RG [BP89, Kan98] dont le principe réside dans le typage qu'il propose des mots de l'expression (et qui définit donc le lexique). On suppose que les deux phrases sont correctes, donc de type S. Puis on type avec des variables, depuis la racine jusqu'aux feuilles, les nœuds des arbres.

Lorsque le résultat d'une règle $\backslash_e$ (respectivement $/_e$) est y, cela signifie que le type du fils gauche doit être x (respectivement y/x) alors que celui du fils droit doit être $x\backslash y$ (respectivement x). Cela permet de typer chacun des arbres, pour obtenir ceux des figures 4.2. Autrement dit, le lexique doit vérifier le typage suivant :

le	: x_1/y_1,x_2/y_2
chat	: y_1,y_2
sourit	: $x_1\backslash S$, z_2
malicieusement	: $z_2\backslash z_2$

D'autre part, on veut un type unique pour chaque mot. Celui-ci doit vérifier chacun des types issus des arbres typés. Cela se réalise en considérant le typage comme un problème d'unification de variables. Ainsi, on cherche σ substitution vérifiant :

– $\sigma(S) = S$;

– $\sigma(A\backslash B) = \sigma(A)\backslash\sigma(B)$;
– $\sigma(A/B) = \sigma(A)/\sigma(B)$

Une telle substitution, appliquée au lexique, ne change pas le langage engendré. On dit qu'elle *unifie* un ensemble T de types si $\forall X, Y \in T, \sigma(X) = \sigma(Y)$.

L'algorithme RG, à partir des types induits par les arbres typés des exemples positifs, cherche donc à unifier tous les types associés à un item lexical (dans le cas des grammaires rigides, l'unificateur le plus général existe toujours). En particulier ici, on recherche σ telle que $\sigma(x_1/y_1) = \sigma(x_2/y_2)$, $\sigma(y_1) = \sigma(y_2)$ et $\sigma(x_1\backslash S) = \sigma(z_2)$.

On obtient

$$\sigma(x_1) = \sigma(x_2) = X \quad \sigma(y_1) = \sigma(y_2) = Y \quad \sigma(z_2) = X\backslash S$$

soit le lexique de la table 4.4[13].

Cet algorithme converge, d'après [Kan98]. Pour une discussion et une implémentation efficace de l'algorithme, voir [Nic99].

Lex$_4$	
le	X/Y
chat	Y
sourit	$X\backslash S$
malicieusement	$(X\backslash S)\backslash(X\backslash S)$

TAB. 4.4 – Définition de **Lex**$_3$

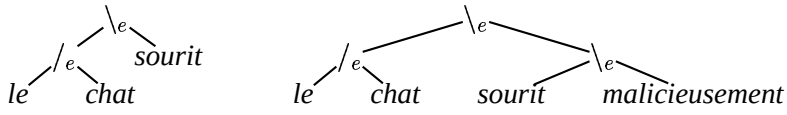


FIG. 4.1 – Exemples positifs pour l'apprentissage

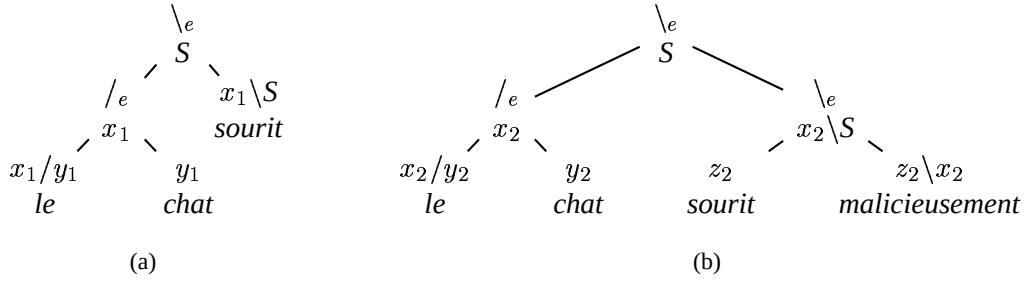


FIG. 4.2 – Arbres typés

[13]Rien ne nous empêche de remplacer X par *np*, Y et *n* pour retrouver le typage habituel.

$\mathbf{Lex}_5$		
Arsène	X	**a**
rit	$X\backslash S$	**rit**
pleure	$X\backslash S$	**pleure**

TAB. 4.5 – Lexique appris à l'aide de la sémantique

4.4.2 Exemples syntaxiques et sémantiques

Nous avons vu dans la section précédente que les exemples fournissaient, en plus de la suite de mots, l'arbre de dérivation. Pour éviter de devoir fournir les arbres de dérivations, [Tel98a] propose que les exemples soient constitués de simples suites de mots, mais avec leur représentation sémantique. On ne fournit donc plus un ensemble d'expressions bien formées, mais un ensemble de paires constituées d'une expression et de sa représentation sémantique.

Ainsi, prenons comme ensemble d'exemples $E = \{(e_1, s_1), (e_2, s_2)\}$ où

$$(e_1, s_1) = (\textit{Arsène rit}, \textbf{rit a}) \text{ et } (e_2, s_2) = (\textit{Arsène pleure}, \textbf{pleure a})$$

Bien que la méthode proposée par [Tel98b, Tel98a] ne requière que la compositionalité sémantique, nous utilisons ici la sémantique de Montague en considérant les expressions sémantiques comme des λ-termes.

Dans une première étape, l'algorithme fait plusieurs hypothèses sur les structures possibles des exemples. On a ici les quatre possibilités de la figure 4.3.

Si l'on ne précise pas les structures correctes, l'algorithme RG donne les quatre lexiques possibles :

Arsène	X
pleure	$X\backslash S$
rit	$X\backslash S$

Arsène	S/X
pleure	X
rit	$(S/X)\backslash S$

Arsène	S/X
pleure	$(S/X)\backslash S$
rit	X

Arsène	S/X
pleure	X
rit	X

Comment donc utiliser la forme sémantique pour les discriminer ? Tout d'abord, les deux possibilités pour *Arsène rit* des figures 4.3(a) et 4.3(b) permettent les deux hypothèses suivantes :

Arsène	X	**a**
rit	$X\backslash S$	**rit**

ou

Arsène	S/X	**rit**
rit	X	**a**

c'est-à-dire que l'on cherche α et β qui vérifient $\beta\alpha =$ **rit a** ou $\alpha\beta =$ **rit a**. Le problème d'unification des catégories syntaxiques s'augmente d'un problème d'unification de λ-termes.

Supposons que l'on ait $(\textit{Arsène} : S/X, \textbf{rit})$. Cet élément lexical doit aussi permettre de calculer dans au moins l'un des deux autres exemples γ**rit** = **pleure a** ou **rit**γ = **pleure a**. Or ceci n'est pas possible car **rit** est une constante qui doit apparaître dans les deux termes. On voit donc comment la sémantique permet de discriminer les possibilités, et on sait qu'alors on doit avoir $(\textit{Arsène} : X, \textbf{a})$, et donc aussi $(\textit{rit} : X\backslash S, \textbf{rit})$. Et de même que précédemment, les conditions sur le deuxième exemple donnent γ**a** = **pleure a** comme seule solution possible. Donc le lexique finalement construit est celui du tableau 4.5.

Cette méthode présente un certain nombre d'avantages. D'une part les exemples ne nécessitent plus de structure mais ils doivent s'accompagner de leur sémantique, d'autre part, pas plus qu'il n'est nécessaire d'indiquer des types syntaxiques de base aux entrées lexicales, il n'est nécessaire d'indiquer des valeurs sémantiques de base. Seule la commodité nous a amenés à donner des noms comme **pleure** ou **rit** aux constantes sémantiques.

[Tel98a] ajoute que d'un point de vue cognitif, la pertinence de cette conception de l'apprentissage rejoint des opinons psycholinguistiques, en évoquant [Pin95] : « Knowing a language is knowing

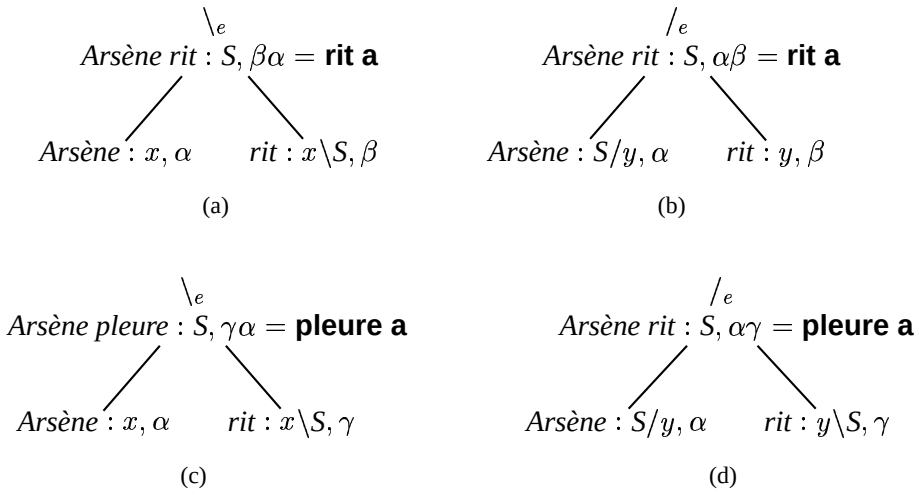


FIG. 4.3 – Hypothèses construites à partir des exemples

how to translate mentalese into strings of words and vice-versa », mais également la nécessité de la répétition syntaxico-sémantique dans l'apprentissage [Qui99, Pin95]. Ainsi, si l'on commente la course de quelqu'un par *John runs*, l'auditeur ne peut pas deviner quel est le prédicat, et quel est l'argument. Cette inférence ne devient possible que si un nouvel exemple est fourni à l'auditeur, tel que *Mary runs,* ou bien *John sleeps.* Ce que l'on retrouve dans l'exemple traité ci-dessus.

Par contre cette approche est encore très incomplète. [Tel98a] relève la complexité de l'algorithme envisagé, au moins exponentielle, en soulignant toutefois que la mesure de complexité intéressante correspond à celle relative au nombre de mots nouveaux dans chaque nouvel exemple. On retrouve là une préoccupation de [Adr92].

Également, la procédure d'unification de λ-termes amène des difficultés, car elle devient très rapidement indécidable [Hue75] (dès le deuxième ordre). Néanmoins, cette approche nous semble intéressante et envisage sous d'autres angles des problèmes que nous avons eu à traiter dans le cas de la génération. En particulier, nous verrons que même si nous faisons de l'unification de λ-termes, nous pouvons montrer que les problèmes posés (moins généraux que l'unification) restent décidables d'une part, et d'autre part qu'une approche par les réseaux de preuve amène une certaine efficacité. La question se pose alors de savoir dans quelle mesure ces résultats se transposent à ce schéma d'apprentissage.

Chapitre 5

Calcul de Lambek

Le chapitre précédent a présenté les grammaires catégorielles avec ses deux règles de réécriture $\backslash_e$ et $/_e$:

$$v\,(v\backslash w) \to w \quad (\backslash_e)$$
$$(v/w)\,w \to v \quad (/_e)$$

En prenant désormais un point de vue plus logique qu'arithmétique, et en considérant le $/$ et le $\backslash$ comme des *implications* (gauche et droite), une signification autre de ces règles apparaît. En effet, supposons que d'une expression Γ on puisse dire qu'elle est de type v ($\Gamma \vdash v$), et qu'une autre expression Δ soit de type $v\backslash w$ ($\Delta \vdash v\backslash w$). La règle $\backslash_e$ nous assure qu'alors l'expression Γ, Δ est de type w ($\Gamma, \Delta \vdash w$).

Ceci évoque très précisément la règle d'élimination de l'implication en déduction naturelle décrite au chapitre 3 :

$$\frac{\Gamma \vdash_{\mathsf{DN}} A \quad \Delta \vdash_{\mathsf{DN}} A \to B}{\Gamma, \Delta \vdash_{\mathsf{DN}} B} \to_e$$

c'est-à-dire la règle du *Modus Ponens.*

Mais la déduction naturelle propose la contrepartie de cette règle : la règle d'*introduction* de l'implication :

$$\frac{\Gamma, A \vdash_{\mathsf{DN}} B}{\Gamma \vdash_{\mathsf{DN}} A \to B} \to_i$$

Or cette règle manque aux grammaires AB. Cela signifie, pour reprendre un exemple du chapitre 4, *Marie lit*, qu'on ne peut pas déduire de la correction de la phrase *Marie lit un livre* que *Marie lit* est de type S/np (en considérant que *un livre* est de type *np*), ce qui conduit à introduire des règles supplémentaires telles que $a \to b/(a\backslash b)$. Qu'en est-il dans un cadre comme celui de la déduction naturelle où l'on peut faire des hypothèses (présence d'un *np* après *Marie lit* qui permet d'obtenir une phrase correcte — on obtient une preuve de *S*) ?

Ce chapitre répond en présentant tout d'abord les règles $/_e$ et $\backslash_e$ plongées dans un système logique similaire à celui de la déduction naturelle, avec les propriétés de décidabilité et d'élimination des coupures qui en découlent. Puis il montre par divers exemples comment ce nouveau calcul étend celui des grammaires AB, résoud élégamment certains problèmes et se combine naturellement à la sémantique de Montague. Cela nous permettra d'évoquer pour la première fois les problèmes de génération avec ce type de grammaire. Enfin, nous montrerons des limitations de ce calcul pour la modélisation et des tentatives pour les dépasser.

5.1 Règles d'inférences et interprétation

5.1.1 Déduction naturelle et calcul des séquents

Les règles que nous présentons sont celles de [Lam58], avec le tableau 5.1 pour une présentation en déduction naturelle sous forme de séquents, et le tableau 5.2 pour une présentation en calcul des séquents.

Suivant les propriétés que nous voulons démontrer, ou les transformations dans d'autres systèmes, nous utiliserons l'une ou l'autre de ces représentations.

Les types sont définis par

$$\mathcal{T} ::= \mathcal{P} | \mathcal{T} \backslash \mathcal{T} | \mathcal{T} / \mathcal{T} | \mathcal{T} \bullet \mathcal{T}$$

Par rapport aux grammaires catégorielles, il y a en plus une conjonction non commutative $\bullet$.

$$A \vdash_{\text{L}_{\text{DN}}} A$$

$$\frac{\Gamma \vdash_{\text{L}_{\text{DN}}} A \quad \Delta \vdash_{\text{L}_{\text{DN}}} A\backslash B}{\Gamma, \Delta \vdash_{\text{L}_{\text{DN}}} B} \backslash_e \qquad \frac{A, \Gamma \vdash_{\text{L}_{\text{DN}}} B}{\Gamma \vdash_{\text{L}_{\text{DN}}} A\backslash B} \backslash_i$$

$$\frac{\Gamma \vdash_{\text{L}_{\text{DN}}} B/A \quad \Delta \vdash_{\text{L}_{\text{DN}}} A}{\Gamma, \Delta \vdash_{\text{L}_{\text{DN}}} B} /_e \qquad \frac{\Gamma, A \vdash_{\text{L}_{\text{DN}}} B}{\Gamma \vdash_{\text{L}_{\text{DN}}} B/A} /_i$$

$$\frac{\Gamma \vdash_{\text{L}_{\text{DN}}} A \bullet B \quad \Delta, A, B, \Delta' \vdash_{\text{L}_{\text{DN}}} C}{\Delta, \Gamma, \Delta' \vdash_{\text{L}_{\text{DN}}} C} \bullet_e \qquad \frac{\Gamma \vdash_{\text{L}_{\text{DN}}} A \quad \Delta \vdash_{\text{L}_{\text{DN}}} B}{\Gamma, \Delta \vdash_{\text{L}_{\text{DN}}} A \bullet B} \bullet_i$$

TAB. 5.1 – Règles du calcul de Lambek sous forme de déduction naturelle

$$\frac{}{A \vdash_{\text{L}} A}\ \text{Axiom} \qquad \frac{\Gamma \vdash_{\text{L}} A \quad \Delta, A, \Delta' \vdash_{\text{L}} C}{\Delta, \Gamma, \Delta' \vdash_{\text{L}} C}\ \text{Cut}$$

$$\frac{\Gamma, B, \Gamma' \vdash_{\text{L}} C \quad \Delta \vdash_{\text{L}} A}{\Gamma, B/A, \Delta, \Gamma' \vdash_{\text{L}} C}\ \mathbf{L}_/ \qquad \frac{\Gamma, A \vdash_{\text{L}} B}{\Gamma \vdash_{\text{L}} B/A}\ \mathbf{R}_/$$

$$\frac{\Delta \vdash_{\text{L}} A \quad \Gamma, B, \Gamma' \vdash_{\text{L}} C}{\Gamma, \Delta, A\backslash B, \Gamma' \vdash_{\text{L}} C}\ \mathbf{L}_\backslash \qquad \frac{A, \Gamma \vdash_{\text{L}} B}{\Gamma \vdash_{\text{L}} A\backslash B}\ \mathbf{R}_\backslash$$

$$\frac{\Gamma, A, B, \Gamma' \vdash_{\text{L}} C}{\Gamma, A \bullet B, \Gamma' \vdash_{\text{L}} C}\ \mathbf{L}_\bullet \qquad \frac{\Gamma \vdash_{\text{L}} A \quad \Delta \vdash_{\text{L}} B}{\Gamma, \Delta \vdash_{\text{L}} A \bullet B}\ \mathbf{R}_\bullet$$

TAB. 5.2 – Règles du calcul de Lambek sous forme de séquents

En fait, de ce système présenté par [Lam58], plusieurs autres peuvent découler. Parmi les plus courants, citons la version sans le produit $\bullet$ (que l'on pourra parfois considérer dans la suite). Les

autres se différencient sur les règles structurelles. On a vu que ce système est non commutatif, mais la virgule « , » est associative. On peut également considérer un système non associatif et non commutatif comme [Lam61].

Une propriété importante du calcul de Lambek associatif (*structural completeness* [Bus88]) indique que s'il existe une dérivation d'un type de base à partir d'une suite de types dont les sous-suites sont munies d'un parenthésage (par exemple *(Marie (lit (le livre)))*), tout autre parenthésage permet une dérivation du même type (par exemple *((Marie lit)(le livre))*) :

$$\dfrac{\begin{array}{c}np \vdash_{\text{L}_{\text{DN}}} np \qquad \dfrac{\dfrac{(np\backslash S)/np \vdash_{\text{L}_{\text{DN}}} (np\backslash S)/np \qquad \dfrac{np/n \vdash_{\text{L}_{\text{DN}}} np/n \quad n \vdash_{\text{L}_{\text{DN}}} n}{(np/n, n) \vdash_{\text{L}_{\text{DN}}} np}\,/_e}{((np\backslash S)/np, (np/n, n)) \vdash_{\text{L}_{\text{DN}}} np\backslash S}\,\backslash_e}{(np, \quad ((np\backslash S)/np, \quad (np/n, \quad n))) \quad \vdash_{\text{L}_{\text{DN}}} S} \\ \textit{(Marie} \quad \textit{(lit} \quad \textit{(le} \quad \textit{livre)))}\end{array}}{}\tag{5.1}$$

$$\dfrac{\begin{array}{c}\dfrac{np \vdash_{\text{L}_{\text{DN}}} np \qquad \dfrac{(np\backslash S)/np \vdash_{\text{L}_{\text{DN}}} (np\backslash S)/np \quad np \vdash_{\text{L}_{\text{DN}}} np}{(np\backslash S)/np, np \vdash_{\text{L}_{\text{DN}}} np\backslash S}\,\backslash_e}{np, (np\backslash S)/np, np \vdash_{\text{L}_{\text{DN}}} S}\,/_i \qquad \dfrac{np/n \vdash_{\text{L}_{\text{DN}}} np/n \quad n \vdash_{\text{L}_{\text{DN}}} n}{(np/n, n) \vdash_{\text{L}_{\text{DN}}} np}\,/_e}{(np, (np\backslash S)/np) \vdash_{\text{L}_{\text{DN}}} S/np}\,/_e \\ ((np, \quad (np\backslash S)/np), \quad (np/n, \quad n)) \quad \vdash_{\text{L}_{\text{DN}}} S \\ \textit{((Marie} \quad \textit{lit)} \quad \textit{(le} \quad \textit{livre))}\end{array}}{}\tag{5.2}$$

La notion usuelle de constituant se trouve changée, car désormais toute décomposition est possible. Cela permet de traiter par exemple la coordination entre non-constituants, comme le montre la section 5.4.2. De même, la dérivation de chacun de ses regroupements peut se justifier : (5.1) dans le cas syntaxique, et (5.2) dans le cas prosodique, car les notions de constituants ne coïncident pas forcément dans les deux cas [Lad92].

L'examen de la dérivation (5.2) montre un usage implicite de l'associativité pour utiliser la règle $/_i$: on passe en fait de $np, ((np\backslash S)/np, np)$ à $(np, (np\backslash S)/np), np$. Considérer un calcul non associatif [Lam61] permet de retrouver la notion habituelle de constituant. Pour permettre de retrouver certains phénomènes, comme celui des constituants prosodiques, on peut assouplir localement la non-associativité, et donner des postulats structurels [Moo96]. Plus généralement, cette approche et les phénomènes linguistiques qu'elle permet de prendre en compte, est présentée à la section 5.5.1.

Nous rappelons que les deux systèmes (calcul des séquents et déduction naturelle), s'ils utilisent les même symboles pour leur syntaxe ont des propriétés différentes, comme indiqué au chapitre 3. En particulier la procédure de normalisation est différente dans les deux systèmes. Pour pouvoir utiliser l'un ou l'autre des deux systèmes, il faut montrer qu'ils sont équivalents. Mais l'équivalence dont on parle ici est celle de la prouvabilité. Cela signifie que l'on veut montrer le théorème suivant :

Théorème 49. *Un séquent* $\Gamma \vdash_L A$ *est prouvable si et seulement si il est racine d'une déduction* $\Gamma \vdash_{\text{L}_{\text{DN}}} A$.

Ce théorème n'est qu'une variante de la correspondance entre calcul des séquents et déduction naturelle telle que nous l'avons décrite au chapitre 3.

5.1.2 Le séquent vide

Nous avons pour l'instant considéré les règles des tableaux 5.1 et 5.2 sans restriction, et nous avons vu que cela permettait d'étendre le langage engendré (pour un même lexique, bien sûr) par

rapport aux grammaires catégorielles. Par contre, puisqu'il s'agit maintenant d'une logique, quel sens donner à une tautologie ? Supposons qu'une formule A soit une tautologie. On a donc $\vdash_{\text{L}_{\text{DN}}} A$. Cela signifie que la chaîne vide ϵ est de type syntaxique A pour toute tautologie A. Et alors ϵ peut être utilisé pour fournir un A dès qu'il y en a besoin.

Par exemple, on a : $\dfrac{n \vdash_{\text{L}_{\text{DN}}} n}{\vdash_{\text{L}_{\text{DN}}} n/n}\,/_i$. Or n/n est justement le type des adjectifs (qui attendent un nom commun pour donner un nom commun). De plus, un adverbe comme *très* modifie les adjectifs, et a comme type $(n/n)/(n/n)$. Dès lors le lexique de la table 5.3 permet de dériver le type *np* aussi bien depuis l'expression *le très gros gâteau* que depuis l'expression **le très gâteau.* En effet, on a bien une preuve de $np/n, (n/n)/(n/n), n \vdash_{\text{L}_{\text{DN}}} np$ avec :

$$\dfrac{\dfrac{np/n \vdash_{\text{L}_{\text{DN}}} np/n \qquad \dfrac{\dfrac{(n/n)/(n/n) \vdash_{\text{L}_{\text{DN}}} (n/n)/(n/n) \qquad \dfrac{n \vdash_{\text{L}_{\text{DN}}} n}{\vdash_{\text{L}_{\text{DN}}} n/n}\,/_i}{(n/n)/(n/n) \vdash_{\text{L}_{\text{DN}}} n/n}\,/_e \qquad n \vdash_{\text{L}_{\text{DN}}} n}{(n/n)/(n/n), n \vdash_{\text{L}_{\text{DN}}} n}\,/_e}{\underset{\textit{le}}{np/n}, \quad \underset{\textit{très}}{(n/n)/(n/n)}, \quad \underset{\textit{gâteau}}{n} \quad \vdash_{\text{L}_{\text{DN}}} np}\,\_e$$

Cela explique que la définition des règles d'inférences du calcul de Lambek s'accompagne d'une restriction : la suite de formules à gauche du signe $\vdash$ ne doit pas être vide (dans $\Gamma \vdash \Delta$, $\Gamma \neq \emptyset$). Cela permet de définir les calculs $\vdash_{\text{L}^*_{\text{DN}}}$ et $\vdash_{\text{L}^*}$ à partir de $\vdash_{\text{L}_{\text{DN}}}$ et $\vdash_{\text{L}}$.

Lex$_6$	
très	$(N/N)/(N/N)$
gâteau	n
le	np/n
gros	N/N

TAB. 5.3 – Définition de **Lex**$_6$

5.1.3 Relations avec la logique linéaire

La règle X de la logique linéaire (voir le tableau 1.4) peut se décomposer en deux règles :

$$\dfrac{\vdash \Gamma, A}{\vdash A, \Gamma}\,\text{cX} \qquad \dfrac{\vdash \Gamma, A, B}{\vdash \Gamma, B, A}\,\text{tX}$$

Le nouveau système **c-MLL**, construit à partir des règles de **MLL**, à l'exception de la règle X remplacée par cX, est celui défini par [Yet90], celui de la logique linéaire cyclique (seule la permutation circulaire est autorisée) pour laquelle on considère également le fragment intuitionniste **ICLL**.

On a alors la proposition suivante :

Proposition 50. *$A_1, \ldots, A_n \vdash_{\text{L}} B$ si et seulement si $\vdash_{\text{IcLL}} A_n^{\perp}, \ldots A_1^{\perp}, B$.*

[Ret96a] en donne une preuve.

Donc :

- le calcul de Lambek est équivalent à la logique linéaire (multiplicative) intuitionniste cyclique ;
- le calcul de Lambek, avec la règle de permutation, est équivalent à la logique linéaire (multiplicative).

C'est vrai également pour chacun de ces calculs considérés sans la séquence vide, c'est-à-dire pour L^* et IcLL^*.

5.2 Réseaux de preuve pour le calcul de Lambek

Puisque le calcul de Lambek se révèle être un fragment non commutatif de la logique linéaire multiplicative intuitionniste, sa présentation sous forme de réseaux apparaît naturellement.

La section 3.2.3 présente les réseaux de preuve pour la logique linéaire multiplicative intuitionniste à l'aide des polarités. Ces réseaux peuvent s'utiliser tels quels si l'on ajoute le moyen de contrôler la non-commutativité. Celle-ci peut s'exprimer également comme une propriété à ajouter au critère de correction.

La présentation des réseaux sous forme de R&B-graphes permet de la formuler de deux manières :

Réseaux sans coupures Un réseau de preuve de **MLL** est un réseau de preuve pour **cMLL** si et seulement si les liens axiomes ne se croisent pas [Roo91, Ret96a]. On dit également que la suite s des formules reliées par des liens axiomes est bien parenthésée (c'est-à-dire s est vide ou bien $s = As_1A^\perp s_2$ avec s_1 et s_2 bien parenthésées). C'est ce critère que nous retiendrons car la partie syntaxique des grammaires de types logiques se fera sans coupure. Bien entendu, l'ordre des conclusions des axiomes dépend de l'ordre des conclusions du réseau.

Réseaux avec coupure Pour exprimer la cyclicité avec des coupures (ce que ne permet pas le bon parenthésage des axiomes), [AM98] propose un critère basé sur l'orientation des liens.

On a donc d'après la proposition 50 : $A_1, \ldots, A_n \vdash_{\mathsf{L}^*} B$ si et seulement s'il existe un réseau de conclusions *ordonnées* $A_n^\perp, \ldots, A_1^\perp, B$ dont les liens axiomes ne se coupent pas.

Proposition 51. *Soit* Π *un réseau du calcul de Lambek. Les énoncés suivant sont équivalents :*
- *tout sous-préréseau*[14] *de* Π *a au moins deux conclusions ;*
- *tout sous-réseau de* Π *a au moins deux conclusions ;*
- *toute séquentialisation de* Π *ne contient que des séquents ayant au moins deux conclusions.*

[Ret96a] donne la preuve de cette proposition. Elle permet de caractériser les réseaux correspondant au calcul de Lambek sans séquence vide.

Exemple 6. Prenons le lexique suivant :

Othello	np
Desdémone	np
aime	$(np\backslash S)/np$

Pour prouver que l'expression *Othello aime Desdémone* est bien une phrase, il suffit de trouver un réseau de conclusions $np^\perp, ((np\backslash S)/np)^\perp, np^\perp, S$ dont les liens axiomes ne se coupent pas. La figure 5.1 montre que c'est possible.

Remarque. Bien que l'ordre des mots sur la figure soit *Desdémone aime Othello,* c'est bien l'expression *Othello aime Desdémone* qui a été analysée. En effet, le passage par négation de la gauche à la droite du séquent (pour obtenir un séquent unilatère) change l'ordre des formules, ce qui se traduit sur les réseaux par une lecture de droite à gauche de l'expression analysée.

Exemple 7. Avec le même lexique que précédemment, l'analyse de *Othello Desdémone aime* échoue car tous les réseaux possibles de conclusions $((np\backslash S)/np)^\perp, np^\perp, np^\perp$ et S ont les liens axiomes qui se croisent (voir figures 5.2).

[14] Un *sous-préréseau* (resp. *sous-réseau*) est un sous-graphe qui est aussi un préréseau (resp. un réseau).

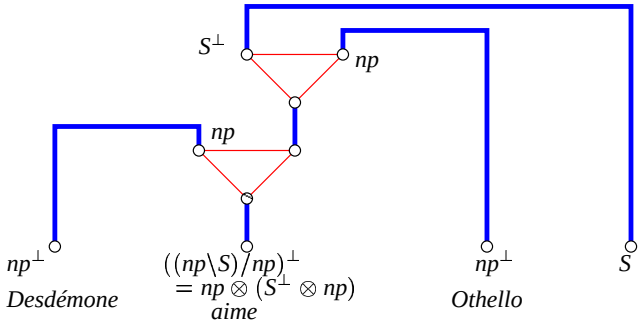


FIG. 5.1 – Analyse de *Othello aime Desdémone*

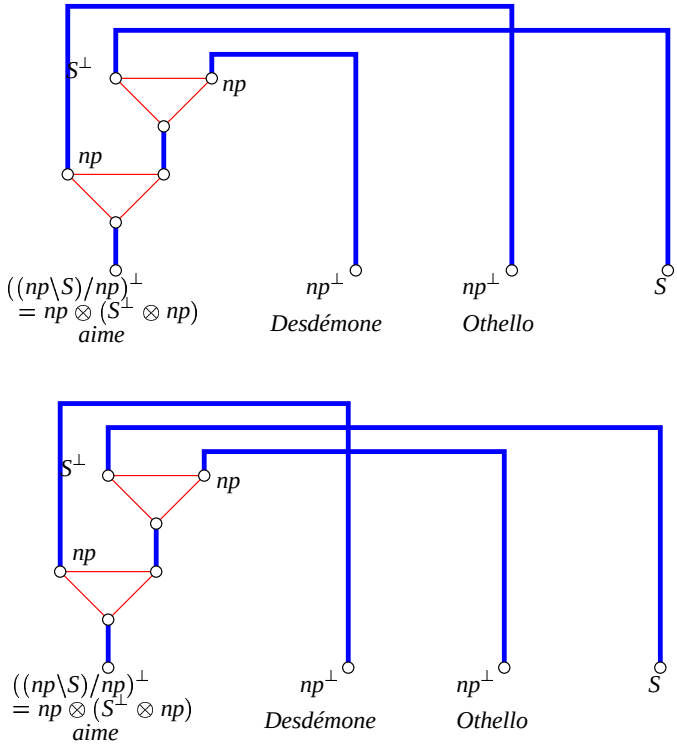


FIG. 5.2 – Les deux réseaux possibles pour *Othello Desdémone aime* ont des liens axiomes qui se croisent

Cependant, nous obtenons là des analyses pour le calcul de Lambek *avec* la séquence vide. [Ret96a] caractérise les réseaux correspondant au calcul de Lambek sans la séquence vide : ce sont les réseaux π tels que tout sous-préréseau (c'est-à-dire tout sous-graphe qui est un préréseau) de π contient au moins deux conclusions.

En reprenant l'exemple de la section 5.1.2 et le lexique du tableau 5.3, on voit que le seul réseau qui permettrait d'analyser *le très gâteau* comme un *np* est celui de la figure 5.3. Or celui-ci contient le sous-réseau de conclusion unique $n \parr n^{\perp}$. Il ne correspond donc pas à une preuve du calcul de Lambek sans séquence vide et doit être rejeté.

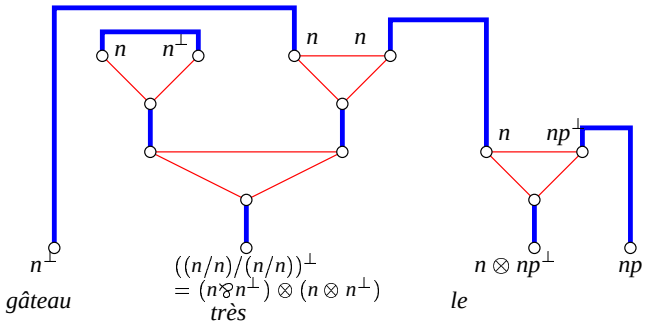


FIG. 5.3 – Analyse de *le très gâteau*

5.3 Pouvoir génératif

5.3.1 Calcul de Lambek et grammaires AB

Puisque nous disposons désormais de la charge et de la décharge d'hypothèses, voyons-en les conséquences sur l'exemple *le livre que Marie lit*. Le problème était que la dérivation de cette expression dans les grammaires AB avec le lexique de la table 5.4 rendait nécessaire l'introduction d'une nouvelle règle (d'élévation de type). Est-ce toujours nécessaire ?

Lex$_3$	
Marie	np
livre	n
le	np/n
lit	$(np\backslash S)/np$
que	$(n\backslash n)/(S/np)$

TAB. 5.4 – Définition de **Lex**$_3$

Avec D_1 la preuve :

$$\dfrac{\dfrac{np \vdash_{\mathrm{L}^*_{\mathrm{DN}}} np \qquad \dfrac{(np\backslash S)/np \vdash_{\mathrm{L}^*_{\mathrm{DN}}} (np\backslash S)/np \quad np \vdash_{\mathrm{L}^*_{\mathrm{DN}}} np}{(np\backslash S)/np, np \vdash_{\mathrm{L}^*_{\mathrm{DN}}} np\backslash S}\,/_e}{np, (np\backslash S)/np, np \vdash_{\mathrm{L}^*_{\mathrm{DN}}} S}\,\backslash_e}{np, (np\backslash S)/np \vdash_{\mathrm{L}^*_{\mathrm{DN}}} S/np}\,/_i$$

la preuve :

$$\dfrac{\dfrac{np/n \vdash_{\mathsf{L}^*_{\mathsf{DN}}} np/n \qquad \dfrac{n \vdash_{\mathsf{L}^*_{\mathsf{DN}}} n \qquad \dfrac{(n\backslash n)/(S/np) \vdash_{\mathsf{L}^*_{\mathsf{DN}}} (n\backslash n)/(S/np) \qquad \begin{array}{c}\vdots D_1\\ np, (np\backslash S)/np \vdash_{\mathsf{L}^*_{\mathsf{DN}}} S/np\end{array}}{(n\backslash n)/(S/np), np, (np\backslash S)/np \vdash_{\mathsf{L}^*_{\mathsf{DN}}} n\backslash n}\backslash_e}{n, (n\backslash n)/(S/np), np, (np\backslash S)/np \vdash_{\mathsf{L}^*_{\mathsf{DN}}} n}/_e}{\begin{array}{cccccc} np/n, & n, & (n\backslash n)/(S/np), & np, & (np\backslash S)/np & \vdash_{\mathsf{L}^*_{\mathsf{DN}}} np\\ le & livre & que & Marie & lit & \end{array}}$$

montre que cette expression est dérivable (comme *np*) sans les règles (4.1), (4.3), (4.2) et (4.4) supplémentaires.

$$A/B\ B/C \to A/C \tag{4.1}$$
$$A\backslash B\ B\backslash C \to A\backslash C \tag{4.3}$$
$$A \to B/(A\backslash B) \tag{4.2}$$
$$A \to (B/A)\backslash B \tag{4.4}$$

En fait, ces règles sont maintenant des théorèmes :

$$\dfrac{\dfrac{A/B \vdash_{\mathsf{L}^*_{\mathsf{DN}}} A/B \qquad \dfrac{B/C \vdash_{\mathsf{L}^*_{\mathsf{DN}}} B/C \quad C \vdash_{\mathsf{L}^*_{\mathsf{DN}}} C}{B/C, C \vdash_{\mathsf{L}^*_{\mathsf{DN}}} B}/_e}{A/B, B/C, C \vdash_{\mathsf{L}^*_{\mathsf{DN}}} A}/_e}{A/B, B/C \vdash_{\mathsf{L}^*_{\mathsf{DN}}} A/C}/_i \qquad \dfrac{\dfrac{A \vdash_{\mathsf{L}^*_{\mathsf{DN}}} A \quad A\backslash B \vdash_{\mathsf{L}^*_{\mathsf{DN}}} A\backslash B}{A, A\backslash B \vdash_{\mathsf{L}^*_{\mathsf{DN}}} B}\backslash_e}{A \vdash_{\mathsf{L}^*_{\mathsf{DN}}} B/(A\backslash B)}/_i$$

$$\dfrac{\dfrac{\dfrac{A \vdash_{\mathsf{L}^*_{\mathsf{DN}}} A \quad A\backslash B \vdash_{\mathsf{L}^*_{\mathsf{DN}}} A\backslash B}{A, A\backslash B \vdash_{\mathsf{L}^*_{\mathsf{DN}}} B}\backslash_e \qquad B\backslash C \vdash_{\mathsf{L}^*_{\mathsf{DN}}} B\backslash C}{A, A\backslash B, B\backslash C \vdash_{\mathsf{L}^*_{\mathsf{DN}}} C}\backslash_e}{A\backslash B, B\backslash C \vdash_{\mathsf{L}^*_{\mathsf{DN}}} A\backslash C}\backslash_i \qquad \dfrac{\dfrac{B/A \vdash_{\mathsf{L}^*_{\mathsf{DN}}} B/A \quad A \vdash_{\mathsf{L}^*_{\mathsf{DN}}} A}{B/A, A \vdash_{\mathsf{L}^*_{\mathsf{DN}}} B}/_e}{A \vdash_{\mathsf{L}^*_{\mathsf{DN}}} (B/A)\backslash B}\backslash_i$$

Tout se passe comme si la preuve s'effectuait en supposant un x : *np* comme objet pour *lit* : $(np\backslash S)/np$ pour dériver *lit x* : $np\backslash S$, puis *Marie lit x* : *S*. Cette expression de type *S* ayant été obtenue en supposant l'existence d'un *np* à droite, l'expression *Marie lit* doit dériver *S* si on lui fournit un *np* à droite, c'est-à-dire que son type est S/np. On trouvera à la figure B.19 de l'annexe B un autre exemple d'élévation de type directement obtenu par la preuve.

Bien d'autres règles proposées pour augmenter le pouvoir génératif des grammaires catégorielles deviennent ainsi des théorèmes de ce calcul et sont naturellement intégrées. On peut les retrouver par exemple dans [Moo97].

L'exemple ci-dessus montre comment, grâce au raisonnement avec hypothèse, le calcul de Lambek permet de rendre compte des constructions avec *dépendances non bornées.* L'annexe B montre par exemple les analyses (voir figures B.15(a), B.15(b) et B.15(c)) de phrases telles que :

Jean déteste le livre que le professeur conseille à Marie de lire
Jean déteste le livre dont le professeur conseille à Marie de lire la préface
Jean parle du livre dont le professeur conseille à Marie de lire la preface

Toutefois, étant donné la nature totalement ordonnée du séquent, on ne peut faire que des extractions périphériques. Ici, la règle qui sert est :

$$\dfrac{\Gamma, A \vdash_{\mathsf{L}^*_{\mathsf{DN}}} B}{\Gamma \vdash_{\mathsf{L}^*_{\mathsf{DN}}} B/A}/_i$$

permettant de typer *Marie lit* avec S/np. Par contre, en l'absence de règle du genre :

$$\frac{\Gamma, A, C \vdash_{\mathsf{L}^*_{\mathsf{DN}}} B}{\Gamma, C \vdash_{\mathsf{L}^*_{\mathsf{DN}}} B/A}\,/_i$$

le simple fait de rajouter à l'extrémité *avidement* (de type $S\backslash S$), empêche d'obtenir une analyse de *que Marie lit avidement.* La section 5.5.1 montre comment étendre le calcul de Lambek pour obtenir ces dérivations.

5.3.2 Calcul de Lambek et grammaires hors contexte

Dans cette section qui décrit les relations entre grammaires hors-contexte et calcul de Lambek, nous n'allons pas donner les preuves des résultats. En effet, si l'un des sens de l'équivalence est assez aisé (les grammaires hors-contexte sont faiblement équivalentes aux grammaires de Lambek), le second par contre est beaucoup plus ardu. Connu comme la conjecture de Chomsky [Cho63], il n'a été démontré que récemment par Pentus [Pen93].

Théorème 52. *Pour toute grammaire hors-contexte sans le ϵ, il existe une grammaire de Lambek telle que les langages engendrés par chacune de ces grammaires coïncident.*

Théorème 53. *Pour toute grammaire de Lambek, il existe une grammaire hors-contexte telle que les langages engendrés par chacune des grammaires coïncident.*

5.4 Analyse sémantique

Bien que constitué d'une mise en commun de résultats déjà décrits (en particulier dans les sections 3.2.1, 3.2 et 3.1.3), ce paragraphe constitue un élément crucial des grammaires de types logiques et de leur articulation entre syntaxe et sémantique, remarqué notamment par [vB83]. Et finalement il définit la pierre angulaire de notre travail autour de la génération en donnant le moyen de passer de la syntaxe à la sémantique et réciproquement.

Nous avons montré comment passer d'une preuve de la logique linéaire intuitionniste à un λ-terme grâce à l'isomorphisme de Curry-Howard. Il s'étend naturellement au cas non commutatif (celui du calcul de Lambek), en associant les mêmes λ-termes pour l'introduction et l'élimination de $/$ et $\backslash$: ceux associés à $\multimap$.

Le deuxième point traduit tout simplement l'utilisation de la sémantique de Montague à travers la correspondance de Curry-Howard, telle que la décrit le chapitre 3.

Dès lors, si l'on a :

- une expression e_1 de type syntaxique A, de type sémantique $\mathcal{H}(A)$ et de forme sémantique s_1 ;
- une expression e_2 de type syntaxique $A\backslash B$, de type sémantique $\mathcal{H}(A) \to \mathcal{H}(B)$ et de forme sémantique s_2,

la preuve $A, A\backslash B \vdash_{\mathsf{L}^*_{\mathsf{DN}}} B$ de ce que l'expression $e_1 e_2$ est de type B se transforme en une preuve de $s_1 : \mathcal{H}(A), s_2 : \mathcal{H}(A) \to \mathcal{H}(B) \vdash_{\mathsf{ILL}} s_2 s_1 : \mathcal{H}(B)$ et la forme sémantique de l'expression $e_1 e_2$ est donc $s_2 s_1$.

Pour appliquer cela à l'analyse syntaxique, il faut donc :

- un lexique dont chaque entrée lexicale associe un mot m_i à un ou plusieurs types syntaxique $T_j^{(i)}$ et à un ou plusieurs λ-termes typés $s_i^{(j)}$ (sa forme sémantique, dans le cadre de la sémantique de Montague) ;
- un isomorphisme $\mathcal{H}$ entre les types syntaxiques et sémantiques. On utilisera, sauf précision du contraire, $\mathcal{H}$ tel que :

$$\mathcal{H}(np) = e \quad \mathcal{H}(S) = t \quad \mathcal{H}(n) = e \to t$$

$$\mathcal{H}(A\backslash B) = \mathcal{H}(A) \to \mathcal{H}(B) \quad \mathcal{H}(A/B) = \mathcal{H}(B) \to \mathcal{H}(A)$$

Il faut bien sûr le compléter si d'autres types syntaxiques s'ajoutent.

Si l'expression $m_\sigma(1)\cdots m_\sigma(n)$ est une expression de type T, il existe une preuve π de

$$T_{j_1}^{(\sigma(1))}, \ldots, T_{j_n}^{(\sigma(n))} \vdash_{\mathsf{L}^*_{\mathsf{DN}}} T$$

dont on obtient, en remplaçant les types syntaxiques par leur transformés suivant $\mathcal{H}$ (les règles d'inférence ne changeant pas), une preuve π' de

$$x_1 : \mathcal{H}(T_{j_1}^{(\sigma(1))}), \ldots, x_n : \mathcal{H}(T_{j_n}^{(\sigma(n))}) \vdash_{\mathsf{ILL}} u : \mathcal{H}(T)$$

La forme sémantique de $m_\sigma(1)\cdots m_\sigma(n)$ est alors donnée par $u[s_{\sigma(1)}/x_1, \ldots, s_{\sigma(n)}/x_n]$.

Notation. Puisque les règles sont inchangées entre la preuve de la correction syntaxique et celle du typage sémantique, on peut directement étiqueter les premières par les λ-termes des secondes. Puis, au lieu de ne faire qu'à la fin la substitution des x_i par les $s_{\sigma(i)}$, on peut tout de suite les utiliser dans le séquent à démontrer.

Illustrons ce fonctionnement sur un premier exemple simple pris dans la section 3.1.3 consacrée à la sémantique de Montague et définissons le lexique $\mathbf{Lex}_7$ du tableau 5.5.

$\mathbf{Lex}_7$			
Entrée	Type syntaxique	Type sémantique	Forme sémantique
Othello	*np*	e	$\mathbf{o}$
Desdémone	*np*	e	$\mathbf{d}$
meurt	$np\backslash S$	$e \to t$	$\lambda x.\mathbf{meurt}x$
aime	$(np\backslash S)/np$	$e \to e \to t$	$\lambda x.\lambda y.(\mathbf{aime}x)y$

TAB. 5.5 – Définition de $\mathbf{Lex}_7$

Cherchons à calculer le sens s de *Othello meurt.* On cherche une preuve de

$$\mathbf{o} : np, \lambda x.\mathbf{meurt}x : np\backslash s : S \vdash_{\mathsf{L}^*_{\mathsf{DN}}} S$$

$$\dfrac{\dfrac{}{\mathbf{o} : np \vdash_{\mathsf{L}^*_{\mathsf{DN}}} \mathbf{o} : np}\ \text{Axiome} \qquad \dfrac{}{\lambda x.(\mathbf{meurt}x) : np\backslash S \vdash_{\mathsf{L}^*_{\mathsf{DN}}} \lambda x.(\mathbf{meurt}x) : np\backslash S}\ \text{Axiome}}{\mathbf{o} : np, \lambda x.(\mathbf{meurt}x) : np\backslash S \vdash_{\mathsf{L}^*_{\mathsf{DN}}} (\lambda x.(\mathbf{meurt}x))\mathbf{o} : S}\ \backslash_e$$

On a donc $s = (\lambda x.(\mathbf{meurt}x))\mathbf{o} \to_\beta^* \mathbf{meurt}\ \mathbf{o}$.

Nous verrons par la suite des exemples plus frappants de la pertinence de l'approche de Montague, notamment dans le contrôle des portées des quantificateurs.

5.4.1 Analyse sémantique et réseaux

Ce processus d'analyse sémantique, décrit pour le calcul des séquents, est bien entendu tout à fait valable avec les réseaux de preuve, en partant de la preuve syntaxique non pas sous forme de séquents mais sous forme de réseau. On a alors deux méthodes.

La première, calquée sur celle des séquents, permet de calculer le sens de l'expression analysée en étiquetant le réseau comme indiqué à la section 3.2.3.0. La figure 5.4 l'illustre sur le même exemple que précédemment (en partant des conclusions négatives auxquelles on attribue la forme sémantique

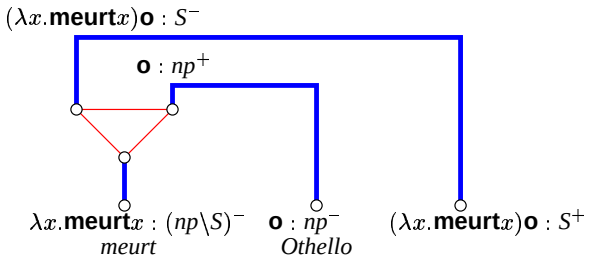


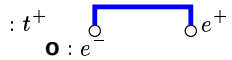
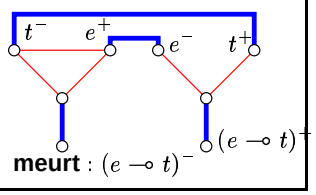
FIG. 5.4 – Analyse sémantique de *Othello meurt* par les réseaux

définie dans le lexique, puis en écrivant directement les résultats de l'unification). On obtient de même $s = (\lambda x.\mathbf{meurt}x)\mathbf{o} \rightarrow^*_\beta \mathbf{meurt}\ \mathbf{o}$.

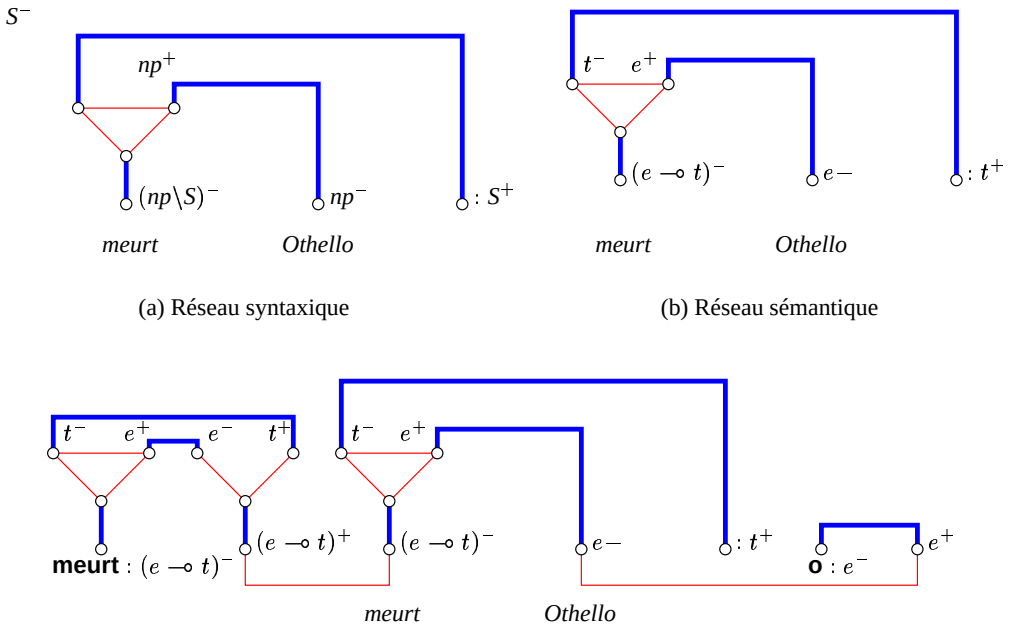
La deuxième méthode, proposée par [dGR96], consiste à se *passer* des λ-termes en leur préférant le réseau qui les encode. On procède de la manière suivante :

1. dans le lexique, les λ-termes sont remplacés par les réseaux (dits *sémantiques*) qui les codent ;
2. après l'analyse syntaxique, on obtient un réseau de preuve π' sur les types syntaxiques (figure 5.5(a)) qui se transforme en réseau de preuve π sur les types sémantiques par $\mathcal{H}$ (figure 5.5(b)), que l'on note par abus de langage $\mathcal{H}(\pi')$. Attention, dans cette transformation, on peut rajouter des sommets et des liens : si l'image $\mathcal{H}(A)$ d'un type syntaxique de base A est d'ordre supérieur ou égal à 2 (par exemple $\mathcal{H}(n) = e \multimap t$), on développe le lien axiome par η-expansion dans le réseau sémantique ;
3. au lieu de faire la substitution des λ-termes du lexique dans le λ-terme qu'on aurait obtenu en lisant le réseau de l'étape précédente, on relie directement par un lien Cut chaque conclusion négative de π, correspondant à une entrée lexicale, au réseau sémantique de cette entrée lexicale (voir figure 5.5(c)) ;
4. on élimine les coupures (figure 5.5(d)).

On a alors un nouveau réseau sémantique qui donne le sens de l'expression analysée. Bien entendu, la lecture de ce réseau en λ-terme donne le même résultat qu'avec la première méthode puisque le processus agit sur des représentations isomorphes. La figure 5.5 illustre, toujours sur le même exemple, cette méthode, avec le lexique du tableau 5.6.

Lex$_8$			
Entrée	Type syntaxique	Type sémantique	Forme sémantique
Othello	np	e	
meurt	$np\backslash S$	$e \rightarrow t$	

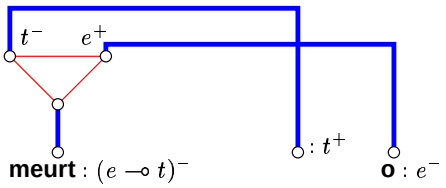
TAB. 5.6 – Définition de **Lex**$_8$



(a) Réseau syntaxique

(b) Réseau sémantique

(c) Réseau sémantique et lien Cut avec les réseaux sémantiques du lexique



(d) Réseau sémantique, après réduction

FIG. 5.5 – Analyse sémantique, uniquement avec les réseaux, de *Othello meurt*

Remarque 4. Les opérations sémantiques se passent dans le fragment intuitionniste *commutatif*, à la différence des opérations syntaxiques. Le problème des liens axiomes qui se croisent, pour les réseaux, ne se pose donc plus.

Remarque 5. Nous avons vu à la section 3.3 comment, en ajoutant les exponentiels aux connecteurs multiplicatif, il était possible de coder des λ-termes *non linéaires* à l'aide des réseaux de preuve.

Par exemple, tenir compte de la non linéarité de la forme sémantique de *tout* :

$$\lambda P.\lambda Q.\forall(Px) \Rightarrow (Qx)$$

demande que x soit typé $!e$ (en fait, on retrouve là la décomposition de l'implication intuitionniste $A \to B$ en $!A \multimap B$ avec les connecteurs de la logique linéaire).

Cela se traduit par l'adoption d'un isomorphime $\mathcal{H}$ qui tienne compte du fait que le type syntaxique est linéaire, mais que le type sémantique puisse ne pas l'être. En l'occurrence, puisque l'on demande désormais à ce que le type des entités soit $!e$, il convient de ce que $\mathcal{H}$ envoie les éléments syntaxiques de type *np* vers $!e$. On utilisera donc comme définition de $\mathcal{H}$ sur les types de base :

$$\begin{aligned}\mathcal{H}(np) &= !e \\ \mathcal{H}(S) &= t \\ \mathcal{H}(n) &= !e \multimap t\end{aligned}$$

5.4.2 Capacité de modélisation

Dans cette section, nous illustrons sur des exemples un peu plus complexes comment certains phénomènes linguistiques peuvent être modélisés. Nous évoquons les aspects syntaxiques, avec la coordination de non-constituants, les aspects sémantiques, avec l'ambiguïté, et nous présentons également une mesure de complexité syntaxique basée sur les réseaux. Cette hypothèse, d'après [Mor00] à la suite des travaux [Joh98], permet entre autres d'expliquer la préférence d'une interprétation dans le cas d'une ambiguïté.

Coordination

Le traitement de la coordination se révèle comme un des apports importants du calcul de Lambek [Dow88, Moo91, Mor94]. De manière générale, la coordination entre constituants (syntagmes nominaux, verbaux, etc.) peut s'exprimer grâce au schéma de type $(X\backslash X)/X$. Par exemple, le typage $(S\backslash S)/S$ permet d'exprimer *Jean aime le livre et Marie déteste le livre*. Par contre, dans *Jean aime et Marie déteste le livre*, la conjonction n'a plus lieu sur des constituants complets, mais sur des constituants auxquels il manque un *np* à droite pour être complets, soit de type S/np.

De la même manière que précédemment, on peut typer les expressions *Jean aime* et *Marie déteste* par S/np. La coordination entre ces deux expressions peut alors s'effectuer avec

$$et : ((S/np)\backslash(S/np))/(S/np)$$

pour donner le type (S/np) que *le livre* : *np* complète en dérivant *S* (on trouvera l'analyse avec les réseaux à la figure B.16).

Soit, avec :

$$D_1 = \dfrac{\dfrac{np \vdash_{L^*_{DN}} np \qquad \dfrac{(np\backslash S)/np \vdash_{L^*_{DN}} (np\backslash S)/np \quad np \vdash_{L^*_{DN}} np}{(np\backslash S)/np, np \vdash_{L^*_{DN}} np\backslash S}\,/_e}{np, (np\backslash S)/np, np \vdash_{L^*_{DN}} S}\,\backslash_e}{\underset{Jean}{np}, \quad \underset{aime}{(np\backslash S)/np} \quad \vdash_{L^*_{DN}} S/np}\,/_i$$

$$D_2 = \dfrac{\dfrac{np \vdash_{L^*_{DN}} np \qquad \dfrac{(np\backslash S)/np \vdash_{L^*_{DN}} (np\backslash S)/np \quad np \vdash_{L^*_{DN}} np}{(np\backslash S)/np, np \vdash_{L^*_{DN}} np\backslash S}\,/_e}{np, (np\backslash S)/np, np \vdash_{L^*_{DN}} S}\,\backslash_e}{\underset{Marie}{np}, \quad \underset{déteste}{(np\backslash S)/np} \quad \vdash_{L^*_{DN}} S/np}\,/_i$$

on obtient :

$$\dfrac{D_1 \qquad \dfrac{((S/np)\backslash(S/np))/(S/np) \vdash_{L^*_{DN}} ((S/np)\backslash(S/np))/(S/np) \quad D_2}{((S/np)\backslash(S/np))/(S/np), np, (np\backslash S)/np \vdash_{L^*_{DN}} (S/np)\backslash(S/np)}\,/_e}{\underset{Jean}{np}, \quad \underset{aime}{(np\backslash S)/np}, \quad \underset{et}{((S/np)\backslash(S/np))/(S/np)}, \quad \underset{Marie}{np}, \quad \underset{déteste}{(np\backslash S)/np} \quad \vdash_{L^*_{DN}} S/np}\,\backslash_e$$

Notons que nous avons introduit un type polymorphe pour la coordination. Le lexique n'est alors plus véritablement fini. La question de savoir si l'analyse reste décidable se pose alors.

Théorème 54 ([Car97]). *L'appartenance au langage défini par un lexique fini étendu avec l'infinité d'instances des coordinations polymorphiques des entrées lexicales est décidable.*

Du point de vue sémantique, comme le montre [Car97], le phénomène de coordination suit l'analyse syntaxique par l'isomorphisem de Curry-Howard. Avec le connecteur habituel $\wedge : t \to t \to t$, on peut définir pour l'exemple précédent :

$$et : \quad ((S/np)\backslash(S/np))/(S/np) \quad \lambda P^{(e\to t)}.\lambda Q^{(e\to t)}.\lambda x^e.(Px) \wedge (Qx)$$

Avec la sémantique calculée pour les expressions :

Jean aime : $\lambda y.(\textbf{aime}\ y)\textbf{j}$

Marie déteste : $\lambda y.(\textbf{déteste}\ y)\textbf{m}$

on obtient pour *Jean aime et Marie déteste* :

$$\begin{aligned}((\lambda P^{(e\to t)}.\lambda Q^{(e\to t)}.\lambda x^e.(Px) \wedge (Qx))(\lambda y.(\textbf{déteste}\ y)\textbf{m})(\lambda y.(\textbf{aime}\ y)\textbf{j}) \\ \to^*_\beta \lambda x.((\lambda y.(\textbf{déteste}\ y)\textbf{m})x) \wedge ((\lambda y.(\textbf{aime}\ y)\textbf{j})x) \\ \to^*_\beta \lambda x.((\textbf{déteste}\ x)\textbf{m}) \wedge ((\textbf{aime}\ x)\textbf{j})\end{aligned}$$

Néanmoins, comme le relève [Mor94, Car97], ce schéma de typage peut s'avérer trop général et nécessiter plus de contrôle structurel pour isoler les éléments de coordinations et les structures avec lesquelles elles peuvent agir. Dans le cas contraire, des phrases comme **[Jean pense que et le frère de] Marie dort* appartiennent au langage. L'annexe B donne un exemple où *John believes Mary or Bill walks* est analysé avec le typage

$$((S/(np\backslash S))\backslash(S/(np\backslash S)))/(S/(np\backslash S))$$

comme *John believes (Mary walks or Bill walks)*, mais également comme *(John believes Mary walks) or (John believes Bill walks)*.

Portée des quantificateurs

L'approche de Montague se révèle également efficace pour exprimer les ambiguïtés dues à la portée des quantificateurs. Ainsi la phrase :

Toute peine mérite un salaire

a-t-elle deux sens :

1. il y a un salaire commun (et unique) que mérite toute peine[15] ;
2. pour chaque peine, il y a un salaire que cette peine mérite.

L'ambiguïté ici naît de ce que chacune des quantifications (celle du sujet : quantification universelle, et celle de l'objet : quantification existentielle) peut s'appliquer à la phrase toute entière. Ainsi, dans une phrase comme :

Lantier mérite un salaire

la quantification qui provient de l'objet s'applique-t-elle à toute la phrase :

$$\exists y(\textbf{salaire}\ y) \wedge ((\textbf{mérite}\ y)\mathbf{l})$$

De même, le sujet peut lui aussi prendre toute la phrase dans son champ :

Toute peine fatigue

s'interprète comme : $\forall x(\textbf{peine}\ x \Rightarrow \textbf{fatigue}\ x)$.

Voyons donc si l'on peut rendre compte des deux interprétations :

1. $\exists y \forall x(\textbf{peine}\ x \Rightarrow (\textbf{salaire}\ y) \wedge ((\textbf{mérite}\ y)x))$
2. $\forall x \exists y(\textbf{peine}\ x \Rightarrow (\textbf{salaire}\ y) \wedge ((\textbf{mérite}\ y)x))$

Pour cela, utilisons le lexique du tableau 5.7, avec les formes sémantiques définies à la section 3.1.3. Notons que nous faisons usage de la non-linéarité, ce qui conduit à adopter le typage et la définition de $\mathcal{H}$ définis à la section 3.3 et à la remarque 5.

Nous optons ici pour une présentation de l'analyse syntaxique sous forme de réseau pour deux raisons : la première est que ce qui engendre l'ambiguïté dans le principe des grammaires de types logiques, à savoir la multiplicité des preuves pour un même séquent, nous semble plus frappant avec les réseaux. D'autre part, cela permet d'introduire la section suivante qui présente une justification, d'après [Mor00], de la préférence pour l'une ou l'autre des interprétations.

Comme le montrent les figures 5.6(a) et 5.7(a), il existe deux preuves différentes de

$$(S/(np\backslash S))/n, np, (np\backslash S)/np, ((S/np)\backslash S)/np, np \vdash_{\mathsf{lcLL}} S$$

(dans le cas commutatif, il existe huit preuves de ce séquent. On ne retient bien entendu ici, pour la syntaxe, que celles dont les liens axiomes ne se croisent pas). L'étiquetage de chacune de ces preuves (figures 5.6(b) et 5.7(b)) montre que l'on a affaire à une véritable ambiguïté (et non une ambiguïté fallacieuse) en donnant les termes suivants :

– $(yx)(\lambda a.(wv)(\lambda b.(za)(b)))$
– et $(wv)(\lambda a.((yx)(\lambda b.(zb)a)))$.

[15] Ce sens peut surprendre. Le négliger pourrait conduire, dans des négociations salariales, à la même déception que celle des ouvriers de la première heure, traités comme ceux de la onzième heure : « Ces derniers venus n'ont fait qu'une heure, et tu les as traités comme nous, qui avons porté le fardeau de la journée (...) » (Mt 20 :12).

Entrée lexicale	Type syntaxique	Type sémantique	Forme sémantique
peine	*np*	$!e$	**peine**
salaire	*np*	$!e$	**salaire**
mérite	$(np\backslash S)/np$	$!e \multimap !e \multimap t$	$\lambda x.\lambda y.(\textbf{mérite}\ x)y$
toute	$(S/(np\backslash S))/n$	$(!e \multimap t) \multimap (!e \multimap t) \multimap t$	$\lambda P.\lambda Q.\forall(\lambda x.(Px \Rightarrow Qx))$
un	$((S/np)\backslash S)/n$	$(!e \multimap t) \multimap (!e \multimap t) \multimap t$	$\lambda P.\lambda Q.\exists(\lambda x.(Px) \wedge (Qx))$

TAB. 5.7 – Lexique $\textbf{Lex}_9$

On a alors :

$$(yx)(\lambda a.(wv)(\lambda b.(za)(b))) \rightarrow_\eta ((yx)(\lambda a.(wv)(za)))$$

$$\begin{aligned}(yx)(\lambda a.(wv)(za))[\lambda P.\lambda Q.\forall(\lambda u_1.(Pu_1 \Rightarrow Qu_1))/w] \\ \rightarrow^*_\beta (yx)(\lambda a.(\lambda Q.\forall(\lambda u_1.(vu_1 \Rightarrow Qu_1)))(za)) \\ \rightarrow^*_\beta (yx)(\lambda a.\forall(\lambda u_1.(vu_1 \Rightarrow (za)u_1)))\end{aligned}$$

$$\begin{aligned}(yx)(\lambda a.\forall(\lambda u_1.(vu_1 \Rightarrow (za)u_1)))[\lambda P.\lambda Q.\exists(\lambda u_2.(Pu_2) \wedge (Qu_2))/y] \\ \rightarrow^*_\beta \exists(\lambda u_2.(xu_2) \wedge ((\lambda a.\forall(\lambda u_1.(vu_1 \Rightarrow (za)u_1)))u_2)) \\ \rightarrow^*_\beta \exists(\lambda u_2.(xu_2) \wedge (\forall(\lambda u_1.(vu_1 \Rightarrow (zu_2)u_1))))\end{aligned}$$

$$\begin{aligned}\exists(\lambda u_2.(xu_2) \wedge (\forall(\lambda u_1.(vu_1 \Rightarrow (zu_2)u_1))))[\lambda u_2.\lambda u_3.(\textbf{mérite}\ u_2)u_3/z] \\ \rightarrow^*_\beta \exists(\lambda u_2.(xu_2) \wedge (\forall(\lambda u_1.(vu_1 \Rightarrow (\textbf{mérite}\ u_2)u_1))))\end{aligned}$$

$$\begin{aligned}\exists(\lambda u_2.(xu_2) \wedge (\forall(\lambda u_1.(vu_1 \Rightarrow (\textbf{mérite}\ u_2)u_1))))[\textbf{salaire}/x, \textbf{peine}/v] \\ = \exists(\lambda u_2.(\textbf{salaire}\ u_2) \wedge (\forall(\lambda u_1.(\textbf{peine}\ u_1 \Rightarrow (\textbf{mérite}\ u_2)u_1))))\end{aligned}$$

et

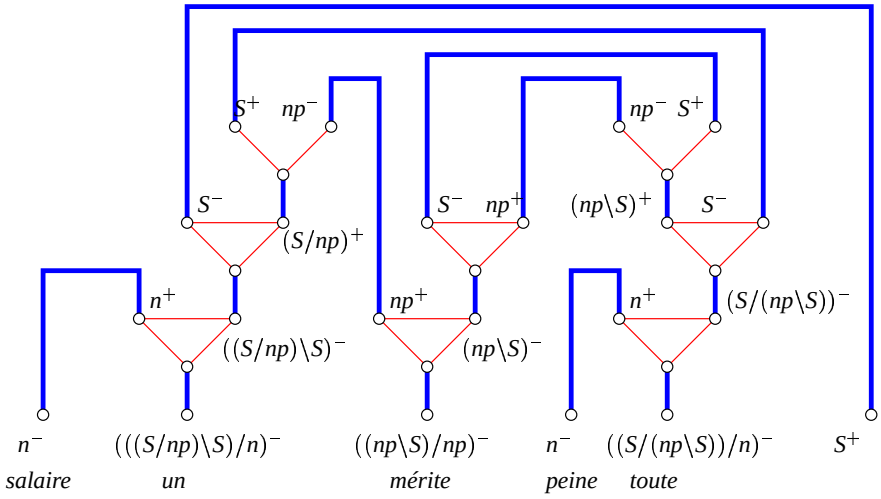
$$\begin{aligned}(wv)(\lambda a.((yx)(\lambda b.(zb)a)))[\lambda P.\lambda Q.\exists(\lambda u_2.(Pu_2) \wedge (Qu_2))/y] \\ \rightarrow^*_\beta (wv)(\lambda a.\exists(\lambda u_2.(xu_2) \wedge ((zu_2)a)))\end{aligned}$$

$$\begin{aligned}(wv)(\lambda a.\exists(\lambda u_2.(xu_2) \wedge ((zu_2)a)))[\lambda P.\lambda Q.\forall(\lambda u_1.(Pu_1 \Rightarrow Qu_1))/w] \\ \rightarrow^*_\beta \forall(\lambda u_1.(vu_1 \Rightarrow ((\lambda a.\exists(\lambda u_2.(xu_2) \wedge ((zu_2)a)))u_1))) \\ \rightarrow^*_\beta \forall(\lambda u_1.(vu_1 \Rightarrow (\exists(\lambda u_2(xu_2) \wedge ((zu_2)u_1)))))\end{aligned}$$

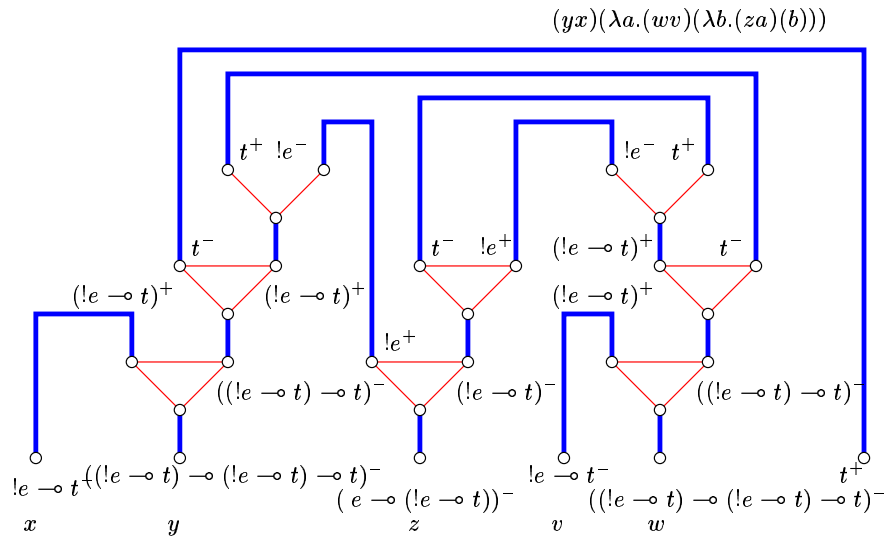
$$\begin{aligned}\forall(\lambda u_1.(vu_1 \Rightarrow (\exists(\lambda u_2(xu_2) \wedge ((zu_2)u_1)))))[\lambda u_2.\lambda u_3.(\textbf{mérite}\ u_2)u_3/z] \\ \rightarrow^*_\beta \forall(\lambda u_1.(vu_1 \Rightarrow (\exists(\lambda u_2(xu_2) \wedge ((\textbf{mérite}\ u_2)u_1)))))\end{aligned}$$

$$\begin{aligned}\forall(\lambda u_1.(vu_1 \Rightarrow (\exists(\lambda u_2(xu_2) \wedge ((\textbf{mérite}\ u_2)u_1)))))[\textbf{salaire}/x, \textbf{peine}/v] \\ = \forall(\lambda u_1.(\textbf{peine}\ u_1 \Rightarrow (\exists(\lambda u_2(\textbf{salaire}\ u_2) \wedge ((\textbf{mérite}\ u_2)u_1)))))\end{aligned}$$

Ce sont (à une α-conversion près) bien les deux λ-termes attendus, et dont l'existence correspond à l'ambiguïté de la phrase.

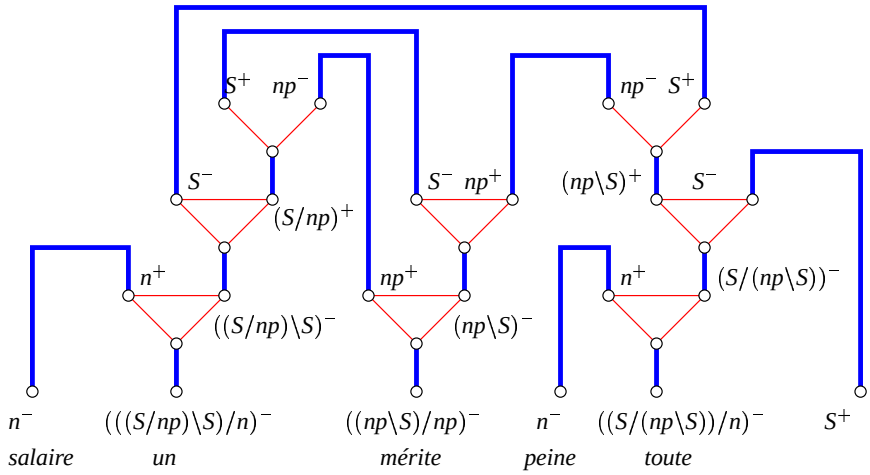


(a) Réseau syntaxique

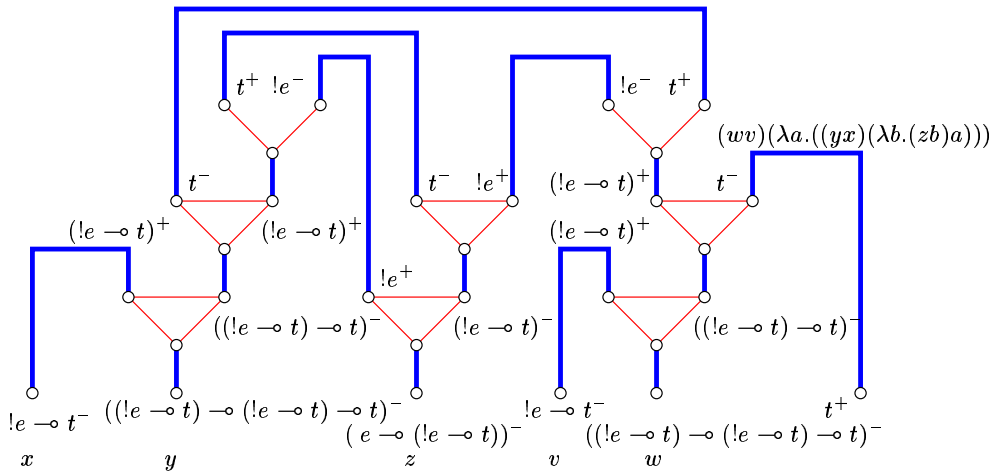


(b) Réseau sémantique

FIG. 5.6 – Une première analyse de *Toute peine mérite un salaire*



(a) Réseau syntaxique



(b) Réseau sémantique

FIG. 5.7 – Une deuxième analyse de *Toute peine mérite un salaire*

Une mesure de complexité

[Mor00] tente d'expliquer pourquoi certaines constructions syntaxiques restent acceptables alors que d'autres perdent cette acceptabilité rapidement lorsque leur taille croît. Ainsi, les phrases :

– *la poule qui fit l'œuf caqueta*
– *la poule qui fit l'œuf qui fit la poule caqueta*
– *la poule qui fit l'œuf qui fit la poule qui fit l'œuf caqueta*

restent plus acceptables que[16] :

– *la maison que le petit cochon habitait s'effondra*
– ?*la maison que le petit cochon que le loup menaçait habitait s'effondra*
– ??*la maison que le petit cochon que le loup que l'appétit tenaillait menaçait habitait s'effondra*

(On trouvera en annexe B des analyses de ces phrases en les figures B.17 et B.18 qui correspondent bien aux résultats attendus après le critère de complexité énoncé ci-après.)

De même, il cherche à justifier que, pour l'ambiguïté de quantification, la préférence est donnée aux interprétations dans lesquelles le quantificateur qui prend le champ le plus grand est celui porté par le premier déterminant. Pour cela, il énonce le *principe d'acceptabilité* :

> L'acceptabilité est inversement proportionnelle à la somme au cours du temps des valences non résolues.

Le temps se réfère au traitement incrémental effectué après chaque mot.

Ce principe permet de classer :

– les sens préférés par ordre croissant de complexité, en présence d'ambiguïté, lorsque la prosodie reste constante ;
– les expressions préférées par ordre croissant de complexité, en présence de synonymie, à sens constant.

L'analyse syntaxique sous forme de réseau permet un calcul très simple de ces valences non résolues. En effet, celui-ci correspond au nombre d'atomes qui n'ont pas encore été liés à leur dual, après chaque mot. Pour le déterminer, il suffit de regarder sur le réseau combien de liens axiomes passent entre chaque mot. Pour les deux exemples des figures 5.6(a) et 5.7(a), cela donne le tableau 5.8 (rappelons qu'il faut lire le réseau de droite à gauche). Au-dessus de chaque mot, nous indiquons pour chacun des sens le nombre de valences non résolues après ce mot.

Cette mesure permet d'expliquer pourquoi le deuxième sens est celui généralement préféré. Remarquons que si dans ce cas la mesure instantanée suffit (le nombre de liens axiomes après chaque mot) car comme le montre le tableau 5.8, à chaque instant le nombre de liens pour a est supérieur au nombre de liens pour b, cela ne semble pas vrai en général, alors que la mesure cumulée (le total) répond mieux aux prévisions.

5.5 Extensions

Si la modélisation à l'aide du calcul de Lambek et de la sémantique de Montague permet de couvrir un certain nombre de phénomènes linguistiques, d'autres, par contre, lui échappent.

En particulier l'interprétation prosodique dans un semi-groupe avec une loi associative et non commutative se traduit par une modélisation dans laquelle l'ordre des mots est très contraint. Or, s'il est vrai que l'ordre des mots n'est en général pas totalement libre, il n'est pas en général complètement figé.

[16] $?e$ où e est une expression indique que l'on n'est pas sûr de la correction de l'expression. $??e$ indique une sûreté encore moindre.

5		a					
4			a				
3		b		ba			
2			b				
1	ba				ba		
0						ba	
	S	toute	peine	mérite	un	salaire	
a : lecture avec $\exists\forall$							Total : 14
b : lecture avec $\forall\exists$							Total : 10

TAB. 5.8 – Mesure de la complexité de chacune des lectures

Ainsi, l'ordre des mots dans certaines phrases n'est-il que partiel : *J'écoute pousser mes cheveux* ou *J'écoute mes cheveux pousser*. Cet ordre peut être plus ou moins libre suivant les langues. D'autre part, l'interprétation par la concaténation impose également la continuité des composants. Cela empêche de rendre compte en français de la négation (***n**'allez **pas** noyer le souffleur*) ou des particules verbales en anglais (*she very soon **finished** it **off*** ou bien *she very soon **finished off** the cake*).

Pour étendre le calcul de Lambek, diverses approches se présentent. Nous évoquerons plus particulièrement l'approche multi-modale [Moo97, Mor94] et l'approche avec un calcul partiellement ordonné [LR95, LR96b]. On a vu combien la structure (la sensibilité aux ressources, la commutativité ou la non-commutativité) importait à la fois pour les systèmes logiques et la capacité de modélisation.

5.5.1 Systèmes multi-modaux

Les systèmes multi-modaux s'appuient sur un système minimal en ce sens : simultanément sensible aux ressources, non commutatif et non associatif. Cela signifie en particulier que dans les règles logiques, un séquent Γ n'est plus considéré comme une suite de formules $A_1, \ldots, A_n$ mais comme un arbre (binaire) ; l'opération $(\cdot, \cdots)$ n'est plus associative, et les règles tiennent compte de ce contexte. On le note alors $\Gamma[\cdot]$ où $\cdot$ est un nœud de l'arbre identifié par *pattern matching*. Par exemple, on notera $\Gamma[(A_1, (A_2, A_3))]$ pour indiquer que la règle s'applique à tout nœud dont le fils droit n'est pas une feuille. Les règles sont décrites dans le tableau 5.9.

$$\frac{\Gamma \vdash_{\text{MM}} A/B \quad \Delta \vdash_{\text{MM}} B}{(\Gamma, \Delta) \vdash_{\text{MM}} A} /_e \qquad \frac{(\Gamma, B) \vdash_{\text{MM}} A}{\Gamma \vdash_{\text{MM}} A/B} /_i$$

$$\frac{\Gamma \vdash_{\text{MM}} A\backslash B \quad \Delta \vdash_{\text{MM}} B}{(\Gamma, \Delta) \vdash_{\text{MM}} B} \backslash_e \qquad \frac{(B, \Gamma) \vdash_{\text{MM}} A}{\Gamma \vdash_{\text{MM}} B\backslash A} /_i$$

$$\frac{\Delta \vdash_{\text{MM}} (A \bullet B) \quad \Gamma[(A, B)] \vdash_{\text{MM}} C}{\Gamma[\Delta] \vdash_{\text{MM}} C} \bullet_e \qquad \frac{\Gamma \vdash_{\text{MM}} A \quad \Delta \vdash_{\text{MM}} B}{\Gamma, \Delta \vdash_{\text{MM}} A \bullet B} \bullet_i$$

TAB. 5.9 – Règles d'inférence pour les opérateurs binaires

D'autre part, on ne dispose plus seulement des seuls connecteurs $\backslash$, $/$ et $\otimes$, mais de familles de connecteurs $\backslash_i$, $/_i$ et $\otimes_i$. Chacune de ces familles suit les mêmes règles. Mais, pour qu'elles puissent interragir, on dispose de postulats d'inclusion, qui sont autant de règles supplémentaires, de même

que de règles spécifiant les propriétés structurelles des connecteurs d'une même famille. Ces règles, à définir suivant la nécessité de la modélisation au même titre que le nom et le nombre de catégories et que le lexique, peuvent prendre les aspects suivants (en supposant que l'on a une famille c commutative et une famille a associative) :

$$\frac{\Gamma[(A,B)^c] \vdash_{\text{MM}} F}{\Gamma[(B,A)^c] \vdash_{\text{MM}} F} \qquad \frac{\Gamma[(A,(B,C)^a)^a] \vdash_{\text{MM}} F}{\Gamma[((A,B)^a,C)^a] \vdash_{\text{MM}} F} \qquad \frac{\Gamma[(A,B)^c] \vdash_{\text{MM}} F}{\Gamma[(A,B)^a] \vdash_{\text{MM}} F}$$

Les deux premières règles indiquent que c et a se comportent respectivement comme des familles de connecteurs commutatifs et associatifs, la dernière règle permettant de transformer tout contexte commutatif en un contexte associatif.

Notons enfin que ce système s'augmente également de familles d'opérateurs unaires qui permettent de préciser finement comment tel mot va autoriser certaines modifications structurelles des autres opérateurs. Le tableau 5.10 décrit les règles pour ces opérateurs unaires. Comme $\Gamma[(A,B)]$ décrit la structure du séquent, avec la virgule pour opérateur binaire, il y a un opérateur unaire sur la structure : $\langle\Gamma\rangle$.

$$\frac{\Delta \vdash_{\text{MM}} \Diamond A \quad \Gamma[\langle A\rangle] \vdash_{\text{MM}} B}{\Gamma[\Delta] \vdash_{\text{MM}} B}\,\Diamond_e \qquad \frac{\Gamma \vdash_{\text{MM}} A}{\langle\Gamma\rangle \vdash_{\text{MM}} \Diamond A}\,\Diamond_i$$

$$\frac{\Gamma \vdash_{\text{MM}} \Box A}{\langle\Gamma\rangle \vdash_{\text{MM}} A}\,\Box_e \qquad \frac{\langle\Gamma\rangle \vdash_{\text{MM}} A}{\Gamma \vdash_{\text{MM}} \Box A}\,\Box_i$$

TAB. 5.10 – Règles d'inférence pour les opérateurs unaires

Cela permet de rendre compte de phénomènes aussi variés que l'extraction de branche droite (comme dans les relatives en anglais ou en français), de l'extraction de branche gauche (comme dans les relatives en néerlandais ou en allemand) [Moo99a], ou les *negative polarity items* [Ber00]. Cette approche fait l'objet d'une implémentation [Moo99b].

Par exemple pour obtenir une dérivation de *le livre que Marie lit avidement*, on veut pouvoir dériver le type *S* de *Marie (lit X)* pour obtenir *(Marie (lit X)) avidement*. Puis, on veut pouvoir modifier la *structure* de l'expression, afin de rendre X extractible, et donc de le placer à l'extrémité. C'est-à-dire obtenir *((Marie lit) X) avidement* puis *((Marie lit) avidement) X*. Ceci bien entendu ne doit être possible qu'en présence du relatif *que*. C'est pourquoi ce dernier sera typé : $rel = (n\backslash n)/(S/\Diamond\Box np)$. Ce typage indique :

- qu'il permet l'ancien typage, et donc les anciennes dérivations (car $\Diamond\Box A \vdash_{\text{MM}} A$ et $A \vdash_{\text{MM}} \Box\Diamond A$) ;
- il spécifie qu'il demande un *np* structuré, sachant que cette structure justement *autorise* la modification structurelle dans la relative.

Ainsi, on accompagne le typage du relatif par les deux postulats suivants :

$$\frac{\Gamma[(A,(B,\langle C\rangle))] \vdash_{\text{MM}} D}{\Gamma[((A,B),\langle C\rangle)] \vdash_{\text{MM}} D}\,P_1 \qquad \frac{\Gamma[((A,\langle B\rangle),C)] \vdash_{\text{MM}} D}{\Gamma[((A,C),\langle B\rangle)] \vdash_{\text{MM}} D}\,P_2$$

Cela permet la dérivation suivante :

$$D_1 = \dfrac{\dfrac{np \vdash_{MM} np \qquad \dfrac{(np\backslash S)/np \vdash_{MM} (np\backslash S)/np \qquad \dfrac{\Box np \vdash_{MM} \Box np}{\langle \Box np\rangle \vdash_{MM} np}\Box_e}{(np\backslash S)/np, \langle \Box np\rangle \vdash_{MM} np\backslash S}/_e}{np, ((np\backslash S)/np, \langle \Box np\rangle) \vdash_{MM} S}\backslash_e}{(np, (np\backslash S)/np), \langle \Box np\rangle \vdash_{MM} S}P_1 \qquad \dfrac{}{S\backslash S \vdash_{MM} S\backslash S}\backslash_e$$
$$\dfrac{((np, (np\backslash S)/np), \langle \Box np\rangle), S\backslash S) \vdash_{MM} S}{((np, (np\backslash S)/np), S\backslash S), \langle \Box np\rangle \vdash_{MM} S}P_2$$

Puis, en utilisant D_1 :

$$\dfrac{(n\backslash n)/(S/\Diamond\Box np) \vdash_{MM} (n\backslash n)/(S/\Diamond\Box np) \qquad \dfrac{\Diamond\Box np \vdash_{MM} \Diamond\Box np \qquad D_1}{((np, (np\backslash S)/np), S\backslash S), \Diamond\Box np \vdash_{MM} S}\Diamond_e}{(np, (np\backslash S)/np), S\backslash S \vdash_{MM} S/\Diamond\Box np}/_i}{(n\backslash n)/(S/\Diamond\Box np), \underset{que}{} ((\underset{Marie}{np}, \underset{lit}{(np\backslash S)/np}), \underset{avidement}{S\backslash S}) \vdash_{MM} n\backslash n}/_e$$

Le choix dans cette approche est d'avoir une logique de base non associative et non commutative, puis d'autoriser des modifications structurelles au cas par cas, en faisant porter par les items lexicaux les caractéristiques qui autorisent ces modifications structurelles définies par des postulats. [MP99] présente également des réseaux de preuve pour les grammaires catégorielles.

5.5.2 Calcul ordonné et mots comme modules

Cette autre approche destinée à étendre le calcul de Lambek repose sur les propriétés de la logique linéaire. En effet, celle-ci permet de mélanger l'utilisation de connecteurs commutatifs et non commutatifs, en particulier pour la logique non commutative [dG96, Rue97] et le calcul ordonné (pomset calculus) [Ret97b]. Seul ce dernier a pour l'instant fait l'objet de développement pour la linguistique informatique.

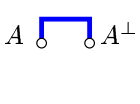
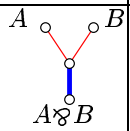
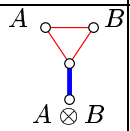
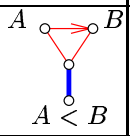
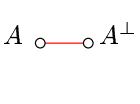
Réseaux ordonnés

Ce calcul est basé sur l'utilisation des réseaux. Il utilise un connecteur, le *précède* ($<$), auto-dual ($(A < B)^\perp = A^\perp < B^\perp$). Les liens sont définis dans le tableau 5.11. La définition des préréseaux est inchangée (en fait, dans le cas général, on peut ajouter un ordre sur les conclusions et les coupures, mais nous n'utilisons pas cette possibilité et pour nous, l'ordre est vide). Les æ-chemins sont désormais orientés, et la définition des réseaux n'est pas modifiée :

Définition 40. Un réseau ordonné est un préréseau qui ne contient pas d'æ-cycle.

À la différence des réseaux multiplicatifs, il n'y a pas pour les réseaux de théorème de séquentialisation. En effet, à ce jour, il n'y a pas de calcul des séquents correspondant, et probablement pas pour cette forme exacte de réseaux. En effet, nous avons montré qu'il n'existait pas d'interpolant pour la preuve de

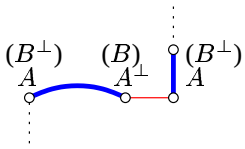
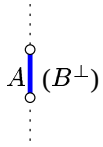
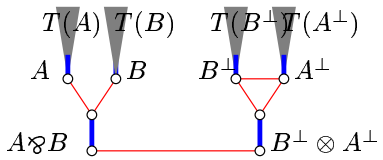
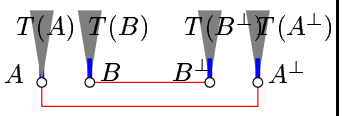
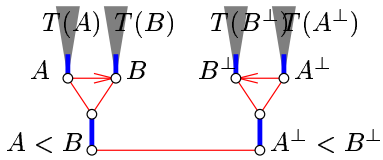
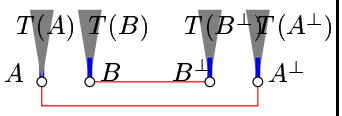
$$(x_5 < x_6) \otimes x_1, ((x_5^\perp < x_4) \parr x_3) < (x_6^\perp \otimes x_2),$$
$$x_{10} < x_1^\perp, (x_7 \parr x_9) < x_2^\perp, (x_7^\perp \parr x_{10}^\perp \parr x_8) \otimes x_3^\perp, (x_8^\perp \parr x_9^\perp) \otimes x_4^\perp$$

Lien	*axiome*	*par*	*tenseur*	*précède*	*cut*
Prémisses	Aucune	A et B	A et B	A et B	A et $A^\perp$
Graphes					
Conclusions	A et $A^\perp$ ($A \in \mathcal{P}$)	$A \parr B$	$A \otimes B$	$A < B$	Aucune

TAB. 5.11 – Les *liens*

utilisant les liens axiomes 1, 2, 3 et 4. Néanmoins, ce calcul des réseaux vérifie l'élimination des coupures telles qu'elles sont définies dans le tableau 5.12 :

Proposition 55 ([Ret93]). *Les liens coupures peuvent être éliminés. Plus précisément, si* Π *est un réseau ordonné dont les conclusions sont* $C_1, \ldots, C_k, \bullet_1, \ldots, \bullet_p$ *(où les* C_i *sont des formules et les* $\bullet_i$ *des coupures), il est possible de réécrire* Π *en* Π' *de conclusions* $C_1, \ldots, C_k$*, sans coupure. De plus, cette réécriture est fortement normalisable et confluente.*

	Redex	Réduit
Avec un axiome		
Entre par et tenseur		
Entre précède et précède		

TAB. 5.12 – Élimination des coupures

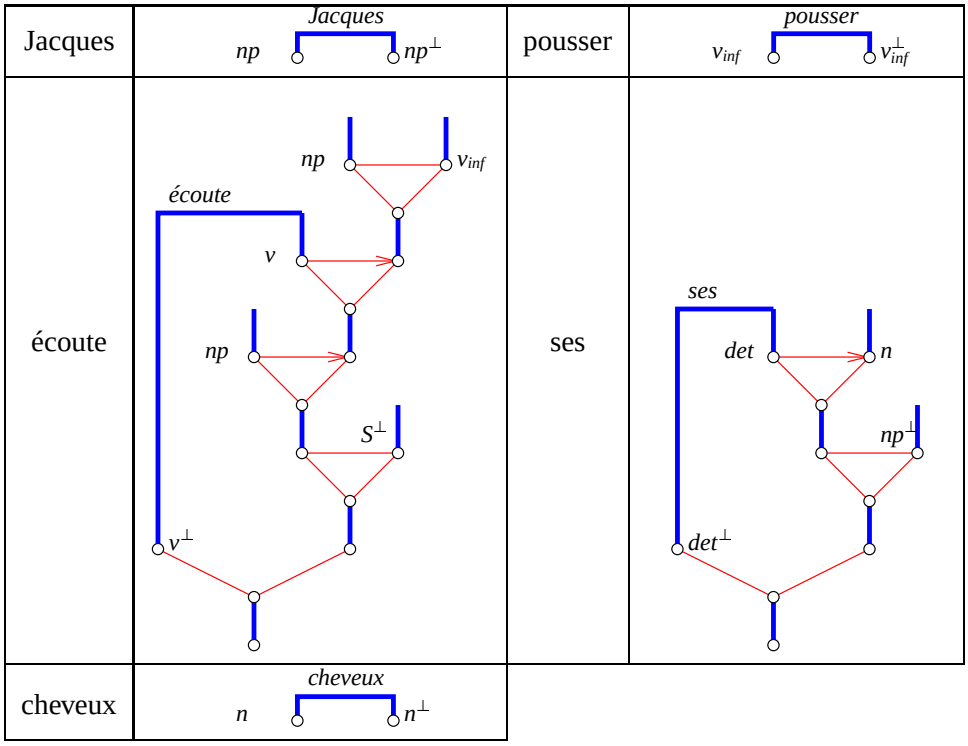
Pour étendre le calcul de Lambek, le principe [LR95] est d'associer dans le lexique les mots non plus à des formules, mais à des modules, c'est-à-dire des structures auxquelles il ne manque que des liens axiomes pour être des préréseaux ou des réseaux. Deux modules se connectent entre eux soit par les liens axiomes manquants, soit par coupure sur deux de leurs conclusions. Les liens précède permettent d'ordonner les axiomes. Dans les modules, ceux-ci sont étiquetés par les mots et permettent donc de calculer un ordre sur ces mots.

Par exemple, avec le lexique du tableau 5.13, on peut construire le réseau de la figure 5.8(c).

L'ordre partiel sur les mots indique que :

$$\begin{aligned} \textit{Jacques} &< \textit{écoute} \\ \textit{ses} &< \textit{cheveux} \\ \textit{écoute} &< \textit{(ses cheveux)} \\ \textit{écoute} &< \textit{pousser} \end{aligned}$$

et rend possible aussi bien *Jacques écoute pousser ses cheveux* et *Jacques écoute ses cheveux pousser*.

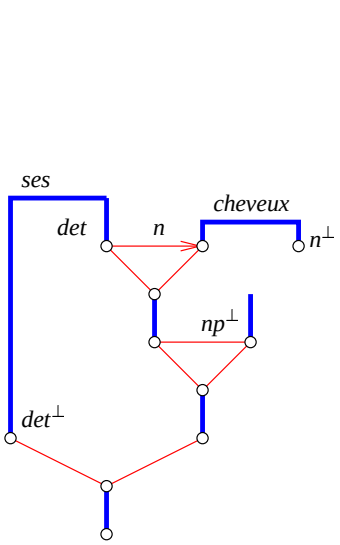


TAB. 5.13 – Lexique avec les réseaux ordonnés

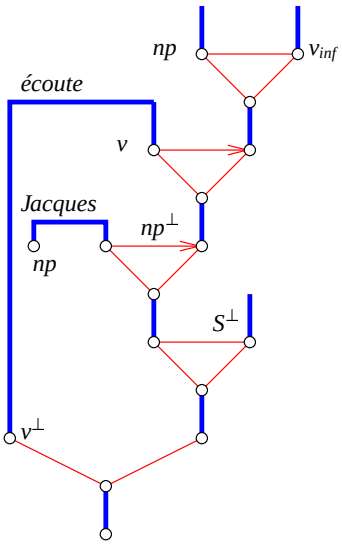
Cette approche permet de modéliser également des phénomènes plus courants tels que le *head wrapping* [LR95]. Cette fois, il s'agit de faire porter par chacune des conclusions d'un lien axiome d'une seule entrée lexicale un élément prosodique propre. De cette façon, les deux éléments prosodiques vont pouvoir « entourer» d'autres éléments venus d'entrées lexicales différentes. Illustrons cette possibilité par un exemple avec la négation en français.

Cette fois, la connexion entre modules qui permet d'entourer des éléments est la coupure. La négation est portée par un module très simple (voir le lexique au tableau 5.14). On remarque que le *ne* et le *pas* sont portés par chacune des conclusions de l'axiome, et que la conclusion du module est $neg^{\perp} \parr neg$.

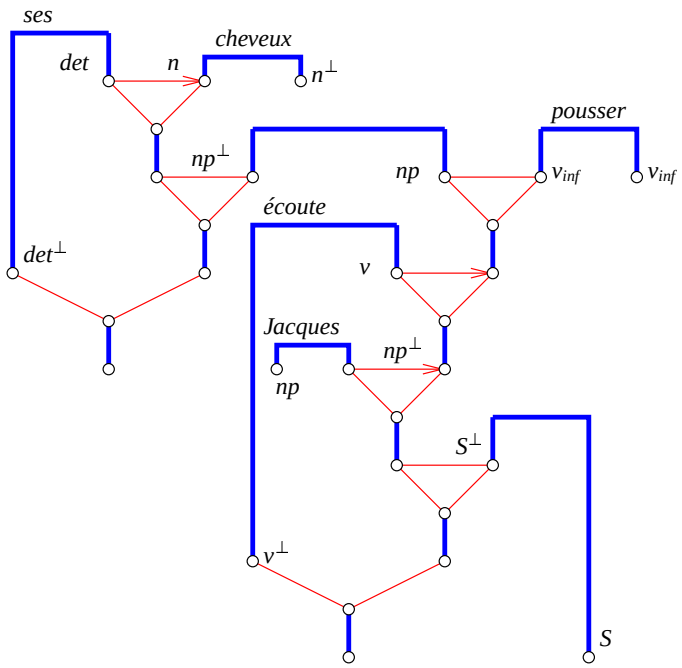
Pour pouvoir accepter ce module, qui n'a pas de prémisse libre, il faut que le module correspondant au verbe puisse l'accepter, c'est-à-dire qu'il ait une conclusion $neg \otimes neg^{\perp}$. De plus, les prémisses de cette conclusion doivent envelopper correctement le module verbal. Ainsi le lexique doit tenir compte de la possibilité pour chaque verbe d'avoir une forme plus générale, capable de tenir compte de la négation, comme l'indique le lexique du tableau 5.14. Remarquons tout de suite, par rapport à l'exemple précédent, que si aucun module de négation n'est connecté au module verbal,



(a) Formation de *ses cheveux*



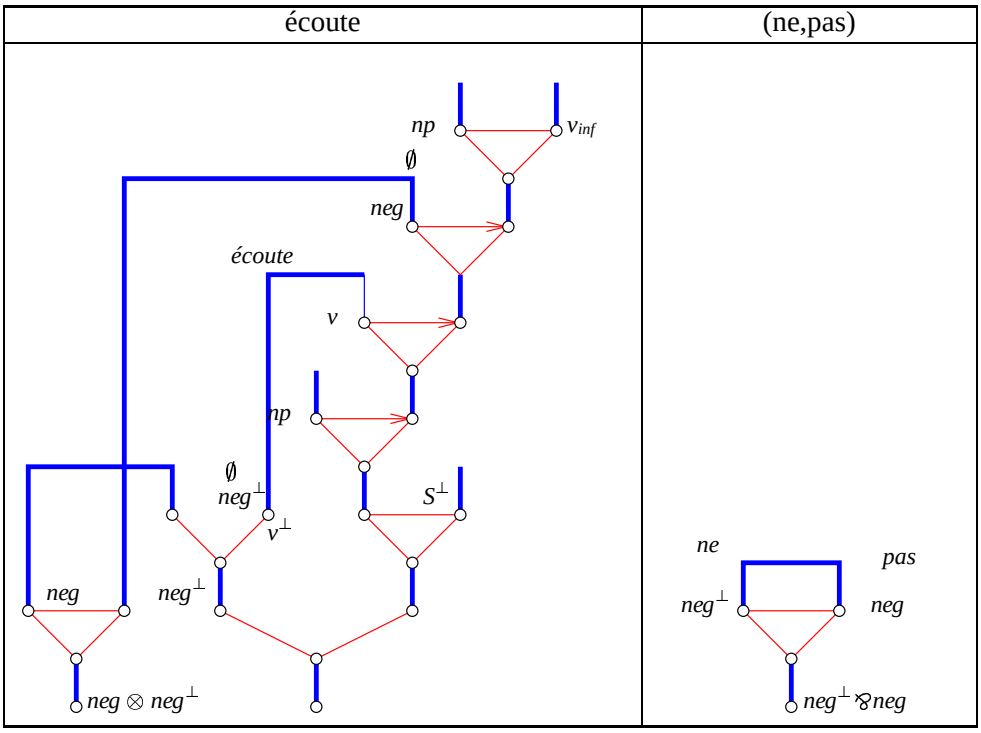
(b) Formation de *Jacques écoute*



(c) Formation de *Jacques écoute ses cheveux pousser*

FIG. 5.8 – Preuve pour *Jacques écoute pousser ses cheveux* et *Jacques écoute ses cheveux pousser*

les éléments prosodiques sont vides et n'ont aucune influence sur le calcul de l'ordre des ensembles. On retrouve ainsi la forme positive.



TAB. 5.14 – Lexique tenant compte de la négation

Le processus de construction de l'analyse reste le même que précédemment, avec la possibilité supplémentaire de connecter les modules par des coupures. On obtient la négation comme le montre la figure 5.9.

Ces exemples montrent comment tirer parti des ordres partiels pour étendre le calcul de Lambek. [LR97] montre également la manière de modéliser des pronoms clitiques en français, où l'objet par exemple prend place devant le verbe :

Pierre le *voit*

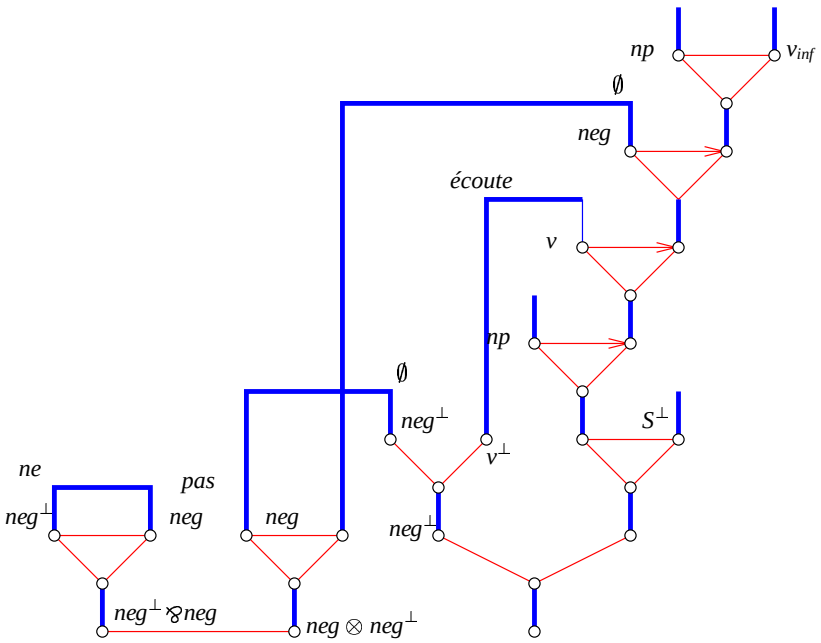
Cette approche permet d'aller au-delà des langages définis par d'autres formalismes, tels que les LTAGs [JLT75, Abe93] (*Lexicalized Tree Adjoining Grammars*, ou grammaires lexicalisées d'arbres adjoints).

En effet, ces grammaires sont des grammaires d'arbres, munies de deux opérations : l'adjonction et la substitution, qui sont des grammaires légèrement contextuelles (*mildly context sensitive grammars*) et permettent d'engendrer des langages tels que $a^n b^n c^n d^n$.

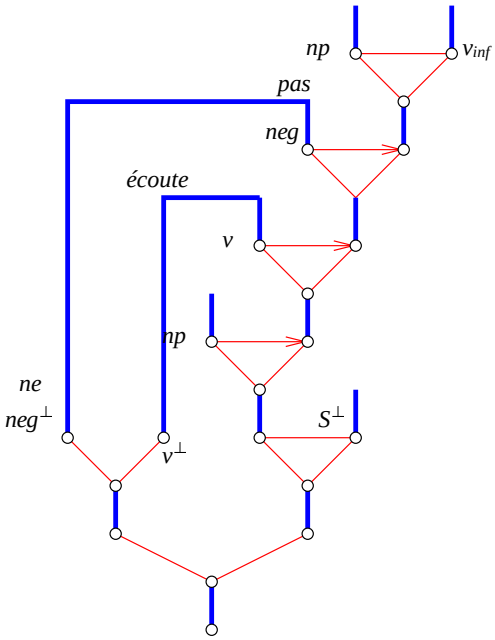
Et nous avons montré [Pog98] qu'un petit fragment du calcul ordonné était équivalent aux LTAGS. Cela montre en particulier que les modules sont au-delà du calcul de Lambek et des langages qu'il engendre. Pour cela nous considérons des modules dont les conclusions sont des formules de $\mathcal{C}$ où $\mathcal{C}$ est défini à partir des atomes $\mathcal{A}$ par :

$$\mathcal{B}_1 ::= \mathcal{A}^{\perp} | \mathcal{A}^{\perp} \parr \mathcal{B}_1 \qquad \mathcal{B}_2 ::= \mathcal{A} | \mathcal{B}_2 < \mathcal{A}$$

$$\mathcal{C} ::= \mathcal{A} | \mathcal{A}^{\perp} | \mathcal{A} \otimes \mathcal{A}^{\perp} | \mathcal{A} \parr \mathcal{A}^{\perp} | \mathcal{B}_1 \parr (\mathcal{B}_2 \otimes \mathcal{A}^{\perp})$$



(a) Connexion des modules par coupure



(b) Élimination de la coupure et propagation des éléments prosodiques

FIG. 5.9 – La négation en français avec le calcul ordonné

On présente une correspondance entre arbres lexicalisés et modules intuitionnistes simplement lexicalisés (SLIPN) qui permet, à partir d'un lexiques d'arbres, de construire un lexique de modules.

De même, les opérations de substitution et d'adjonction sont simulées par des opérations correspondantes (à partir de coupures) sur les modules.

On a alors le théorème suivant :

Théorème 56 ([Pog98]). *Tout arbre dérivé T correspond à un SLIPN, ce dernier s'obtenant à partir des connexions correspondantes aux opérations d'adjonction et de substitution entre les arbres du lexiques qui permettent la dérivation de T.*

La preuve de ce théorème, ainsi que la définition des correspondances entre arbres et modules, et entre opérations sur les arbres et opérations sur les modules sont données en annexe C.

Outre une étude précise des phénomènes linguistiques dont cette approche permettrait de rendre compte, et de la classe des langages qu'elle engendre, il reste encore à définir comment la description syntaxique permettrait de retrouver une analyse sémantique. En effet, il n'y a pas de correspondance évidente à la Curry-Howard. Néanmoins, cette extension conservatrice du calcul de Lambek permet de traiter un plus grand nombre de phénomènes que ce dernier.

Enfin, ce principe de considérer des preuves incomplètes ou des modules associés aux mots se retrouve dans l'approche des arbres de preuve partiels (*partial proof trees*) [JK97b, JK97a]. Cette fois, ce sont des arbres de preuve en déduction naturelle qui sont associés aux mots. Là aussi des opérations permettent de les combiner pour obtenir des preuves complètes.

5.5.3 Traits morphologiques

Les différents lexiques utilisés jusqu'à présent ne tenaient pas compte des traits morphologiques tels que le genre ou le nombre. La correction syntaxique n'en tenait donc pas compte davantage. Cette section montre comment intégrer ces paramètres.

Une première idée consiste à ajouter des traits du premier ordre (c'est-à-dire des traits atomiques) aux items lexicaux, et de vérifier que lors de la dérivation, ces traits s'unifient. Le lexique s'augmenterait ainsi :

le	np/n	gen=Masc, nbre=Sg
train	n	gen=Masc, nbre=Sg
maison	n	gen=Fém, nbre=Sg

et permettrait la dérivation de *le train* comme *np*, mais pas de **le maison* comme *np*, dérivations toutes deux autorisées sans les traits.

Ce principe rejoint celui des grammaires d'unification telles que LFG, GPSG ou HPSG. Malheureusement, la grammaire devient dans ce cas un système hybride utilisant déduction et unification. En fait, [vB90, Mor94] montrent comment ce principe d'unification pour les grammaires de types logiques (et en fait également pour les grammaires d'unification) peut s'exprimer dans un système déductif enrichi : celui des types dépendants.

Dans ce cadre, les formules atomiques se composent des catégories propositionnelles et de catégories prédicatives sur des termes composés de *traits* constants, variables et fonctionnels. Ainsi, la catégorie d'un syntagme nominal comme *la maison* deviendra $np(f)$, ou même $NP(3(f))$ si l'on veut indiquer que le nombre peut être raffiné par le genre.

Puisque dans nos travaux, et en particulier dans le prototype, nous n'avons pas du tout fait usage de cette possibilité de traitement des genres et des nombres, nous ne voulons pas décrire ici précisément le système des types dépendants, mais juste en donner un aperçu. Si l'on a un type *Gen* (avec m et f des variables de ce type par exemple), dire que $np(m)$ et $np(f)$ sont bien des types, cela signifie qu'on a un type, noté $\Pi g^{Gen} np(g)$ qui permet de construire à partir de tout élément g de type *Gen*

un nouveau type $np(g)$. On voit que les déclarations des types mélangent types et termes dans les formules. Les règles d'introduction et d'élimination du type $\Pi\alpha^T\, T'(\alpha)$ sont donc les suivantes :

$$\frac{\Gamma \vdash t : \Pi x^T\, T' \quad \Gamma \vdash t' : T}{\Gamma \vdash tt' : T'[t/x]}\,\Pi_e \qquad \frac{\Gamma, x : T \vdash t : T'}{\Gamma \vdash \lambda x.t : \Pi x^T\, T'}\,\Pi_i$$

Le lexique :

bon	$n(m)/n(m)$
une	$np(f)/n(m)$
un	$np(m)/n(m)$
après-midi	$\Pi g^{Gen}\, n(g)$

permet la construction des dérivations suivantes :

$$\frac{\beta : np(m)/n(m) \qquad \dfrac{\alpha : \Pi g^{Gen}\, n(g) \vdash \alpha : \Pi g^{Gen}\, n(g) \quad m : Gen \vdash m : Gen}{\alpha : \Pi g^{Gen}\, n(g) \vdash \alpha m : n(m)}\,\Pi_e}{\underset{un}{\beta : np(m)/n(m)}, \quad \underset{après\text{-}midi}{\alpha : \Pi g^{Gen}\, n(g)} \quad \vdash \beta(\alpha m) : np(m)}\,/_e$$

$$\frac{\beta : np(f)/n(f) \qquad \dfrac{\alpha : \Pi g^{Gen}\, n(g) \vdash \alpha : \Pi g^{Gen}\, n(g) \quad f : Gen \vdash f : Gen}{\alpha : \Pi g^{Gen}\, n(g) \vdash \alpha f : n(f)}\,\Pi_e}{\underset{une}{\beta : np(f)/n(f)}, \quad \underset{après\text{-}midi}{\alpha : \Pi g^{Gen}\, n(g)} \quad \vdash \beta(\alpha f) : np(f)}\,/_e$$

$$\frac{\beta : np(m)/n(m) \qquad \dfrac{\gamma : n(m)/n(m) \qquad \dfrac{\alpha : \Pi g^{Gen}\, n(g) \vdash \alpha : \Pi g^{Gen}\, n(g) \quad m : Gen \vdash m : Gen}{\alpha : \Pi g^{Gen}\, n(g) \vdash \alpha m : n(m)}\,\Pi_e}{\gamma : np(m)/n(m), \alpha : \Pi g^{Gen}\, n(g) \vdash \beta(\alpha m) : n(m)}\,/_e}{\underset{un}{\beta : np(m)/n(m)} \quad \underset{bon}{\gamma : n(m)/n(m)} \quad \underset{après\text{-}midi}{\alpha : \Pi g^{Gen}\, n(g)} \quad \vdash \beta(\gamma(\alpha m)) : np(m)}$$

mais évidemment pas **une bon après-midi.*

Il conviendrait d'être plus précis sur les règles de formation des types et des λ-termes, mais cette section donne ainsi un aperçu de la manière dont les traits morphologiques peuvent être pris en compte d'un point de vue de la syntaxe et de la sémantique, tout en restant dans des systèmes logiques (en particulier avec la correspondance de Curry-Howard). Contrairement aux grammaires d'unification, l'unification n'est pas un critère de grammaticalité, mais tout simplement (éventuellement) le moyen de mettre en œuvre la recherche de preuve avec des types dépendants. Pour un traitement plus exhaustif des possibilités de cette approche, nous renvoyons le lecteur à [Mor94] ou [Ran98].

Chapitre 6

Génération

Lors de l'utilisation des grammaires de types logiques pour modéliser le comportement des langues naturelles, l'essentiel de la réflexion s'est porté sur la vérification de la grammaticalité des phrases, et leur analyse sémantique, comme l'attestent les très nombreux travaux cités. Le présent chapitre cherche à déterminer les propriétés de réversibilité de ces grammaires et les moyens, à partir d'énoncés sémantiques, d'engendrer des expressions, dans une ou plusieurs langues (suivant le choix du lexique), qui respectent cet énoncé.

Globalement, on recherche les mots à utiliser (sélection des items lexicaux) et la manière de les combiner (réalisation syntaxique). Reformulé avec les principes des grammaires de types logiques, ce dernier objectif s'énonce : « à partir du type syntaxique d'élements lexicaux donnés, trouver une preuve de S ». Cela suggère de partager le traitement en deux :

- d'une part en choisissant les items lexicaux susceptibles d'apparaître dans la phrase ;
- d'autre part en cherchant un arrangement de ces items lexicaux qui forme une phrase syntaxiquement correcte (donc trouver une preuve) dont le sens est bien celui attendu.

Bien entendu, chacun de ces traitements peut (voire doit) faire intervenir au plus tôt la notion de preuve et la notion de sens calculé. Ce chapitre évoque plus précisément ce point dans la section 6.4.5, et l'on voit que l'on va être guidé par les constantes typées présentes dans les items lexicaux et dans le sens final.

Néanmoins, ce chapitre se concentre essentiellement sur le deuxième traitement, appelé *réalisation syntaxique*, qui suppose choisis les items lexicaux.

Peu de travaux reflètent cette problématique globale pour les grammaires de types logiques, alors que différents travaux la présentent pour les grammaires d'unification en général et les grammaires catégorielles d'unification en particulier. Après une description formelle des problèmes de génération et de réversibilité des grammaires à la section 6.1 qui se base sur [Dym92, Dym94], nous donnerons quelques-unes des réponses apportées pour d'autres types de grammaires. Plus précisément, la section 6.2.1 souligne les points communs et les différences entre ces approches et celle que nous proposons. La section 6.2.2 présente le principal des travaux dévolus à la réalisation syntaxique pour les grammaires de types logiques, à partir d'une représentation sémantique à la Montague et la correspondance de Curry-Howard, et en discute les avantages et les inconvénients. Puis la section 6.3 reformule le problème grâce aux réseaux sémantiques de la section 5.4.1. Cette reformulation nous permet de proposer une solution se basant sur une recherche de preuve originale, à l'aide de la géométrie de l'interaction présentée à la section 2.5. En particulier, nous pouvons caractériser les lexiques permettant une réalisation syntaxique en temps polynomial.

6.1 Grammaires réversibles et génération

Habituellement, Les approches de la génération en langues naturelles distinguent deux phases : la première, dite *stratégique* [McK85] ou *conceptuelle* [vN93], définit le message à communiquer et en fournit une représentation. La deuxième, dite *tactique* ou *grammaticale* s'attache à construire une expression (une suite de mots lorsque la communication est écrite) à partir de la représentation qui lui est fournie.

Seule la partie tactique nous intéresse ici. Nous supposons construite la représentation, sous forme de λ-terme ou de réseau, qui sert de base à la génération d'expressions. En particulier, elle peut résulter de l'analyse sémantique d'une autre expression dans la même langue ou dans une autre.

6.1.1 Motivations

Pour traiter précisément de la réversibilité et des problèmes formels qui en découlent, la section suivante donnera les définitions des différentes notions de réversibilité et des problèmes calculatoires qui s'y rattachent. Néanmoins, dans un premier temps, nous considérons que la propriété de réversibilité d'une grammaire exprime sa capacité à être utilisée aussi bien en analyse qu'en génération.

Suivant [vN93], les raisons pour obtenir une telle grammaire sont linguistiques, psychologiques et techniques. D'un point de vue linguistique, la caractérisation d'un langage que permet une grammaire ne privilégie ni la compréhension d'expressions ni la production d'expressions. De plus, le but d'une grammaire étant de définir les relations entre des expressions et des sens, une théorie devrait permettre de prévoir le sens qu'une grammaire associe à une expression, et réciproquement de prévoir quelles expressions elle associe à un sens donné.

Le fait qu'un locuteur peut parfaitement comprendre des phrases que jamais lui-même ne produirait semble infirmer d'un point de vue psychologique la pertinence des grammaires réversibles. Cependant, diverses raisons semblent indiquer que ce comportement ne relève pas tant du niveau linguistique que d'autres niveaux cognitifs. Par exemple, si le contexte est assez précis, un lecteur peut parfaitement comprendre une phrase contenant un mot qui lui est inconnu. La capacité d'apprentissage dépasse la simple compréhension du langage. De même (pour reprendre un exemple de [vN93]), comprendre la théorie de la relativité telle que décrite par Einstein n'implique certainement pas avoir été capable de la formuler à sa place.

D'un point de vue technique enfin, le développement de grammaires réversibles offre des avantages. Méthodologiquement d'une part, et c'est un argument qu'on retrouve dans [Dym92], l'écriture de la grammaire nécessite une analyse plus abstraite des phénomènes linguistiques qui doit pouvoir se retrouver aussi bien dans l'algorithme d'analyse que celui de génération. On échappe ainsi au risque d'écrire une grammaire trop proche des propriétés de l'algorithme d'analyse ou de génération.

D'autre part, les grammaires pour analyseurs sont généralement *surgénératrices*, c'est-à-dire qu'elles associent des formes logiques à des expressions en fait agrammaticales (voir la discussion sur la coordination à la section 5.4.2) ce qui peut conduire à des fausses ambiguïtés. La réversibilité permet d'éviter ce risque, sur les générateurs également — sauf qu'il s'agit généralement de *sous-génération.*

Cela permet encore d'assurer la cohérence entre le langage reconnu et celui produit. Dans le cas d'une interface homme/machine par exemple, une grammaire réversible reconnaissant le même langage que celui qu'elle produit permet à un utilisateur de prévoir les phrases que le système pourra analyser, et qui sont donc à privilégier, en s'inspirant des phrases que le système produit.

6.1.2 Différentes notions de réversibilité et problèmes de génération

Si la réversibilité semble a priori une propriété souhaitable, il reste à la définir précisément. Une première idée, guidée par l'implémentation, pourrait être que la réversibilité se définit comme la propriété d'un unique programme d'être utilisé aussi bien en analyse qu'en génération. Certainement, un candidat idéal pour ce type d'approche, serait un programme à clauses définies, dont l'analyse de l'expression X pourrait être donnée par y vérifiant la clause $r(X, y)$, et la génération d'une expression x de sens Y vérifiant la clause $r(x, Y)$.

Cependant, que se passe-t-il si à cette requête le programme ne finit pas, ne donne pas de réponse, ou en donne une infinité ? Quelle part de ce comportement attribuer au programme lui-même et à la grammaire qu'il implémente ?

Pour étudier les propriétés algorithmiques intrinsèques de la grammaire, [Dym92, Dym94] introduit divers concepts, initialement liés à la programmation logique, mais dont la validité s'étend naturellement. Pour une grammaire G, on considère la relation $r_G(x, y)$ telle que $r_G(x, y)$ si et seulement si l'expression x admet comme représentation sémantique y. On considère alors les problèmes suivants [Dym92] :

- le *problème de p-énumération*, ou *problème d'analyse*, est le problème consistant à énumérer, pour toute valeur de x fixée, les y tels que $r_G(x, y)$. Le *problème de p-acceptation* ou *problème de décision pour l'analyse* est le problème consistant à vérifier à x fixé qu'il existe y tel que $r_G(x, y)$.
  La p-énumération est *finiment énumérable* si à x fixé, il existe un programme qui énumère tous les y tels que $r_G(x, y)$ et qui termine.
  S'il existe un programme soit capable de dire en temps fini qu'à x donné il n'existe pas de y tel que $r_G(x, y)$, soit capable d'énumérer les y tels que $r_G(x, y)$ sans nécessairement finir — par exemple s'il y a un nombre infini de solutions — on dit que le problème de p-énumération est *découvrable* ;
- le *problème de g-énumération*, ou *problème de génération*, est le problème consistant à énumérer, pour toute valeur de y fixée, les x tels que $r_G(x, y)$. Le *problème de g-acceptation* ou *problème de décision pour la génération* est le problème consistant à vérifier à y fixé qu'il existe x tel que $r_G(x, y)$.
  La g-énumération est *finiment énumérable* si à y fixé, il existe un programme qui finit et qui énumère tous les x tels que $r_G(x, y)$.
  S'il existe un programme soit capable de dire en temps fini qu'à y donné il n'existe pas de x tel que $r_G(x, y)$, soit capable d'énumérer les x tels que $r_G(x, y)$ sans nécessairement finir — par exemple s'il y a un nombre infini de solutions — on dit que le problème de g-énumération est *découvrable*.

Dans ces définitions, on a pris soin de ne rien dire des programmes P_p et P_g qui peuvent instencier ces propriétés sur les différents problèmes pour une grammaire donnée. Plus que la notion d'*uniformité d'implémentation*, c'est-à-dire savoir si l'on peut trouver $P_p = P_g$, on va s'intéresser à la capacité pour les grammaires de types logiques d'induire de bonnes propriétés des problèmes de génération et d'analyse. Selon la terminologie de [Dym92], on va s'intéresser à la *réversibilité intrinsèque* de ces grammaires.

Définition 41. Une grammaire G est (intrinsèquement) *finiment réversible* si et seulement si :

1. l'analyse est finiment énumérable ;
2. la génération est finiment énumérable.

Une grammaire G est donc finiment réversible s'il existe un programme P_p et un programme P_g, pas nécessairement les mêmes, tels que :

1. à x fixé, P_p renvoie la liste finie des y tels que $r_G(x, y)$ et s'arrête ;
2. à y fixé, P_g renvoie la liste finie des x tels que $r_G(x, y)$ et s'arrête.

En particulier, l'existence de P_p et celle de P_g sont *indépendantes* en général.

Ainsi on trouvera dans [Dym91, Dym92] :

– une grammaire pour laquelle l'analyse est finiment énumérable et la g-acceptation n'est pas décidable (construite à partir du théorème de Matiyasevich [Cut80] qui apporte une réponse négative au dixième problème de Hilbert : « Existe-t-il un algorithme capable de résoudre les équations diophantiennes ? ») ;
– une grammaire dont la génération est finiment énumérable et la p-acceptation n'est pas décidable (construite à partir du théorème de Church [CL93b] prouvant l'indécidabilité du calcul des prédicats).

Les propriétés de réversibilité d'une grammaire ne peuvent donc s'obtenir qu'à la condition que cette grammaire vérifie un certain nombre de contraintes. Les sections suivantes indiquent pour différentes classes de grammaires les résultats de réversibilité les concernant. Malgré la différence entre les grammaires d'unification et les grammaires de types logiques, la rareté du traitement de la génération pour ces dernières nous incite à relever quelques approches de la génération pour les premières, en soulignant certaines similarités.

6.2 Génération et types de grammaires

Une des raisons de la rareté des approches de la génération à partir d'une représentation sémantique sous forme de λ-terme tient à la difficulté d'inverser le processus de réduction des λ-termes, comme l'indique [vEM92] : « It is simpler and more efficient to use the feature system and unification to do explicitly what lambda expressions and lambda reduction do implicitly, that is, assign a value to a variable embedded in a logical form expression. »

De même, [Als92] justifie l'utilisation de grammaires d'unification par : « If the semantic rules had been more in the style of traditional Montague semantics, generation from structures that had undergone lambda reductions would have presented search difficulties because the reductions would have to be applied in reverse during the generation process. »

La section 6.4 infirme une partie de ces arguments en proposant un moyen efficace d'inverser le processus, sans recours à l'unification de λ-termes. Néanmoins cela justifie que nous consacrions la section 6.2.1 aux grammaires d'unification avant d'aborder la proposition de [MM97] à la section 6.2.2 pour le calcul de Lambek avec une représentation sémantique à la Montague.

6.2.1 Grammaires d'unification

Parmi les grammaires d'unification, notre intérêt s'est porté naturellement sur les grammaires catégorielles d'unification (UCG) [Usz86a, Usz86b, Kar86]. Cependant, les techniques mises en œuvre pour la génération dans ce cadre s'inspirent d'autres travaux sur la génération dans le cadre plus général des grammaires d'unification, en particulier [Shi88] et les grammaires d'unification à base de règles hors-contexte (LFG, PATR). Elles s'appuyent sur des techniques d'analyse de ces grammaires, en particulier l'algorithme de *chart-parsing* de Earley [Ear86]. C'est pourquoi dans cette section, nous présentons tout d'abord cette approche, puis d'autres qu'elle a inspirées et les problèmes propres aux grammaires catégorielles d'unification. Nous conclurons en nous intéressant également aux résultats de décidabilité pour LFG.

Une même architecture pour l'analyse et la génération

Nous avons déjà remarqué que les grammaires logiques offrent un intérêt pour les problèmes de réversibilité, du fait entre autre de l'implémentation commune des algorithmes d'analyse et de génération. S'appuyant sur [PW83, SPKK86], qui montrent comment l'analyse peut être vue comme une déduction logique prouvant la grammaticalité d'une expression (on parle alors de *déduction de Earley* pour la stratégie déductive qui procède de manière analogue à l'algorithme de Earley), [Shi88] suggère que la génération soit vue comme la preuve de l'existence d'une expression vérifiant un certain critère.

Ainsi, sous le contrôle des règles de grammaire, il s'agit de prouver qu' « une expression α est une phrase de sens S » soit avec α donnée (analyse), soit S donné (génération). [Shi88] montre que les deux règles d'inférence de la déduction de Earley, prédiction et complétion, peuvent être utilisées de la même manière pour l'analyse comme la génération, et même comment différentes stratégies de déduction peuvent être implantées, en particulier celle de [PW83] qui permet de simuler l'algorithme de Earley.

À partir des axiomes que constituent les règles de la grammaire, l'objectif de [Shi88] est de raffiner la recherche de la preuve d'un théorème particulier parmi tous ceux de la clôture de la grammaire et des prémisses sous les règles de déduction. Le premier point consiste à conserver les lemmes déjà prouvés, selon les principes de la programmation dynamique, et éviter ainsi les dérivations redondantes. Le deuxième point consiste à éviter l'indéterminisme du processus de recherche de preuve en donnant une priorité à chaque lemme qui doit être prouvé. Le troisième point consiste à avoir un ensemble d'axiomes correspondant à la tâche attendue — analyse ou génération — ainsi qu'un critère de réalisation de cette tâche. En génération par exemple, le critère de réalisation est que le sens recherché subsume le sens de l'expression engendrée.

En génération comme en analyse, cette approche fortement influencée par l'analyse et l'algorithme de Earley, repose sur une définition des items basée sur la position de la fin d'une sous-expression dans une expression. Si en analyse la concaténation des divers éléments valide cette stratégie, elle pose des problèmes en génération. Ainsi, dans un premier temps, [Shi88] envisage-t-il d'avoir comme axiomes initiaux l'ensemble des éléments lexicaux (tous les mots de la grammaires) à chaque position possible dans l'expression finale. Ce problème se résout en ne tenant pas compte, pour la génération, de la position des sous-expressions dans la phrase. En contrepartie, l'avantage de la méthode est perdu et la génération n'est plus orientée. Le calcul de la priorité prend alors toute son importance, pour éviter les éléments dont le sens ne puisse pas être finalement subsumé par le sens recherché. La condition de *monotonie sémantique* alors imposée est que *le sens de chaque sous-expression subsume une partie du sens recherché*

Cette condition est très importante et impose une contrainte pour la grammaire. En effet, par exemple dans le cas des particules verbales en anglais, on peut attribuer [SvNMP89] à *up* la représentation **up**, mais **up** ne contribue pas à la représentation de *call up*, qui a un sens propre **call-up** ou **phone** (voir note 17). Le même problème se pose avec les expressions idiomatiques, comme *casser sa pipe* qui a un sens qui lui est propre : le sens figuré **mourir**. Dans ce cas, la condition de priorité interdira la génération de telles expressions, et le programme pourra analyser des expressions qu'il ne saura pas engendrer.

Seules les grammaires ayant la propriété de monotonie sémantique assurent la réversibilité avec cette approche. Ce sont les grammaires dont pour toute expression grammaticale, la représentation sémantique de chacune des sous-expressions immédiates subsume une partie de l'expression complète. Remarquons que, dans *le cas d'une représentation sous forme de λ-termes linéaires*, cette hypothèse est *vérifiée*.

Néanmoins, cette approche de bas en haut permet d'éviter la récursion gauche d'une approche de

haut en bas naïve. Par exemple celle qui existe dans une règle, dont la syntaxe s'inspire de celle de Prolog, comme

$$\texttt{vp}(\texttt{Subcat}) - \texttt{S} \longrightarrow \texttt{Comp}, \texttt{vp}([\texttt{Comp}|\texttt{Subcat}]) - \texttt{S} \tag{6.1}$$

Cette règle correspond aux relatives en allemand ou en néerlandais où le verbe suit les compléments qu'il sous-catégorise. Chaque expression `syn – sem` sépare la partie syntaxique `syn` de la partie sémantique `sem`.

En ajoutant les règles :

$$\texttt{np} - \texttt{j} \longrightarrow [\texttt{Johann}] \tag{6.2}$$

$$\texttt{np} - \texttt{i} \longrightarrow [\texttt{Ingrid}] \tag{6.3}$$

$$\texttt{vp}([\texttt{np} - \texttt{Obj}, \texttt{np} - \texttt{Subj}]) - \texttt{l}(\texttt{Subj}, \texttt{Obj}) \longrightarrow [\texttt{liebt}] \tag{6.4}$$

on peut calculer la relation $\texttt{vp}([\,]) - \texttt{l(j,i)}$ à partir de l'expression

$$[\texttt{Johann}], [\texttt{Ingrid}], [\texttt{liebt}]$$

en commençant par prouver

$$\texttt{np} - \texttt{j}, \texttt{np} - \texttt{i}, \texttt{vp}([\texttt{np} - \texttt{Obj}, \texttt{np} - \texttt{Subj}]) - \texttt{l}(\texttt{Subj}, \texttt{Obj})$$

puis la règle (6.1) permet d'instancier `Obj` avec `i`, et amène à prouver

$$\texttt{np} - \texttt{j}, \texttt{vp}([\texttt{np} - \texttt{Subj}]) - \texttt{l}(\texttt{Subj}, \texttt{i})$$

qui donne grâce à la règle (6.1) l'instanciation de `Subj` par `j` et le résultat $\texttt{vp}([\,]) - \texttt{l}(\texttt{j}, \texttt{i})$.

Réciproquement, une approche de haut en bas de la génération d'un *vp* `l(j,i)`, à cause de la règle (6.1), n'aboutirait pas car cette règle force la liste de sous-catégorisation à s'augmenter tant qu'une entrée lexicale ne la limite pas. Or cette dernière ne peut être trouvée du fait de la récursion. Notons que d'autres travaux que nous ne détaillerons pas plus se sont penchés sur le problème [DI88, Wed88, Dym92], avec des approches telles que la spécialisation, à partir d'une seule grammaire, en une grammaire équivalente pour l'analyse et une grammaire équivalente pour la génération.

La réponse aux deux problèmes de la récursion gauche et de la monotonie sémantique s'opère par une vision plus éloignée des procédures d'analyse sémantique (droite-gauche) et privilégie les informations données par la forme sémantique. Au lieu d'utiliser la tête syntaxique d'un constituant, c'est la tête sémantique qui va guider la génération. La section suivante donne quelques-unes des approches selon ce point de vue.

Utilisation de la tête sémantique et UCG

L'approche que présente cette partie est commune à de nombreux travaux. Elle rejoint les préoccupations de génération aussi bien pour les grammaires d'unification à squelette hors-contexte que les grammaires catégorielles d'unification. En effet, dans les deux cas, l'accent va être mis sur la représentation foncteur/argument.

L'idée principale de cette approche repose sur la volonté de guider au maximum la génération par l'information sémantique. Ainsi que l'a montré la règle (6.1), un simple procédé unidirectionnel ne suffit pas car la *tête sémantique,* c'est-à-dire l'élément de la partie droite de la règle qui partage sa sémantique avec l'élément de la partie gauche, n'est pas nécessairement tout à gauche ou tout à droite.

Or, il est l'élément le plus approprié à compléter, car il est celui qui apporte l'information sémantique pour ses arguments.

Dans la proposition de [SvNMP89], les règles de la grammaire se séparent en deux catégories : les *chain rules*, dont la forme sémantique d'un des élements de la partie droite de la règle est identique à celle de la partie gauche, comme la règle (6.1), et les *non-chain rules*. Cela permet d'identifier le *nœud pivot* qui se définit comme étant le plus bas dans l'arbre de dérivation, éventuellement à construire, de même sémantique que tous les nœuds qui lui sont supérieurs et ainsi jusqu'à la racine. Ce nœud ne peut pas avoir été produit par une *chain rule*, sinon cela contredirait sa définition, et peut être utilisé comme tête sémantique.

Ce nœud partage l'arbre, et donc l'algorithme, en deux parties : une partie de bas en haut qui construit l'arbre avec les *chain rules*, et une partie de haut en bas avec les *non-chain rules*. En effet, il se décompose de la manière suivante : à partir de la représentation sémantique pour laquelle il faut engendrer une expression, la racine, on cherche une règle dont la partie gauche s'unifie avec la racine et qui soit une *non-chain rule*, pour définir le pivot. L'algorithme est répété récursivement sur chacun des éléments de la partie droite de la règle. On procède donc de haut en bas. La partie de bas en haut cherche à relier le pivot à la racine en utilisant des *chain rules* dont le pivot est tête sémantique. Là encore, après le choix de la tête sémantique, les éléments restants de la partie droite sont engendrés récursivement.

Ainsi, en reprenant l'exemple de la section précédente, pour engendrer une expression $\texttt{vp}([\,])$ de sens $\texttt{l}(\texttt{j},\texttt{i})$, l'unique *non-chain rule* est la règle (6.4). Puisque la partie droite n'a pas de non-terminaux, l'algorithme ne repart pas sur la partie droite. Le pivot est donc $\texttt{vp}([\texttt{np}-\texttt{i},\texttt{np}-\texttt{j}])-\texttt{l}(\texttt{j},\texttt{i})$ (voir figure 6.1(a)) qu'il faut rattacher par des *chain rules* à la racine. Pour cela, on utilise une première fois la règle 6.1 qui permet d'obtenir (voir figure 6.1(b)) :

– un nouveau nœud à relier à la racine : $\texttt{vp}([\texttt{np}-\texttt{j}])$;
– un nouveau non-terminal à engendrer : `Comp`, instancié par $\texttt{np}-\texttt{i}$.

L'étape suivante (voir figure 6.1(c)) utilise encore la règle 6.1 pour enfin relier $\texttt{vp}([\texttt{np}-\texttt{j}])$ à la racine tout en produisant un nouveau non-terminal $\texttt{np}-\texttt{j}$. Les deux instanciations restantes sont lexicales et permettent d'obtenir l'arbre de dérivation de la figure 6.1(d), soit l'expression *Johann Ingrid liebt.*

Cette nouvelle approche répond à la fois au problème de la monotonie sémantique[17] et de la récursion gauche évoquée précédemment. Cependant, comme [SvNMP89] le souligne, un autre cas de récursion peut apparaître, car la sélection de la tête sémantique ne se fait que sur la base du partage de la représentation sémantique avec la partie gauche. Si deux non-terminaux de la partie droite partagent cette représentation sémantique, lequel choisir ? Le mauvais choix pouvant induire, dans le processus de bas en haut, la non-terminaison de l'algorithme. Prenons la *chain rule*

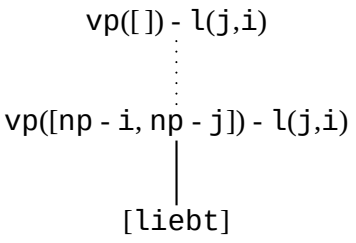
$$\texttt{nbar}-\texttt{N} \longrightarrow \texttt{nbar}-\texttt{N},\texttt{sbar}-\texttt{N} \tag{6.5}$$

Si $\texttt{sbar}-\texttt{N}$ est choisi comme pivot, l'algorithme est relancé sur $\texttt{nbar}-\texttt{N}$ et peut de nouveau choisir la règle (6.5). Les bonnes propriétés provenant de l'imitation de l'algorithme de Earley sont perdues.

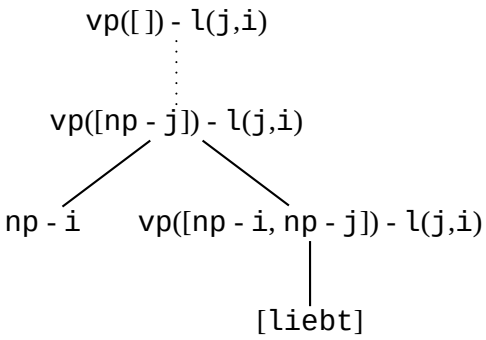
Pour pallier ce problème, [Ger91] propose un algorithme combinant l'approche de [Shi88] basée sur l'algorithme de Earley mais également guidé par la tête sémantique, que nous ne développerons pas ici. Une approche encore plus uniforme que nous ne détaillons pas non plus, avec la même grammaire et le même programme pour l'analyse et la génération se trouve dans [Neu94].

Les algorithmes de génération pour les grammaires catégorielles d'unification utilisent la même approche [GH90, CRZ89], en parlant plutôt de foncteur que de tête sémantique. La justification vient

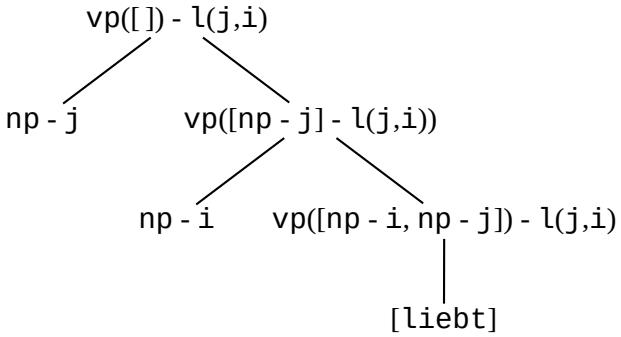
[17] [SvNMP89] donne un exemple de génération de l'expression *John calls friends up*, qui contient un verbe à particule. Pour cela, l'entrée lexicale correspondant à *call up* sous-catégorise *up* en gardant *calls* comme réalisation syntaxique. Le sens `up` n'apparaît pas sémantiquement, mais la règle nécessite la réalisation syntaxique de *up* : $\texttt{vp}([\texttt{np}-\texttt{Obj},\texttt{p}-\texttt{up},\texttt{np}-\texttt{Subj}])-\texttt{call\_up}(\texttt{Subj},\texttt{Obj})\longrightarrow[\texttt{calls}]$, avec $\texttt{p}-\texttt{up}\longrightarrow[\texttt{up}]$.



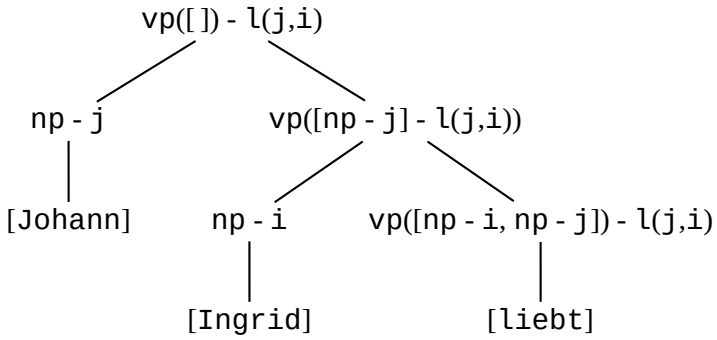
(a) Choix du pivot



(b) Première étape du pivot vers la racine



(c) Deuxième étape du pivot vers la racine



(d) Instanciation par les items lexicaux

FIG. 6.1 – Les différentes étapes de la génération guidée par la tête sémantique pour $\texttt{l}(\texttt{j}, \texttt{i})$

évidemment de la correspondance entre les règles de réduction syntaxique et les règles d'application fonctionnelle du côté sémantique. Comme dans les autres cas, l'algorithme combine une approche de bas en haut dans la recherche de l'élément lexical dont le prédicat correspond à celui de la racine, et une approche de haut en bas qui recompose les différentes étapes de réductions possibles [CRZ89].

Là encore, la difficulté survient lorsqu'un élément lexical ou qu'une règle de réduction, dans le cas des grammaires catégorielles augmentées, modifie la représentation sémantique sans laisser de trace. On rejoint là les problèmes de *vestigial semantics* [Shi88] qui requéraient la monotonie sémantique, et plus précisément les problèmes de *sémantique de l'identité* (*identity semantics*) [CRZ89] pour les grammaires catégorielles d'unification.

Du point de vue lexical, cela apparaît par exemple avec les conjonctions introduisant des subordonnées complétives (*complementiser*) comme *que* en français, ou *that, whether* en anglais, et dont la représentation sémantique habituelle est $\lambda x.x$ [Mor94]. On peut évoquer également les pronoms réfléchis comme *se* : $\lambda P.\lambda x.(Px)x$. Notre proposition, décrite à la section 6.4, rencontre les mêmes difficultés dans son étape de sélection des items lexicaux. Par contre, l'utilisation du calcul de Lambek plutôt que des règles d'extension des grammaires catégorielles permet d'éviter la sémantique de l'identité associée aux règles de réécriture unaire.

Par exemple, la règle d'élévation de type

$$X \rightarrow Y/(X\backslash Y)$$

s'accompagne de la transformation de la sémantique α associée à X en $\lambda P.P\alpha$. Dès lors, la correspondance entre foncteur syntaxique et foncteur sémantique n'est plus assurée : on ne peut pas affirmer qu'un élément de type syntaxique $X\backslash Y$ de sémantique $\beta\alpha$ provient de l'application de $(X\backslash Y)/Z$ ou $Z\backslash(X\backslash Y)$ de sémantique $\lambda x.\beta x$ à un élément de type syntaxique Z et de sémantique α. En effet, il peut très bien provenir de l'application de $(X\backslash Y)/Y$ de sémantique $\lambda P.P\alpha$ à Y de sémantique β. Autrement dit, pour engendrer une expression de sémantique $\beta\alpha$, il peut falloir commencer par engendrer celle de représentation α, l'argument, puis lui appliquer une règle de réduction, pour l'appliquer à l'expression de représentation β. Dans notre proposition, l'utilisation du calcul de Lambek transforme ces règles en simples théorèmes. La procédure de recherche de preuve mise en œuvre en tient compte alors naturellement et de manière indifférenciée dans son déroulement.

Pour tenir compte de ces limitations, [CRZ89] ajoute à sa procédure un essai systématique de ces règles de réduction et de ces items lexicaux. Dès lors la terminaison de l'algorithme requiert une condition sur la grammaire, celle de *offline parsability*[18].

Ces différentes approches partagent une volonté de guider la génération à l'aide des représentations sémantiques. S'il s'exprime de manières diverses (monotonie sémantique, récursion gauche lorsque tous les éléments de la partie droite partagent leur sémantique avec la partie gauche, règles unaires), le même problème de la trace sémantique laissée par chaque mot d'une expression lors de la composition se manifeste pour chacune des solutions, et nous en retrouvons une forme dans notre proposition de la section 6.4.

La manière dont la contrainte de monotonie sémantique a pu être levée, dans [SvNMP89] par la sous-catégorisation des parties discontinues (le *up* de *call up*), ou dans une grammaire permettant des

[18]Dans une grammaire d'unification G dont les règles ont la forme $t_0 \rightarrow t_1 \ldots t_n$ où t_0 est un terme de la logique du premier ordre et les t_i sont des termes ou des symboles terminaux, les t_i qui sont des termes sont appelés *termes supérieurs* (*top-level terms*) de la règle. Si aucun d'entre eux n'est une variable, la suppression de leurs arguments donne une règle $c_0 \rightarrow c_a \ldots c_n$ où chaque c_i est soit un symbole de relation soit un symbole terminal. En appliquant cela à toutes les règles de la grammaire, on obtient une grammaire hors-contexte. Si cette dernière est finiment ambiguë, alors G est *offline parsable* [KB82, PW83]. Par exemple, si G contient la règle $p(M) \rightarrow p(s(M))$ où M est une variable, s un symbole de fonction et p un symbole de relation, le squelette hors-contexte contient la règle $p \rightarrow p$ permettant une ambiguïté infinie (un nombre infini de dérivations de p). Et G n'est pas *offlineparsable* [Haa89].

opérations syntaxiques plus complexes que la simple concaténation [vN93, p. 100], semble indiquer un lien fort entre la modélisation des phénomènes syntaxiques et les propriétés de réversibilité intrinsèque des grammaires. Notons toutefois que si certaines grammaires rendent compte des constituants discontinus en perdant la monotonie sémantique [SvNMP89], ce n'est pas le cas pour d'autres, telles les grammaires de types logiques étendues pour traiter ces phénomènes. Dans [Mor94, p. 113] par exemple, les deux chaînes *rang* et *up* restent, pour le verbe *ring up*, dans la même unité lexicale, avec une forme sémantique **phone** qui permet à la grammaire de garder sa monotonie sémantique.

Problèmes en LFG

Les travaux en génération pour LFG montrent combien les propriétés de réversibilité intrinsèque dépendent des hypothèses de départ. En utilisant comme représentation sémantique une structure de traits[19], contenant un trait sémantique défini par une formule logique, la distance que l'on autorise entre cette structure initiale et celle de l'expression engendrée influe sur les résultats. Plus que les méthodes de génération pour LFG, très inspirées de ce que décrit la section précédente et qui demanderaient une plus grande description du formalisme, cette section présente quelques résultats de décidabilité du problème de génération.

Dans une grammaire LFG G, et pour une chaîne s, la relation de dérivation Δ_G se caractérise ainsi : $\Delta_G(s, c, \phi, f)$ si et seulement si G assigne à s la c-structure c à laquelle ϕ associe la f-structure f.

La c-structure représente l'arbre de dérivation de s grâce aux règles hors-contexte de G. Celles-ci sont augmentées d'équations dans un langage de description de traits qui permettent, lors de leur utilisation, de définir ϕ et f.

Par exemple, si pour une règle hors contexte $Y \to Y_1 \dots Y_n$ on associe les f-structures x_0 à Y et x_i à Y_1 (autrement dit, $\phi(Y) = x_0, \phi(Y_i) = x_1$), on peut avoir les règles :

				det	$\to$	*a*
S	$\to$	*np*	*vp*			x_0.SPEC = INDEF
		x_0.SUBJ $= x_1$	$x_0 = x_2$			x_0.NUM = SING
		x_1.CASE = NOMI		*n*	$\to$	*student*
np	$\to$	*det*	*n*			x_0.PRED = 'STUDENT'
		$x_0 = x_1$	$x_0 = x_1$			x_0.NUM = SING
vp	$\to$	*v*		*v*	$\to$	*fell*
		$x_0 = x_1$				x_0.PRED = 'FALL$\langle$(SUBJ)$\rangle$'
						x_0.TENSE = PAST

Elles permettent de construire la c-structure et la f-structure de la figure 6.2. Le problème de l'analyse est ici de trouver, étant donné s, une c-structure c, une f-structure f et ϕ tels que $\Delta_G(s, c, \phi, f)$. Pour la génération, l'idée est de partir d'une f-structure f_0 pour engendrer une expression s. Dans [Wed88], cette f-structure doit être égale à celle que construirait l'analyse de l'expression. Dans [Koh92], si f est la f-structure associée à s, on peut avoir également que f subsume f_0 ou bien que les deux s'unifient.

[Wed99] a montré que le problème de prouver l'existence d'une expression s et d'une f-structure f que G lui associe, étant donné f_0 avec la condition que f_0 subsume f est indécidable. Par contre, si l'on considère des *descriptions de traits*[20] déductivement équivalentes[21], le problème de savoir si,

[19] Comme en LFG, PATR [SUP+83] ou en HPSG [PS94].

[20] On peut considérer les équations associées aux règles comme la conjonction de formules F, avec des fonctions unaires (SUBJ(x_0) pour x_0.SUBJ par exemple), des symboles de relation ($=$, etc.), et exprimer la f-structure comme un modèle satisfaisant la structure de trait $\exists x_0 \dots x_l \bigwedge F$.

[21] Tout modèle de l'une est modèle de l'autre et réciproquement.

étant donné une description de trait, il existe une expression s avec une structure de traits déductivement équivalente satisfaisable est décidable [Wed95].

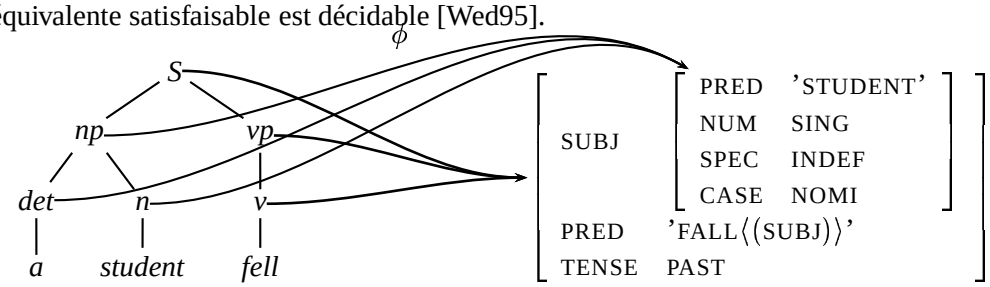


FIG. 6.2 – Correspondance entre c-structure et f-structure

6.2.2 Grammaires de types logiques : une première approche avec les systèmes déductifs étiquetés

Aussi bien en analyse qu'en génération, il s'agit de construire une preuve à partir des types syntaxiques des items lexicaux considérés. [MM97] propose un cadre uniforme pour l'analyse et la génération en utilisant les systèmes déductifs étiquetés (*labelled deductive systems* [Gab96]).

Sans vouloir décrire complètement un nouveau système, donnons brièvement le principe (appliqué au calcul de Lambek). Si L est un semi-groupe dans lequel sont interprétés les types syntaxiques et $T(A)$ l'interprétation du type A (voir la section 3.1.2), on peut construire l'interprétation $D(A)$ de chaque type A comme un sous ensemble de $L \times T(A)$ à partir des interprétations des types de base et des relations suivantes (seul le fragment implicatif est considéré) :

$$D(A\backslash B) = \{(s,m) | \forall (s',m') \in D(A), (s'+s, m(m')) \in D(B)\}$$
$$D(B/A) = \{(s,m) | \forall (s',m') \in D(A), (s+s', m(m')) \in D(B)\}$$

Chaque formule A est alors étiquetée avec une paire $(s,m) \in D(A)$. De même, on peut étiqueter chaque lien d'un préréseau.

Remarque. On retrouve exactement, pour les λ-termes, l'étiquetage proposé par [Roo91] et présenté à la section 3.2.3. Comme nous le verrons, le critère de correction reste sensiblement le même.

[MM97] énonce alors la condition pour obtenir un réseau comme la réussite de l'unification des étiquettes prosodiques ou sémantiques, l'une impliquant l'autre. Mais l'intérêt de considérer un système déductif étiqueté provient surtout de pouvoir guider la recherche de preuve grâce aux étiquettes, et ainsi réduire l'espace de recherche. La propagation des contraintes d'unification, au cours de la recherche de preuve, permet d'élaguer l'arbre de recherche dès que l'unification échoue.

Ainsi, en phase d'analyse, la prosodie de la phrase est connue et va permettre de guider la recherche de preuve. Puis le λ-terme associé s'en déduira facilement. Pour la génération, le λ-terme décrivant le sens de la phrase est connu et va permettre de guider la recherche de preuve, et la prosodie de l'expression s'en déduira facilement.

Analyse et génération : exemple

Illustrons ce processus sur un exemple, en analyse et en génération. La figure 6.3 montre la recherche de preuve pour l'analyse de *Vladimir attend Godot*, avec le lexique du tableau 6.1. Dans un premier temps, on construit chacun des arbres syntaxiques. La recherche de preuve consiste alors à apparier judicieusement les feuilles avec des formules duales. On a initialement les variables prosodiques a et b, et les variables sémantiques γ, α et β. La forme prosodique de l'expression est connue :

c'est celle qui étiquette l'unique conclusion positive. Si l'on essaye de relier cet atome S^+ avec chacun des autres, outre que dans le cas présent il n'y a qu'un seul atome dual, seul celui-ci permet d'unifier $b+$ *attend* $+a$ avec *Vladimir attend Godot.* On a donc :

$$\begin{aligned} a &= \textit{Godot} \\ b &= \textit{Vladimir} \\ \gamma &= (\textbf{attend}\ \alpha)\beta \end{aligned}$$

On propage la contrainte le long de l'arbre syntaxique avant d'arriver à la deuxième figure. Les contraintes d'unification prosodique sont telles que parmi les deux possibilités d'association des np^+ et des np^-, une seule est permise.

Tous les liens étant désormais présents, on peut procéder au calcul de la sémantique. On a alors :

$$\begin{aligned} \alpha &= \textbf{g} \\ \beta &= \textbf{v} \\ \gamma &= (\textbf{attend}\ \alpha)\beta \\ &= (\textbf{attend g})\textbf{v} \end{aligned}$$

Vladimir	*np*	**v**
Godot	*np*	**g**
attend	$(np\backslash S)/np$	(**attend g**)**v**

TAB. 6.1 – Lexique

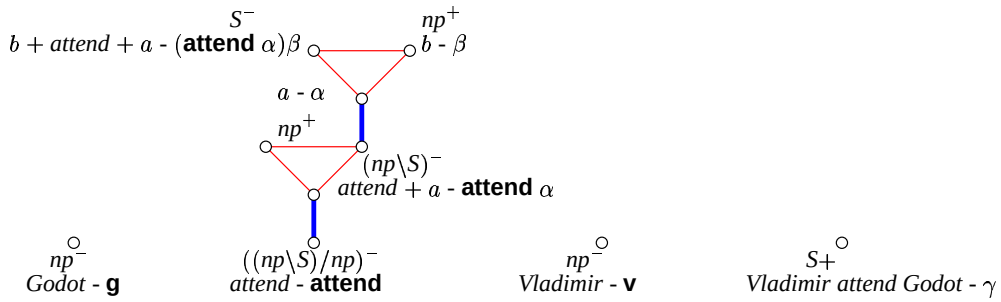
Considérons maintenant le processus inverse, illustré par les figures 6.4. Cette fois, les variables prosodiques sont a, b et c, tandis qu'il n'y a plus que deux variables sémantiques α et β. De la même manière que précédemment, on part de la sémantique de l'expression, (**attend g**)**v**, qui étiquette l'unique conclusion positive, et on cherche une unification possible. Dans ce premier cas, cela ne pose pas de problème et on obtient l'équation (**attend** $\alpha)\beta =$ (**attend g**)**v** sur les formes sémantiques. La section suivante évoque les problèmes liés à l'unification des termes prosodiques mais surtout des λ-termes. Ici, on obtient :

$$\begin{aligned} \alpha &= \textbf{g} \\ \beta &= \textbf{v} \\ c &= b+\ \textit{attend}\ +a \end{aligned}$$

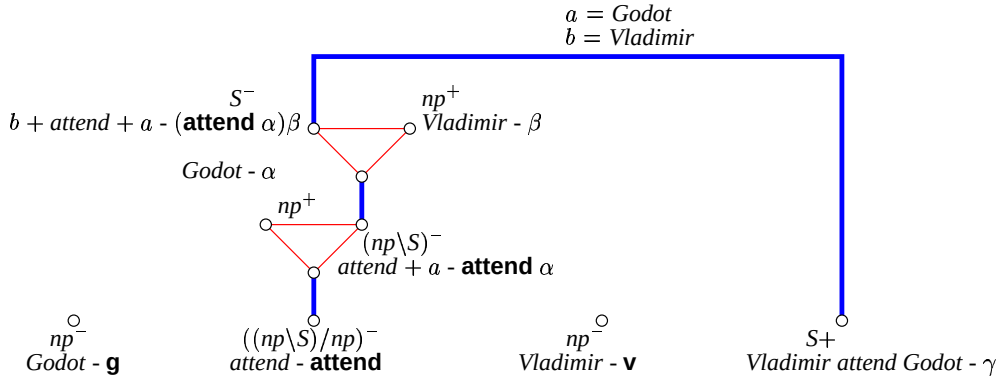
De nouveau, la propagation des résultats de l'unification permet d'obtenir la deuxième figure et parmi les deux possibilités restantes pour l'association des *np* duaux, les contraintes d'unification sur les λ-termes imposent les liens et les unifications prosodiques :

$$\begin{aligned} a &= \textit{Godot} \\ b &= \textit{Vladimir} \\ c &= b+\ \textit{attend}\ +a \\ &= \textit{Vladimir attend Godot} \end{aligned}$$

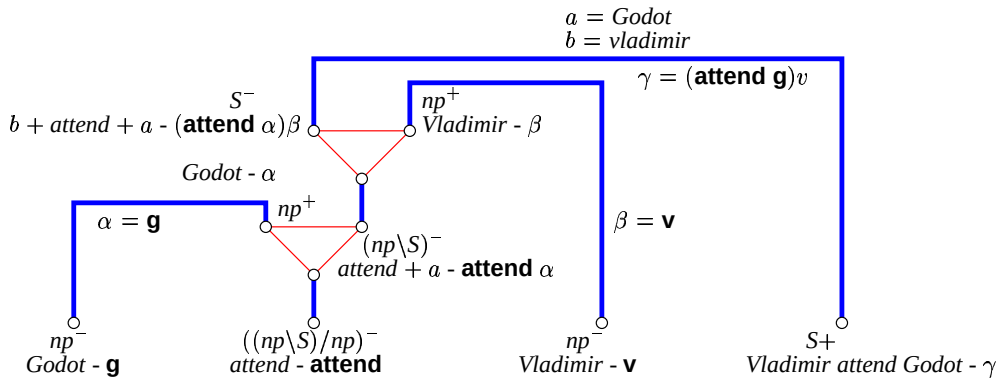
Remarque. L'utilisation des étiquettes prosodiques permet aussi de se passer du critère de bon parenthésage des liens axiomes : l'ordre des mots est calculé de manière interne et non plus suivant l'ordre des conclusions.



(a) Départ pour la construction de la preuve pour l'analyse syntaxique

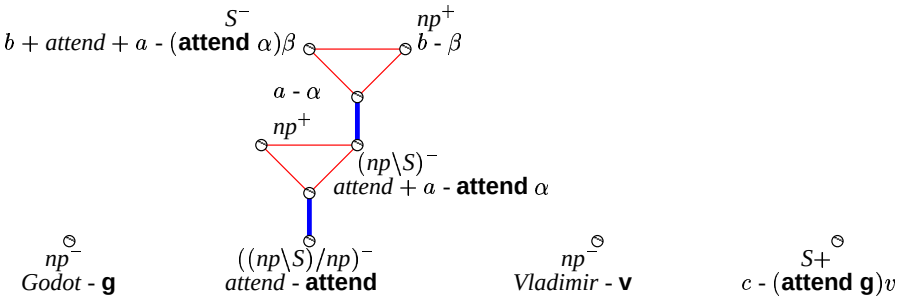


(b) Premier lien

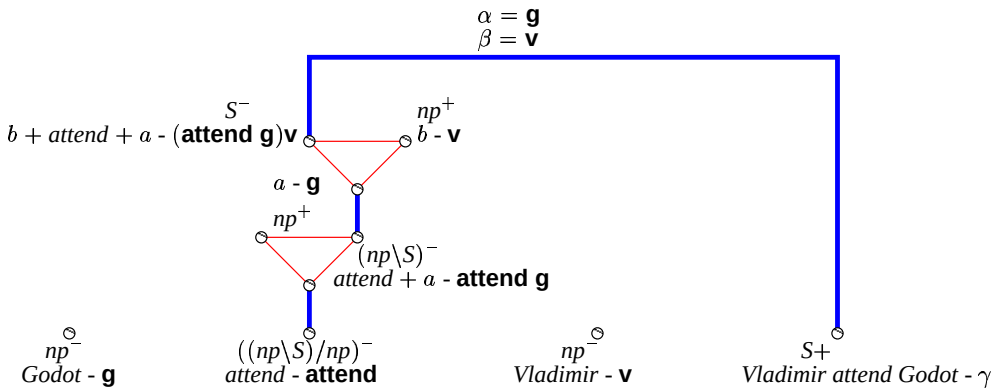


(c) Unification pour obtenir la forme sémantique

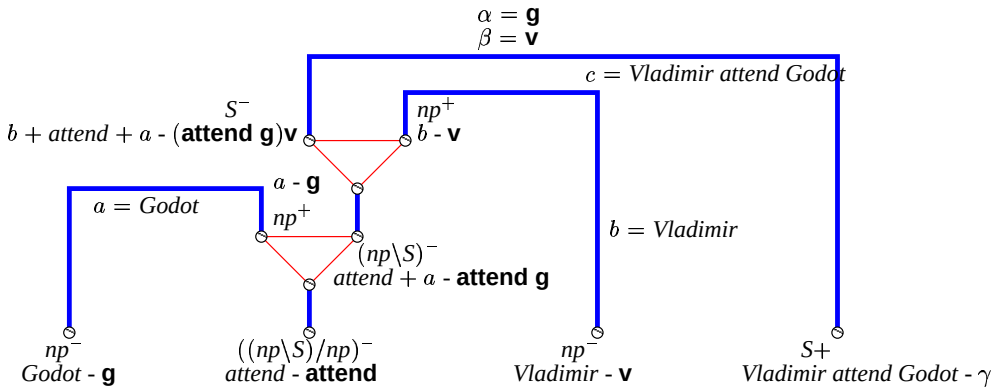
FIG. 6.3 – Analyse sémantique de *Vladimir attend Godot*



(a) Départ pour la construction de la preuve pour la génération



(b) Premier lien



(c) Unification pour obtenir la forme prosodique

FIG. 6.4 – Génération de (**attend g**)**v**

L'utilisation des systèmes déductifs étiquetés permet donc :
– d'uniformiser les procédures d'analyse et de génération ;
– de guider la recherche et d'élaguer au plus tôt les branches ne conduisant pas à une solution.

Discussion

Dans ce processus, l'unification constitue l'élément moteur pour guider la recherche, aussi bien dans le cas de l'analyse que de la génération. Dans le cas des termes prosodiques, il s'agit d'unification au premier ordre, c'est-à-dire que l'on considère l'égalité entre les termes, avec une opération $+$ associative. Elle est décidable, quoiqu'infinitaire[22].

Le problème se pose de façon plus aiguë à propos de la génération. En effet, l'unification de λ-termes n'est en géneral pas décidable [Hue73, Dow01]. Elle n'est même pas décidable au deuxième ordre [Gol81, Dow01]. Il s'agit donc de préciser quels sont les problèmes d'unification que cette méthode rencontre.

Pour ceci, [MM97] restreint les λ-termes utilisés soient à la fois :
– des λ-termes linéaires (nous ferons de même dans un premier temps) ;
– des λ-termes dont les variables libres n'ont au plus que deux occurrences ;
– des λ-termes d'ordre inférieur à deux.

En effet, dans ce cas, le problème est décidable [Lev96] et on peut appliquer l'algorithme.

Notons toutefois que ce sont des problèmes de filtrage plutôt que des problèmes généraux d'unification qui se posent. Cependant, là encore, si le troisième ordre est décidable [Dow92], le problème est ouvert pour les ordres supérieurs en général. Pour les termes linéaires d'ordre supérieur, ce n'est que très récemment que [dG00] a montré que le problème du filtrage était décidable, mais NP-complet.

Ajoutons enfin que même en cas de décidabilité, il n'existe pas en général d'unificateur plus général. Ainsi le problème $\{x\mathbf{f} = \mathbf{f}(\mathbf{f}\ \mathbf{a})\}$ admet-il comme solution les deux substitutions incomparables $\lambda z.z(\mathbf{f}\ \mathbf{a})/x$ et $\lambda z.\mathbf{f}(z\mathbf{a})/x$.

Pourtant, la réalisation syntaxique, à partir d'items lexicaux donnés, n'est qu'une recherche de preuve dans le calcul de Lambek, dont on sait qu'il est décidable. Et si l'information donnée par la forme sémantique à obtenir peut en guider la recherche, elle ne doit pas la rendre plus difficile.

Les sections suivantes présentent l'approche que nous proposons. Pour la réalisation syntaxique, seule la contrainte de linéarité subsiste. En effet, l'ordre des termes considérés, ansi que leur typage qui peut désormais comprendre la conjonction, n'intervient plus. En outre, dans certains cas définis par le lexique, la réalisation syntaxique se fait en temps polynomial. Pour cela, nous utilisons la représentation des formes sémantiques par les réseaux, telle que la définit la section 5.4.1

6.3 Reformulation avec les réseaux sémantiques

Concentrons-nous sur la réalisation syntaxique et résumons le problème. Dans sa forme classique, étant donnés les items lexicaux m_i, de catégories syntaxiques c_i, de formes sémantiques s_i et une forme sémantique S_0 (et une catégorie syntaxique C_0) on cherche un réseau syntaxique Π tel que :
– les conclusions de Π sont $c^-_{\sigma(1)}, \ldots c^-_{\sigma(n)}, C^+_0$;
– si t est le λ-terme associé à Π via l'isomorphisme de Curry-Howard, alors

$$t[s_1/x_1, \ldots s_n/x_n] = S_0$$

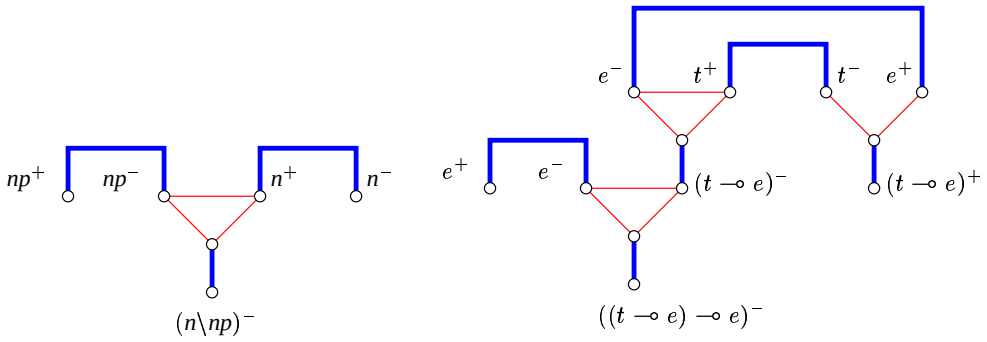
[22]Cela signifie que certains problèmes d'unification admettent un nombre infini de solutions. Par exemple, l'équation $x + a = a + x$ admet comme solutions $x \to a$, $x \to a + a$, $x \to a + (a + a)$, etc.

En reprenant la même démarche que celle de la section 5.4.1 qui montre comment utiliser les réseaux à la place des λ-termes dans l'analyse sémantique d'une phrase, le problème revient à, étant donnés les items lexicaux m_i, de catégories syntaxiques c_i, de formes sémantiques π_i (en réseaux) et une forme sémantique Π_0 (en réseau, et une catégorie syntaxique C_0) chercher Π (le même que ci-dessus) tel que :

- les conclusions de Π sont $c^-_{\sigma(1)}, \dots c^-_{\sigma(n)}, C_0^+$;
- le calcul du réseau sémantique à partir de $\mathcal{H}(\Pi)$ et des π_i par coupure et élimination des coupures donne Π_0.

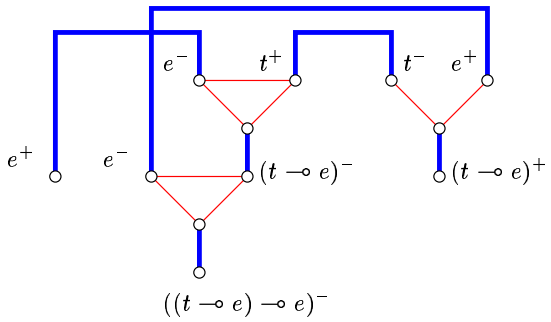
Or, dès que les conclusions de Π sont connues, Π ne dépend que de l'association des atomes duaux. Il suffit donc déterminer l'appariement A_Π des atomes duaux de Π.

Remarque. Le passage de Π à $\mathcal{H}(\Pi)$ peut changer le graphe sous-jacent par η-expansion (voir figure 6.5(b)). Soient deux réseaux Π_1 et Π_2 tels que $\mathcal{H}(\Pi_1)$ privé de ses liens axiomes soit égal à Π_2 privé ses liens axiomes. On dit que A_{Π_2} est *compatible* avec A_{Π_1} si et seulement si $\mathcal{H}(\Pi_1) = \Pi_2$. Si l'on a Π_1 comme à la figure 6.5(a), l'appariement de la figure 6.5(b) est compatible, celui de la figure 6.5(c) ne l'est pas (en définissant, pour cet exemple, $\mathcal{H}(n) = t \multimap e$).



(a) Réseau syntaxique (Π_1)

(b) Appariement compatible ($\Pi_2 = \mathcal{H}(\Pi_1)$)

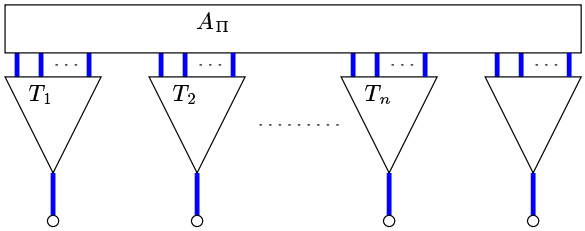


(c) Appariement incompatible ($\Pi_2 \neq \mathcal{H}(\Pi_1)$)

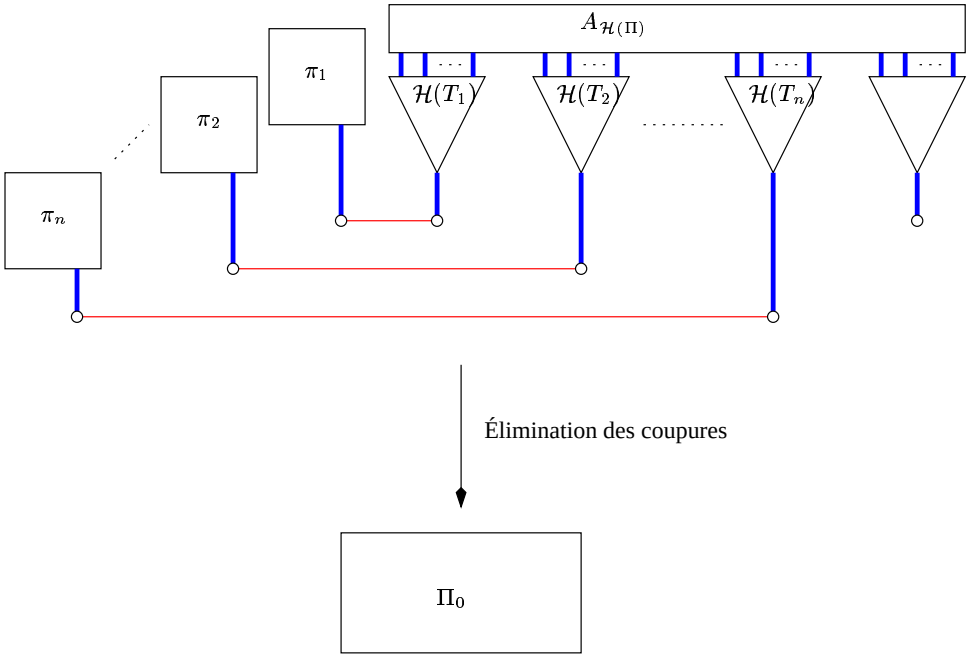
FIG. 6.5 – Exemples d'appariements compatibles et incompatibles

Finalement, le problème peut se traduire comme sur la figure 6.6 où la seule inconnue est A_Π (et donc $A_{\mathcal{H}(\Pi)}$), et les T_i sont les arbres des catégories syntaxiques de chacun des items lexicaux, et les $\mathcal{H}(T_i)$ leur image sémantique. Outre cette contrainte, il faut bien entendu que A_Π n'ait pas de liens axiomes qui se croisent.

Notation. Lorsque dans la section 6.4.1 nous ferons référence à ce problème de réalisation syntaxique, nous appellerons en général $\mathcal{F}$ la forêt $\cup_i\{\mathcal{H}(T_i)\} \cup \{T_0\}$, $\mathcal{F}^+$ le réseau constitué de $\mathcal{F}$ et de $A_{\mathcal{H}(\Pi)}$, $\overline{U}$ le réseau constitué de $\mathcal{F}^+$ et des π_i reliés par des liens coupures, et σ ces liens coupures.



(a) Réseau syntaxique



(b) Réseau sémantique

FIG. 6.6 – Contraintes vérifiées par A_Π

Pour résoudre ce problème, nous allons faire une recherche de preuve (commutative) sur la partie sémantique et trouver un appariement A, puis vérifier qu'il existe un appariement A' pour le réseau syntaxique tel que A soit compatible avec A'.

Il suffira alors de vérifier qu'il existe un ordre des conclusions (et donc des mots) tel qu'avec les mots dans cet ordre, A' n'a pas de liens axiomes qui se croisent.

6.4 Proposition

S'appuyant sur les conclusions précédents, nous décrivons notre proposition en deux parties :

– d'une part la *réalisation syntaxique*, qui relève de la recherche de preuve et qui *suppose connus les items lexicaux*. Ce problème se ramène à la résolution d'une équation matricielle ;
– d'autre part le *choix des items lexicaux*, qui relève de propriétés d'invariances des réseaux lors de l'élimination des coupures.

6.4.1 Recherche de preuve et formule d'exécution

La technique utilisée pour la recherche de preuve n'est pas propre aux types sémantiques e et t mais est plus générale. Son originalité vient d'une part de ce que pour la guider, elle utilise le résultat (connu) de la réduction du réseau recherché, et d'autre part qu'elle utilise la formule d'exécution de la géométrie de l'interaction (voir section 2.5). Dans un premier temps cependant, nous ne considérons que des réseaux de la logique linéaire multiplicative.

Par contre, si la technique est générale, pour permettre une expression simple de la solution dans le cas qui nous intéresse, nous allons tirer parti de toutes les caractéristiques du problème de génération.

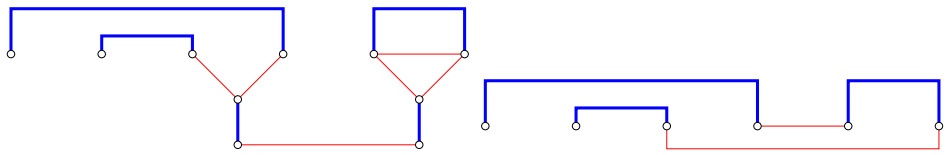
Rappelons que la formule d'exécution :

$$\mathrm{Res}(U,\sigma) = (1-\sigma^2)U(1+\sum_{1}^{\infty}(\sigma U)^k)(1-\sigma^2)$$

exprime comment calculer l'interprétation matricielle d'un réseau ($\mathrm{Res}(U,\sigma)$) résultant de l'élimination des coupures (interprétées par σ) du réseau (interprété par U) à partir de ces mêmes interprétations.

Comme nous l'avons vu, l'élimination des coupures pour le fragment multiplicatif distingue l'élimination des coupures entre liens par et tenseur et entre les axiomes. Ce processus étant parfaitement déterministe, tout appariement correct pour le réseau avant l'élimination des coupures est correct pour le réseau obtenu en propageant l'élimination des coupures jusqu'à ce qu'il n'y ait plus que des coupures entre axiomes.

Donc, si nous trouvons un appariement pour le réseau réduit à l'exception des liens axiomes, il suffit de vérifier qu'avec les arbres des formules il forme bien un réseau. C'est un critère de vérification dont on a vu qu'il était quadratique avec les réseaux comme R&B-graphes. La vérification ultérieure est nécessaire car si un réseau se réduit bien en un réseau, un préréseau qui n'est pas un réseau peut se réduire en réseau, comme le montre la figure 6.7.



(a) Préréseau avant réduction (b) Réseau après réduction

FIG. 6.7 – L'élimination des coupures sur un préréseau qui n'est pas un réseau peut produire un réseau

Dans cette section qui vise à reformuler la formule d'exécution, nous considérons :
– $\overline{U}$ un réseau de preuve avec des liens coupures uniquement sur des liens axiomes ;
– U l'interprétation matricielle des liens axiomes ;
– σ l'interprétation matricielle des liens coupures :
– $\overline{\Pi_0}$ la forme normale de $\overline{U}$, interprétée par Π_0 ;

– pour un réseau quelconque $\overline{\Pi}$, $A_{\overline{\Pi}}$ est l'appariement des atomes duaux de $\overline{\Pi}$, et pour un atome a, $A_{\overline{\Pi}}(a)$ est l'unique atome dual de a relié à celui-ci par un lien axiome.

U, σ et Π_0 sont tous dans $\mathcal{M}_s(\mathbf{bool})$, l'ensemble des matrices carrées de dimension s à coefficients booléens. Comme dans la section 2.5, la base est $(e_i)_{i\in[1,s]}$, chaque e_i correspondant à la conclusion d'un lien axiome. Par abus de langage, on parlera également de e_i pour désigner le sommet auquel il correspond.

Supposons qu'il existe $p \in [1, s]$ tel que pour tout $i \in [1, p]$, e_i et $A_{\overline{U}}(e_i)$ ne soient pas prémisses d'un lien coupure. On a alors (avec $U_0 \in \mathcal{M}_p(\mathbf{bool})$) :

$$U = \begin{bmatrix} U_0 & 0 \\ 0 & U_1 \end{bmatrix} \quad \text{et} \quad \sigma = \begin{bmatrix} 0 & 0 \\ 0 & \sigma_1 \end{bmatrix}$$

Alors $\mathrm{Res}(U, \sigma) = \left[\begin{smallmatrix} U_0 & 0 \\ 0 & \mathrm{Res}(U_1,\sigma_1) \end{smallmatrix}\right]$. Cela signifie qu'un lien axiome dont aucune conclusion n'est prémisse d'un lien coupure est inchangé par élimination des coupures. On peut donc considérer sans perte de généralité que tous les liens axiomes ont au moins une conclusion prémisse d'un lien coupure.

Remarque. Ci-dessous, comme dans la plupart des démonstrations qui vont suivre, on considère des matrices blocs. De plus, si nécessaire, lorsque $X \in \mathcal{M}_{u,v}(\mathbf{bool})$, on note $X^{(u,v)}$.

Les (e_i) se répartissent en trois catégories dans $\overline{U}$:

– $\forall i \in [1, p]$, e_i n'est pas prémisse d'un lien Cut (et donc, d'après l'hypothèse énoncée ci-dessus, $A_{\overline{U}}(e_i)$ est prémisse d'un lien Cut) ;
– $\forall i \in [p + 1, 2p]$, e_i est prémisse d'un lien coupure mais pas $A_{\overline{U}}(e_i)$;
– $\forall i \in [2p + 1, 2p + n]$, e_i et $A_{\overline{U}}(e_i)$ sont prémisses d'un lien Cut.

On obtient alors ($s = 2p + n$) :

$$U = \begin{bmatrix} 0 & U_1^{(p,p)} & 0 \\ {}^tU_1^{(p,p)} & 0 & 0 \\ 0 & 0 & U_3^{(n,n)} \end{bmatrix} \quad \sigma = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \sigma_1^{(p,p)} & \sigma_2^{(p,n)} \\ 0 & \sigma_3^{(n,p)} & \sigma_4^{(n,n)} \end{bmatrix} \quad \Pi_0 = \begin{bmatrix} \Pi_1^{(p,p)} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Rappelons que :

– $U_1\,{}^tU_1 = {}^tU_1U_1 = 1$;
– si $\sigma_4 = 0$ alors $n \leq p$;
– comme il y a exactement un et un seul 1 par ligne et par colonne, si $\sigma_4 = 0$, chaque colonne de σ_3 a un 1 et le rang de σ_2 est n.

Nous pouvons maintenant énoncer le théorème crucial de cette partie :

Théorème 57. *Soit $\overline{U}$ un réseau de preuve correct, dont les prémisses de tous les liens coupures sont conclusions de liens axiomes. Soit $\overline{\Pi_0}$ le réseau résultant de l'élimination des coupures de $\overline{U}$. Si l'on interprète les liens axiomes de $\overline{U}$ par U, les liens axiomes de $\overline{\Pi_0}$ par Π_0 et les coupures de $\overline{U}$ par σ, en reprenant les définitions ci-dessus de $U_1, U_3, \sigma_1, \sigma_2, \sigma_3, \sigma_4$ et Π_1 et $A = ({}^tU_1\Pi_1 - \sigma_1\,{}^tU_1)U_1$ et $X = (U_3 - \sigma_4)^{-1}$, alors X est bien définie et les trois relations suivantes sont équivalentes :*

$$\mathit{Res}(U, \sigma) = (1 - \sigma^2)U(1 - \sigma U)^{-1}(1 - \sigma^2) \tag{6.6}$$

$$({}^tU_1\Pi_1 - \sigma_1\,{}^tU_1)U_1 = \sigma_2U_3(1 + \sum_{k\geq 1}(\sigma_4U_3)^k)\,{}^t\sigma_2 \tag{6.7}$$

$$A = \sigma_2X\,{}^t\sigma_2 \mathit{et}\, U_3 = X^{-1} + \sigma_4 \tag{6.8}$$

Démonstration. Ce théorème est démontré en annexe A, car avec le corollaire 62, σ_4U_3 est inversible et comme $U_3U_3 = 1$, $U_3 - \sigma_4 = (1 - \sigma_4U_3)U_3^{-1}$, d'où $X = U_3(1 - \sigma_4U_3)^{-1}$. □

Replaçons-nous maintenant dans l'optique de la recherche de preuve que nous évoquions pour la génération. Le réseau recherché vérifie les hypothèses du théorème précédent. De plus, chacun des liens axiomes de l'appariement, à l'exception de ceux qui sont prémisses de l'unique conclusion positive, sont prémisses d'un lien coupure. En conséquence, les liens axiomes recherchés sont décrits par U_3 alors que U_1, Π_1 et σ_1, donc A, sont connus.

La recherche de la preuve, c'est-à-dire la *réalisation syntaxique*, s'effectue donc à l'aide de la relation (6.8) du théorème en résolvant l'équation en X. Du nombre de solution à cette équation dépend le nombre de réalisations syntaxiques possibles, c'est pourquoi nous discutons maintenant de la résolution de cette équation. Elle fait apparaître deux cas, selon que σ_4 est nulle ou pas, dans lesquels il esiste une solution unique ou bien un ensemble de solutions. Par contre σ_4 est déterminée par la donnée forme des sémantiques des items lexicaux donnés. Autrement dit, dès le choix des items lexicaux, le comportement de l'algorithme de réalisation syntaxique (solution unique ou plusieurs solutions) est connu.

Remarque. Lorsqu'on dit que U_1 est connu, ce n'est pas tout à fait exact. En particulier, les liens vers l'unique conclusion positive (en général *S*, mais cela peut être une formule complexe) ne sont pas connus a priori. Cependant, il est facile de les calculer si $\sigma_4 = 0$, et comme nous le verrons, c'est suffisant.

Pour ce faire, il suffit de regarder à quels atomes de Π_0 les atomes de la sortie positive sont reliés. Ceux-ci sont reliés par un lien axiome puis par un lien coupure à un atome de $\mathcal{H}(Pi)$. Ce sont ces derniers que l'on relie aux atomes de la conclusion positive de U (qui est la même que celle de Π_0).

6.4.2 Résolution de l'équation : si $\sigma_4 = 0$

Dans l'équation :

$$A = \sigma_2 X \, {}^t\sigma_2 \tag{6.9}$$

où A est connu, de même que σ_2, on cherche X inversible qui, grâce à la relation

$$U_3 = X^{-1} + \sigma_4 \tag{6.10}$$

doit permettre de trouver le réseau recherché. La difficulté provient de ce qu'en général la matrice $\sigma_2 \in \mathcal{M}_{p,n}(\mathbf{bool})$ n'est pas une matrice carrée ni inversible.

Néanmoins, si $r \leq \min(p, n)$ est le rang de σ_2 (c'est le nombre de 1 qu'il y a dans σ_2), alors il existe $p \in \mathcal{M}_p(\mathbf{bool})$ et $Q \in \mathcal{M}_n(\mathbf{bool})$ telles que $\sigma_2 = P I_r Q$ (avec $I_r \in \mathcal{M}_{p,n}(\mathbf{bool})$ telle que pour tout $i, j \in [1, r]^2$, $(I_r)_{ij} = 1$ si $i = j$ et $(I_r)_{ij} = 0$ si $i \neq j$). Alors, si $B = P^{-1} A \, {}^tP^{-1}$ et $Y = Q X \, {}^tQ$, résoudre (6.9) est équivalent à résoudre

$$B = I_r Y \, {}^t I_r \tag{6.11}$$

$$= \begin{bmatrix} Y_{r,r} & 0 \\ 0 & 0 \end{bmatrix} \tag{6.12}$$

avec B connu (et $Y_{r,r} = (Y_{ij})_{1 \leq i,j \leq r}$). Alors, avec $Y \in \mathcal{M}_n(\mathbf{bool})$, symétrique (car X symétrique), on a $X = Q^{-1} Y \, {}^tQ^{-1}$ (Q est également connu). Cependant, si $r < n$, Y n'est pas totalement déterminée par B. Or, si $p < n$, la configuration de σ (un seul 1 par ligne et par colonne) montre que $\sigma_4 \neq 0$. On a donc les cas :

- $r = n$, donc $p \geq n$ et $\sigma_4 = 0$;
- $r < n$, alors $\sigma_4 \neq 0$.

De plus, si $\sigma_4 = 0$, on a nécessairement $n \leq p$ et alors $r = n$. Comme B détermine complètement Y si et seulement si $n = r$ et $p \geq n$, B détermine complètement Y si et seulement si $\sigma_4 = 0$. Dans ce cas, de plus, d'après (6.10), $U_3 = X^{-1}$. Or $U_3^2 = 1$ et donc $U_3 = X$.

On voit que si $\sigma_4 = 0$, le problème de recherche de preuve se résoud très simplement (polynomialement) par un simple produit de matrices.

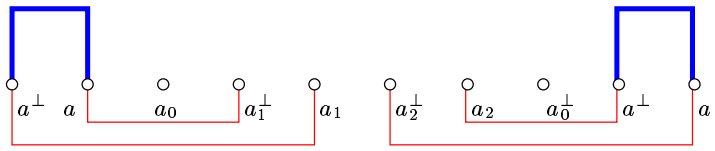
Remarque. Essayons de donner un sens pour le problème de génération à $\sigma_4 = 0$. σ_4 indique les liens coupures qui existent entre des conclusions de liens axiomes dont l'autre conclusion est aussi prémisse d'un lien coupure. Toutes les coupures sont entre des axiomes recherchés et des axiomes des π_i (en particulier, il n'y a aucune coupure entre deux liens axiomes de $A_{\mathcal{H}(\Pi)}$). Donc si $\sigma_4 \neq 0$, cela signifie que du côté d'un π_i, il y a un lien axiome dont l'autre conclusion est prémisse d'une coupure.

Autrement dit, la connaissance des items lexicaux permet a priori de savoir que le calcul de la solution (et la preuve de son existence) se fait en temps polynomial. Le temps de calcul est caractérisé par la forme des réseaux sémantiques des entrées lexicales.

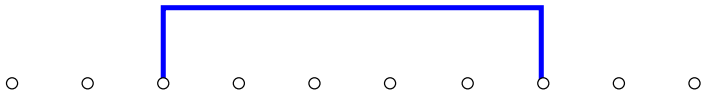
6.4.3 Résolution de l'équation : si $\sigma_4 \neq 0$

Avant de proposer une solution pour le cas $\sigma_4 \neq 0$, regardons pourquoi cela amène une indétermination et donc, a priori, plusieurs solutions. La figure 6.9(a) montre une solution au problème de la figure 6.8 (les liens axiomes proposés sont en pointillé. Les éléments donnés par l'énoncé sont en trait plein, en particulier les liens axiomes aux extrémités. Ils correspondent à des liens appartenant à des réseaux sémantiques d'items lexicaux). Les deux liens coupures sont dans σ_4.

On s'aperçoit que leurs extrémités libres ($a_1^\perp$ et $a_2^\perp$, et a_1 et a_2) jouent tout à fait le même rôle, et suggèrent une autre solution évidente : celle de la figure 6.9(b) obtenue à partir de celle de la figure 6.9(a).



(a) Données initiales

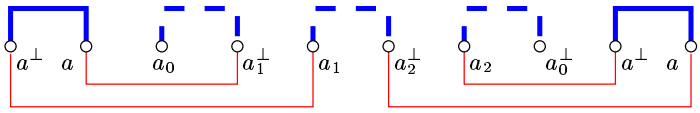


(b) Réseau à obtenir après élimination des coupures

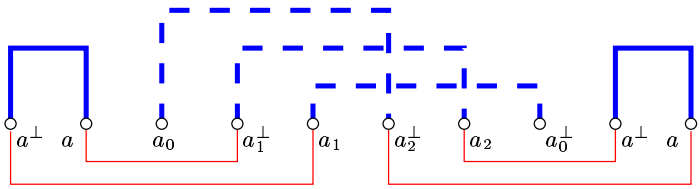
FIG. 6.8 – Énoncé du problème

En fait, on peut même aller plus loin. Résolvons le problème sans tenir compte de ce « détour» (indiquons-le en pointillé et faisons comme si $\sigma_4 = 0$). Nous obtenons le résultat de la figure 6.10. Il suffit alors de couper en deux le lien axiome obtenu, puis de le brancher des deux manières possibles sur les deux sommets extrémités d'un des détours (a_1 et $a_1^\perp$ par exemple). On a alors deux liens axiomes qui peuvent chacun être coupés pour être branchés sur le deuxième détour (a_2 et $a_2^\perp$).

Cette dernière remarque nous suggère la manière de résoudre le problème lorsque $\sigma_4 \neq 0$. En effet, soit un appariement A solution du problème P. A' obtenu à partir de A en éliminant toutes



(a) Première solution



(b) Deuxième solution

FIG. 6.9 – Les deux solutions

les coupures qui provoquent des détours est solution du problème P' obtenu en supprimant tous les sommets qui sont sur un détour. Or P' est soluble de la manière indiquée dans la section précédente car, pour lui, $\sigma_4 = 0$ désormais (puisqu'on a supprimé les détours). Remarquons que ce problème est directement calculable depuis le problème P.

Réciproquement, en partant d'une solution A' de P', il suffit, pour chaque détour entre deux sommets de formule a et $a^\perp$ de casser un lien axiome de A' entre deux sommets a et $a^\perp$ et de connecter chacune des moitiés aux extrémités du détour. Cela donne A'' et on itère le processus jusqu'à ce qu'il n'y ait plus de détour d'extrémités a et $a^\perp$.

Autrement dit, si $\sigma_4 \neq o$:

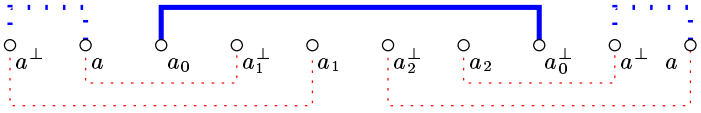
- on vérifie l'existence et on calcule une éventuelle solution de base en temps polynomial ;
- étant donné un type atomique pour lequel la solution de base donne l liens axiomes, et pour lequel il y a dans le problème initial $d \geq 1$ détours, on sait engendrer à partir du type de base les $l*(l+1)*\cdots*(l+(d-1))$ appariements pour ce type de base qui constituent une solution ;
- si k est le nombre de types utilisés dans le réseau (typiquement deux : e et t) et l_k le nombre de liens pour la catégorie k et d_k le nombre de détours pour la catégorie k, il y a $\sum_{i=1}^{k} \frac{(l_k+(d_k-1))!}{(l_k-1)!}$ solutions.

La vérification de l'existence d'une preuve est rapide, mais l'espace des solutions peut être très grand.

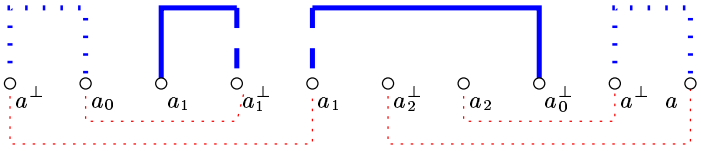
Remarque. Puisqu'un détour correspond à un axiome dont les deux conclusions sont prémisses de l'unique conclusion positive d'un réseau sémantique π_i, et que celle-ci est reliée à un arbre correspondant à une entrée lexicale, les deux extrémités d'un détour font partie de la même entrée lexicale.

Nous avons montré que le problème de réalisation syntaxique conduit à une recherche de preuve. Conduite dans le fragment commutatif, cette recherche admet la vérification de l'existence d'une solution en temps polynomial (relativement au nombre total d'atomes du réseau avant élimination des coupures). Le calcul effectif d'une solution commutative (parmi toutes celles possibles) est également polynomial.

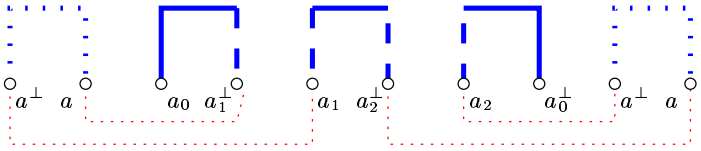
Par contre, le nombre de solutions dépend de la structure des réseaux sémantiques associés aux items lexicaux utilisés, ce nombre pouvant être très grand. Cependant, seules les solutions correctes d'un point de vue non commutatif nous intéressent. Aussi la construction (dans le cas $\sigma_4 \neq 0$) des solutions doit-elle tenir compte des contraintes de bon parenthésage des liens axiomes.



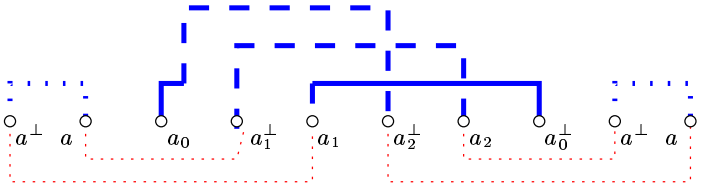
(a) Solution du problème sans « détour »



(b) La même s'apprêtant à tenir compte du premier détour



(c) La même s'apprêtant à tenir compte du deuxième détour, en coupant un axiome



(d) La même s'apprêtant à tenir compte du deuxième détour, en coupant l'autre axiome

FIG. 6.10 – Génération de solutions *commutatives*

6.4.4 Appariement syntaxique et ordre des mots

La recherche de preuve décrite dans les sections précédentes relève du fragment commutatif de la logique linéaire. Toutefois, puisque les solutions doivent s'appliquer au réseau syntaxique, outre les problèmes de compatibilité des appariements, les solutions retenues ne doivent pas avoir de liens axiomes qui se croisent.

Pour l'instant, aucune contrainte sur l'ordre des mots, et donc des arbres syntaxiques Y_i, puis des $\mathcal{H}(T_i)$, n'a été exprimée. De fait, ce sont les liens axiomes définis par les solutions de la recherche de preuve commutative qui vont induire l'ordre des mots. Plutôt que de vérifier le bon parenthésage des liens axiomes, il s'agit de placer les mots de façon à ce que ce bon parenthésage soit vérifié. Pour cela, nous procédons en deux temps : le premier pour vérifier qu'à l'intérieur d'un mot, le parenthésage des liens est correct, le second pour placer les mots relativement les uns aux autres.

Correction à l'intérieur d'un mot

Dans ce prétraitement, pour chaque mot (et donc chaque $\mathcal{H}(T_i)$), on ne considère que les axiomes dont les deux conclusions sont prémisses de $\mathcal{H}(T_i)$. La différence principale avec le placement des mots les uns par rapport aux autres, c'est qu'à l'intérieur d'un mot, l'ordre est parfaitement défini. Il ne s'agit que d'une vérification d'un bon parenthésage. Si cela ne se fait pas en temps linéaire (par rapport à la vérification d'un bon parenthésage habituel), c'est tout simplement qu'il faut s'assurer que lorsqu'on rencontre la conclusion d'un lien axiome, l'autre conclusion fait bien partie du même arbre.

Notation. Lues de gauche à droite, les feuilles de chaque arbre $\mathcal{H}(T_i)$, qui sont des atomes, forment un *mot d'atomes.*

Définition 42. Par analogie avec les groupes libres, on dit que m est *réductible* si :
- $m = aA_{\mathcal{H}(\Pi)}(a)$;
- $m = am'A_{\mathcal{H}(\Pi)}(a)$ et m' est réductible ;
- $m = m_1m_2$ et m_1 et m_2 sont réductibles.

Pour un mot d'atomes m, la propriété de bon parenthésage implique : $m = m_1 \dots m_p$ avec pour tout $k \in [1,p]$, $m_k = a_1^{(k)} \cdots a_{i_k}^{(k)} m'_k a_{j_k}^{(k)} \cdots a_{n_k}^{(k)}$ (avec la convention que si $i = 0$ alors $m_k = m'_k a_{j_k}^{(k)} \cdots a_{n_k}^{(k)}$ et si $j = i+1$, $m_k = a_1^{(k)} \cdots a_{i_k}^{(k)} m'_k$) et pour tout $(k,l) \in [1,p] \times ([1,i_k] \cup [j_k,n])$, $A_{\mathcal{H}(\Pi)}(a_l^{(k)})$ n'est pas dans m et m'_k est réductible.

Alors $a_1 \cdots a_{i_1}^{(1)} a_{i_1}^{(1)} a_{j_1}^{(1)} \cdots a_{j_p}^{(p)} \cdots a_{n_p}^{(p)}$ est la forme *réduite* de m, celle qui va réellement jouer un rôle dans l'ordonnancement des items lexicaux. Par extension, on dit également que m est réductible.

Le prétraitement consiste donc à vérifier que chaque mot d'atome est réductibe et à calculer sa forme réduite. Si l est le nombre d'entrées lexicales, et p le plus grand nombre de feuilles d'une entrée lexicale, retrouver la deuxième conclusion d'un lien axiome peut se faire en $(\log lp)$. Puis l'application d'un algorithme habituel de vérification de bon parenthésage étant linéaire, le prétraitement s'effectue sur tous les mots en $lp(\log lp)$.

Si cette première partie n'échoue pas, on applique la suite de l'algorithme sur les mots d'atomes réduits, c'est-à-dire qu'il n'y a plus que des liens entre différentes entrées lexicales.

Correction et ordonnancement des mots

Dans cette section, tous les mots d'atomes sont considérés comme réduits. Pour ordonner les items lexicaux, nous utilisons la propriété suivante :

Proposition 58. *Soit un réseau (non commutatif) correct, avec $\preceq_o$ l'ordre total sur les mots (celui correspondant à l'unique conclusion positive étant le plus grand), et $\preceq \subset \preceq_0$ un ordre partiel sur des mots d'atomes. Soit m_0 un mot d'atomes et $m = a_1 \cdots a_n$, $m \preceq m_0$, un mot d'atomes tel que a_i est relié par un lien axiome à un atome de m_0. Appelons m_k le mot dont un des atomes est relié à a_k(voir figure 6.11). Soit alors l'ordre $\preceq'$ défini par :*

- $m_{k_1} \preceq' m_{k_2}$ *si et seulement si :*
  - $(k_1, k_2) \in [1, i-1] \times [1, i-1]$ *et* $k_2 \leq k_1$
  - *ou* $(k_1, k_2) \in [1, i-1] \times \{i\}$)
  - *ou* $(k_1, k_2) \in [i, n]^2$ *et* $k_1 \leq k_2$
  - *ou* $(k_1, k_2) \in [1, n] \times \{0\}$)
- $m_k \preceq m_0$

Alors la clôture transitive de $\preceq \cup \preceq'$ est un ordre total inclus dans $\preceq_0$.

Démonstration. C'est une conséquence directe du bon parenthésage et de sa définition. □

L'algorithme d'ordonnancement des mots s'inspire de cette propriété appliquée récursivement à partir de l'ordre vide et du mot correspondant à l'unique conclusion positive, puis on remplace $\preceq$ par $\preceq \cup \preceq'$ et m_0 par m_i et pour tout k, m_i par m_k. Puisque le réseau est connexe, on arrive à ordonner tous les mots les uns par rapport aux autres.

FIG. 6.11 – Ordre induit sur les mots par les axiomes

L'algorithme 1 vérifie qu'un ordre existe qui rend le réseau correct pour le fragment non commutatif et le construit. Il débute par

$$\textbf{Ordre}(m_0, \textbf{longueur}(m_0), \textbf{Gauche}, G, \textbf{null}, \{\text{mot} = m_0; \text{indice} = \textbf{longueur}(m_0)\})$$

où m_0 est le mot correspondant à la conclusion positive et G est l'ensemble des mots moins m_0. L'ordre est obtenu en un temps en $lp\log(lp)$. De plus, **mot**(x) est une fonction qui renvoie le mot auquel l'atome $A_{\mathcal{H}(\Pi)}(x)$ appartient et **place**(x) calcule la position de l'atome $A_{\mathcal{H}(\Pi)}(x)$ dans **mot**(x).

6.4.5 Le choix lexical

Jusqu'à présent, nous n'avons considéré que la réalisation syntaxique car sur elle portait essentiellement les difficultés relevées dans [MM97]. La procédure que nous proposons reste tout à fait standard, construisant un arbre de possibilités qu'il s'agit ensuite de tester. Par essence, le procédé s'avère donc exponentiel.

Dans l'analyse sémantique d'une expression, on constate que les constantes typées qui étiquettent les conclusions négatives réapparaissent toutes dans le réseau final. En effet, ces conclusions ne font jamais partie des coupures donc restent présentes au cours de l'analyse sémantique. C'est une conséquence directe de l'élimination des coupures puisqu'on a la preuve sous forme d'un réseau du séquent $\vdash [\Delta]\Gamma$ que l'on réduit pour obtenir la preuve de $\vdash \Gamma$ sans coupure. Il s'ensuit que toute constante typée présente dans la forme sémantique (avec éventuellement sa multiplicité) à réaliser est présente (avec sa multiplicité) dans les formes sémantiques des items lexicaux utilisés[23]

Théorème 59. *Soit un lexique dont la forme sémantique de chaque entrée lexicale :*

[23]Cela correspond à l'utilisation de λ-termes linéaires. Sans linéarité, pour conserver cette propriété de retrouver les

Algorithme 1 Ordre($LastWord, i,$ orientation, $S, LeftBound, RightBound$)

$L' \leftarrow [\,]; L'' \leftarrow [\,]; NewWord =$ **word**($LastWord[i]$); $j =$ **place**($LastWord[i]$)
if $NewWord \notin S$ **then** {Si le mot est déjà ordonné}
  if orientation $=$ Left **then**
    if $LeftBound$.mot $=$ $NewWord$ et $j \neq LeftBound$.indice **then**
      FALSE
    else if $LeftBound$.mot $=$ $NewWord$ et $i > 0$ **then**
      Ordre($LastWord, i-1,$ orientation, $S, LeftBound, \{$mot $=$ $LastWord$; indice $= i\}$)
    else if $LeftBound$.mot $=$ $NewWord$ et $j \not< $ longueur($LeftBound$.mot) $- 1$ **then**
      Ordre($LeftBound, j+1,$ Right, $S, \{$mot $=$ $LeftBound$.mot; indice $= j\}, \{$mot $=$ $LastWord$; indice $= i\}$)
    else
      FALSE
    end if
  else if $RightBound$.mot $=$ $NewWord$ et $j+1 \neq RightBound$.indice **then** {l'orientation est Right}
    FALSE
  else if $LeftBound$.mot $=$ $NewWord$ et $i <$ longueur($LastWord$) $- 1$ **then**
    Ordre($LastWord, i+1,$ orientation, $S, \{$mot $=$ $LastWord$; indice $= i\}, RightBound$)
  else if $LeftBound$.mot $=$ $NewWord$ et $j > 0$ **then**
    Ordre($RightBound, j-1,$ Left, $S, \{$mot $=$ $LastWord$; indice $= i\}, \{$mot $=$ $RightBound$.mot; indice $= j\}$)
  else
    FALSE
  end if
else if orientation=Right **then** {$NewWord$ n'a pas encore été ordonné}
  if $j > 0$ **then**
    $L' \leftarrow$ **Ordre**($NewWord, j-1,$ Left, $S \backslash NewWord, \{$mot $=$ $LastWord$; indice $= i\}, \{$mot $=$ $NewWord$; indice $= j\}$)
  end if
  if $j <$ longueur($NewWord$) $- 1$ **then**
    $L'' \leftarrow$ **Ordre**($NewWord, j+1,$ Right, $S \backslash NewWord, \{$mot $=$ $NewWord$; indice $= j\}, RightBound$)
  end if
else {$NewWord$ n'a pas encore été ordonné et l'orientation est Left}
  if $j > 0$ **then**
    $L' \leftarrow$ **Ordre**($NewWord, j-1,$ Left, $S \backslash NewWord, LeftBound, \{$mot $=$ $NewWord$; indice $= j\}$)
  end if
  if $j <$ longueur($NewWord$) $- 1$ **then**
    $L'' \leftarrow$ **Ordre**($NewWord, j+1,$ Right, $S \backslash NewWord, \{$mot $=$ $NewWord$; indice $= j\}, \{$mot $=$ $LastWord$; indice $= i\}$)
  end if
end if
Return $L' + [NewWord] + L''$

– *est linéaire*
– *comprend au moins une constante typée.*

Alors le processus complet de génération d'une expression (étant donnée une forme sémantique trouver les expressions syntaxiquement correctes qui la réalise) est décidable.

Démonstration. D'après ce qui précède, il existe un multi-ensemble fini d'items lexicaux pouvant être utilisés, ce qui donne un espace de recherche fini. Comme la réalisation syntaxique est décidable, le processus complet est décidable. □

Dans le cas général (il existe des items n'ayant pas de constante typée dans leur forme sémantique), nous n'avons aucune certitude. Dans la suite néanmoins, nous ferons l'hypothèse que toutes les formes sémantiques possèdent au moins une constante typée. Notons toutefois que pour un item lexical n'ayant aucune constante, toutes les conclusions des liens axiomes sont prémisses de la conclusion (positive). Leur contribution s'apparente donc à celle des « détours ». Autrement dit, si la sélection des items lexicaux *avec constante* ne permet pas la résolution de l'équation pour $\sigma_4 = 0$, aucune expression, même en tenant compte des items lexicaux sans constantes, ne peut être engendrée par la grammaire.

En même temps que la sélection des items lexicaux se fait la numérotation des atomes qui permet, lors de la recherche de preuve, de faire correspondre les atomes avant et après la réduction (en fait, il s'agit de s'assurer que les colonnes des matrices correspondent bien au même atome dans U et dans $\mathbf{Res}(U, \sigma)$, comme l'illustre la section 6.5). À chaque atome, on associe un triplet (dit de *références*) $(\text{label}, \text{adresse}, \text{num})$. Le label correspond à celui de la conclusion si celle-ci est négative, il est vide sinon (ϵ). L'adresse correspond à l'adresse de l'atome dans l'arbre syntaxique de la formule. Le numéro est instancié au cours du traitement.

Si l'on a un ordre total (dont $(\epsilon, \epsilon, \text{num})$ est le maximum) sur les références (ne tenant pas compte de num), chaque axiome peut s'exprimer par un couple $(\text{ref}_1, \text{ref}_2)$ (avec ref_1 plus petit que ref_2). On peut alors exprimer l'appariement de tout réseau π par un multi-ensemble $A(\pi)$ d'axiomes ainsi décrits.

On dit que deux références r_1 et r_2 s'unifient si elles ont même label et même adresse. Un axiome (r_1, r_2) s'unifie avec (r'_1, r'_2) si l'on a l'un des cas suivants

– r_1 et r'_1 et r_2 et r'_2 s'unifient ;
– r_1 et r'_1 s'unifient et r_2 a pour label ϵ.

Supposons que l'on veuille engendrer une expression de forme sémantique Π_0, la première étape consiste à numéroter chacun de ses atomes (comme on le souhaite). On définit ensuite $A_0 = A(\Pi_0)$. Soit alors π un réseau sémantique du lexique. A_0 est modifié de la manière suivante :

– pour tout $(r_1, r_2) \in A(\pi)$, on essaye d'unifier (r_1, r_2) avec un élément (r'_1, r'_2) (ou (r'_2, r'_1)) de A_0 :
  – si r_1 et r'_1 et r_2 et r'_2 s'unifient, on donne à r_1 et r_2 les numéros de r'_1 et r'_2 et $A_0 = A_0 \backslash \{(r'_1, r'_2)\}$. Cette étape permet de s'assurer que tous les axiomes de π dont les conclusions sont prémisses d'une conclusion négative (donc étiquetée par une constante typée) se retrouvent dans Π_0, ce qui est nécessaire car ces conclusions, comme on l'a vu, ne sont pas modifiées lors de l'analyse sémantique ;
  – si r_1 et r'_1 s'unifient et r_2 a pour label ϵ, $A_0 = A_0 \backslash \{(r'_1, r'_2)\}$ et on donne à r_1 le numéro de r'_1 et si le label de r'_2 n'est pas ϵ, $A_0 = A_0 \uplus \{(r'_2, (\epsilon, \epsilon, 0))\}$. Cette étape permet de dire que si un axiome de π dont une conclusion est premisse d'une conclusion négative étiquetée mais dont l'autre conclusion est prémisse de la conclusion positive de π, il faudra par la

constantes des réseaux sémantiques lexicaux dans le réseau final, il ne faut pas que du matériel sémantique puisse disparaître, par exemple avec des λ-termes comme $\lambda x.\lambda y.x$. De tels termes ne peuvent pas exister si l'on interdit l'utilisation de la règle d'affaiblissement, comme le propose [Per90].

suite trouver un autre axiome du même genre (dans un réseau sémantique du lexique) dont la première référence s'unifiera à r_2' ;

– si un élément de $A(\pi)$ n'a pas réussi à s'unifier, alors π ne correspond pas.

Si l'on applique cette procédure au réseau sémantique de chaque item lexical (éventuellement plusieurs fois car un même item lexical peut être utilisé plusieurs fois, y compris avec le même réseau sémantique), soit on échoue systématiquement et le problème n'a pas de solution, soit $A_0 = \emptyset$ et on a un ensemble d'items lexicaux et leur forme sémantique π_i dont les atomes (qui ne sont pas dans la conclusion positive) sont numérotés pour correspondre avec ceux de π_0.

Évidemment, cela n'assure pas que chaque ensemble de π_i donne une solution au problème de la génération pour Π_0. Il faut encore essayer les étapes de réalisation syntaxique et d'ordonnancement des mots. Par contre toute solution est présente sous forme d'un ensemble de π_i. En particulier, si le lexique le permet, on obtiendra des paraphrases.

6.4.6 Toutes les étapes

Cette section résume la proposition pour la génération dans sa totalité. C'est celle mise en œuvre dans l'implémentation. La plupart des étapes a été décrite dans les sections précédentes.

On suppose donné un lexique dont les réseaux sémantiques :

– possèdent au moins une constante typée ;

– font partie du fragment intuitionniste ;

et un réseau sémantique (intuitionniste) Π_0 dont on veut donner une expression syntaxique.

Pour ceci :

1. on choisit les items lexicaux de la manière décrite ci-dessus. Cela conduit à un ensemble de multi-ensembles d'items lexicaux avec leur réseau sémantique dont les atomes sont numérotés pour correspondre à celle de Π_0 ;
2. pour chaque ensemble de réseaux sémantiques, on numérote les autres atomes (de façon à respecter les contraintes sur les (e_i)), et on forme U et σ comme dans la section 6.4.1 (en particulier en éliminant toutes les coupures entre liens par et liens tenseur). On vérifie et on calcule, par des opérations matricielles, la ou les solutions commutatives ;
3. on vérifie que ces solutions en sont bien lorsqu'on revient au *réseau complet* (avec les arbres syntaxiques) puisque la résolution s'est faite avec des coupures apparaissant uniquement sur des liens axiomes. On peut ajouter la vérification que ce sont des réseaux du calcul de Lambek sans séquence vide (voir section 5.2) ;
4. pour chaque solution restante, il convient encore pour passer au cas non commutatif de vérifier qu'il existe un ordre correct des mots dans l'expression, et de le calculer le cas échéant ;
5. à ce stade, si l'on n'a pas de solution, c'est qu'il n'en existait pas. Il peut y en avoir plusieurs, des paraphrases. Pour leur donner un ordre de préférence, on peut les classer par ordre croissant de complexité suivant la mesure de [Mor00] décrite à la section 5.4.2. Cependant, si l'expression ainsi choisie est ambiguë, cela ne garantit pas Π_0 est son interprétation préférée. Une procédure plus rigoureuse, mais plus couteuse car demandant de calculer toutes les interprétations de toutes les expressions engendrées, serait de choisir l'expression pour laquelle l'interprétation par Π_0 est la moins complexe.

Ainsi, nous avons :

– une opération de choix des items lexicaux qui dépend des constantes présentes dans le réseau sémantique cible ;

– une opération de réalisation syntaxique, matricielle, dont la complexité est déterminée par les items lexicaux choisis qui fixe la valeur de σ_4.

À première vue, deux étapes introduisent essentiellement de la complexité : celle du choix lexical, et celle des éléments lexicaux pour lesquels $\sigma_4 \neq 0$. Jusqu'à présent, seuls de petits lexiques, plutôt propres à tester le fonctionnement du prototype, ont permis des expérimentations. Si l'étape du choix lexical ne semble, pour des raisons de complétude, que pouvoir bénéficier d'heuristiques (par exemple, on pourrait ordonner les ensembles potentiels de solutions par nombre décroissant d'items lexicaux utilisés, séparer les candidats pour lesquels $\sigma_4 = 0$ et les autres, etc.), il conviendrait d'établir la multiplicité maximum d'une constante typée dans un lexique pour connaître réellement la conséquence de ce choix sur la performance.

Il en est en partie de même pour le nombre d'items lexicaux pour lesquelles $\sigma_4 \neq 0$, et leur fréquence dans le lexique pour savoir si cela rend le problème intraitable en pratique. Néanmoins, pour cette étape, la complexité gagnerait sans doute à intégrer plus tôt dans le processus l'étape d'ordonnancement. En effet, ne serait-ce qu'à partir de l'ordre partiel donné par les liens qui ne font pas de détours, la construction des solutions en intégrant chaque détour doit respecter cet ordre partiel. Comme les deux extrémités d'un même détour appartiennent au même mot, on peut ainsi avoir des indications sur les liens qui doivent être cassés par un détour, ou bien ceux qui ne le peuvent pas du tout. Une réflexion autour de cette considération permettrait peut-être, si cela s'avère nécessaire, d'améliorer la complexité de la partie réalisation syntaxique du processus de génération.

Par rapport à la méthode qui, étant donné un ensemble d'entrées lexicales avec leur multiplicité, construirait toutes les phrases correctes constituées de ces mots et comparerait leur représentation sémantique à la formule initiale, l'étape 2 est cruciale. En effet, elle permet de ne construire que les assemblages de mots (ce ne sont pas encore des phrases, il reste à vérifier s'il est possible d'ordonner correctement ces mots) dont l'analyse sémantique est bien celle souhaitée. Dans le cas où les items lexicaux sont tels que $\sigma_4 = 0$, il y a donc au plus une solution calculée en temps quadratique, relativement au nombre d'atomes. Bien qu'elle soit inconnue, la complexité du calcul de Lambek est probablement supérieure. Cette inconnue empêche une comparaison précise dans le cas où $\sigma_4 \neq 0$, car il faut alors filtrer un nombre exponentiel de solutions commutatives. Notons toutefoisque même avec la methode construction et vérification, la recherche de preuve dans le calcul de Lambek ne doit pas se faire sur un ordre connu des mots mais également trouver les ordonnancement des mots corrects et les preuves qui vont avec.

6.5 Exemple

Pour cet exemple, nous utilisons le lexique du tableau 6.2[24]. Supposons que nous voulions engendrer une expression de sens $(\mathbf{l}\ \mathbf{j})(\mathbf{w}\ \mathbf{g})$, soit le réseau Π_1 de la figure 6.12(a) (pour lequel la première étape a consisté à numéroter les atomes). Sans décrire l'étape de sélection et de numérotation, on obtient les items lexicaux avec la numérotation de la figure 6.12(b).

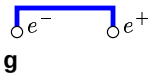
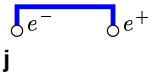
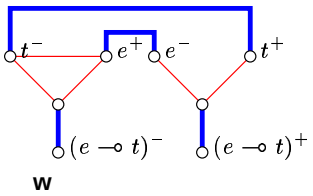
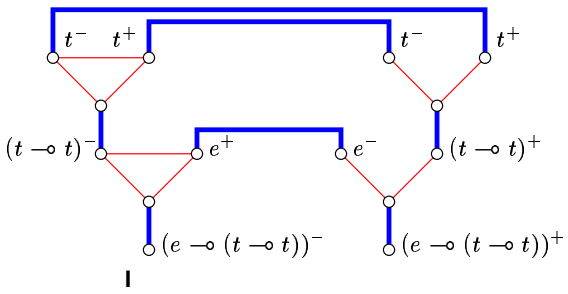
Pour la numérotation, rappelons que l'on décompose les matrices de la manière suivante :

$$U = \begin{bmatrix} 0 & U_1^{(p,p)} & 0 \\ {}^tU_1^{(p,p)} & 0 & 0 \\ 0 & 0 & U_3^{(n,n)} \end{bmatrix} \quad \sigma = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \sigma_1^{(p,p)} & \sigma_2^{(p,n)} \\ 0 & \sigma_3^{(n,p)} & \sigma_4^{(n,n)} \end{bmatrix} \quad \Pi_0 = \begin{bmatrix} \Pi_1^{(p,p)} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

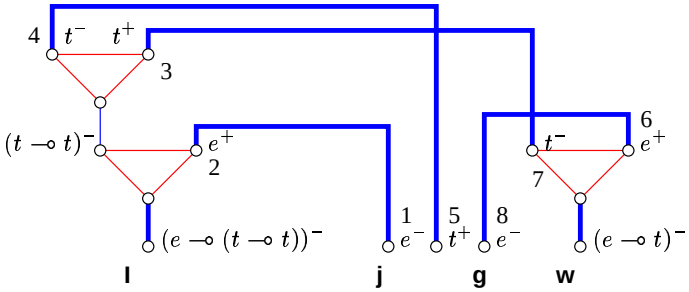
avec ici :

- $p = 8$
- $n = 6$

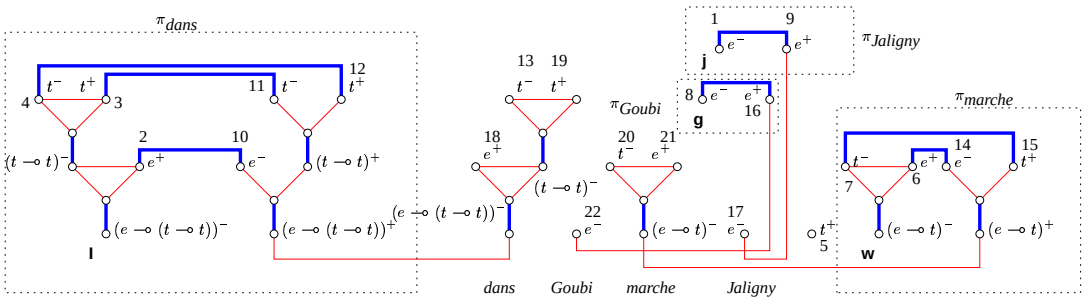
[24] Goubi est un personnage de roman, inspiré toutefois d'une personne réelle, et Jaligny-sur-Besbre est un village de l'Allier.

	type syntaxique	forme sémantique	réseau sémantique
Goubi	*np*	$\mathbf{g} : e$	
Jaligny	*np*	$\mathbf{j} : e$	
marche	$np\backslash S$	$\lambda x.\mathbf{w}\ x : e \multimap t$	
dans	$(S\backslash S)/np$	$\lambda x.\lambda y.(\mathbf{l}\ x)y : e \multimap t \multimap t$	

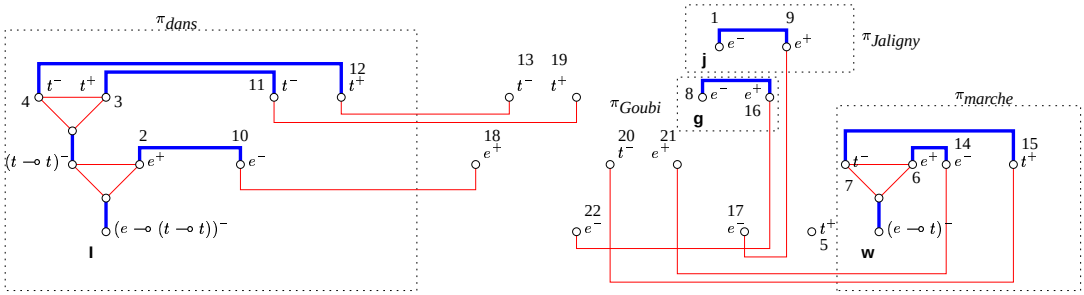
TAB. 6.2 – Lexique $\mathbf{Lex}_{10}$



(a) Π_1 numéroté



(b) Coupure et marquage des atomes avant de calculer U_1



(c) Calcul de U_1

FIG. 6.12 – Les données du problème

Il faut maintenant lier la conclusion positive numérotée 5. Dans Π_1 (figure 6.12(a)), on voit que l'atome 5 est relié à l'atome 4. Pour savoir à quel atome le relier dans $\overline{U}$, on cherche dans le réseau où l'on a réduit toutes les coupures entre liens par et tenseurs (figure 6.12(c)) quel est l'atome relié à 4 après la coupure. C'est l'atome 13. U_1 est maintenant connue, qui indique les liens entre les atomes compris entre 1 et 8 (qui ne sont pas prémisses d'un lien coupure) et ceux compris entre 9 et 16 (prémisses d'un lien coupure) :

1 et 9	2 et 10	3 et 11	4 et 12
5 et 13	6 et 14	7 et 15	8 et 16

Bien entendu, Π_1 et σ sont également connues. On a finalement[25] :

$$U_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \qquad \Pi_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

$$\sigma_1 = \begin{array}{c} \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ 16 \end{array} \begin{array}{c} \begin{array}{cccccccc} 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \qquad \sigma_2 = \begin{array}{c} \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ 16 \end{array} \begin{array}{c} \begin{array}{cccccc} 17 & 18 & 19 & 20 & 21 & 22 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

$$= PI_6Q$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$* I_6 * \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

[25] On a indiqué à côté et au-dessus des matrices, quand nécessaire, les numéros des atomes auxquels les lignes et les colonnes correspondent.

Et donc

$$A = ({}^tU_1\Pi_1 - \sigma_1\,{}^tU_1)U_1 \qquad B = P^{-1}A\,{}^tP^{-1}$$

$$= \begin{bmatrix} 0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1\\ 0&0&1&0&0&0&0&0\\ 0&0&0&0&0&1&0&0 \end{bmatrix} \qquad = \begin{bmatrix} 0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&1&0&0&1&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0 \end{bmatrix}$$

Soit, finalement

$$\begin{aligned} X &= Q^{-1}Y\,{}^tQ^{-1}\\ &= U_3\\ &= \begin{array}{c} \\ 17\\18\\19\\20\\21\\22 \end{array} \begin{array}{c} \begin{array}{cccccc} 17 & 18 & 19 & 20 & 21 & 22 \end{array}\\ \begin{bmatrix} 0&1&0&0&0&0\\ 1&0&0&0&0&0\\ 0&0&0&1&0&0\\ 0&0&1&0&0&0\\ 0&0&0&0&0&1\\ 0&0&0&0&1&0 \end{bmatrix} \end{array} \end{aligned}$$

N'oublions pas que U_3 indique les liens entre les atomes numérotés de 17 à 22, soit les liens entre les atomes :

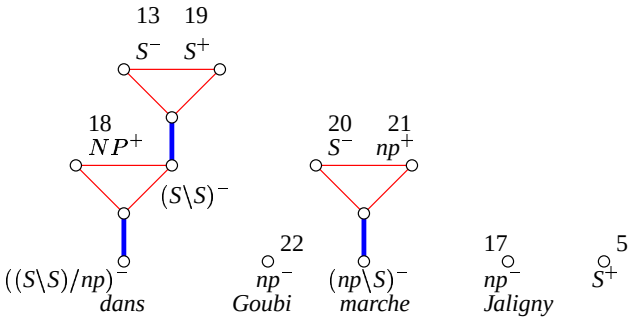
17 et 18 19 et 20 21 et 22

La figure 6.13(a) montre les arbres syntaxiques des items lexicaux, avec la numérotation calculée à partir de la numérotation sémantique. La figure 6.13(b) pose les liens définis par U_3. Il ne reste plus qu'à vérifier qu'il existe un ordre des conclusions pour lequel ce réseau est un réseau non commutatif, et le calculer. Cela se fait aisément pour obtenir le réseau de la figure 6.13(c).

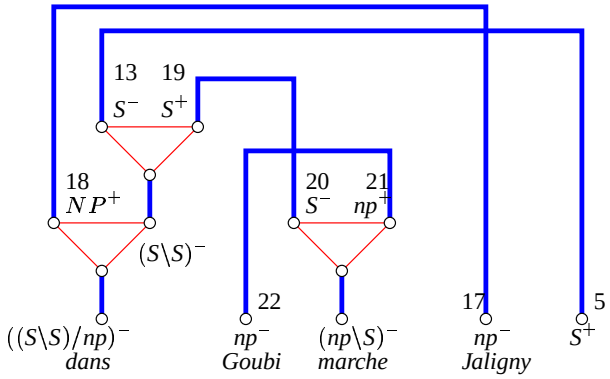
On trouvera en annexe B des exemples de générations qui mettent en avant la génération de paraphrases et la génération en anglais d'expressions dont la représentation sémantique a été calculée à partir d'une expression en français. Et également, à la figure B.20, le traitement automatique de cet exemple.

6.6 Commentaires et perspectives

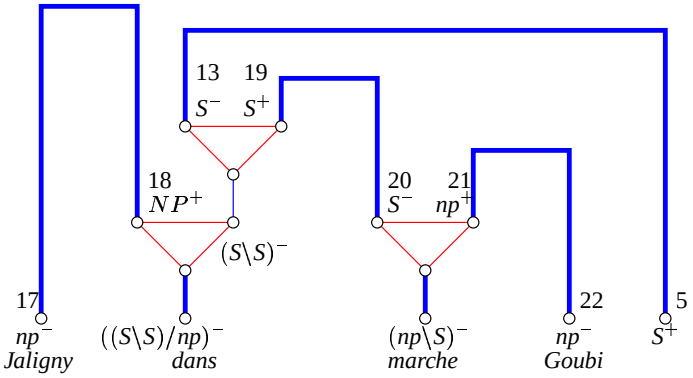
Nous avons montré que les grammaires de types logiques qui vérifient certaines propriétés — la linéarité des λ-termes des formes sémantiques lexicales et la présence pour chacun d'eux d'au moins une constante — sont intrinsèquement réversibles. Nous donnons en outre pour ces grammaires un algorithme original de recherche de preuve guidée par un réseau. Ce réseau cible permet de réaliser la recherche d'une preuve en temps polynomial, alors que la recherche de preuve au sens habituel en logique linéaire multiplicative est un problème NP-complet. Toutefois, d'autres parties de l'algorithme, telles que le choix des items lexicaux, restent exponentielles. Tout en apportant des solutions à certains problèmes de génération avec les grammaires de types logiques, notre approche met en évidence de nouvelles questions.



(a) La forêt syntaxique numérotée



(b) U_3 appliqué à la forêt syntaxique



(c) Calcul de l'ordre des mots

FIG. 6.13 – Report des liens calculés sur l'arbre syntaxique

Avec un nouvel éclairage sur l'utilisation de λ-termes, nous esquivons le problème de l'inversion de la β-réduction grâce à une approche uniformisée de la syntaxe et de la génération par les réseaux. Quelle est la robustesse de ce résultat ? Nous avons vu, notamment avec [SvNMP89][26] et [vN93][27], l'importance de la capacité de modélisation des phénomènes syntaxiques par la grammaire sur les propriétés de réversibilité de cette grammaire. Or, même s'il ne porte que sur le calcul de Lambek, ce résultat a pour intérêt de s'appuyer sur des propriétés logiques des réseaux de preuve. En ce sens, puisque différentes approches pour étendre la couverture du calcul de Lambek s'appuyent sur la logique linéaire, comme le montre la section 5.5, elles devraient pouvoir profiter de la réponse apportée ici au problème de génération. Réciproquement, la capacité plus fine d'analyse syntaxique de ces grammaires devrait avoir une influence sur les propriétés nécessaires à la réversibilité (voir note 17) en offrant une analyse de phénomènes linguistiques (comme les constituants discontinus) qui préserve des propriétés sémantiques de la grammaire utiles à la génération (comme la monotonicité dans le cas illustré par l'exemple de *call up* de représentation sémantique monotone **phone** chez [Mor94], mais pas chez [SvNMP89]).

Parallèlement à cette extension de la couverture syntaxique, les propriétés requises sur les réseaux sémantiques ne limitent pas l'usage de ceux-ci aux seuls réseaux intuitionnistes. S'ils suivent un minimum de propriétés (essentiellement savoir quelle conclusion du réseau sémantique et quelle conclusion syntaxique relier par une coupure), il n'est pas nécessaire qu'ils restent intuitionnistes et correspondent à des λ-termes. Cela montre la généralité de cette méthode de recherche de preuve d'une part, et d'autre part qu'une éventuelle proposition de représentation sémantique avec des réseaux de preuve en général, s'il en existe de pertinente, permettrait de garder les propriétés propices à la génération.

La section 3.3 montre que les réseaux de preuve peuvent aussi exprimer la non-linéarité, grâce aux connecteurs exponentiels, et coder des λ-termes non linéaires. La question de la prise en compte de ces cas pour la génération se pose donc naturellement. La géométrie de l'interaction apporte une réponse théorique à cette question : la formule d'exécution y est toujours valable. Par contre, l'interprétation des preuves fait qu'on est amené à y résoudre des équations sur des opérateurs linéaires. Une solution consisterait peut-être à donner un modèle dans lequel les équations seraient plus facilement résolubles. Du point de vue de l'uniformité de la représentation, nous avons parcouru une première étape en étendant la syntaxe des R&B-graphes aux réseaux de **MELL**.

Toujours dans cette optique, mentionnons également que le fait d'exprimer la sémantique d'un pronom réfléchi comme *se* (*himself*) par un réseau plutôt que par $\lambda P.\lambda x.(Px)x$ permet de faire apparaître un lien contraction. À l'analyse, cela permet clairement de distinguer les phrases *Jean se rase* de *Jean rase Jean* (où il y a deux entités *Jean*). Réciproquement, à la génération, la présence du lien contraction permet de détecter la présence d'un pronom réfléchi.

On rejoint le problème, éludé jusqu'à présent, de l'équivalence logique [vN93, Shi93]. Ce problème consiste à obtenir un générateur capable de produire des expressions non seulement à partir d'une représentation sémantique canonique, mais encore à partir de toute forme logique qui lui est équivalente. Sans vouloir entrer dans les détails, reprenons deux aspects de ce problème exposés dans [Shi93]. Le premier est celui du calcul même de l'équivalence logique, qui n'est simplement pas toujours possible au premier ordre. Le second tient plus à la pertinence de l'utilisation de la logique du premier ordre, et de sa définition de l'équivalence, comme représentation. En particulier, si p est la représentation d'une expression s, elle est logiquement équivalente à $p \wedge (q \vee \neg q)$ (l'interprétation par exemple de l'expression s' : *s qu'il pleuve ou qu'il ne pleuve pas*). Pourtant, il est peu probable que l'on veuille exprimer indifféremment s ou s'. Notre approche ne traite pas ces problèmes. Par contre,

[26] Avec la non-monotonie sémantique de *call up*.

[27] Avec des règles de combinaison des mots plus complexes que la seule concaténation.

sa robustesse (dans le sens où elle n'est pas réservée aux seuls réseaux intuitionnistes et aux seules constantes de la logique des prédicats) devrait permettre d'envisager des extensions qui puissent y répondre en partie.

Vis-à-vis des solutions du problème de génération pour les grammaires d'unification, nous ne tenons compte que de la partie sémantique pour la sélection des items lexicaux. Elle débute le processus, lequel se poursuit par le calcul *simultané* de tous les liens entre les constantes, c'est-à-dire de toutes les dépendances foncteur-argument. La forme sémantique guide plus globalement la recherche par le réseau cible que la tête sémantique obtenue à chaque étape. Cela, avec le fait de considérer l'aspect logique des grammaires catégorielles, permet par rapport aux grammaires catégorielles d'unification, d'éviter les problèmes liés aux règles supplémentaires, telles que l'élévation de type. On présente donc une résolution partielle des problèmes liés à la vacuité sémantique. Par contre, lors du choix lexical, le problème lié à l'absence de constantes dans la forme sémantique reste entier. Notons que ce problème est évoqué dans [Kay96] sans proposition pour le résoudre. Une solution serait de tenir compte également de la partie syntaxique apportée par les items lexicaux. Par exemple, si *que* est de catégorie syntaxique $Comp/S$ pour une représentation sémantique $\lambda x.x$, si un verbe *dire* de type syntaxique $(np\backslash S)/Comp$ a été sélectionné grâce à la constante **dit**, lorsque toutes les sélections lexicales ont eu lieu, on peut s'apercevoir qu'aucun élément n'apporte le type syntaxique *Comp*, et que ce dernier doit être introduit par un élément lexical dont la représentation sémantique ne comporte aucune constante. Mais seule l'expérimentation permettrait de vérifier si cette particularisation syntaxique des éléments lexicaux sans constante sémantique suffit. (Le lexique proposé par [Mor94] vérifie cette propriété — *to, that* — sauf pour les pronoms réfléchis. Mais ces derniers peuvent être repérés dans les réseaux par l'utilisation du lien contraction, ce que ne permettent pas les λ-termes.)

Nous voulons également souligner des perspectives qui s'éloignent du problème pur de la réversibilité. La première, suivant les remarques de la section 4.4, concerne l'apprentissage à partir de données syntaxiques *et* sémantiques. La procédure de génération que nous avons proposée permet d'inférer une preuve, c'est-à-dire un λ-terme qui, appliqué à chacune des entrées lexicales, conduit à la représentation sémantique attendue sous forme normale. Nous pensons que cette procédure peut s'étendre au problème d'apprentissage décrit par [Tel98a] pour lequel on cherche également à inférer un λ-terme, malgré la présence dans ce cas d'une plus grande indétermination.

La seconde prespective sort du champ linguistique et insiste sur la relation entre réseaux et λ-termes. De la même manière que les réseaux ont apporté une solution efficace à un problème jusque là posé en terme d'unification de λ-termes (la réalisation syntaxique), on peut se demander si d'autres problèmes tels que le filtrage gagneraient algorithmiquement à cet éclairage. Dans la suite de [dG00], c'est un problème qui commence à être étudié. Il exprime l'une des données récurrentes de notre travail : l'influence qu'exercent l'une sur l'autre l'étude des grammaires de types logiques et celle de la théorie de la démonstration.

Annexe A

Preuve du théorème 57

Dans cette annexe, qui contient la démonstration du théorème 57, nous reprenons les hypothèses et les notations du chapitre 6 : $\overline{U}$ est un réseau correct, dont les coupures ont lieu entre liens axiomes, et dont la réduction donne $\overline{\Pi_0}$. De plus :

- U interprète les liens axiomes de $\overline{U}$;
- σ interprète les liens axiomes de $\overline{U}$;
- Π_0 interprète la forme réduite de $\overline{U}$.

Les (e_i) se répartissent en trois catégories dans $\overline{U}$:

- $\forall i \in [1,p]$, e_i n'est pas prémisse d'un lien Cut (et donc, d'après l'hypothèse énoncée ci-dessus, $A_{\overline{U}}(e_i)$ est prémisse d'un lien Cut) ;
- $\forall i \in [p+1,2p]$, e_i est prémisse d'un lien coupure mais pas $A_{\overline{U}}(e_i)$;
- $\forall i \in [2p+1,2p+n]$, e_i et $A_{\overline{U}}(e_i)$ sont prémisses d'un lien Cut.

On obtient alors :

$$U = \begin{bmatrix} 0 & U_1^{(p,p)} & 0 \\ {}^tU_1 & 0 & 0 \\ 0 & 0 & U_3^{(n,n)} \end{bmatrix} \quad \sigma = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \sigma_1^{(p,p)} & \sigma_2^{(p,n)} \\ 0 & \sigma_3^{(n,p)} & \sigma_4^{(n,n)} \end{bmatrix} \quad \Pi_0 = \begin{bmatrix} \Pi_1^{(p,p)} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Rappelons que :

- $U_1\,{}^tU_1 = {}^tU_1U_1 = 1$;
- si $\sigma_4 = 0$ alors $n \leq p$;
- comme il y a exactement un et un seul 1 par ligne et par colonne, si $\sigma_4 = 0$, chaque colonne de σ_3 a un 1 et le rang de σ_2 est n.

Lemme 60. *Soit $X \in \mathcal{M}_s(\textbf{\textit{bool}})$. Si X est nilpotente ($\exists n_0, X^{n_0} = 0$) alors $(1-X)$ est inversible et $(1-X)^{-1} = 1 + \sum_{k\geq 1} X^k$*

Démonstration. $\sum_{k\geq 1} X^k = \sum_{k=1}^{n_0} X^k$ est bien défini et

$$\begin{aligned}(1-X)(1+\sum_{k\geq 1} X^k) &= 1 + X + \sum_{k\geq 2} X^k - X - X\sum_{k\geq 1} X^k \\ &= 1\end{aligned}$$

□

Lemme 61. *Soit $X \in \mathcal{M}_s(\textbf{\textit{bool}})$ telle que $X = \left[\begin{smallmatrix} 0^{(p,p)} & 0^{(p,m)} & 0^{(p,n)} \\ a^{(m,p)} & 0^{(m,m)} & c^{(m,n)} \\ b^{(n,p)} & 0^{(n,m)} & d^{(n,n)} \end{smallmatrix}\right]$, alors pour tout $k \geq 2$*

$$X^k = \begin{bmatrix} 0^{(p,p)} & 0^{(p,m)} & 0^{(p,n)} \\ cd^{k-2}b^{(m,p)} & 0^{(m,m)} & cd^{k-1}{}^{(m,n)} \\ d^{k-1}b^{(n,p)} & 0^{(n,m)} & d^k{}^{(n,n)} \end{bmatrix}$$

Démonstration. Par récurrence sur k :

$$\begin{aligned} X^2 &= \begin{bmatrix} 0^{(p,p)} & 0^{(p,m)} & 0^{(p,n)} \\ a^{(m,p)} & 0^{(m,m)} & c^{(m,n)} \\ b^{(n,p)} & 0^{(n,m)} & d^{(n,n)} \end{bmatrix} * \begin{bmatrix} 0^{(p,p)} & 0^{(p,m)} & 0^{(p,n)} \\ a^{(m,p)} & 0^{(m,m)} & c^{(m,n)} \\ b^{(n,p)} & 0^{(n,m)} & d^{(n,n)} \end{bmatrix} \\ &= \begin{bmatrix} 0^{(p,p)} & 0^{(p,m)} & 0^{(p,n)} \\ cb^{(m,p)} & 0^{(m,m)} & cd^{(m,n)} \\ db^{(n,p)} & 0^{(n,m)} & d^2{}^{(n,n)} \end{bmatrix} \end{aligned}$$

Supposons que $X^k = \left[\begin{smallmatrix} 0^{(p,p)} & 0^{(p,m)} & 0^{(p,n)} \\ cd^{k-2}b^{(m,p)} & 0^{(m,m)} & cd^{k-1}{}^{(m,n)} \\ d^{k-1}b^{(n,p)} & 0^{(n,m)} & d^{n\,(n,n)} \end{smallmatrix}\right]$ alors

$$X^{k+1} = \begin{bmatrix} 0^{(p,p)} & 0^{(p,m)} & 0^{(p,n)} \\ a^{(m,p)} & 0^{(m,m)} & c^{(m,n)} \\ b^{(n,p)} & 0^{(n,m)} & d^{(n,n)} \end{bmatrix} * \begin{bmatrix} 0^{(p,p)} & 0^{(p,m)} & 0^{(p,n)} \\ cd^{k-2}b^{(m,p)} & 0^{(m,m)} & cd^{k-1}{}^{(m,n)} \\ d^{k-1}b^{(n,p)} & 0^{(n,m)} & d^{k\,(n,n)} \end{bmatrix}$$
$$= \begin{bmatrix} 0^{(p,p)} & 0^{(p,m)} & 0^{(p,n)} \\ cd^{k-1}b^{(m,p)} & 0^{(m,m)} & cd^{k\,(m,n)} \\ d^k b^{(n,p)} & 0^{(n,m)} & d^{k+1\,(n,n)} \end{bmatrix}$$

□

Corollaire 62. *Soit $X \in \mathcal{M}_s(\textbf{\textit{bool}})$ telle que $X = \left[\begin{smallmatrix} 0^{(p,p)} & 0^{(p,m)} & 0^{(p,n)} \\ a^{(m,p)} & 0^{(m,m)} & c^{(m,n)} \\ b^{(n,p)} & 0^{(n,m)} & d^{(n,n)} \end{smallmatrix}\right]$ soit nilpotente. X est nilpotente (et donc $1-X$ inversible) si et seulement si d est nilpotente (et $1-d$ inversible). Dans ce cas :*

$$(1-X)^{-1} = \begin{bmatrix} 1^{(p,p)} & 0^{(p,m)} & 0^{(p,n)} \\ a + c(1-d)^{-1}b^{(m,p)} & 1^{(m,m)} & c(1-d)^{-1\,(m,n)} \\ (1-d)^{-1}b^{(n,p)} & 0^{(n,m)} & (1-d)^{-1\,(n,n)} \end{bmatrix}$$

Démonstration. Avec le lemme 61, c'est une équivalence triviale. □

Lemme 63. *Avec ces définitions de U, σ et Π_0, nous avons :*

$$\textit{Res}(U,\sigma) = (1-\sigma^2)U(1-\sigma U)^{-1}(1-\sigma^2)$$
$$\Leftrightarrow$$
$$({}^tU_1\Pi_1 - \sigma_1\,{}^tU_1)U_1 = \sigma_2 U_3(1 + \textstyle\sum_{k\geq 1}(\sigma_4 U_3)^k)\,{}^t\sigma_2$$

Démonstration. Notons pour simplifier :

$$a = \sigma_1\,{}^tU_1$$
$$b = {}^t\sigma_2\,{}^tU_1$$
$$c = \sigma_2 U_3$$
$$d = \sigma_4 U_3$$

Ensuite, par définition de $\mathrm{Res}(U,\sigma)$, de U et de σ : $\mathrm{Res}(U,\sigma) = (1-\sigma^2)U(1-\sigma U)^{-1}(1-\sigma^2)$ si

et seulement si ((σU) est nilpotente et)

$$
\begin{aligned}
\Pi &= \begin{bmatrix} 1^{(p,p)} & 0^{(p,m)} & 0^{(p,n)} \\ 0^{(m,p)} & 0^{(m,m)} & 0^{(m,n)} \\ 0^{(n,p)} & 0^{(n,m)} & 0^{(n,n)} \end{bmatrix} \\
&\quad * \begin{bmatrix} 0^{(p,p)} & {U_1}^{(p,m)} & 0^{(p,n)} \\ {}^tU_1{}^{(m,p)} & 0^{(m,m)} & 0^{(m,n)} \\ 0^{(n,p)} & 0^{(n,m)} & {U_3}^{(n,n)} \end{bmatrix} \\
&\quad * \begin{bmatrix} 1^{(p,p)} & 0^{(p,m)} & 0^{(p,n)} \\ a + c(1-d)^{-1}b^{(m,p)} & 1^{(m,m)} & c(1-d)^{-1(m,n)} \\ (1-d)^{-1}b^{(n,p)} & 0^{(n,m)} & (1-d)^{-1(n,n)} \end{bmatrix} \\
&\quad * \begin{bmatrix} 1^{(p,p)} & 0^{(p,m)} & 0^{(p,n)} \\ 0^{(m,p)} & 0^{(m,m)} & 0^{(m,n)} \\ 0^{(n,p)} & 0^{(n,m)} & 0^{(n,n)} \end{bmatrix} \qquad \text{d'après le corollaire 62} \\
&= \begin{bmatrix} 0^{(p,p)} & {U_1}^{(p,m)} & 0^{(p,n)} \\ 0^{(m,p)} & 0^{(m,m)} & 0^{(m,n)} \\ 0^{(n,p)} & 0^{(n,m)} & 0^{(n,n)} \end{bmatrix} \\
&\quad * \begin{bmatrix} 1^{(p.p)} & 0^{(p,m)} & 0^{(p,n)} \\ a + c(1-d)^{-1}b^{(p,p)} & 0^{(m,m)} & 0^{(m,n)} \\ (1-d)^{-1}b^{(n,p)} & 0^{(n,m)} & 0^{(n,n)} \end{bmatrix} \\
&= \begin{bmatrix} U_1(a + c(1-d)^{-1}b)^{(p,p)} & 0^{(p,m)} & 0^{(p,n)} \\ 0^{(m,p)} & 0^{(m,m)} & 0^{(m,n)} \\ 0^{(n,p)} & 0^{(n,m)} & 0^{(n,n)} \end{bmatrix}
\end{aligned}
$$

Ce qui est équivalent à $\Pi_1 = U_1(\sigma_1\,{}^tU_1 + \sigma_2 U_3(1 + \sum_{k\geq 1}(\sigma_4 U_3)^k)\,{}^t\sigma_2\,{}^tU_1)$.

Enfin, puisque $U_1\,{}^tU_1 = 1 = {}^tU_1U_1$ nous avons

$$({}^tU_1\Pi_1 - \sigma_1\,{}^tU_1)U_1 = \sigma_2 U_3(1 + \sum_{k\geq 1}(\sigma_4 U_3)^k)\,{}^t\sigma_2$$

□

Annexe B

Implémentation

Parallèlement aux résultats, théoriques et nécessaires, prouvant entre autres la possibilité d'utiliser les grammaires de types logiques pour des tâches de génération, nous avons développé un prototype permettant l'expérimentation autour des grammaires basées sur la logique linéaire. Cette annexe décrit le prototype, essentiellement par ses fonctionnalités et des exemples d'utilisation. Ces derniers illustrent aussi des exemples linguistiques traités dans la partie principale de la thèse.

La première section reprend les différentes fonctionnalités du prototype, au niveau du prouveur automatique et au niveau de l'analyseur et générateur. Elle décrit également les formats des entrées. La seconde section présente une vue très générale de l'implémentation, fidèle aux principes qui sous-tendent les grammaires de types logiques. La dernière partie traite des exemple linguistiques en analyse et en génération, se partageant entre des exemples repris des différentes parties de la thèse et de nouveaux exemples.

B.1 Description

L'objectif de ce prototype est multiple. D'une part, il donne une plateforme d'expérimentation de grammaire autour d'un prouveur automatique en particulier en montrant les réseaux obtenus, et d'autre part il permet de concrétiser les résultats théoriques obtenus autour de la génération. Nous présentons donc les différentes parties relatives à la logique pure, à l'analyse et à la génération. La présentation des réseaux seraient inutile dans le but d'une simple analyse de textes plus conséquents, qui nécessiterait la mise au point d'une grammaire plus réaliste, mais ce n'était pas l'objectif recherché.

B.1.1 Prouveur automatique

Le prouveur automatique se base sur la représentation des réseaux par des R&B-graphes. Cela nous permet d'avoir à la fois un prouveur pour **MLL** (avec les connecteurs $\parr$ et $\otimes$) et pour le calcul ordonné (avec les connecteurs $\parr$, $\otimes$ et $<$) car ces prouveurs utilisent le même critère de correction. La partie interfacée graphiquement se limite à ces fragments, mais pour les utilisations linguisitques, elle est étendue au cas non-commutatif.

L'interface se présente comme à la figure B.1. La partie utile pour le prouveur automatique contient un champ *Formula* pour donner la formule à prouver. Le bouton *All* permet de chercher toutes les preuves de cette formule et le bouton *Creep* permet de s'arrêter à chacune des étapes (correcte ou incorrecte). Le bouton *Cut* permet d'éliminer les coupures d'un réseau, et enfin le bouton *Postscript* permet d'engendrer un fichier postscript avec le réseau courant.

Les conventions syntaxiques suivantes ont été utilisées :

- la négation $A^\perp$ est remplacée par ~A ;
- le symbole $\parr$ est remplacé par @ ;
- le symbole $\otimes$ est remplacé par $*$;
- on garde le symbole $<$;
- pour spécifier que dans un même préréseau deux atomes duaux A et $A^\perp$ sont reliés par un lien axiome, on les indice avec la même valeur (ex : A_1 et ~A_1) ;
- les conclusions d'un même préréseau sont séparées par des virgules (ex : $a_1, (\sim a_1) * b_1, \sim b_1$, voir figure B.2(a)) ;
- les conclusions de deux préréseaux sont séparés par des points-virgules (ex : $a_1, (\sim a_1) * b_1, \sim b_1; a_1, (\sim a_1) * b_1, \sim b_1$@, foir figure B.2(b)) ;
- pour indiquer une coupure entre deux formules F et G, on utilise $F!G$ comme une conclusion du préréseau, et le lien coupure a la même forme que le lien tenseur.

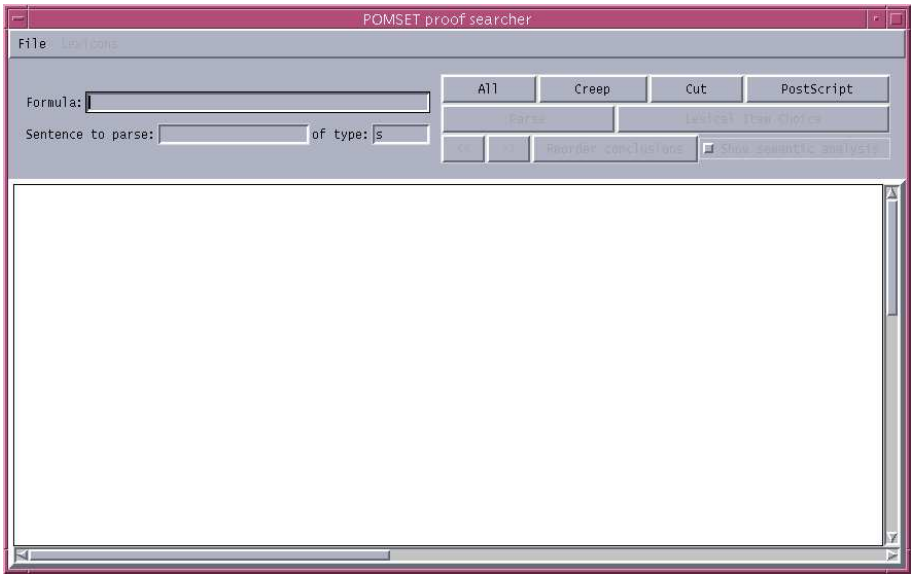
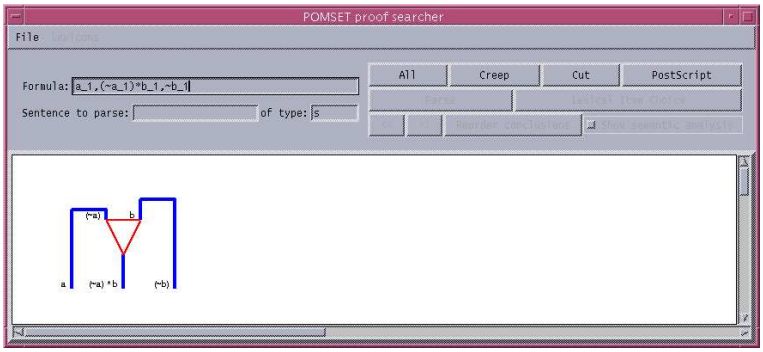
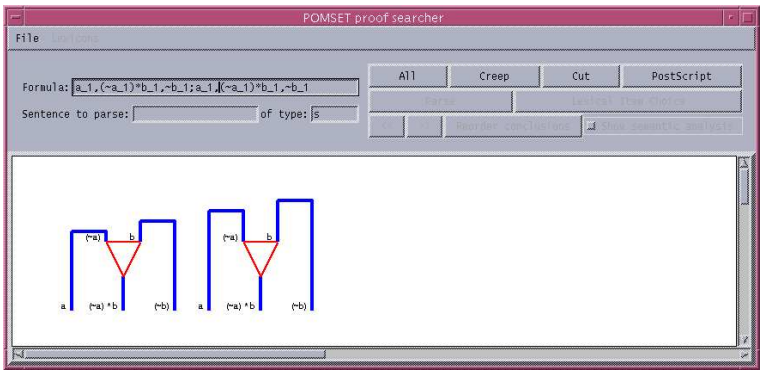


FIG. B.1 – Interface



(a) Formule : $a_1, (\sim a_1) * b_1, \sim b_1$



(b) Formule : $a_1, (\sim a_1) * b_1, \sim b_1; a_1, (\sim a_1) * b_1, \sim b_1$

FIG. B.2 – Exemple d'utilisation des formules

Si tous les liens axiomes ont été précisés, la recherche de preuve se limite à la vérification que le préréseau est correct. Pour faire une recherche de preuve plus générale, il suffit de ne pas indiquer les liens entre atomes duaux et le prouveur cherchera les associations qui donnent un réseau correct. Lorsqu'un réseau est incorrect, le æ-cycle est mis en avant par un petit disque (vert) sur les sommets du cycle, comme le montre le figure B.3. La figure B.4 montre que l'opérateur $<$ est auto-dual, et la figure B.5 montre l'utilisation d'une coupure et son élimination.

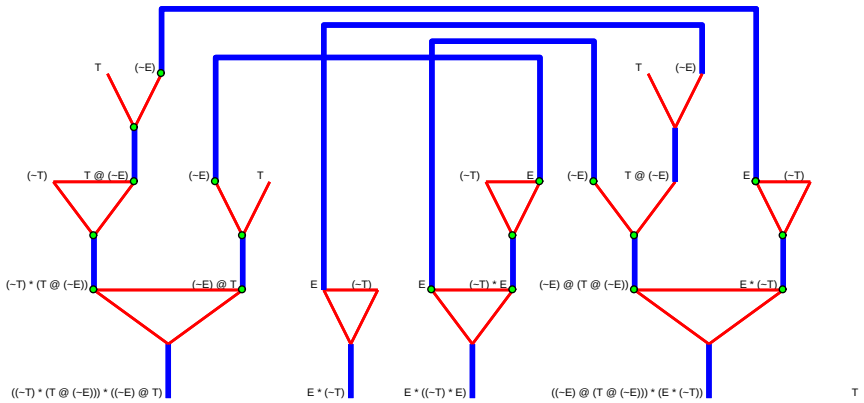


FIG. B.3 – Exemple de préréseau incorrect

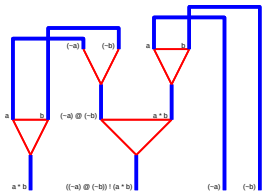
(a) Preuve de $a < b, a^{\perp} < b^{\perp}$

(b) Préréseau incorrect $a < b, b^{\perp} < a^{\perp}$

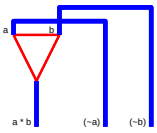
FIG. B.4 – $<$ est un opérateur auto-dual

B.1.2 Analyseur

Si l'analyseur se base sur le prouveur automatique, seuls trois critères ont pour l'instant été définis pour les systèmes déductifs que l'on peut associer à un lexique : le calcul ordonné, **MLL** et le calcul de Lambek. De même une seule stratégie d'analyse a été implantée : celle qui consiste à rechercher les liens axiomes entre atomes duaux. Il serait tout à fait envisageable d'en implanter une (par exemple pour les modules, voir section 5.5) avec des connexions par des coupures. Par ailleurs nous n'avons pas pris en compte le traitement des traits morphologiques, ce que la suggestion de la section 5.5.3 nous aurait permis de faire grâce aux types dépendants.



(a) Preuve de $\vdash [a \otimes b] a \otimes b, a^{\perp}, b^{\perp}$



(b) Élimination de la coupure et preuve de $\vdash a \otimes b, a^{\perp}, b^{\perp}$

FIG. B.5 – Élimination d'une coupure

La première étape (voir figure B.6) consiste à sélectionner un lexique. Il est alors possible d'utiliser le bouton *Parse* pour vérifier si l'expression entrée dans la zone *Sentence to parse* est bien du type indiqué dans la zone *of type*.

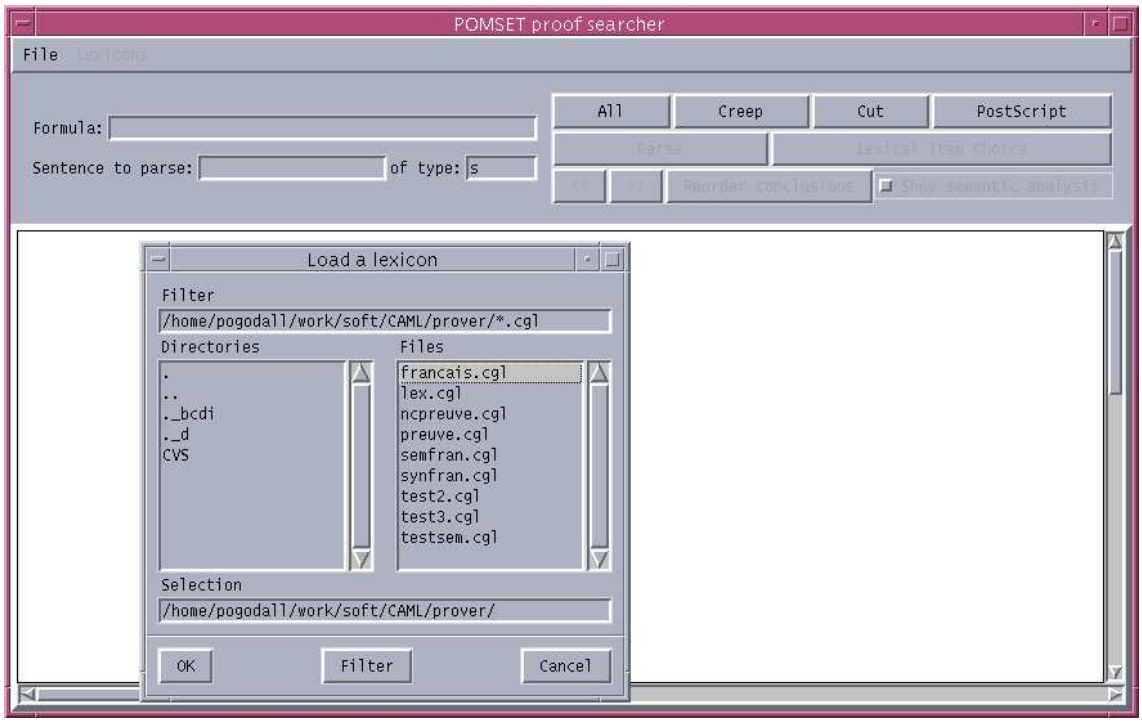


FIG. B.6 – Sélection d'un lexique

Le lexique est construit de la manière suivante :

- la première ligne contient la déclaration LEXICON puis le système déductif (*mll* ou *ncmll* pour la version non commutative) ;
- chaque ligne correspond à une ou plusieurs entrées lexicales ;
- les caractères situés sur un ligne après le caractère % ne sont pas pris en compte ;
- chaque ligne à la forme suivante : $m_1, \ldots, m_n : t_1 | \ldots | t_p$ signifiant que tous les mots m_i ont

les types t_p, avec $n, p \geq 1$;
- les formules pour les types utilisent les mêmes principes que les formules du prouveur ainsi que les connecteurs $\backslash$ et $/$;

La figure B.7 donne le contenu du lexique français utilisé dans les exemples. Notons que l'analyse de la phrase entrée pour séparer les items lexicaux et les retrouver dans le lexique est très rudimentaire, simplement basée sur le caractère blanc comme séparateur et une recherce de chaque mot dans le lexique. A priori, l'analyseur proprement dit pourrait recevoir des données de la part d'un autre module de tokenization et d'analyse morphologique basés par exemple sur des techniques à états finis [KKZ92, Kar94, KCGS96, Cha94]. Différents exemples d'analyses, correspondant à des exemples traités dans le développement de la thèse, apparaissent en dernière section de cette annexe.

```
LEXICON : ncmll
Jean, Marie, Othello, Desdemone: np
achete, deteste, voit, trouve, lit, aime,
      habite, merite : (np\s)/np
tenaillait, menacait, habitait, fit :(np\s)/np
marche, parle, vit, jure : (np\s)/pp | np\s
s'effondra, caqueta, eclosit:(np\s)/pp | np\s
lire : (np\s)/np
le, la, l' :np/n
professeur, preface, maison, cochon, apetit, loup,
     poule, oeuf, homme, femme, livre, faim, charretier :n
qui :(n\n)/(s/np) | (n\n)/(np\s)
cherche :(np\s)/ (( (np\s) /np) \(np\s))
tout, toute:(s/(np\s))/n | ((s/np)\s)/n
un, une: ((s/np)\s)/n | (s/(np\s))/n
se:((np\s)/np)\(np\s)
et, ou:(s\s)/s | ((np\s)\(np\s))/(np\s) | (np\np)/np
 | (n\n)/n | (((np\s)/np)\((np\s)/np))/((np\s)/np)
 | ((s/np)\(s/np))/(s/np) | ((s/(np\s))\(s/(np\s)))/(s/(np\s))
grand, rouge, petit, blanc, sourd : n/n
que :(n\n)/(np\s) | (n\n)/(s/np)
dans :((np\s)\(np\s))/np
a : (np\s)/(np\s) | pp/np
tres : (n/n)/(n/n)
de : pp/np | vp/(np\s)
du : pp/n | (n\n)/n
dont : (n\n)/(s/pp) | (n\n)/s
dit : (np\s)/pp
conseille: ((np\s)/vp)/pp
avidement: s\s
```

FIG. B.7 – Exemple de lexique pour du français

B.1.3 Génération

Le générateur utilise les mêmes principes que l'analyseur. Le lexique est un peu différent puisqu'il comporte également une entrée pour le réseau sémantique des items lexicaux.

Le format d'une entrée lexicale (illustré par le lexique de la figure B.8) est le suivant :

$$m_1, \ldots, m_n : t_1 \,\&\, r_1 > c_1^{(1)}, \ldots, c_{l_1}^{(1)} | \ldots | t_k \,\&\, r_k > c_1^{(k)}, \ldots, c_{l_k}^{(k)}$$

où les m_i sont des mots, les t_i leurs types syntaxiques et pour chaque t_i un réseau sémantique r_i dont les conclusions sont étiquetés par les constantes $c_j^{(i)}$, dans l'ordre indiqué pour chaque réseau. La figure B.9 présente un lexique équivalent pour de l'anglais et qui permet de passer d'une langue à l'autre.

```
LEXICON-SEM : ncmll
Goubi : np & (~e_1),e_1 > g
Paris : np & (~e_1),e_1 > p
Jaligny : np & (~e_1),e_1 > j
homme : n & (~t_1)*e_1,(~e_1)@t_1 > H
femme : n & (~t_1)*e_1,(~e_1)@t_1 > F
aime : (np\s)/np & ((~t_1)*e_1)*e_2,
       (~e_2)@((~e_1)@t_1) > A
a, dans : (s\s)/np & ((~t_1)*t_2)*e_1,
       (~e_1)@((~t_2)@t_1) > I
marche : np\s & (~t_1)*e_1,(~e_1)@t_1 > w
vit : np\s & (~t_1)*e_1,(~e_1)@t_1 > l
habite : (np\s)/np & ((~t_1)*t_2)*e_1,
       (~t_2)*e_2,e_1\(e_2\t_1) > I, l
tout,toute : (s/(np\s))/n & (((~e_1)@(~e_2))@t_1)*(~t_2),
             t_3*(t_4*(~t_1)), (e_2*(~t_3))@((e_1*(~t_4))@t_2)
             > forall, implies
             | ((s/np)\s)/n & (((~e_1)@(~e_2))@t_1)*(~t_2),
             t_3*(t_4*(~t_1)), (e_2*(~t_3))@((e_1*(~t_4))@t_2)
             > forall, implies
un,une : ((s/np)\s)/n & (((~e_1)@(~e_2))@t_1)*(~t_2),
         ((~t_1)*t_4)*t_3, (e_2*(~t_3))@((e_1*(~t_4))@t_2)
         > exists, and
        | (s/(np\s))/n & (((~e_1)@(~e_2))@t_1)*(~t_2),
        t_3*(t_4*(~t_1)), (e_2*(~t_3))@((e_1*(~t_4))@t_2)
         > exists, and
```

FIG. B.8 – Exemple de lexique, avec la sémantique, pour du français

Après une analyse syntaxique avec un lexique qui comporte une partie sémantique, différents boutons deviennent accessibles. Le bouton *Show semantic analysis* (voir figure B.10) permet de montrer le réseau correspondant à l'analyse sémantique (voir figure B.11) puis de revenir à l'analyse syntaxique (les mots utilisés du lexique sont en italique, tandis que les constantes sémantiques qui étiquettent les conclusions sont normales). Le bouton *Lexical Item Choice* permet, lorsqu'on a obtenu une analyse sémantique, de rechercher dans le lexique toutes les expressions qui, *sans tenir compte de l'ordre des mots*, auraient la même sémantique. Le bouton *Reorder conclusions* permet de filtrer parmi toutes ces expressions celles qui, en plus, permettent d'engendrer une expressions sans que les liens axiomes se croisent. Par exemple, l'expression obtenue apres une action sur *Lexical Item Choice*, à partir de la sémantique de la figure B.11, et indiquée à la figure B.12 ne passe pas le filtre de l'ordonnancement des mots : les types syntaxiques utilisés pour *tout* ne vont pas (celui qui correspond à un *np* objet est aussi utilisé pour le *np* sujet).

S'il y a plusieurs solutions, par exemple plusieurs analyse syntaxiques dans le cas de l'analyse de *tout homme aime une femme*, on peut passer de l'une à l'autre grâce aux boutons « et ». Ainsi, la figure B.13 présente-t-elle la deuxième analyse pour *tout homme aime une femme*.

```
LEXICON-SEM : ncmll
Goubi: np & (~e_1),e_1 > g
Paris: np & (~e_1),e_1 > p
Jaligny: np & (~e_1),e_1 > j
man : n & (~t_1)*e_1,(~e_1)@t_1 > H
woman : n & (~t_1)*e_1,(~e_1)@t_1 > F
loves : (np\s)/np & ((~t_1)*e_1)*e_2,
        (~e_2)@((~e_1)@t_1) > A
in : (s\s)/np & ((~t_1)*t_2)*e_1,
        (~e_1)@((~t_2)@t_1) > I
walks: np\s & (~t_1)*e_1,(~e_1)@t_1 > w
lives: np\s & (~t_1)*e_1,(~e_1)@t_1 > l
inhabits : (np\s)/np & ((~t_1)*t_2)*e_1,
           (~t_2)*e_2,e_1\(e_2\t_1) > I, l
every : (s/(np\s))/n & (((~e_1)@(~e_2))@t_1)*(~t_2),
        t_3*(t_4*(~t_1)), (e_2*(~t_3))@((e_1*(~t_4))@t_2)
        > forall, implies
        | ((s/np)\s)/n & (((~e_1)@(~e_2))@t_1)*(~t_2),
        t_3*(t_4*(~t_1)), (e_2*(~t_3))@((e_1*(~t_4))@t_2)
        > forall, implies
a : ((s/np)\s)/n & (((~e_1)@(~e_2))@t_1)*(~t_2),
    ((~t_1)*t_4)*t_3, (e_2*(~t_3))@((e_1*(~t_4))@t_2)
    > exists, and
    | (s/(np\s))/n & (((~e_1)@(~e_2))@t_1)*(~t_2),
    t_3*(t_4*(~t_1)), (e_2*(~t_3))@((e_1*(~t_4))@t_2)
    > exists, and
```

FIG. B.9 – Exemple de lexique, avec la sémantique, pour de l'anglais

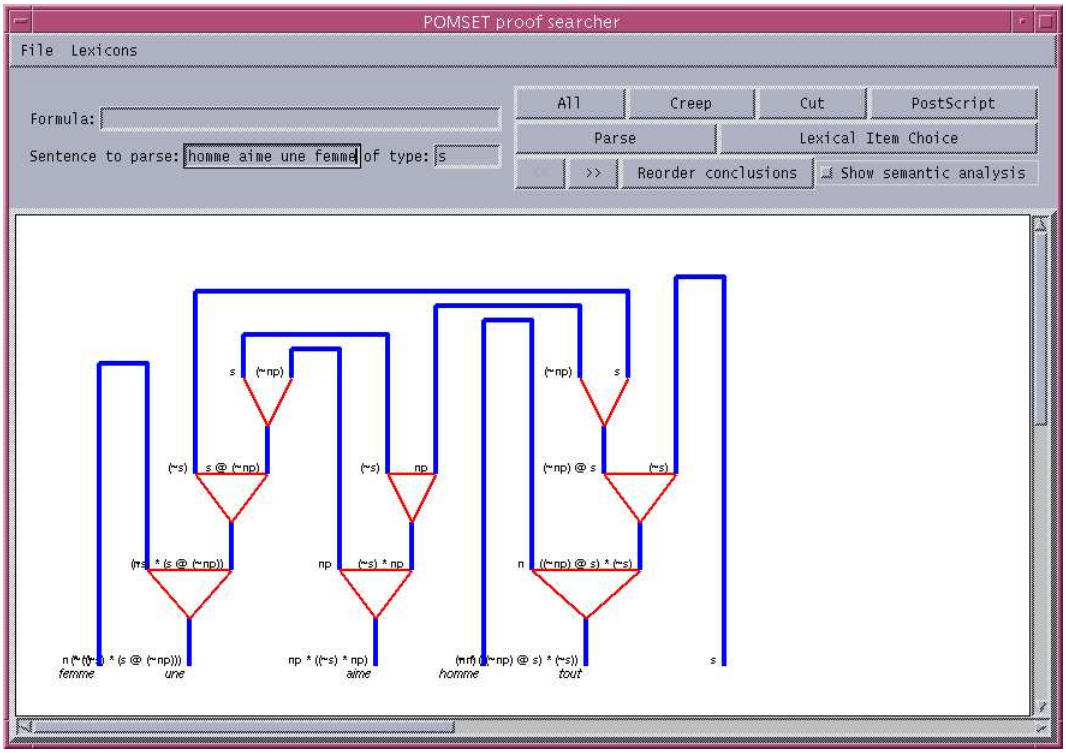


FIG. B.10 – Une première analyse syntaxique de *tout homme aime une femme*

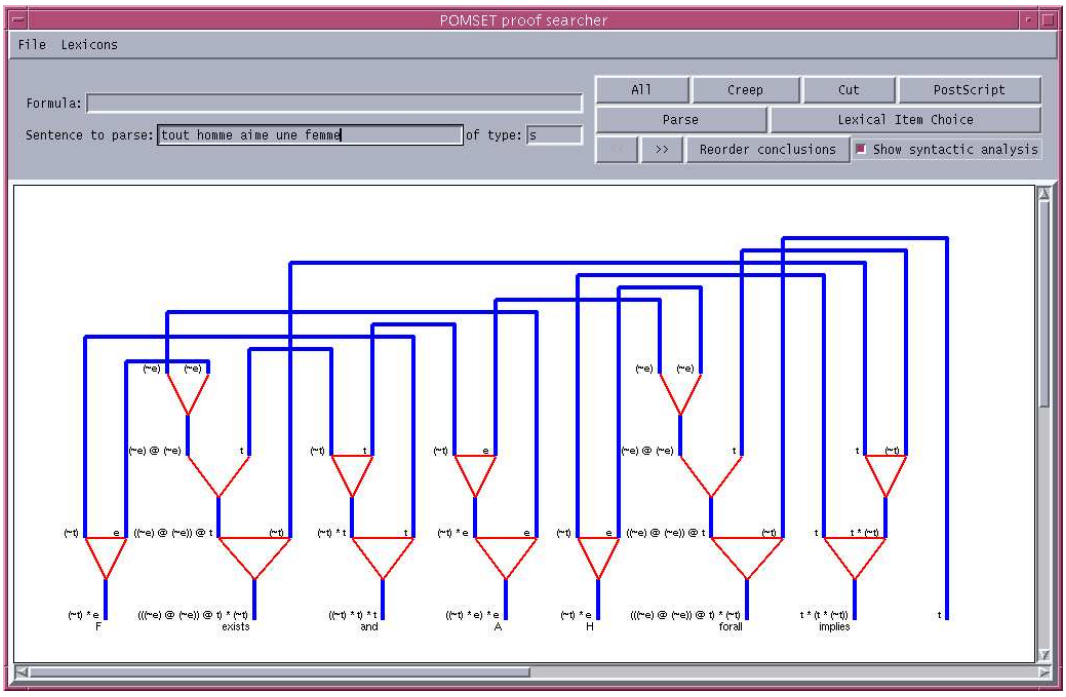


FIG. B.11 – Une première analyse sémantique de *tout homme aime une femme*

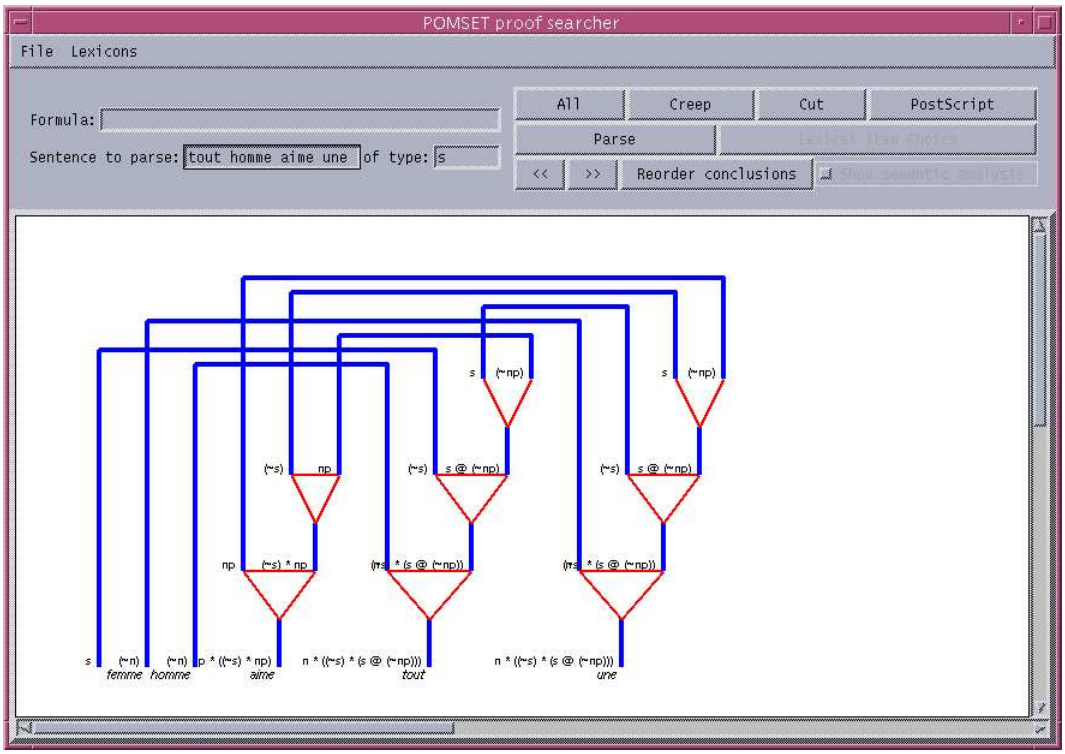


FIG. B.12 – Une expression qui, sans tenir compte de la non-commutativité, exprime le même sens que *tout homme aime une femme*

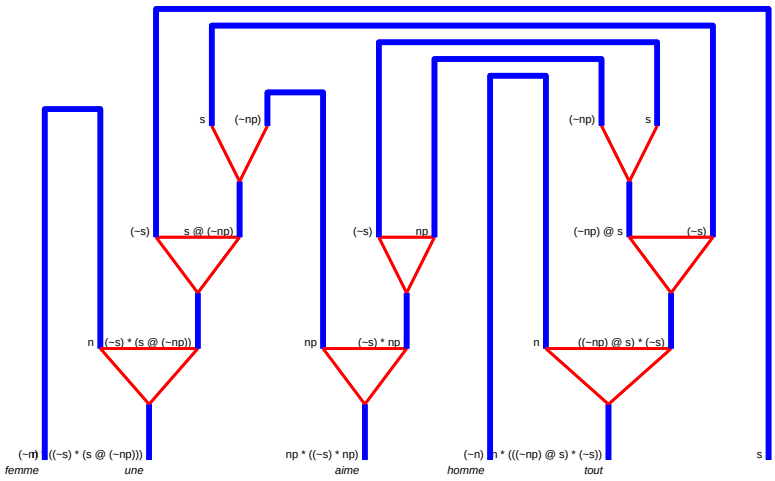


FIG. B.13 – Une deuxième analyse de *tout homme aime une femme*

B.2 Architecture

Cette section décrit d'un point de vue très général l'architecture du programme, qui reste très proche des principes théoriques de fonctionnement des grammaires de types logiques. Le choix du langage s'est porté sur Objective Caml [CMP00], un langage fonctionnel fortement typé qui intègre un puissant système de modules et des objets. Il nous a permis de manipuler aussi bien des structures comme les graphes que de faire les calculs matriciels nécessaires à la génération. Il dispose d'un interpréteur et de deux compilateurs : en *byte-code* (code-octet) ou en natif avec des performances des exécutables similaires aux meilleurs compilateurs disponibles.

La figure B.14 décrit cette architecture. Les boîtes en pointillé schématisent les ressources. Les boîtes arrondies, en gras, schématisent les différents fonctionnalités qui s'articulent entre-elles. Les autres boîtes correspondent aux éléments de base qui permettent par combinaison de remplir ces fonctionnalités. Le tout représente environ 5000 lignes de code, dont 1000 pour l'interface graphique réalisée grâce au module LabelTk d'interface entre OCaml et Tk.

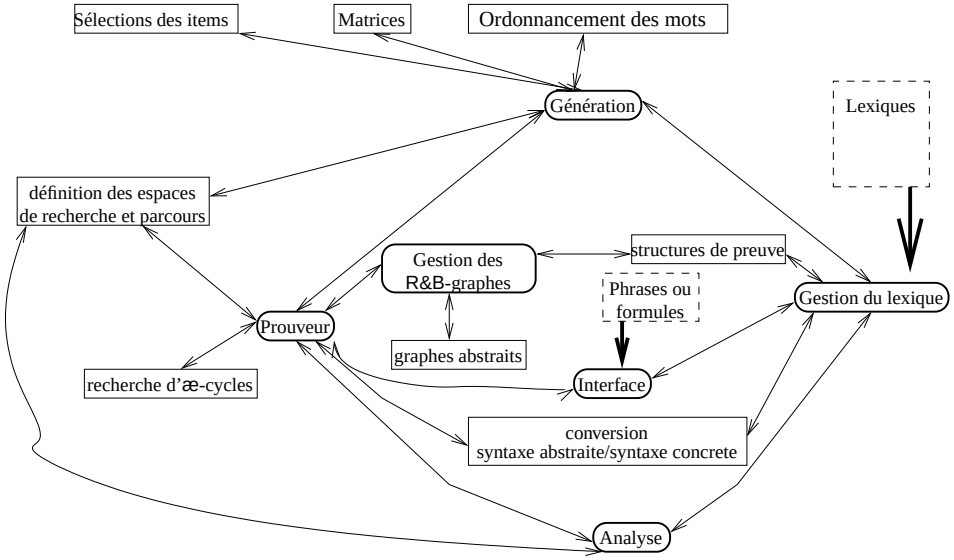


FIG. B.14 – Architecture du prototype

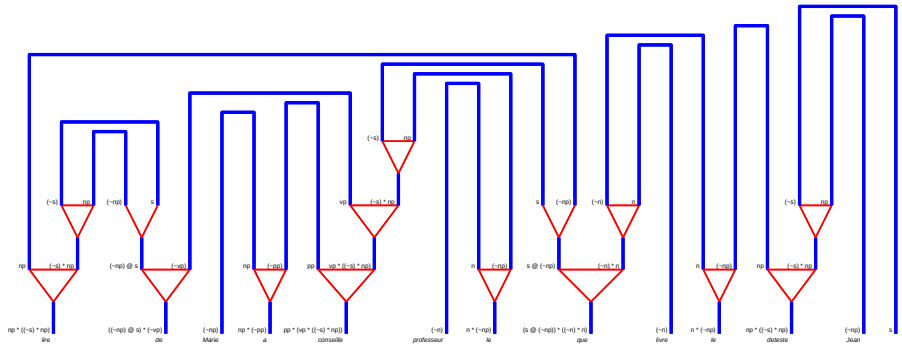
B.3 Plus d'exemples

Cette section permet de voir les résultats de l'approche des grammaires de types logiques par les réseaux de preuve et du prototype sur des exemples un peu plus complexes. On y retrouve des exemples évoqués dans le développement de la thèse ou bien de nouveaux exemples.

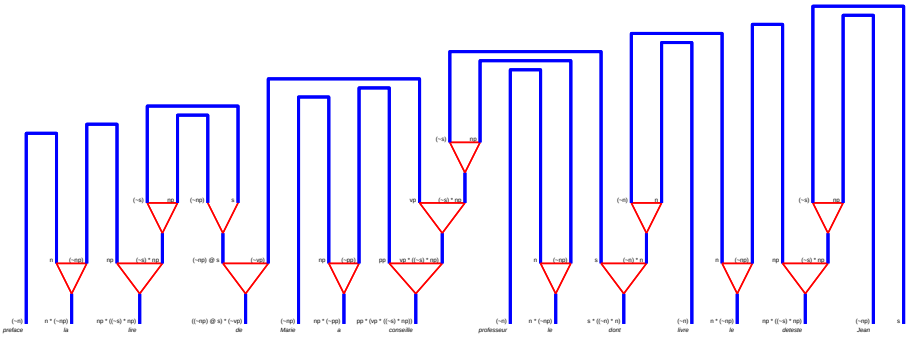
B.3.1 Analyse

Les exemples de la figure B.15 montrent un exemple évoqué à la section 5.3 de la manière dont les grammaires de types logiques rendent compte des dépendances non bornées. L'exemple de la figure B.16 illustre de plus la conjonction de non-consituants.

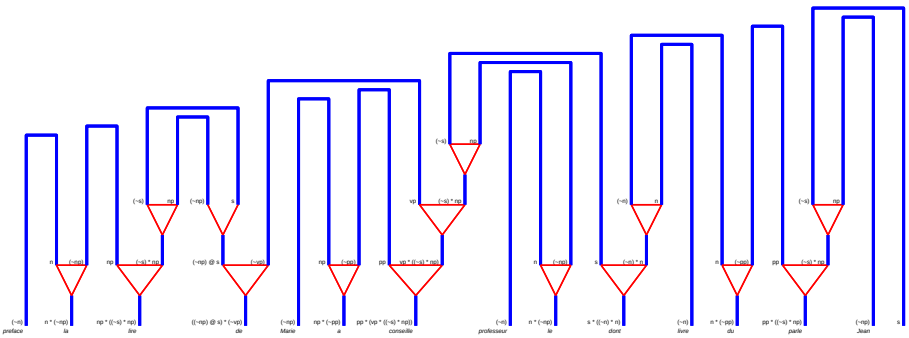
Les exemples suivants, aux figures B.17 et B.18 illustrent quant à elles la notion de complexité d'analyse d'une expression telle qu'elle a été définie par [Mor00] et décrite à la section 5.4.2. Ainsi



(a) Analyse de *Jean déteste le livre que le professeur conseille à Marie de lire*



(b) Analyse de *Jean déteste le livre dont le professeur conseille à Marie de lire la préface*



(c) Analyse de *Jean parle du livre dont le professeur conseille à Marie de lire la préface*

FIG. B.15 – Exemple de dépendances non bornées (section 5.3)

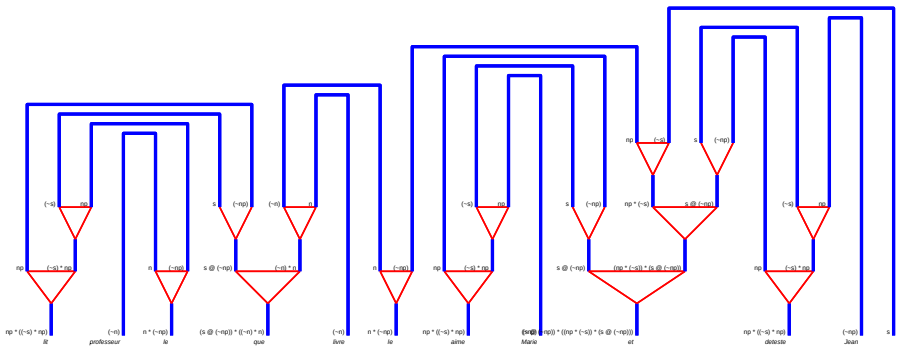


FIG. B.16 – Analyse de *Jean déteste et Marie aime le livre que le professeur lit*, exemple de conjonction (section 5.4.2)

on peut justifier qu'un certain type de constructions grammaticales emboîtées reste compréhensible lorsque le nombre de mots augmente, tandis que pour d'autres constructions, cela devient rapidement intraitable.

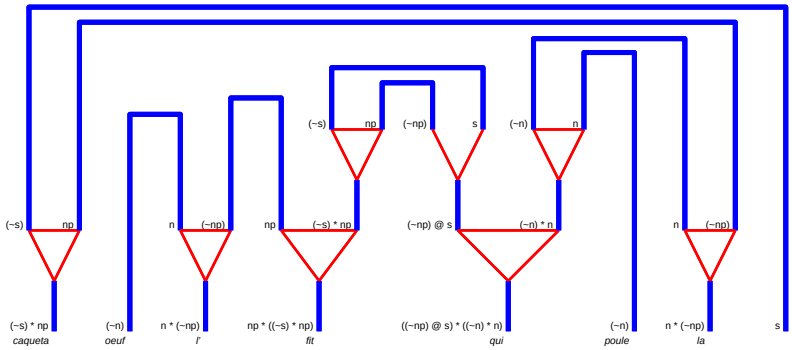
Le dernier exemple montre que l'élévation de type se fait naturellement dans la preuve. Dans la figure B.19, pour pouvoir coordoner *tout charretier* et *Jean*, le type *np* de ce dernier doit être élevé à $S/(np\backslash S)$ (c'est-à-dire $S \parr (S^{\perp} \otimes np)$), le même type que *tout charretier* issu de la combinaison des types $(S/(np\backslash S))/n$ de *tout* et *n* de *charretier*.

B.3.2 Génération

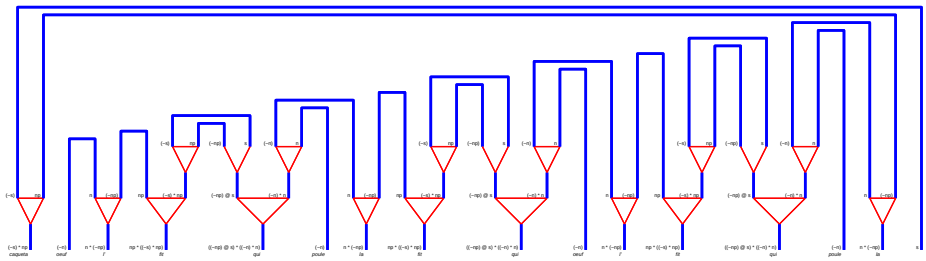
Le premier des exemples reprend celui qui expose la méthode de la section 6.5. Tout d'abord, à partir du lexique anglais (pour ne pas tricher), nous calculons la représentation sémantique de l'expression que nous voulons engendrer (voir figure B.20(a)). Puis, après avoir sélectionné le lexique français, on demande un choix lexical qui offre deux possibilités étant donné le lexique de la figure B.8 : *à* et *dans* (figures B.20(b) et B.20(c)). L'ordonnancement des mots montre que les deux solutions sont acceptables[28] (figures B.20(d) et B.20(e)).

En utilisant le même principe que précédemment, nous pouvons obtenir l'analyse syntaxique et l'analyse sémantique de *Goubi habite Jaligny*, à figure B.21 à partir du lexique en français. Cette analyse sémantique nous permet à son tour, en sélectionnant cette fois le lexique anglais, d'engendrer deux expressions après ordonnancement des items lexicaux choisis : *Goubi lives in Jaligny* et *Goubi inhabits Jaligny* (figure B.22), c'est-à-dire deux paraphrases, et dans une langue différente.

[28] Rappelons que sur les réseaux syntaxiques, l'expression se lit de droite à gauche.

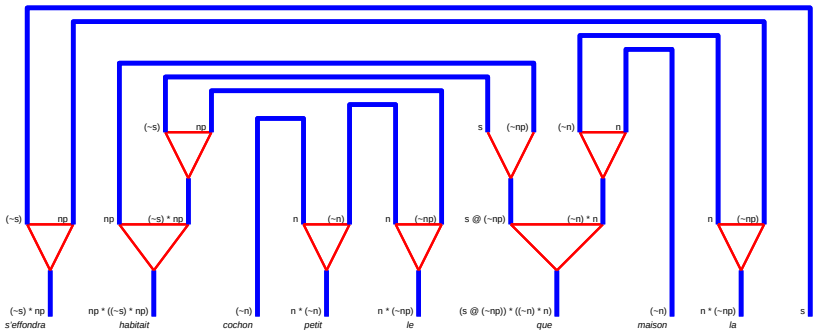


(a) Analyse de *la poule qui fit l'œuf caqueta*

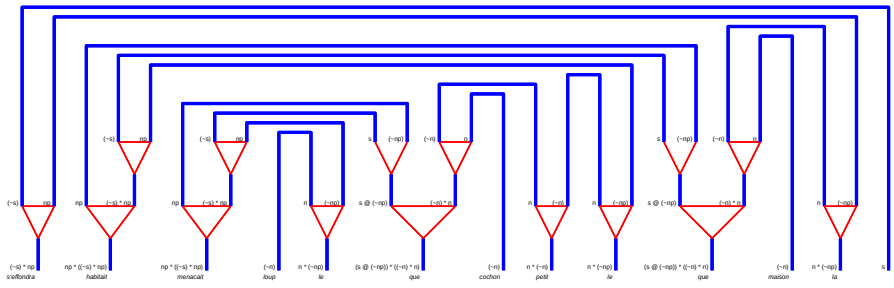


(b) Analyse de *la poule qui fit l'œuf qui fit la poule qui fit l'œuf caqueta*

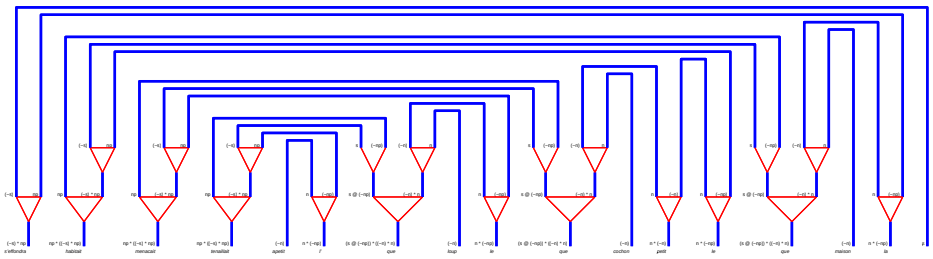
FIG. B.17 – Complexité constante (voir section 5.4.2)



(a) Analyse de *la maison que le petit cochon habitait s'effondra*



(b) Une analyse de*la maison que le petit cochon que le loup menaçait habitait s'effondra*



(c) Une analyse de*la maison que le petit cochon que le loup que la faim tenaillait menaçait habitait s'effondra*

FIG. B.18 – Complexité croissante (voir section 5.4.2)

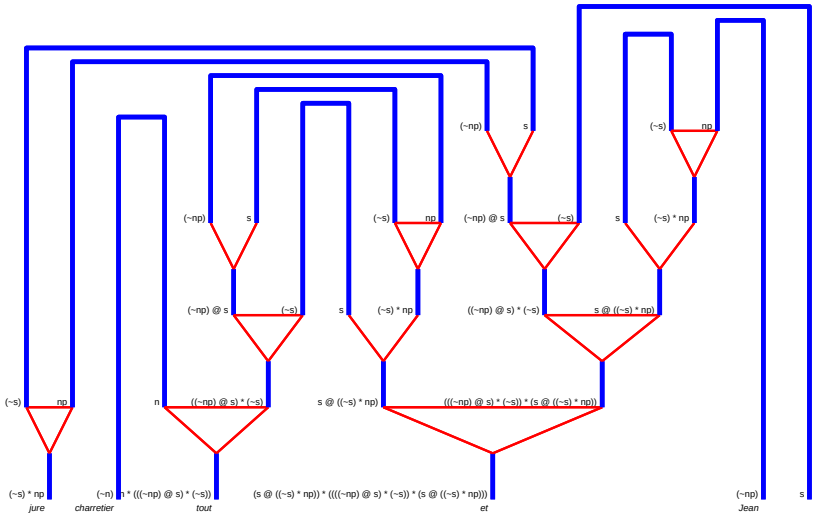
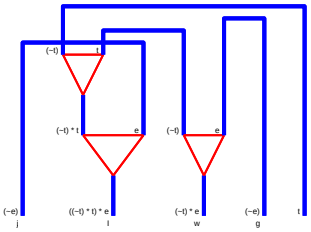
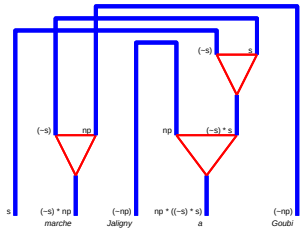


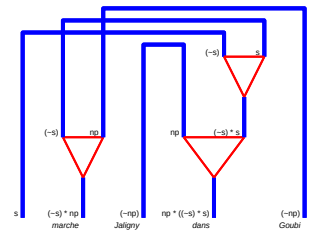
FIG. B.19 – Élévation de type



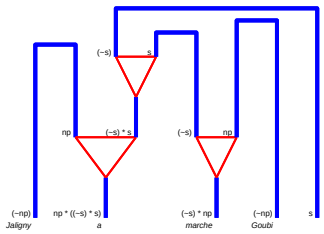
(a) Réseau sémantique cible Π_0



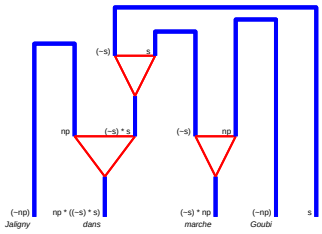
(b) Premier choix d'items lexicaux



(c) Deuxième choix d'items lexicaux

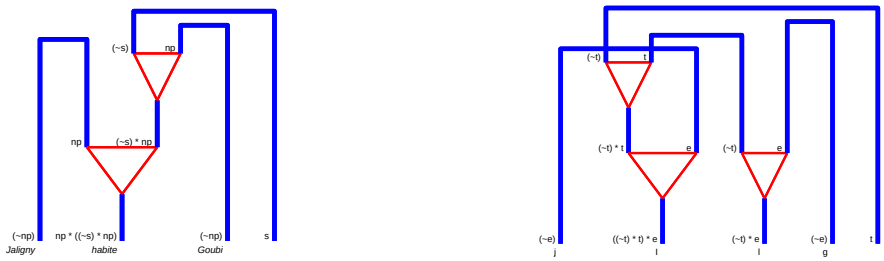


(d) Première expression



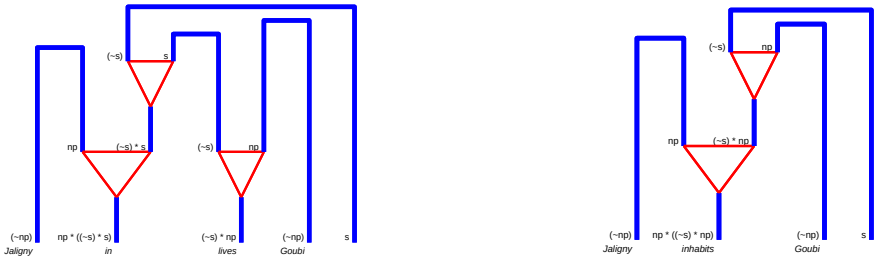
(e) Deuxième expression

FIG. B.20 – Génération de l'exemple de la section 6.5



(a) Analyse syntaxique (b) Analyse sémantique

FIG. B.21 – Analyse de *Goubi habite Jaligny*



(a) Première traduction (b) Deuxième traduction

FIG. B.22 – Les deux expressions équivalentes en anglais

Annexe C

Réseaux de preuves intuitionnistes simplement lexicalisés et TAGs

Introduction

First introduced by [Ret93], pomset linear logic can deal with linguistic aspects by inducing a partial order on words. [LR95] uses this property : it defines modules (or partial proof-nets) which consist in entries for words, describing both the category of the word and its behavior when interacting with other words. Then the natural question of comparing the generative power of such grammars with Tree Adjoining Grammars [JLT75], as [JK96] pointed some links out, arises.

To answer this question, we propose a logical formalization of TAGs in the framework of linear logic proof-nets. We aim to model trees and operations on these trees with a restricted part of proof-nets (included in the intuitionistic ones), and we show how this kind of proof-nets expresses equivalently TAG-trees.

The first section presents all the definitions. Then, in the second section, we propose a fragment of proof-nets allowing the tree encoding and the third section defines the way we model operations on proof-nets. As replying to the second section, the fourth one allows us to come back from proof-nets to trees. Finally, section C.5 shows examples of how the definitions and properties work.

C.1 Definitions

C.1.1 TAG

First, extending the original definition of TAG [JLT75] with the substitution operation as in [SAJ88, AFV96], we get :

Definition 1. A *TAG* is a 5-uple (V_N, V_T, S, I, A) where :

1. V_N is a finite set of non-terminal symbols,
2. V_T is a finite set of terminal symbols,
3. S is a distinguished non-terminal symbol, the *start* symbol,
4. I is a set of *initial* trees,
5. A is a set of *auxiliary* trees.

Initial trees represent basic sentential structures or basic categories. They have non-terminal nodes to be substituted for and serve as arguments to themselves or to auxiliary trees. A leaf (marked with a $*$) with the same label as the root node characterizes the auxiliary trees. An elementary tree is either an initial tree or an auxiliary tree.

The TAGs we are considering here will always be such that every elementary tree has at least a (terminal) node labeled by a terminal symbol, so that the TAGs are *lexicalized.*

Second, for refering trees and nodes in these trees, we use the notations [JLT75] defined for trees on the finite aphabet V ($V = V_N \cup V_T$) :

Definition 2. γ is a *tree over* V iff it is a function from D_γ into V where the domain D_γ is a finite subset of $J^\star$ such that :

1. if $q \in D_\gamma, p < q,$ then $p \in D_\gamma$;
2. if $p \cdot j \in D_\gamma, j \in J,$ then $p \cdot 1, p \cdot 2, \ldots, p \cdot (j-1) \in D_\gamma$

where $J^\star$ is the free monoid generated by J the set of all natural numbers, $\cdot$ is the binary operation, 0 is the identity and for $q \in J^\star, p \leq q$ iff there is a $r \in J^\star$ such that $q = p \cdot r$, and $p < q$ iff $b \leq q$ and $p \neq q$.

We call elements in D_γ adresses of γ. If $(p, X) \in \gamma$, then we say that X is the label of the node at the adress p in γ. We write it $\gamma(p) = X$.

Third, we require another property :

Proposition 64 (ϖ). *A tree γ satisfies the ϖ property iff $\forall p \in D_\gamma$ such that $\gamma(p) \in V_T$ then $p = q \cdot 1$ and $q \cdot 2 \notin D_\gamma$.*

It means that for a tree, if a node is terminal, labeled by a terminal symbol, then it is the unique daughter of its mother-node. Performing the two operations (substitution and adjunction) preserves this property.

But considering TAGs whose elementary trees have the ϖ property does not restrict the generated language. Indeed, if G is a TAG whose elementary trees do not have the ϖ property, $T(G)$ is the set of all the trees that the two operations produce in the TAG G, $L(G)$ is the language that G generates (the set of strings as sequences of terminal symbol-labeled leaves of trees in $T(G)$) and if G_1 is the TAG made from G in order to get the elementary trees have the ϖ property, then we have no special relation between $T(G)$ and $T(G_1)$. Nevertheless, we have $L(G) \subset L(G_1)$.

Fourth, another restriction, similar to the restriction from [AFV96], is to avoid the use of trees γ such that :

$$\exists p \in D_\gamma, \gamma(p) = \gamma(p \cdot 1) \text{ and } p \cdot 2 \notin D_\gamma$$

It means there is no tree that have an X-labeled node whose unique leaf is also an X-labeled node.

C.1.2 Lexicalized Proof-Nets

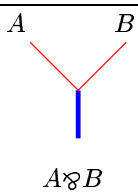
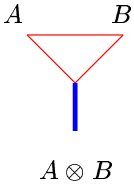
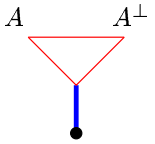
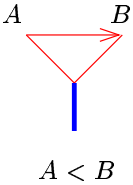
Proof-nets in linear logic have become familiar [Gir87a, Ret93, Abr95]. In this paper, we refer to [Ret96b]'s notations of proof-nets, extended to the ordered calculus [Ret97b]. It defines proof-nets as bicolored (Red and Blue, or Regular and Bold) graphs with the five links corresponding to the axiom, the tensor ($\otimes$), the before ($<$), the par ($\parr$) and the cut (`Cut`). This calculus enjoys cut-elimination [Ret93], a crucial property for our modeling.

Let us remind the main definitions :

Definition 3 (RB-graphs). A RB-graph is a graph with couloured edges (blue and red, or bold and regular). B-edges are undirected. The R-edges may be undirected or directed, in which case we call them R-arcs.

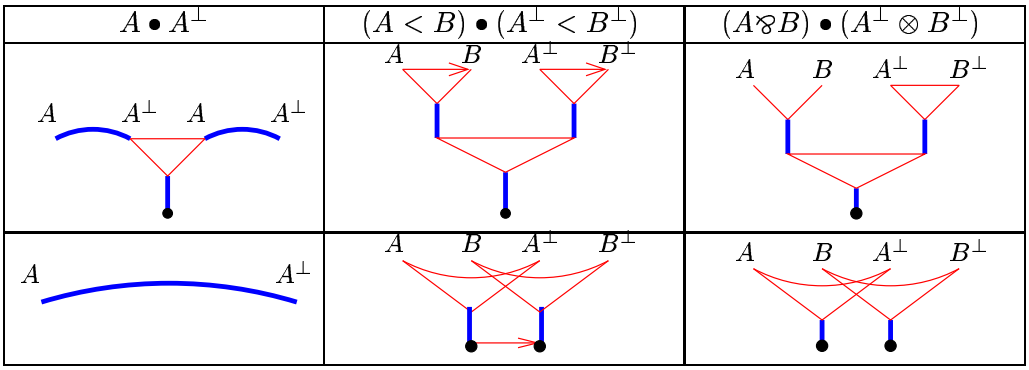
Definition 4 (Links). There are five sorts of links, defined as RB-graphs (see table C.1).

TAB. C.1 – Definitions of the links

Name	axiom	$\parr$	$\otimes$	`Cut`	$<$
Premises	none	A and B	A and B	A and $A^\perp$	A and B
R&B-graph					
Conclusions	A and $A^\perp$	$A \parr B$	$A \otimes B$	none	$A < B$

Definition 5 (Proof-structure). A proof structure is a RB-graph such that any B-edge is the conclusion of exactly one link and the premise of at most one link (the B-edges which are not a premise of

TAB. C.2 – Rewriting rules on proof-nets for cut-elimination



any link are called conclusions of the proof-structure, they contain all the cuts), provided with a set of R-arcs between conclusions which defines a strict partial order.

Definition 6 (Proof-net). An ordered proof-net is a proof-structure which contains no alternate elementary circuit[29].

We speak about correctness criterion, or correctness checking to speak about the absence of any alternate elementary circuit in a proof-structure, so that we know wether a proof-structure is a proof-net or not.

Proposition 65 (Cut-elimination). *Cuts can be eliminated. More precisely : let* Π *be a proof-net whose conclusions are* $F_1, F_2, \ldots, F_k, \bullet_1, \ldots, \bullet_p$ *ordered by* $\mathfrak{R}$ *(where* $F_1, \ldots, F_k$ *are formulas and all the* $\bullet_i$ *are cuts) it is possible to rewrite* Π *as* Π' *with conclusions* $F_1, \ldots, F_k$ *ordered by the restriction of* $\mathfrak{R}$ *to these formulas. Moreover, this rewriting enjoys strong normalisation and confluence [Ret93].*

Table C.2 shows the rewriting rules on proof-nets.

We do not consider all proof-nets, but only those taking their formulas in the $\mathcal{C}$ language defined as follows : $\mathcal{A}$ is an alphabet of atomic formulas (we shall take $\mathcal{A} = V_N$) and

$$\mathcal{B}_1 ::= \mathcal{A}^\perp | \mathcal{A}^\perp \parr \mathcal{B}_1 \qquad \mathcal{B}_2 ::= \mathcal{A} | \mathcal{B}_2 < \mathcal{A}$$

$$\mathcal{C} ::= \mathcal{A} | \mathcal{A}^\perp | \mathcal{A} \otimes \mathcal{A}^\perp | \mathcal{A} \parr \mathcal{A}^\perp | \mathcal{B}_1 \parr (\mathcal{B}_2 \otimes \mathcal{A}^\perp)$$

Moreover, we always set the $\mathfrak{R}$ partial order relation to $\emptyset$.

In addition to logical formulas, we also decorate proof-nets with *labels* from a finite set of terminal symbols. Then, as a restriction of the lexicalized intuitionistic labeled proof-nets defined in [LR96b], we define :

Definition 7.

1. **Output :** An *output* is either a B-edge that is labeled by a positive atom or the conclusion of a par-link between two atoms dual one from another (we call such a conclusion a *par-gate*).
2. **Intuitionistic proof-net :** An *intuitionistic proof-net (IPN)* is a proof-net with one and only one output.
3. **Simply lexicalized proof-net :** A *simply lexicalized (SLIPN)* is a lexicalized IPN the conclusions of which are of the form :

[29] a path of even length, starting and ending on the same vertex, using only once every other vertex and with alternating blue and red edges.

(a) atoms or dual of atoms,

(b) output,

(c) $A_{j_1}^{\perp} \parr \cdots \parr A_{j_n}^{\perp} \parr ((A_1 < \cdots < A_m) \otimes Y^{\perp})$ with $j_i \in [1, m]$ ($j_i \leq j_k$ iff $i \leq k$) and every $A_{j_i}^{\perp}$ is labeled by a string w_{j_i} ($A_i \in V_N$ and $w_j \in V_T$),

(d) $X \otimes X^{\perp}$ with X atomic (we call such a conclusion a *tensor-gate*).

Note that neither the lexicalization nor the intuitionnistic property contribute to the proof-structure correctness checking. The correctness criterion does not change (since it's (almost) the only one to handle the before-link, we would rather keep it). On the other hand, the intuitionistic feature allow the use of (a variant of) intuitionnistic paths [Lam94]. It is stable under the operations we are considering and paths enable the decoding from proof nets to trees (see section C.4.2).

C.2 From Trees to Proof-Nets

This section defines for each elementary tree of a TAG a corresponding SLIPN with an induction on the height of the trees. The set of atoms for logical formulas comes from V_N, and labels come from V_T.

C.2.1 Initial Trees

We first give the general idea of this encoding on the tree T_2 of table C.5. Forgetting the lexicalized part, we can read this tree as a terminal node (N) preceding another terminal node (V) to produce their mother-node (P). We can express this idea with a formula of pomset logic : $(N < V) \multimap P$. But do not forget that this is a brick from which we want to derive S. Moreover, proof-nets correspond to one-sided sequent, so that, actually, we are more intersted in the dual of such formula, namely : $(N < V) \otimes P^{\perp}$. Thus, we shall have a SLIPN with this latter sub-formula, other connectives dealing with the lexicalization.

Table C.3 sums up the translation. Note that for $h = 1$, the two latter cases do not belong to the considered TAG. Nevertheless, we require the definition of their corresponding SLIPNs for the next steps of the induction. The case $h = 2$ only presents the case where lexicalized subtrees' height (at least one exists) is 1 and other subtrees' height is 0, for we deal with the cases where other subtrees' height can be 1 in the next general case. For this latter, we possibly have $\{i_1, \ldots, i_p\} = \emptyset$ and in figure C.1(b) the $\Pi_{j_1}, \ldots, \Pi_{j_m}$ are the inductivily built SLIPNs corresponding to the subtrees of γ at $X_{j_1}, \ldots, X_{j_m}$.

Definition 8. For every initial tree T, we call *corresponding initial SLIPN*, or T *transformation*, the SLIPN defined as in table C.3 and figure C.1.

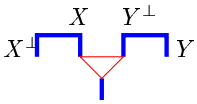
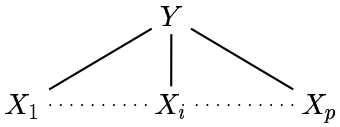
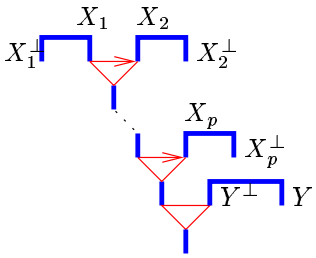
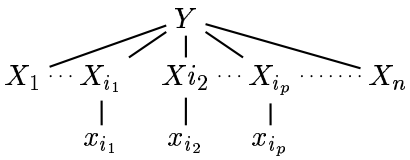
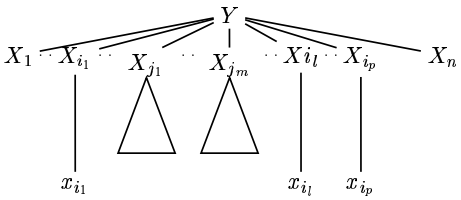
Remarque 6.

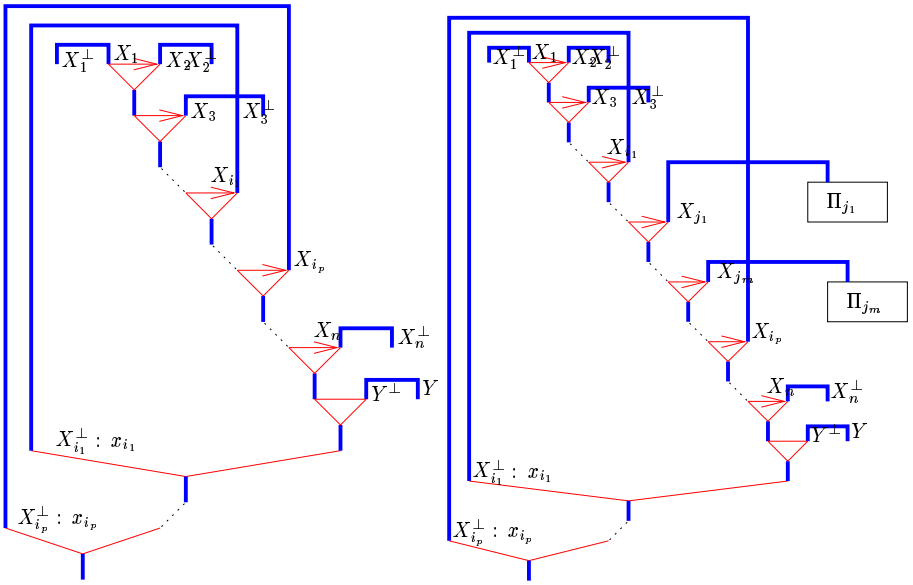
1. The unique output of every SLIPN corresponds to the root node of the tree, and every terminal node, labeled by a non-terminal symbol, of a tree corresponds to a conclusion of the corresponding SLIPN.
2. There is a one-to-one maping between the non-terminal symbol labeled nodes of a tree and the axiom links of the corresponding SLIPN.

C.2.2 Auxiliary Trees

Let γ be an auxiliary tree and let us define γ^+ as the same tree as γ except for its X^* node replaced with an X node. We call r the adress of N^* in γ so that $\gamma(r) = X^*$ and $\gamma^+(r) = X$. Then, following the definition in the previous section, we can define Π^+ the SLIPN corresponding to γ^+. And, as

TAB. C.3 – Initial trees maping

	Trees	Proof-nets
$h = 1$		
		
		
$h = 2$		(see figure C.1(a))
general case		(see figure C.1(b))



(a) SLIPNs corresponding to trees of height 2

(b) SLIPNs corresponding to trees in the general cases

FIG. C.1 – SLIPNs for higher trees

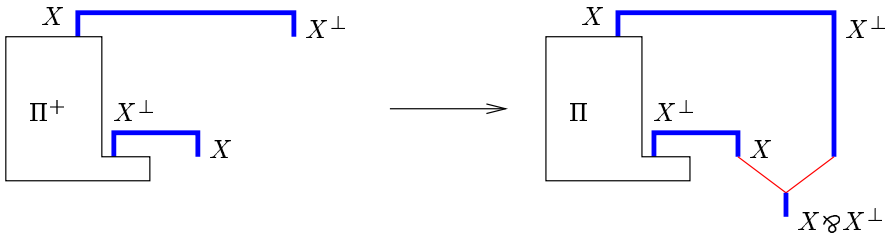


FIG. C.2 – Auxiliary trees

γ is an auxiliary tree, $\gamma^+(r) = \gamma^+(0)$ and Π^+ has a conclusion X (corresponding to $\gamma^+(0)$) and a conclusion $X^\perp$ (corresponding to $\gamma^+(r)$).

Thus we define Π the corresponding SLPIN to γ as the proof-net built from Π^+ in binding with a $\parr$-link its X and its $X^\perp$ conclusions (see figure C.2). Π is a (correct) SLIPN.

Definition 9. For every auxiliary tree γ, we call *auxiliary corresponding SLIPN*, or γ *transformation*, the SLIPN defined as above. Then, for every elementary tree, we call *corresponding SLIPN* the initial or auxiliary SLIPN corresponding to that tree.

C.3 Elementary Operations

This section deals with a particular case of the next section but focuses on the core operations which we can refer to during the generalisation.

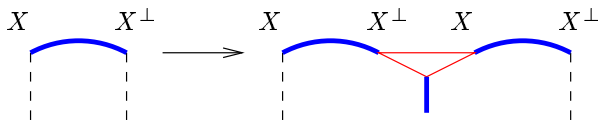


FIG. C.3 – Adding a tensor-gate

C.3.1 The Substitution Operation

Let γ_1 and γ_2 be two trees such that we can substitute γ_2 to a terminal node X (whose adress is r) of γ_1, and let Π_1 and Π_2 be their corresponding SLIPNs. Then $\gamma_1(r) = \gamma_2(0)$, and (cf. remark 6) Π_1 has a conclusion $X^\perp$, and the output of Π_2 is X.

Thus we can bind these conclusions with a cut-link and yield a new SLIPN from which we eliminate the cut and obtain a new SLIPN Π.

Definition 10. For every tree γ resulting from the substitution of γ_2 to a node of γ_1, we define its *corresponding SLIPN* Π as above.

C.3.2 The Adjunction Operation

Preparing the Target Tree.

In order to allow the adjunction of an auxiliary tree on a target tree, we need to modify a little bit this latter.

Let γ be the tree on which we want to perform an adjunction, r the adress of the node where to perform the adjunction, and Π the corresponding SLIPN. As noted in remark 6, Π contains an axiom link $\gamma(r)$ ⌐ $\gamma(r)^\perp$ corresponding to the node $\gamma(r)$. So we can split this link into two axiom-links linked with a tensor-link and we obtain (with $X = \gamma(r)$) a new SLIPN Π' as shown in figure C.3.

Proposition 66. *Adding a tensor-gate preserves the correctness of the proof-net.*

Actually, such a conclusion is an instance of cuts (as a cut is equivalent to a conclusion $(\exists X)(X \otimes X)$). Then if this tensor-gate remains unused, cut-elimination amounts to delete this tensor-gate and come back to a simple axiom-link. The second example of section C.5 uses this feature at the very end of the derivation.

Performing the Operation.

In this section, we define the SLIPN corresponding to the result of adjoining the tree γ_2 to γ_1. In this preliminary case, both γ_1 and γ_2 are elementary trees (γ_2 is an auxiliary tree and γ_1 is the target tree). And Π_1 and Π_2 correspond to them.

We assume Π_1 already has a tensor-gare added on the axiom corresponding to the node where we want to adjoin γ_2. So that if X labels the stared-node of γ_2, then X also labels the node of γ_1 receiving the adjunction and we have a tensor-gate conclusion $X \otimes X^\perp$ for Π_1 and a par-gate $X \parr X^\perp$ (the output) for Π_2 (merely by construction).

Thus, we can bind Π_1 and Π_2 with a cut-link and eliminate this cut to obtain a new SLIPN (because there is no modification on the lexicalized parts, still no alternate elementary circuit and still only one output : γ_1's one).

Definition 11. For every tree γ resulting from the adjunction of the auxiliary tree γ_2 on γ_1, we call the *corresponding SLIPN* the SLIPN built as above.

C.3.3 Operations on Derived Trees

Up to now, we defined a way of modeling elementary trees in the framework of SLIPNs, and a way of combining these SLIPNs to model the adjunction and substitution operations on elementary trees. We now are about to extend this modeling on derived trees, so that a SLIPN will correspond to every tree (elementary or derived) of a TAG.

Definition 12. For a tree γ and an elementary tree E_0, we call *derivation* of γ the pair

$$< E_0, ((\diamond_1, E_1, \gamma_1), \dots, (\diamond_n, E_n, \gamma_n)) >$$

such that $\gamma_n = \gamma$ and for every $i \in [1, n]$, γ_i results from the operation $\diamond_i$ (adjunction or substitution) between γ_{i-1} and the elementary tree E_i.

n is *the length* of the derivation.

Remarque 7. For a derived tree, the derivation is not necessarily unique.

Definition 13. Let γ be a derived tree from the derivation d. Then we can define the *d-SLIPN corresponding* to γ, built only with cut and cut-elimination (between unlabeled conclusions) from the SLIPNs corresponding to the elementary trees of the derivation.

Actually, proving the existence of this SLIPN interests us more than the simple definition, as it also gives its construction's steps.

Démonstration. We prove the existence of $\Pi^{(d)}$ by induction. We also prove the property that if γ has a terminal node (except for the stared node of an auxiliary tree), labeled by a non terminal symbol X, then $\Pi^{(d)}$ has a pendant conclusion $X^\perp$ (corresponding to this node, hence not labeled neither).

1. if $l = 0$: γ is an elementary tree, and we already defined its transformation $\Pi^{(0)}$. And its construction also proves the property of the pendant conclusion.
2. if $l > 0$: Let $d_{l-1} =< \gamma_0, ((\diamond_1, E_1, \gamma_1), \dots, (\diamond l-1, E_{l-1}, \gamma_{l-1})) >$ and $\Pi^{(d_l-1)}$ be the SLIPN corresponding to γ_{l-1} in the d-derivation.
   (a) If $\diamond_l$ is the substitution of a leaf X of γ_{l-1} by E_l (whose transformation is π_l) then $\Pi^{(d_{l-1})}$ has an axiom-link $X \ulcorner\urcorner X^\perp$ in which $X^\perp$ is a pendant conclusion (induction hypothesis), and π_l has an axiom-link $X \ulcorner\urcorner X^\perp$ in which X is a pendant conclusion ($E_l(0) = X$). Then we can link these two pendant conclusions with a cut-link, and eliminate it. This yields a new SLIPN. It also proves the property of pendant conclusion, as every terminal node of the new tree, labeled with a non-terminal symbol, already was terminal in one or another of the two trees so that (by induction hypothesis) they already had the property.
   (b) if $\diamond_l$ is the adjunction on the leave X of γ_{l-1} of the auxiliary tree E_l (whose transformation is π_l) then γ_{l-1} has an axiom-link $F \ulcorner\urcorner F^\perp$ we can replace (with respect to the SLIPNs class belonging) with two axiom-links linked together with a tensor-link (i.e. we add a tensor gate $X \otimes X^\perp$ as for adjunctions on elementary trees). π_l has a par-gate $X \parr X^\perp$ so that we can bind the two gates with a cut-link, and then eliminate this latter. We obtain a new SLIPN, and as above, the induction proves the property of the pendant conclusion.

□ □

Remarque 8. This shows that cuts are only between atomic formulas or tensor and par-gate. Actually, the grammar given for the conclusions of SLIPNs indicates that no other cut can occur.

During this section, we made the assumption of allowing adjunctions at every time on every node. Of course, sometimes we do not want such possibilities. Allowing the tensor-gate addition only in the lexicon, and not during the derivation, brings a solution to this option.

Section C.5 shows examples for both cases. In particular, the mildly-context sensitivity of TAGs, generating $\{a^n b^n c^n d^n\}$, illustrates the second case.

TAB. C.4 – Polarities of the conclusions

$\otimes$	$\circ$	$\bullet$
$\circ$	$\circ$	$\bullet$
$\bullet$	$\bullet$	–

$\parr$	$\circ$	$\bullet$
$\circ$	–	$\circ$
$\bullet$	$\circ$	$\bullet$

$<$	$\circ$	$\bullet$
$\circ$	$\circ$	–
$\bullet$	–	–

C.4 From SLIPNs to Trees

So far, we explained how, given a TAG and a derived tree in this TAG, we could obtain a SLIPN that we qualify as corresponding. But we now have to see how this SLIPN actually corresponds so that we shall henceforth be able to handle only proof-nets and translate the results on trees.

C.4.1 Polarities

Let us define a positive polarity ($^\circ$) and a negative one ($^\bullet$). Every formula is inductiveley polarized as follows : if α is an atom then α° and $\alpha^{\perp\bullet}$. Then, for each link we define the polarity of the conclusion from premises' ones as in table C.4.

We call *input* the negative conclusions, and *output* the positive one (which is coherent with the previous definition of the output).

The grammar on SLIPNs' conclusions shows that SLIPNs are polarized.

C.4.2 Reading of SLIPNs

We give an algorithm for the reading of any cut-free SLIPN, based on formulas' polarities. It uses a very simple principle : following from a starting point (namely the output) the positive polarities, we define a path across the proof-net. And every time the path crosses an axiom-link, we add a node to the tree under construction. Actually, we build both the function γ and D_γ. Of course, the path can not cross twice the same axiom-link (such a possibility would occur only with par-gate).

As we shall see later, the path never cross a par-link (except par-gates, from the positive conclusion), always enter a tensor-link through a negative premise and always enter a before-link throuh the positive conclusion. So, because of the before-link, the path is not linear (both premises, positive ones, are likely to be the next on the path), and we define the *first branch* as the path from the positive premise of the before-link at the beginning of the arrow, and the *second branch* as the path from the other premise.

Then, when adding a new node on the tree, its mother-node is the the last node met on the same branch of the path (in a before-link, both premises are on the same branch as the conclusion, but they are themselves on different branchs ; other connectives do not create branchs). So that if its mother-node's adress is p, then the new node's adress is $p \cdot j$ with $j \in \mathbb{N}^*$ and for all i such that $0 < i < j$, $p \cdot i \in D_\gamma$

Then, we state the algorithm as follows :

1. Enter the net through the only output (so, if X is the output, $0 \in D_\gamma$, and $(0, X) \in \gamma$).
2. Follow the path defined by the positive polarities until reaching an atom (when a before-link is crossed, first choose the premise at the beginning of the red arc) and cross it. If its conclusions are X and $X^\perp$, we define its adress p as precised above (wrt the branchs) and $p \in D_\gamma$ and $(p, X) \in \gamma$.
3. (a) if the input is lexicalized, then lexicalize the last written node of the tree under construction (if the lexicalization is x, we then add $p \cdot 1 \in D_\gamma$ and $(p \cdot 1, x) \in \gamma$). Either there is no

more link after, or this input is premise of a par-link whose other polarities are negative. In both cases, come back to the last before-link the path did not go through the two branchs and make as in 2.

(b) else juste follow as in 2

4. Stop when the path joined all the atoms.

The next section will show that every SLIPN built from elementary SLIPNs can be read such a way and that the path cross every axiom-link.

Remarque 9.
1. This reading provides a unique correspondance between any axiom link of the SLIPN and a node (labeled by a non terminal symbol). Moreover, if the negative conclusion of an axiom-link is pendant and not lexicalized, then it corresponds to a terminal node of the tree.
2. Two different SLIPNs can have the same reading. It underlines the importance of making precise a base of elementary SLIPNs (corresponding to the elementary trees of a given TAG).

C.4.3 From SLPINs, Back to Trees

We now have both a maping from trees to SLIPNs, and a maping from SLIPNs to trees. It remains us to see if the composition of these mapings gives the identity.

This consists in three steps :

1. check that the reading of a SLIPN corresponding to an elementary tree is the same as the elementary tree ;
2. check that the reading of a SLIPN corresponding to the substitution between two trees is the resulting tree ;
3. check that the reading of a SLIPN corresponding to the adjunction between two trees is the resulting tree.

Moreover, given a TAG, the basic bricks we consider are SLIPNs corresponding to elementary trees of this TAG. Then the only way to build new SLIPNs is binding them with cut between unlabeled conclusions.

Let us remind the definition of subtrees and supertrees as in [JLT75]

Definition 14. Let γ be a tree and $p \in D_\gamma$. Then

$$\gamma/p = \{(q, X) | (p \cdot q, X) \in \gamma, q \in J^\star\}$$

$$\gamma\backslash p = \{(q, X) | (q, X) \in \gamma, p \not< q\}$$

γ/p is called the *subtree of* γ at p and $\gamma\backslash p$ is called the *supertree of* γ at p. Further, for $p \in J^\star$

$$p \cdot \gamma = \{(p \cdot q, X) | (q, X) \in \gamma\}$$

Proposition 67. *$\gamma = \gamma\backslash p \cup p \cdot (\gamma/p)$ for every tree γ and $p \in D_\gamma$.*

Remarque 10. If the reading of a SLIPN Π gives γ, and if we make the path begin at any positive conclusion of an axiom link that corresponds to the node at adress p in γ, then the reading algorithm returns γ/p. And of course, the reading of Π, with a pruning at the same axiom link returns $\gamma\backslash p$.

Elementary Reading.

Let γ be an elementary tree. To prove the reading of the SLIPN corresponding to γ being γ itself, we simply proceed by induction, with the same steps as for the building of elementary SLIPNs.

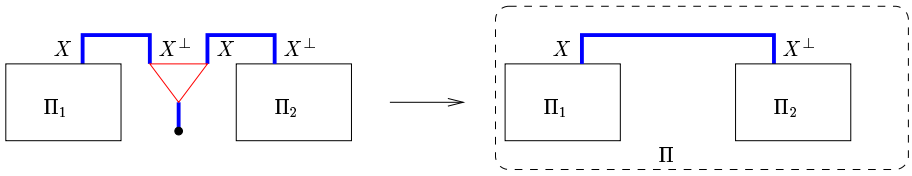


FIG. C.4 – Substitution

Substitution.

Let us consider two SLIPNs Π_1 and Π_2, whose readings are respectively γ_1 and γ_2. We assume Π_1 was built (with cuts) from elementary SLIPNs and Π_2 is an elementary SLIPN itself.

Proposition 68. *If a negative (not labeled) atomic conclusion of Π_1 and a positive atomic conclusion of Π_2 support a cut-linking, then the corresponding terminal node of γ_1 accepts a substitution by the root of γ_2. And the reading of the new SLIPN, after cut-elimination, corresponds exactly to the resulting tree.*

Démonstration. Let p be the adress of X in the reading γ_1 of Π_1, where X corresponds to the axiom link of figure C.4 (on the left). The adress of X in the reading γ_2 of Π_2 is 0 (the root node) and the reading of γ_2 starts at this X axiom-link. On the other hand, the reading of γ_1 stops at X for its branch. After the cut and the cut-elimination, the reading of the new SLIPN (on the right of figure C.4) starts at the output, which also was the output of Π_1, and continues like for γ_1 until the new X axiom-link is reached. Its adress in the new tree γ is also p. There, the reading of γ_2 takes place. So that, as defined in the algorithm, if γ is the reading of the new SLIPN,

$$\forall q \in D_{\gamma_2}, \gamma(p \cdot q) = \gamma_2(q)$$

and nothing changes for the remaining reading : it is the same as for γ_1 (because $\gamma_1 = \gamma \backslash p$). Then the reading γ of Π is such that

$$\gamma = \gamma_1 \cup p \cdot \gamma_2$$

which corresponds to the definition of the substitution of the $\gamma_1(p)$ node with γ_2. □ □

Adjunction.

As above, let us consider two SLIPNs Π_1 and Π_2, whose readings are respectively γ_1 and γ_2. We assume Π_1 was built (with cuts) from elementary SLIPNs and Π_2 is an elementary SLIPN itself.

Proposition 69. *If a tensor-gate of Π_1 and the unique positive conclusion of Π_2 support a cut-linking, then the node corresponding to the tensor-gate on γ_1 accepts an adjunction of γ_2. And the reading of the new SLIPN, after cut-elimination, corresponds exactly to the resulting tree.*

Démonstration. In the following, when dealing with γ_1, we speak about the reading of Π_1 *without* the tensor-gate.

Let us consider Π on figure C.5 (the SLIPN on the right). The input of Π is the same as Π_1. As the positive conclusion of an axiom-link always occurs before the negative conclusion in the path, then new tree γ from Π, after reaching X in Π_1 cross the axiom-link to Π_2, so that $\gamma/p \neq \gamma_1/p$ but $\gamma \backslash p = \gamma_1 \backslash p$ (see remark 10). Yet, The $X^\perp$ on Π_2 is the other conclusion of the starting axiom-link for γ_2. So that at the p adress, for γ, we read γ_2. Then for every $q \in D_{\gamma_2}$, $\gamma(p \cdot q) = \gamma_2(q)$.

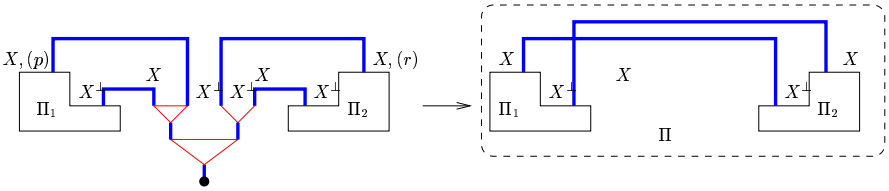


FIG. C.5 – Adjunction

Moreover, reaching the X of Π_2, the path does not stop but continue with the remaining part of γ_1, namely γ_1/p. So that at the new adress of X of Π_2 in γ we add γ_1/p. And the new adress of X in γ is $p \cdot r$ (with r the adress of X^* in γ_2).

Then

$$\gamma = \gamma_1 \backslash p \cup p \cdot \gamma_2 \cup p \cdot r \cdot \gamma_1/p$$

which is the definition of the adjunction of γ_2 on γ_1 at X. □ □

Eventually, we can state the next propositions :

Proposition 70. *Every derived tree (from an elementary tree lexicon) corresponds to the reading of a SLIPN, the latter resulting from* `Cut` *operations between SLIPNs corresponding to the elementary trees of the lexicon.*

Reciprocally, with a lexicon of elementary SLIPNs corresponding to trees, with the restriction of `Cut` operations on formulas that are not lexicalized, the reading of the resulting SLIPNs are the derived trees.

Démonstration. This is immediate after the previous propositions. □ □

C.5 Examples

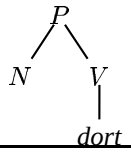
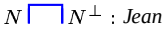
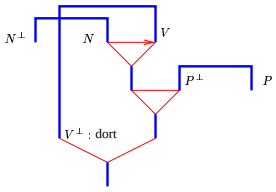
C.5.1 Substitution and Adjunction

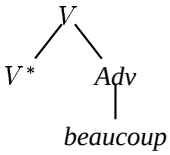
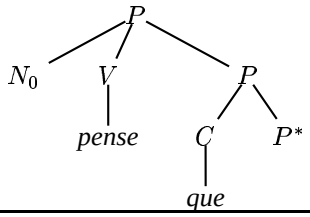
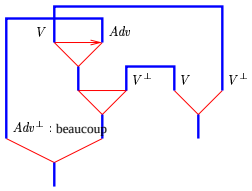
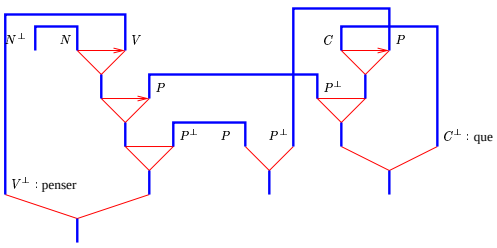
First let us define from the lexicon of the TAG the corresponding elementary SLIPNs. We assume the lexicon of table C.5. This lexicon can yields the trees of figure C.6 (for the first tree : substituting N in T_2 with T_1, then adjoining T_3 on the result. For the second tree : continue with the adjunction of T_4). But we can also make this derivation on the SLIPNs as shown in the figures C.7 and C.8.

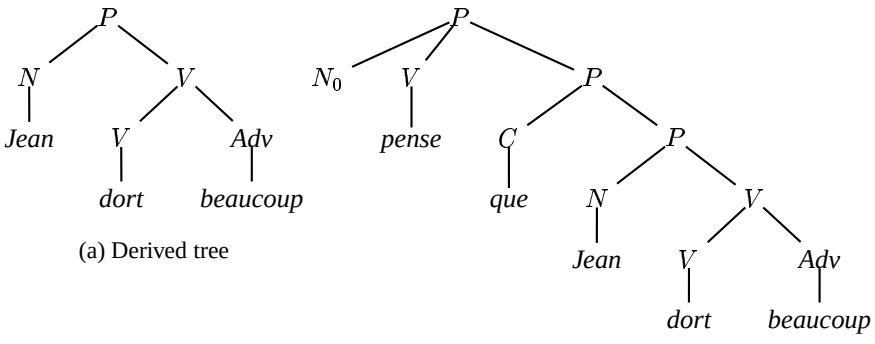
Let us see how to read the SLIPN of figure C.7(e), and obtain the derived tree of figure C.6(a) : first the path enters the net through the atom P (the unique output) and marks P as a node. Then it follows the positive polarities and reaches a before-link with two branches. It first chooses the positive formula at the beginning of the red (regular) arc and crosses an axiom-link. There is a negative lexicalized atom. So on the tree we add a lexicalized (*Jean*) node N. Then doing the same with the other premise of the before-link we get a new atom V and the negative conclusion of the axiom-link is not lexicalized. So we have a junction which will produce new branches under the V node. They are two simple branches : one is a (lexicalized by *dort*) V, and the other is a (lexicalized by *beaucoup*) Adv.

To have a deeper adjunction, let us continue with the adjunction of T_4. Figure C.8 shows the different steps of the operation. But we leave the reader check that polarizing the resulting SLIPN of figure C.8(c) and reading it leads to the tree of figure C.6(b).

TAB. C.5 – Lexicon

	Jean(T_1)	Dort(T_2)
Trees		
SLIPNs		

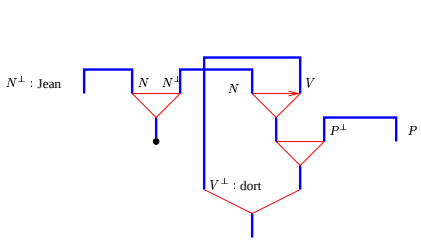
	Beaucoup(T_3)	Pense que(T_4)
Trees		
SLIPNs		



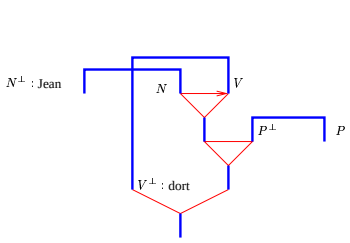
(a) Derived tree

(b) Derived tree

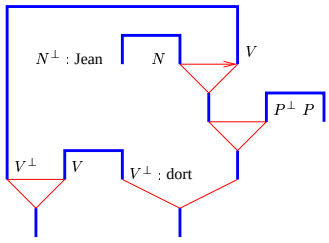
FIG. C.6 – Resulting trees



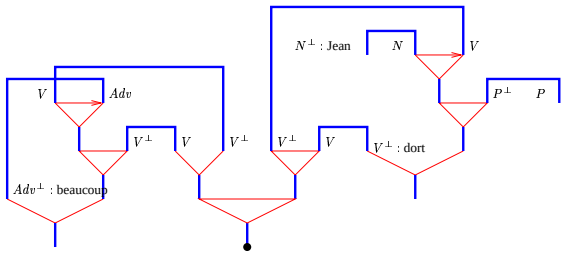
(a) Substitution of the N node of *dort* by *Jean*



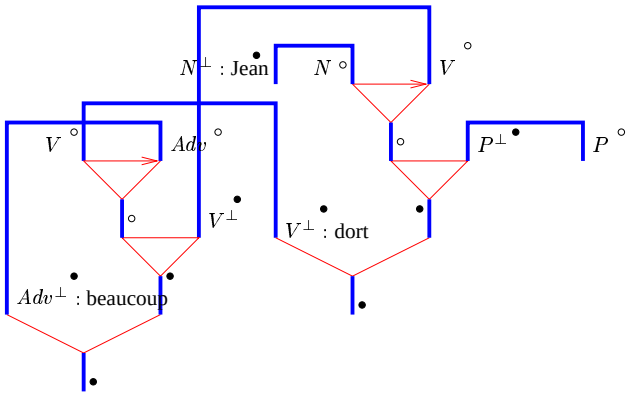
(b) Substitution (continued) : cut-elimination



(c) Addition of a tensor-gate on *Jean dort*

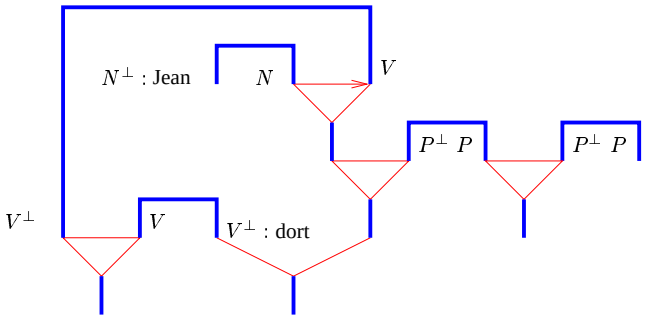


(d) Adjunction of *beaucoup* on *Jean dort*

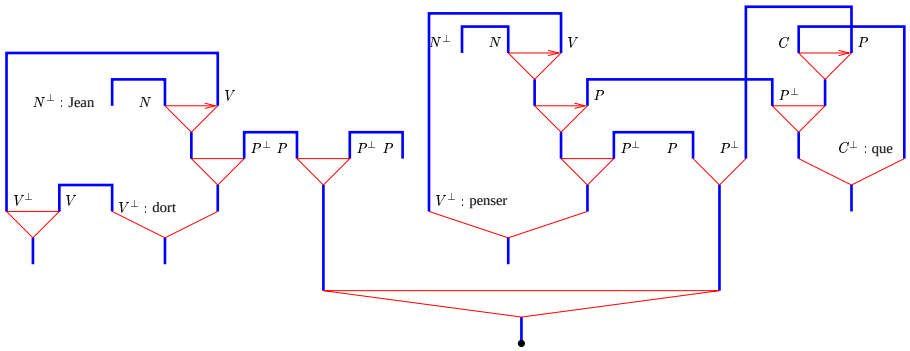


(e) Adjunction (continued) : cut-elimination and polarization

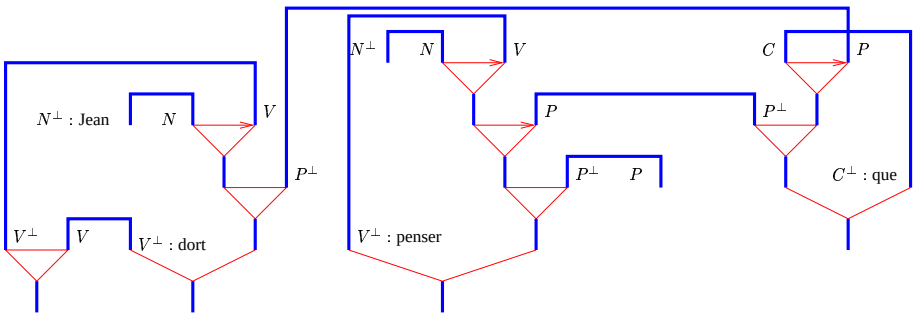
FIG. C.7 – Operating on SLIPNs



(a) Addition of a tensor-gate



(b) Adjunction of *pense que* on *Jean dort beaucoup*



(c) Adjunction (continued) : cut-elimination

FIG. C.8 – Operating on SLIPNs (continued)

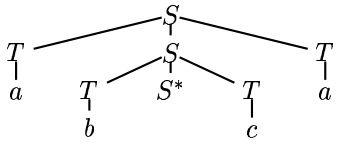
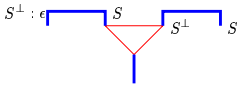
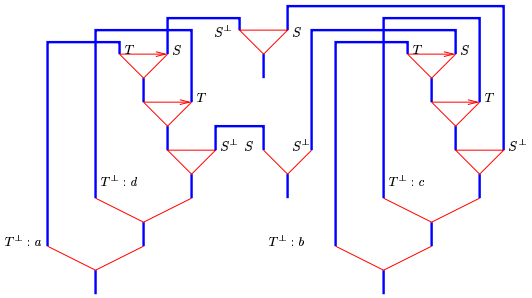
C.5.2 A Formal Language

As in the previous section, we first define the lexicon of table C.6. Note that in this lexicon, the tensor gate belongs to the lexical item, so that we shall never use a tensor-gate addition during a derivation.

We only initiate the use of this lexicon with an adjunction of T_2 on another instance of T_2 (figure C.9(a)), then an adjunction on T_1, resulting in the SLPIN of figure C.9(c). At every adjunction on T_2, a new tensor-gate appear (provided by T_2), as the former (on the derived SLIPN) disappears with the adjunction operation (the cut between the par-gate and the tensor gate of T_2). As the reader can check, the reading actually corresponds to our expectations and generates the word $aabbccdd$.

Thus, without splitting any axiom-link and adding any tensor-gate during the derivation, we can generate the language $\{a^n b^n c^n d^n\}$.

TAB. C.6 – Lexicon

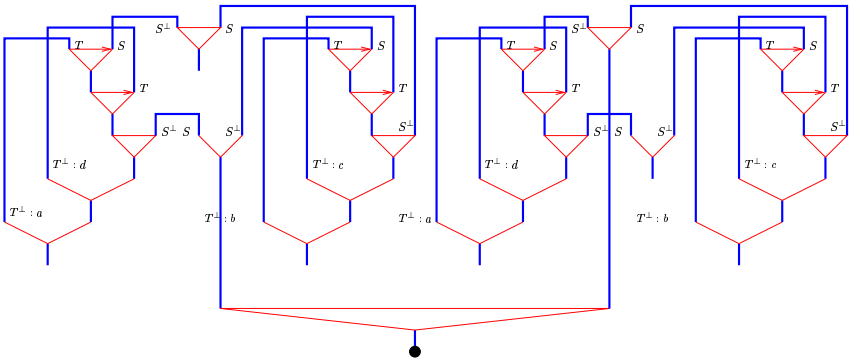
	Empty (T_1)	Main tree (T_2)
Trees		
SLIPNs		

Conclusion

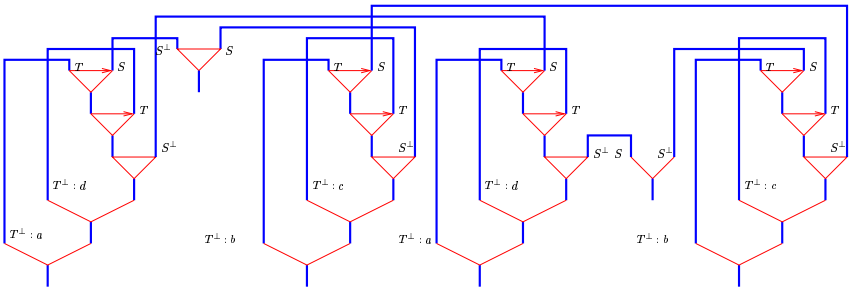
Using a restricted fragment of pomset intuitionistic proof-nets, we showed how to generate the same language as a TAG. This indicates how a more generic formalization (namely [LR97]'s one) allows both keeping generative power and dealing with some linguistic phenomena not by lexical rewriting rules on trees, but by lexical definitions. For instance, we can compare the modeling of clitics in [Abe93] or in [LR97].

We also want to underline that we do not really use the partial order capabilities of pomset proof-nets : the before-links arrange totally the atoms in order. Of course, this results straightforwardly from the fact that the order in the trees is total, so that the same occurs in the SLIPNs with respect to the before-links.

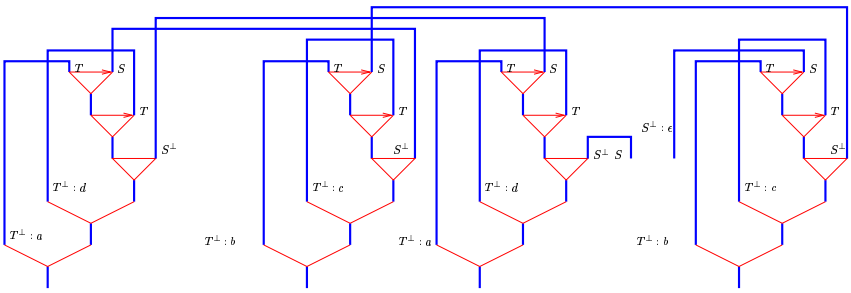
Moreover, we use both commutative and non-commutative connectors, and the building of the path defines the order of the lexical items. The path performs the splitting of the sequent required in the rules (especially the adjunction rule) of [AFV96].



(a) Adjunction of T_2 on another instance of T_2



(b) Cut-elimination



(c) Adjunction onT_1 and cut-elimination

FIG. C.9 – Generating *aabbccdd*

Finally, to know how to express the semantics, at least two possibilities arise : to see it as for intuitionistic proof-nets [dGR96], or as having an alternative expression like with the derivation trees (trees that track the operations performed during a derivation).

Bibliographie

[Abe93] Anne Abeillé. *Les nouvelles syntaxes.* Armand Colin Éditeur, Paris, Armand Colin Éditeur, 103, boulevard St-Michel - 75240 Paris, 1993.

[Abr91] Vito Michele Abrusci. Phase semantics and sequent calculus for pure non-commutative classical linear propositional logic. *Journal of Symbolic Logic*, 56(4) :1403–1451, décembre 1991.

[Abr95] Vito Michele Abrusci. Non-commutative proof nets. Dans *Advances in Linear Logic*, édité par J.-Y. Girard, Y. Lafont, et L. Regnier, pages 271–296. Cambridge University Press, 1995. Proceedings of the Workshop on Linear Logic, Ithaca, New York, June 1993.

[AD89] Harvey Abramson et Veronica Dahl. *Logic Grammars*. Symbolic Computation. Springer-Verlag, 1989.

[Adr92] Pieter Willem Adriaans. *Language Learning from a Categorial Perspective.* Thèse de doctorat, University of Amsterdam, Amsterdam, The Netherlands, 1992.

[AFV96] Vito Michele Abrusci, Christophe Fouqueré, et Jacqueline Vauzeilles. Tree adjoining grammars in non-commutative linear logic. Dans *LACL'96*, édité par Christian Retoré, volume 1328 de *LNCS/LNAI*, pages 96–117, New-York, USA, septembre 1996. Springer-Verlag Inc.

[Ajd35] Kazimierz Ajdukiewicz. Die syntaktische Konnexitat. *Studia Philosophica*, 1 :1–27, 1935. English translation in Storrs McCall (ed), *Polish Logic 1920-1939*, Oxford University Press, pp. 207-231.

[Als92] Hiyan Alshawi. *The Core Language Engine.* ACL-MIT Press Series in Natural Language Processing. MIT Press, 1992.

[AM98] Vito Michele Abrusci et Elena Maringelli. A new correctness criterion for cyclic proof nets. *Journal of Logic, Language, and Information*, 7(4) :449–502, 1998.

[Ang80] Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45, 1980.

[Asp91] Andrea Asperti. A linguistic approach to deadlock. Rapport technique, Département de Mathématiques et d'Informatique de l'École Normale Supérieure, octobre 1991.

[Ber00] Raffaella Bernardi. Polarity items in resocurce logics : A comparison. Dans *Proceedings of the Student Session, Esslli 00*, 2000.

[BH50] Yehoshua Bar-Hillel. On syntactical categories. *Journal of Symbolic Logic*, 15 :1–16, 1950.

[BHGS64] Yehoshua Bar-Hillel, C. Gaifman, et E. Shamir. On categorial and phrase structure grammars. Dans *Language and Information. Selected Essays on their Theory and Application*, édité par Y. Bar-Hillel, pages 99–115. Addison-Wesley, Reading, MA, 1964.

[Boi88] Patrice Boizumault. *PROLOG, l'implantation.* Études et recherches en informatiques. Masson, 1988.

[BP89] Wojciech Buszkowski et Gerald Penn. Categorial grammars determined from linguistic data by unification. Rapport Technique TR-89-05, Department of Computer Science, University of Chicago, juin 1989.

[Bus87] Wojciech Buszkowski. Discovery procedures for categorial grammar. Dans *Categories, Polymorphism, and Unification*, édité par Ewan Klein et Johan van Benthem, pages 36–64. Centre for Cognitive Science, University of Edinburgh and Institute for Language, Logic, and Information, University of Amsterdam, Edinburgh and Amsterdam, 1987.

[Bus88] Wojciech Buszkowski. Generative power of categorial grammars. Dans *Categorial Grammars and Natural Language Structures*, édité par Richard T. Oehrle, Emmon Bach, et Deirdre Wheeler, pages 69–94. Reidel, Dordrecht, 1988.

[Car97] Bob Carpenter. *Type-Logical Semantics.* The MIT Press, 1997.

[Cas88] Claudia Casadio. Semantic categories and the development of categorial grammars. Dans Oehrle et al. [OBW88], pages 95–123.

[Cha94] Jean-Pierre Chanod. Finite-state composition of french verb morphology. Rapport technique mltt-04, Xerox Research Centre Europe, Grenoble, 1994.

[Cho63] Noam Chomsky. Formal properties of grammars. Dans *Handbook of Mathematical Psychology*, volume 2, pages 323–418. John Wiley and Sons, New York, 1963.

[CL93a] René Cori et Daniel Lascar. *Logique mathématique*, volume I. Masson, 1993.

[CL93b] René Cori et Daniel Lascar. *Logique mathématique*, volume II. Masson, 1993.

[CMP00] Emmanuel Chailloux, Pascal Manoury, et Bruno Pagano. *Développement d'applications avec Objective Caml*. Éditions O'Reilly, 2000.

[CRZ89] Jonathan Calder, Mike Reape, et Henk Zeevat. An algorithm for generation in unification categorial grammar. Dans *proceedings of the Conference of the European chapter of the ACL*, pages 233–240, 1989.

[Cut80] Nigel J. Cutland. *Computability.* Cambridge University Press, 1980.

[dG95] Philippe de Groote. *The Curry-Howard Isomorphism*, volume 8 de *Cahiers du centre de logique*. Academia-Erasme, Louvain-La-Neuve (Belgique), 1995.

[dG96] Philippe de Groote. Partially commutative linear logic : sequent calculus and phase semantics. Dans *Proofs and Linguistic Categories, Application of Logic tot the Analysis and Implementation of Natural Langugae Proceedings 1996 Roma Workshop*, édité par Vito Michele Abrusci et Claudia Casadio, pages 199–208. Cooperativa Libraria Universitaria Editrice Bologna, 1996.

[dG99] Philippe de Groote. The non-associative lambek calculus with product in polynomial time. Dans *Automatic Reasoning with Analytic Tableaux and Related Methods*, édité par Neil V. Murray, volume 1617 de *Lecture Notes in Artificial Intelligence*, pages 128–139. Springer Verlag, 1999.

[dG00] Philippe de Groote. Linear higher-order matching is np-complete. Dans *Rewriting Techniques and Applications, RTA'00*, édité par L. Bachmair, volume 1833 de *LNCS*, pages 127–140. Springer, 2000.

[dGR96] Philippe de Groote et Chritian Retoré. On the semantic readings of proof-nets. Dans *Formal Grammar*, édité par Glyn Morrill Geert-Jan Kruijff et Dick Oehrle, pages 57–70, Prague, août 1996. FoLLI.

[DI88] Marc Dymetman et Pierre Isabelle. Reversible logic grammars for machine translation. Dans *proceedings of the 2nd International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages*, 1988.

[DJS94] Vincent Danos, Jean-Baptiste Joinet, et Harold Schellinx. On the linear decoration of intuitionistic derivations. *Archives for Mathematical Logic*, 33(6), 1994.

[Dow88] David Dowty. Type raising, functional composition, and non-constituent conjunction. Dans *Categorial Grammars and Natural Language Structures*, édité par Richard T. Oehrle, Emmon Bach, et Deirdre Wheeler, pages 153–197. Reidel, Dordrecht, 1988.

[Dow92] Gilles Dowek. Third order matching is decidable. Dans *7th Annual IEEE Symposium of Logic in Computer Science*, pages 2–10, 1992.

[Dow01] Gilles Dowek. *Higher-Order Unification and Matching*, chapter 16. Volume 2 de Robinson et Voronkov [RV01], 2001.

[DR89] Vincent Danos et Laurent Regnier. The structure of multiplicatives. *Archive for Mathematical Logic*, 26, 1989.

[DWP81] David R. Dowty, Robert E. Wall, et Stanley Peters. *Introduction to Montague Semantics.* Kluwer Academic Publishers, 1981.

[Dym91] Marc Dymetman. Inherently reversible grammars, logic programming and computability. Dans *proceedings of the ACL Workhop : Reversible Grammar in Natural Language Processing*, 1991.

[Dym92] Marc Dymetman. *Transformations de Grammaires Logiques et Réversibilité.* Thèse de doctorat, Université Joseph Fourier, Grenoble, France, 1992.

[Dym94] Marc Dymetman. Inherently reversible grammars. Dans *Reversible Grammars in Natural Language Processing*, édité par Tomek Strzalkowski, chapter 2, pages 33–57. Kluwer Academic Publishers, 1994.

[Ear86] Jay Earley. An efficient context-free parsing algorithm. Dans *Readings in Natural Language Processing*, édité par Barbara J. Grosz, Karen Sparck-Jones, et Bonnie Lynn Webber, pages 25–33. Morgan Kaufmann, Los Altos, 1986.

[FGS93] Christine Froidevaux, Marie-Claude Gaudel, et Michèle Soria. *Types de données et algorithmes.* Ediscience International, Paris, 1993.

[Gab96] Dov M. Gabbay. *Labelled Deductive Systems.* Oxford Logic Guides. Oxford University Press, 1996.

[Gal95] Jean Gallier. *On the Correspondence between Proofs and λ-Terms*, chapter 4, pages 55–138. Volume 8 de de Groote [dG95], 1995. ftp://ftp.cis.upenn.edu/pub/papers/gallier/cahiers.dvi.Z.

[Gea72] P. T. Geach. A program for syntax. Dans *Semantics of Natural Language*, édité par D. Davidson et G. Harman, pages 483–497. Reidel, Dordrecht, 1972.

[Gen69] Gerhard Gentzen. Investigations into logical deductions, 1935. Dans *The Collected Papers of Gerhard Gentzen*, édité par M. E. Szabo, pages 68–131. North-Holland Publishing Co., Amsterdam, 1969.

[Ger91] Dale Gerdemann. *Parsing and Generation of Unification Grammars*. Thèse de doctorat, University of Illinois, Cognitive Science, 1991. Technical Report CS-91-06.

[GH90] Dale Gerdemann et Erhard W. Hinrichs. Functor-driven natural language generation with categorial-unification grammars. Dans *prooceedings of the International Conference on Computational Linguistics*, pages 145–150, 1990.

[Gir87a] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50 :1–102, 1987.

[Gir87b] Jean-Yves Girard. *Proof Theory and Logical Complexity*. Studies in Proof Theory. Bibliopolis, 1987.

[Gir89] Jean-Yves Girard. Geometry of interaction I : Interpretation of system F. Dans *Logic Colloquium '88*, édité par C. Bonotto, R. Ferro, S. Valentini, et A. Zanardo, pages 221–260. North-Holland, 1989.

[Gir95a] Jean-Yves Girard. Geometry of interaction III : Accomodating the additives. Dans Girard et al. [GLR95], pages 329–389. ftp://iml.univ-mrs.fr/pub/girard/GOI3.ps.gz.

[Gir95b] Jean-Yves Girard. Linear logic : Its syntax and semantics. Dans Girard et al. [GLR95], pages 1–42. ftp://iml.univ-mrs.fr/pub/girard/Synsem.ps.gz.

[GLR95] Jean-Yves Girard, Yves Lafont, et Laurent Regnier. *Advances in Linear Logic*, volume 222 de *London Mathematical Society Lecture Note Serie*. Cambridge University Press, 1995. Proceedings of the Workshop on Linear Logic, Ithaca, New York, June 1993.

[GLT88] Jean-Yves Girard, Yves Lafont, et P. Taylor. *Proofs and Types*. Numéro 7 dans Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1988.

[Gol67] E. Mark Gold. Language identification in the limit. *Information and Control*, 10 :447–474, 1967.

[Gol81] Warren D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13(2), 1981.

[Gue99] Guerrini. Correctness of multiplicative proof nets is linear. Dans *LICS : IEEE Symposium on Logic in Computer Science*, 1999.

[Haa89] Andrew Haas. A generalization of the offline parsable grammars. Dans *proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 237–242, 1989.

[How80a] W. A Howard. *The Formulæ-as-Types Notion of Construction*, pages 479–490. Dans Hindley et Seldin [HS80], 1980.

[How80b] W. A. Howard. *To H. B. Curry : Essays on combinatory logic, Lambda Calculus and Formalism*, chapter The Formulæ-as-Types Notion of Construction, pages 479–490. Academic Press, 1980.

[HS80] J. Roger Hindley et Jonathan P. Seldin. *To H. B. Curry : Essays on combinatory logic, Lambda Calculus and Formalism*. Academic Press, 1980.

[HU79] John E. Hopcroft et Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 1979.

[Hue73] Gérard P. Huet. The undecidability of unification in third order logic. *Information and Control*, 22(3), 1973.

[Hue75] Gérard P. Huet. A unification algorithm for typed lambda -calculus. *Theoretical Computer Science*, 1 :27–57, juin 1975.

[JK96] Aravind K. Joshi et Seth Kulick. Partial proof trees, resource sensitive logics and syntactic constraints. Dans *LACL'96*, édité par Christian Retoré, volume 1328 de *LNCS/LNAI*, pages 21–42, New-York, USA, septembre 1996. Springer-Verlag Inc.

[JK97a] Aravind K. Joshi et Seth Kulick. Partial proof trees as building blocks for a categorial grammar. *Linguistics and Philosophy*, 20(6) :637–667, décembre 1997.

[JK97b] Aravind K. Joshi et Seth Kulick. Partial proof trees, resource sensitive logics and syntactic constraints. Dans Retoré [Ret97a], pages 21–42.

[JLT75] Aravind K. Joshi, Leon S. Levy, et Masako Takahashi. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1) :136–163, 1975.

[Joh98] Mark Johnson. Proof nets and the complexity of processing center embedded constructions. *Journal of Logic, Language and Information*, 7(4), 1998.

[Joi93] Jean-Baptiste Joinet. *Étude de la normalisation du calcul des séquents classiques à travers la logique linéaire*. Thèse de doctorat, Université de Paris VII, 1993.

[Kan91] Max I. Kanovich. The multiplicative fragment of linear logic is NP-complete. ITLI Prepublication Series X-91-13, University of Amsterdam, 1991.

[Kan96] Makoto Kanazawa. Identification in the limit of categorial grammars. *Journal of Logic, Language, and Information*, 5(2) :115–155, 1996.

[Kan98] Makoto Kanazawa. *Learnable Classes of Categorial Grammars*. CSLI and FoLLI, 1998.

[Kar86] Lauri Karttunen. Radical lexicalism. Rapport Technique CSLI-86-68, Center for the Study of Language and Information, Stanford University, décembre 1986.

[Kar94] Lauri Karttunen. Constructing lexical transducers. Dans *Proceedings of the 15th International Conference on Computational Linguistics*, Kyoto, August 1994.

[Kay96] Martin Kay. Chart generation. Dans *proceedings of the conference of the European Chapter of the ACL*, pages 200–204, 1996.

[KB82] Ronald M. Kaplan et Joan Bresnan. Lexical-Functional Grammar : A formal system for grammatical representation. Dans *The Mental Representation of Grammatical Relations*, édité par Joan Bresnan, pages 173–281. MIT Press, 1982.

[KCGS96] Lauri Karttunen, Jean-Pierre Chanod, Gregory Grefenstette, et Anne Schiller. Regular expressions for language engineering. *Natural Language Engineering*, 2 :303–328, 1996.

[KKZ92] Lauri Karttunen, Ronald M. Kaplan, et Annie Zaenen. Two-level morphology with composition. Dans *Proceedings of the 14th International Conference on Computational Linguistics*, Nantes, July 1992.

[Koh92] Dieter Kohl. Generation from under- and overspecified structures. Dans *proceedings of the International Conference on Computational Linguistics*, pages 686–692, 1992.

[Kop95] Alexey P. Kopylov. Decidability of linear affine logic. Dans *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, pages 496–504, 1995.

[Kri90] Jean-Louis Krivine. *Lambda-calcul : types et modèles*. Études et recherches en informatique. Masson, 1990.

[Lad92] D. Robert Ladd. An introduction to intonational phonology. Dans *Papers in Laboratory Phonology II : Gesture, Segment, Prosody*, édité par G.J. Docherty et D.R. Ladd, pages 321–334. Cambridge University Press, 1992.

[Lam58] Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65(3) :154–170, 1958.

[Lam61] Joachim Lambek. On the calculus of syntactic types. Dans *Structure of Language and its Mathematical Aspects*, édité par R. Jacobsen, Proceedings of Symposia in Applied Mathematics, XII. American Mathematical Society, 1961.

[Lam94] François Lamarche. Proof nets for intuitionistic linear logic : Essantial nets. Rapport technique, Imperial College, 1994.

[Lec92] Alain Lecomte. Proof-nets and dependencies. Dans *Proceedings of the International Conference on Computational Linguistics*, volume 1, pages 394–400, 1992.

[Lec96] Alain Lecomte. Grammaire et théorie de la preuve : une introduction. *Traitement Automatique des Langues*, 37(2) :1–37, 1996.

[Lev96] Jordi Levy. Linear second-order unification. Dans *Proceedings of the 7th International Conference on Rewriting Techniques and Applications (RTA-96)*, édité par Harald Ganzinger, volume 1103 de *LNCS*, pages 332–346, Berlin, juillet 1996. Springer-Verlag.

[LMSS92] Patrick Lincoln, John Mitchell, Andre Scedrov, et Natarajan Shankar. Decision problems for propositional linear logic. *Annals of Pure and Applied Logic*, 56 :239–311, avril 1992. Also in the Proceedings of the 31th Annual Symposium on Foundations of Computer Science, St Louis, Missouri, October 1990, IEEE Computer Society Press. Also available as Technical Report SRI-CSL-90-08 from SRI International, Computer Science Laboratory.

[LR95] Alain Lecomte et Christian Retoré. Pomset logic as an alternative categorial grammar. Dans *Formal Grammar*, Barcelona, 1995.

[LR96a] François Lamarche et Christian Retoré. Proof-nets for the lambek calculus – an overview. Dans *Proceedings 1996 Roma Workshop. Proofs and Linguistic Categories*, édité par Vito Michele Abrusci et Claudio Casadio, pages 241–262. Editrice CLUEB, Bologna, avril 1996.

[LR96b] Alain Lecomte et Christian Retoré. Words as modules : a lexicalised grammar in the framework of linear logic proof nets. Dans *International Conference on Mathematical Linguistics*, Tarragona, Spain, 1996. John Benjamins.

[LR97] Alain Lecomte et Christian Retoré. Logique de ressources et réseaux syntaxiques. Dans *TALN'97*, édité par D. Genthial, pages 70–83, Pôle langage naturel, Grenoble, juin 1997.

[LR99] Alain Lecomte et Christian Retoré. Towards a minimal logic for minimalism. Dans *Proceedings of Formal Grammar 1999*, édité par Geert-Jan M. Kruijff et Richard T. Oehrle, 1999.

[LR01] Alain Lecomte et Christian Retoré. Extending lambek grammars : a logical account of minimalist grammars. Dans *Proceedings of the Annual Meeting of the ACL*, 2001.

[McK85] Kathleen R. McKeown. *Text Generation.* Studies in Natural Language Processing. Cambrifge University Press, 1985.

[MM97] Josep M. Merenciano et Glyn V. Morrill. Generation as deduction on labelled proof nets. Dans Retoré [Ret97a], pages 310–328.

[Möh89] Rolf H. Möhring. Computationally tractable classes of ordered sets. Dans *Algorithms and Order*, édité par Ivan Rival, volume 255 de *NATO ASI series C*, pages 105–193. Kluwer, 1989.

[MOM89] N. Marti-Oliet et J. Meseguer. From petri nets to linear logic. Dans *Category Theory and Computer Science*, volume 389 de *LNCS*, pages 313–340. Springer, 1989.

[Mon74] Richard Montague. *Formal Philosophy : Selected Papers of Richard Montague*. Yale University Press, New Haven, CT, 1974.

[Moo91] Michael Moortgat. *Categorial Investigations : Logical and Linguistic Aspects of the Lambek Calculus.* Mouton de Gruyter / Foris Publications, 1991.

[Moo96] Michael Moortgat. Multimodal linguistic inference. *Journal of Logic, Language, and Information*, 5(3–4) :349–385, 1996.

[Moo97] Michael Moortgat. Categorial type logics. Dans van Benthem et Meulen [vBM97], pages 93–177.

[Moo99a] Michael Moortgat. Constants of grammatical reasoning. Dans *Constraints and Resources in Natural Language Syntax and Semantics*, édité par Gosse Bouma, Geert-Jan M. Kruijff, Erhard Hinrichs, et Richard Oehrle. CSLI, stanford, 1999.

[Moo99b] Richard Moot. Grail : an interactive parser for categorial grammars. Dans *Proceedings of VEXTAL'99*, pages 255–261, 1999. http://www.let.uu.nl/~Richard.Moot/personal/grail.html.

[Mor94] Glyn V. Morrill. *Type Logical Grammar Categorial Logic of Signs.* Kluwer Academin Publishers, 1994.

[Mor00] Glyn V. Morrill. Incremental processing and acceptability. *Computational Linguistics*, 26(3) :319–338, septembre 2000.

[MP99] Richard Moot et Quintijn Puite. Proof nets for multimodal categorial grammars. Dans *Proceedings of Formal Grammar 1999*, édité par Geert-Jan M. Kruijff et Richard T. Oehrle, pages 103–114, 1999.

[MTV93] M. Masseron, C. Tollu, et J. Vauzeilles. Generating plans in linear logic i : Actions as proofs. *Theoretical Computer Science*, 113(2) :349–370, 1993.

[Neu94] Günter Neumann. *A Uniform Computational Model for Natural Language Parsing and Generation.* Thèse de doctorat, University of the Saarland, Saarbrücken, 1994.

[Nic99] Jacques Nicolas. Grammatical inference as unification. Rapport Technique 3632, INRIA, mars 1999.

[OBW88] Richard T. Oehrle, Emmon Bach, et Deirdre Wheeler. *Categorial Grammars and Natural Language Structures*. D. Reidel Publishing Company, Dordrecht, 1988.

[Pen93] Mati Pentus. Lambek grammars are context free. Dans *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science*, édité par Robert L. Constable, pages 371–373, Montreal, Canada, juin 1993. IEEE Computer Society Press.

[Per90] Fernando C. N. Pereira. Categorial semantics and scoping. *Computational Linguistics*, 16(1) :1–10, 1990.

[Pin95] Steven Pinker. *The Language Instinct.* Penguin Books, 1995.

[Pog98] Sylvain Pogodalla. Lexicalized proof nets in pomset logic and TAGs. Dans *proceedings of Logical Aspects of Computational Linguistics.* Université Pierre Mendès-France (Grenoble 2) and LORIA (Nancy), France, décembre 1998. http://www.xrce.xerox.com/research/mltt/publications.

[Pog99] Sylvain Pogodalla. Génération à l'aide de réseaux de preuve sémantiques. Dans *actes de Génération Automatique de Textes*, pages 77–91. Université Stendhal (Grenoble 3), 1999. http://www.xrce.xerox.com/research/mltt/publications.

[Pog00a] Sylvain Pogodalla. Generation in the lambek calculus framework : an approach with semantic proof nets. Dans *proceedings of the 1st Meeting of the North American Chapter of the ACL*, mai 2000. http://www.xrce.xerox.com/research/mltt/publications.

[Pog00b] Sylvain Pogodalla. Generation, lambek calculus, montague's semantics and semantic proof nets. Dans *proceedings of the International Conference on Computational Linguistics*, août 2000. http://www.xrce.xerox.com/research/mltt/publications.

[Pog00c] Sylvain Pogodalla. Generation with semantic proof nets. Rapport Technique 3878, INRIA, janvier 2000. http://www.inria.fr/RRRT/publications-eng.html, http://www.xrce.xerox.com/research/mltt/publications.

[Pol97] Jean-Yves Pollock. *Langage et sognition. Introduction au programme minimaliste de la grammaire générative.* Psychologie et sciences de la pensée. Presses Universitaires de France, 1997.

[PS94] Carl Pollard et Ivan Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago, 1994. Draft distributed at the Third European Summer School in Language, Logic and Information, Saarbrücken, 1991.

[PW83] Fernando C. N. Pereira et David H. D. Warren. Parsing as deduction. Dans *proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 137–144, 1983.

[Qui99] Willard Van Orman Quine. *Le mot et la chose.* Champs. Flammarion, 1999. Traduction française de *Words and Objects*, 1960.

[Ran98] Aarne Ranta. Syntactic calculus with dependent types. *Journal of Logic, Language and Information*, 7(4) :413–431, 1998.

[Reg92] Laurent Regnier. *λ-calcul et réseaux.* Thèse de doctorat, Université Paris VII, janvier 1992.

[Ret93] Christian Retoré. *Réseaux et séquents ordonnés.* Thèse de doctorat, University of Paris VII, 1993.

[Ret96a] Christian Retoré. Calcul de lambek et logique linéaire. *Traitement Automatique des Langues*, 37(2) :39–70, 1996.

[Ret96b] Christian Retoré. Perfect matchings and series-parallel graphs : multiplicatives proof nets as r&b-graphs. Dans *Linear Logic 96 Tokyo Meeting*, édité par J.-Y. Girard, M. Okada, et A. Scedrov, volume 3 de *Electronic Notes in Theoretical Computer Science*, avril 1996.

[Ret97a] Christian Retoré. *Logical Aspects of Computational Linguistics : First International Conference, LACL '96, Nancy, France, September 23–25, 1996 : selected papers*, volume 1328 de *Lecture Notes in Artificial Intelligence and Lecture Notes in Computer Science*, New York, NY, USA, 1997. Springer-Verlag Inc.

[Ret97b] Christian Retoré. Pomset logic : a non-commutative extension of classical linear logic. Dans *TLCA'97*, volume 1210 de *LNCS*, pages 300–318, 1997.

[Ret99] Christian Retoré. Handsome proofnets : R&b-graphs, perfect matchings and series-parallel graphs. Rapport Technique RR-3652, INRIA, mars 1999.

[Roo91] Dirk Roorda. *Resource Logics : Proof-theoretical Investigations.* Thèse de doctorat, University of Amsterdam, septembre 1991.

[RS99] Christian Retoré et Edward Stabler. Resource logics and minimalist grammars. Rapport Technique RR-3780, INRIA, 1999.

[Rue97] Paul Ruet. *Non-Commutative Logic and Concurrent Constraint Programming*. Thèse de doctorat, Université Denis Diderot, Paris 7, 1997. http://www.dmi.ens.fr/~ruet/PAPIERS/these.ps.

[RV01] Alan Robinson et Andreï Voronkov. *Handbook of Automated Reasoning*, volume 2. Elsevier Science, 2001.

[SAJ88] Yves Schabes, Anne Abeillé, et Aravind K. Joshi. Parsing strategies with 'lexicalized' grammars. Dans *proceedings of the 12th International Conference on Computational Linguistics*, volume 2, pages 579–583, 1988.

[Shi88] Stuart M. Shieber. A uniform architecture for parsing and generation. Dans *Proceedings of the 12th International Conference on Computational Linguistics*, volume 2, pages 614–619, Budapest, août 1988.

[Shi91] Takeshi Shinohara. Inductive inference of monotonic formal systems from positive data. *New Generation Computing*, 8(4) :371–384, 1991.

[Shi93] Stuart M. Shieber. The problem of logical-form equivalence. *Computational Linguistics*, 19(1) :179–190, 1993.

[SPKK86] Stuart M. Shieber, Fernando C. N. Pereira, Lauri Karttunen, et Martin Kay. A compilation of papers on unification-based grammar formalisms, part ii. Rapport Technique CSLI-86-48, Center for the Study of Language and Information, Stanford University, avril 1986.

[Sta97] Edward Stabler. Derivational minimalism. Dans *Proceedings of the 1st International Conference on Logical Aspects of Computational Linguistics (LACL-96)*, édité par Christian Retoré, volume 1328 de *LNAI*, pages 68–95, Berlin, septembre 1997. Springer.

[SU98] Morten Heine B. Sørensen et Pawel Urzyczyn. Lectures on the curry-howard isomorphism. Available as DIKU Rapport 98/14, 1998. ftp://ftp.diku.dk/diku/semantics/papers/D-368.ps.gz.

[SUP+83] Stuart M. Shieber, H. Uszkoreit, Fernando C. N. Pereira, J. J. Robinson, et M. Tyson. The formalism and implementation of PATR-II. Dans *Research on Interactive Acquisition and Use of Knowledge*. SRI International, Menlo Park, CA, 1983.

[SvNMP89] Stuart M. Shieber, Gertjan van Noord, Robert C. Moore, et Fernando C. N. Pereira. A semantic-head-driven generation algorithm for unification-based formalisms. Dans *proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 7–17, 1989.

[Tar90] Alfred Tarski. *Logic, Semantics, Metamathematics : Papers from 1923-38*. Hackett Publishing Co, 1990.

[Tel98a] Isabelle Tellier. Learning to understand. Rapport Technique IT-320, LIFL, 1998.

[Tel98b] Isabelle Tellier. Meaning helps learning syntax. Dans *Proceedings of the 4th International Colloquium on Grammatical Inference (ICGI-98)*, édité par Vasant Honavar et Giora Slutzki, volume 1433 de *LNAI*, pages 25–36, Berlin, juillet 1998. Springer.

[Tes88] Lucien Tesnière. *Éléments de syntaxe structurale*. Éditions Klincksieck, 1988.

[Usz86a] Hans Uszkoreit. Categorial unification grammars. Rapport Technique CSLI-86-66, Center for the Study of Language and Information, Stanford University, décembre 1986.

[Usz86b] Hans Uszkoreit. Categorial unification grammars. Dans *proceedings of the International Conference on Computational Linguistics*, pages 187–194, 1986.

[vB83] Johan van Benthem. The semantics of variety in categorial grammar. Rapport Technique 83-29, Department of Mathematics, Simon Fraser University, Vancouver, 1983.

[vB90] Johan van Benthem. Categorial grammar meets unification. Rapport Technique CSLI-90-142, Center for the Study of Language and Information, Stanford University, 1990.

[vBM97] J. F. A. K. van Benthem et Alice Ter Meulen. *Handbook of Logic and Language*. MIT Press, 1997.

[vEM92] Jan van Eijck et Robert C. Moore. *Semantic Rules for English*, chapter 5. Dans Alshawi [Als92], 1992.

[vN93] Gertjan van Noord. *Reversibility in Natural Language Processing.* Thèse de doctorat, University of Utrecht, 1993.

[Wed88] Jürgen Wedekind. Generation as structure driven derivation. Dans *proceedings of the International Conference on Computational Linguistics*, pages 732–737, 1988.

[Wed95] Jürgen Wedekind. Some remarks on the decidability of the generation problem in lfg- and patr-style unification grammars. Dans *proceedings of the Conference of the European chapter of the ACL*, pages 45–52, 1995.

[Wed99] Jürgen Wedekind. Semantic-driven generation with lfg-and patr-style grammars. *Computatio9nal Linguistics*, 25(2) :277–281, 1999.

[Yet90] David N. Yetter. Quantales and (noncommutative) linear logic. *Journal of Symbolic Logic*, 55(1) :41–64, mars 1990.

Proofnets and Generation for Type Logical Grammars

Abstract

The study of the connection between syntax and semantics that type logical grammars set has given priority to parsing—from syntax to semantics. This thesis underlines how generation—from semantics to syntax—can also benefit from this close connection.

It relies on the logical study of these grammatical models and points out the use of linear logic and its proofnets. Around Lambek calculus, an intuitionistic non-commutative fragment of linear logic, we study the behavior of the extensions of this calculus as syntactic models, in particular with pomset calculus. For instance, we show that a fragment of this latter generates the same language class as tree adjoining grammars.

On the other hand, the appropriateness of syntax, with the notion of proof, to Montague's semantics, with lambda-terms, appears in the Curry-Howard correspondence. Using proofnets enables us to show that for Lambek calculus and linear semantic representations, with at least one constant, the generation problem is decidable and that these grammars are intrinsically reversible. We characterize the semantic forms that enable polynomial syntactic realization, so that we can propose a full generation method in this framework.

These results, and their implementation as well, use the underlying proof theory, in particular the graph presentation of proofnets. Thus, we get a uniform framework for parsing and generation. For the objective of taking non-linear semantic terms into account thanks to exponential connectives of linear logic, we give a new syntax and a new correctness criterion for proofnets with exponentials as graphs.

Keywords: Categorial grammars, generation, linear logic, proofnets.

Résumé

L'étude de la relation entre syntaxe et sémantique qu'établissent les grammaires de types logiques a essentiellement privilégié le sens de l'analyse — syntaxe vers sémantique. Cette thèse souligne le profit que la génération — sémantique vers syntaxe — tire de l'étroitesse de cette relation.

Elle s'appuie sur l'étude logique des ces modèles grammaticaux et met en avant l'utilisation de la logique linéaire et de ses réseaux de preuve. Autour du calcul de Lambek, un fragment intuitionniste de la logique linéaire non commutative, nous étudions le comportement des extensions de ce calcul en tant que modèles syntaxiques, notamment avec le calcul ordonné. Nous montrons par exemple qu'un fragment de ce dernier permet d'engendrer la même classe de langage que les grammaires d'arbres adjoints.

D'autre part, l'adéquation de la syntaxe, portée par la notion de preuve, à la sémantique de Montague, portée par la notion de lambda-terme, s'illustre dans la correspondance de Curry-Howard. L'utilisation des réseaux de preuve nous permet de montrer que, pour le calcul de Lambek et pour des représentations sémantiques linéaires avec une constante au moins, le problème de génération est décidable et que ces grammaires sont intrinsèquement réversibles. Nous caractérisons les formes sémantiques permettant une réalisation syntaxique polynomiale. Aussi pouvons-nous proposer une méthode complète de génération dans ce cadre.

Ces résultats, de même que l'implémentation dont ils ont fait l'objet, exploitent la théorie de la démonstration sous-jacente et en particulier les réseaux de preuve sous forme de graphes. Nous obtenons ainsi un cadre uniforme pour l'analyse et la génération. Pour le conserver, dans l'optique d'une prise en compte sémantique de termes non linéaires grâce aux connecteurs exponentiels de la logique linéaire, nous donnons une nouvelle syntaxe et un nouveau critère de correction pour les réseaux avec exponentiels sous forme de graphes.

Mots-clés : Grammaires catégorielles, génération, logique linéaire, réseaux de preuve.