THEORIES LOGIQUES DE S. LESNIEWSKI
ET
SYSTEMES EVOLUTIFS DE CONNAISSANCES

Thèse présentée à la Faculté des Lettres,
Institut de philosophie, Chaire de logique
Université de Neuchâtel
pour l'obtention du grade de docteur ès Lettres
par

Frédéric Godet

acceptée sur proposition du jury :

Pr Denis Miéville, professeur à l'Université de Neuchâtel, directeur de thèse

Pr Nadine Gessler, chargée d'enseignement à l'Université de Neuchâtel, rapporteur

Pr François Grize, professeur honoraire à l'Université de Lausanne, rapporteur

Pr Pierre Joray, professeur à l'Université de Rennes, rapporteur

Soutenue le 27 avril 2012

UNIVERSITE DE NEUCHATEL
2012

UNIVERSITÉ DE
NEUCHÂTEL

Faculté des Lettres et Sciences Humaines
Espace Louis-Agassiz 1
CH - 2000 Neuchâtel
Tél : +41 (0)32 718 17 00
Fax : +41 (0)32 718 17 71
secretariat.lettres@unine.ch

IMPRIMATUR

La Faculté des lettres et sciences humaines de l'Université de Neuchâtel, sur les rapports de M. Denis Miéville, directeur de thèse, professeur ordinaire de logique à l'Université de Neuchâtel ; Mme Nadine Gessler, chargée d'enseignement à la Chaire de logique de l'Université de Neuchâtel ; M. François Grize, professeur honoraire à l'Institut des Systèmes d'information de l'Université de Lausanne ; M. Pierre Joray, professeur à l'UFR de philosophie de l'Université de Rennes 1, autorise l'impression de la thèse présentée par M. Frédéric Godet en laissant à l'auteur la responsabilité des opinions énoncées.

Neuchâtel, le 27 avril 2012

Le doyen
Patrick Vincent

Remerciements

Nous sommes particulièrement reconnaissants à Corinne, Pierre et Quentin Godet de la patience et de la compréhension qu'ils ont bien voulu nous témoigner durant notre travail de recherche.

Nous tenons à remercier chaleureusement :

- Corinne Godet pour ses relectures. D'autant plus que les différentes versions du manuscrit qu'elle a revu ne ressemblent en rien à un roman ;
- Laurent Uhler pour les conseils de mise en forme qu'il a bien voulu nous donner ;
- Pr Denis Miéville (notre directeur de thèse) qui nous a fait découvrir S. Leśniewski, qui nous a accompagnés durant cinq ans et sans qui notre travail n'aurait pas abouti ;
- Pr François Grize (rapporteur et membre du jury de notre thèse) en charge des questions informatiques qui nous a conseillés et encouragés tout au long de notre recherche avec pertinence et endurance ;
- Pr Nadine Gessler (rapporteur et membre du jury de notre thèse) qui a mis le doigt avec perspicacité sur certaines incohérences de notre raisonnement et permis ainsi d'apporter la rigueur escomptée pour ce travail ;
- Pr Pierre Joray (rapporteur et membre du jury de notre thèse) qui a dispensé son temps et sa science aux aspects globaux des résultats de notre recherche.

Sans oublier, un des grands logiciens et sémiologues, Jean-Blaise Grize, notre premier directeur de thèse, qui, il y a plus de vingt ans, nous a stimulé à inscrire cette thèse à l'Université de Neuchâtel et à qui nous rendons hommage.

Mots clés :

déduction logique, démonstration, formalisation, grammaire formelle, langage formel, langage pivot, logique de S. Leśniewski, logique des classes, métalangage, moteur de réécriture, preuve, ontologie, réécriture de termes, système de S. Leśniewski, système formel, système logique, théorie des ensembles, transposition.

Résumé :

Les patrons des entreprises fondent leurs décisions sur la base d'indicateurs qui sont alimentés et calculés par des tableaux de bord. Les systèmes informatiques d'aide à la décision augmentent la complexité des rapports et conduisent lors des restructurations des affaires à des malfaçons. Cette recherche propose de formaliser les structures qui participent à la construction de ces indicateurs à l'aide de l'ontologie, un des systèmes formels de Stanisław Leśniewski. La démarche donne les moyens de transposer les entités linguistiques du domaine de connaissances vers le système qu'elle exploite. Elle a pour objectif de mettre en œuvre le contrôle et la rigueur nécessaire d'un système formel pour éviter l'intrusion d'anomalies. L'ontologie répond à ce niveau de besoin de formalisation grâce à sa capacité de quantification des propriétés. Pour faciliter la tâche aux non spécialistes des logiques que sont les financiers et afin de leur permettre de profiter des logiques leśniewskiennes, le développement d'un outil d'aide automatisé a répondu au cahier des charges que s'est donné la recherche. Le niveau actuel de connaissances informatiques ne permet pas de traduire ces besoins directement dans un programme d'ordinateur. Pour traiter le métalangage que constitue l'ontologie, il a fallu définir un langage de réécriture de termes. C'est un langage grammatical qui parle des grammaires et autres règles qui régissent l'ontologie. Lui-même est défini, à l'aide de la théorie des langages formels, par une grammaire qui le pilote. A partir de cette étape intermédiaire, il a été possible de développer un logiciel prototype qui traite des phrases de l'ontologie et des règles qui constituent ce système. Afin de vérifier la faisabilité et la conformité du cahier des charges de la solution, le travail de recherche a fourni un ensemble de grammaires dédiées aux systèmes de S. Leśniewski. La puissance du langage de réécriture a permis d'aborder d'autres domaines de connaissances comme la taxonomie, des traitements encyclopédiques, la gestion de données informatiques.

Table des matières

1. Introduction

Ce chapitre a pour objectif de présenter notre démarche. Il donne les étapes de notre recherche et comment elles s'articulent.

La croissance spectaculaire de l'informatique offre aux décideurs des secteurs économiques un éventail important de logiciels qui contribuent à l'amélioration de la gestion des entreprises. Dans les années 80, l'intérêt des entrepreneurs s'est porté sur des outils automatisés qui leur permettent d'améliorer le pilotage des leurs affaires. Pour parler de ces outils, les spécialistes du domaine parlent de tableaux de bord. La mise en œuvre de ce type de logiciels traite, à partir d'indicateurs, le résultat des affaires. Les structures qui établissent les liens entre les objets mesurés ainsi que le volume des informations pris en compte peuvent conduire à des représentations complexes. Elles nécessitent une attention particulière pour éviter la production de valeurs erronées. Malgré quarante ans d'expérience, les rapports issus des tableaux de bord contiennent très souvent des anomalies. Evidemment, ces défauts engagent les décideurs à prendre des orientations fortuites. Nous nous sommes penché sur la question dans la perspective de décrire ces erreurs et d'en expliquer les causes. Cette analyse fait l'objet de notre chapitre 2. *Exposé du problème, p. 19*. Nous dépistons différents types d'anomalies que nous classons en quatre ensembles de difficultés : *difficultés de calculs*, *difficultés de taxonomie*, *difficultés de restructuration* et *difficultés du maintien d'une propriété*[1]. Nous nous basons sur un exemple de tableau de bord qui nous sert de fil conducteur dans notre développement.

A partir de ce diagnostic, nous nous mettons en quête d'un traitement qui permet aux spécialistes d'éviter les anomalies et les difficultés recensées. Il s'est agi d'explorer la littérature pour trouver un *système* qui nous assure la rigueur nécessaire pour s'affranchir de malfaçons lors du pilotage des entreprises. Notre but est d'appliquer la mécanique d'un *système* qui par sa construction ne tolère aucun risque de génération d'ambiguïtés, de contradictions ou de dérive sémantique. Eviter ces risques et répondre à ce besoin de rigueur, nous conduit à nous orienter du côté des mathématiques, de la logique et, plus particulièrement, des *systèmes formels*. Avant d'entreprendre la sélection d'un *système*, nous avons défini la notion de *domaine de connaissances*, celui qui est délimité, en particulier, par nos exemples de tableaux de bord. Cette première étape nous facilite la préparation au processus de *transposition*, nécessaire à traduire ce domaine dans le *système* que nous choisissons. Dans notre chapitre 3. *Recherche d'un formalisme, p. 27*, parmi quatre *systèmes* : théorie des ensembles, logique des classes, logique des prédicats et les logiques de Stanisław Leśniewski[2], que nous abrégeons *SL*, nous sélectionnons la théorie des ensembles et l'*ontologie* des *SL* qui répondent aux besoins de *formalisation* que nous rencontrons pour traiter les tableaux de bord de façon formelle. Nous ne traitons pas la théorie des ensembles qui ne pose pas de problème particulier pour le traitement de nos exemples.

En revanche, comme tout notre développement se base sur les *SL*, nous présentons deux de ces logiques, la *protothétique* et l'*ontologie*. Afin de mieux comprendre, la spécificité des *SL*, dans le chapitre 4. *Systèmes logiques de Leśniewski, p. 35*, nous commençons par faire un survol des *systèmes formels* traditionnels, section à laquelle nous nous référons pour expliciter les *SL*.

[1] Ces quatre ensembles de difficultés sont définis dans notre développement. Chaque fois qu'une notion nécessite une définition spécifique et est utilisée à plusieurs reprises dans notre texte, nous la mettons en italique. Pour faciliter la lecture et retrouver la définition d'une notion, le lecteur peut se reporter à l'index en fin d'ouvrage. Au numéro de page indiquée par l'occurrence qui contient la mention : (définition) dans une entrée d'index, le lecteur trouve la définition de la notion en question.

[2] Pour *logique(s) de Stanisław Leśniewski*, nous utilisons des synonymes : *logique(s) de S. Leśniewski, logique(s) leśnievskienne(s), système(s) de S. Leśniewski, système de Leśnievski, système(s) leśniewskien(s) (SL).*

Nous introduisons ensuite la *protothétique* et les notions nécessaires à l'appréhender, sachant que cette logique sert de base à l'*ontologie*.

Une grande partie de ce chapitre est dédié à l'*ontologie* qui répond à notre besoin de *formalisation* des tableaux de bord. Au fur et à mesure de la présentation de cette logique, nous introduisons, pour commencer une illustration du procédé de *transposition* que nous retenons et, ensuite, la mise en œuvre de la mécanique leśniewskienne qui nous assure de la rigueur que nous recherchons. C'est à travers les processus de *déduction* que nous pouvons mettre en œuvre la mécanique formelle qui nous offre une solution de traitement à nos tableaux de bord. Après avoir mis en œuvre l'*ontologie* pour notre *domaine de connaissances*, nous pouvons conclure qu'elle répond parfaitement à notre besoin de *formalisation*.

Des *systèmes formels*, comme les *SL*, sont constitués d'un nombre important de notions, de *définitions inductives* et de *règles d'inférences*. Le *système* croît et évolue lors de sa construction par l'apport de *thèses*. On parle de construction développementale. La mise en œuvre de ces matériaux intellectuels nécessite une attention certaines afin d'éviter le non-respect d'une étape de construction ou de *déduction*. Par conséquent, l'utilisation des *SL* nécessitent un minimum de compétences et de pratique que les logiciens ont acquis, mais que les spécialistes des tableaux de bord n'ont, en général, pas dans l'éventail de leurs capacités.

Pour faciliter le travail à ces novices, à savoir l'utilisation de ces *systèmes*, nous proposons : *Une solution d'aide informatisée* dans le chapitre 5*, p. 81*. Nous commençons par faire une analyse des besoins et par établir un cahier des charges qui doit déterminer l'envergure de la *solution* informatique que nous proposons. Etant données les connaissances actuelles de l'informatique en ce qui concerne les théories de la compilation, il n'est pas imaginable de convertir directement dans un programme informatique le langage *protothétique* ou *ontologique* de Leśniewski. Par conséquent, nous passons par des étapes intermédiaires.

La première étape considère l'ensemble de toutes les *phrases*[3] possibles des *SL*. Elles constituent un langage (*protothétique* et *ontologique*) que nous appelons *LO*. Comme notre recherche a la volonté de garder une dimension formelle tout au long de sa démarche, nous faisons appel à la théorie des *langages formels* que nous abrégeons : *LF*. Afin de s'y référer dans la suite de notre développement, nous présentons un survol de cette théorie dans la section *5.3. Les langages formels (LF), p. 85*. Cette étape nous permet de spécifier des sous-ensembles de *phrases* appartenant au langage, *LO*, par des grammaires (*protothétiques* et *ontologiques*) que nous appelons des GO_i.

Dans une deuxième étape, il s'est agi de définir un langage qui parle de ces grammaires GO_i, un langage grammatical ou encore un langage de réécriture que nous abrégeons par *LRE*. Comme là encore, nous faisons appel à la théorie des *langages formels*, nous avons dû définir une grammaire qui détermine le *LRE*, que nous appelons : *GRE*, mis pour grammaire de réécriture.

Une fois le *LRE* défini et dans une troisième étape, nous pouvons envisager le développement d'une *solution* informatique, le logiciel prototype *RE*. A l'issue de cette partie de notre recherche, en confrontant notre cahier des charges, nous pouvons conclure que le *LRE* et le *RE* y répondent.

A partir de là, il reste à vérifier concrètement la faisabilité de l'approche au corpus « linguistique » de l'*ontologie*. Nous présentons dans le chapitre 6*. GOi destinées aux logiques de Leśniewski, p. 151* un ensemble d'exemples de grammaires, les GO_i, destinées aux *SL*. Le

[3] Nous utilisons la notion *phrase*, qui sera approfondie dans notre démarche, comme une structure linguistique construite selon des règles (de grammaire) ayant un sens (tenant compte autant de l'axe syntaxique que de l'axe sémantique). Une phrase est composée de mots formés de graphèmes ou de phonèmes. Les mots participent à la construction syntaxique et ont donc un signifiant et un signifié. Sous l'angle des langages formels, *phrases* et mots sont définis comme des unités abstraites qui constituent le langage, ne tenant compte que de l'axe syntaxique. Dans le cas des métalangages que sont les systèmes formels et les théories logiques leśniewskiennes, les phrases peuvent prendre les formes d'un langage naturel, d'une formule mathématique, d'une expression logique, voire même d'une règle de grammaire.

résultat de la programmation de ces grammaires nous permet de conclure que le *LRE* et le *RE* répondent aussi à notre objectif. Nous avons donné ces exemples de grammaires, GO_i, dans les annexes.

Dans notre chapitre 7. *Perspectives et développements futurs, p. 175,* nous relevons, d'abord, que la puissance du *LRE* ouvre des horizons intéressants pour d'autres *domaines de connaissances*. Ensuite, nous donnons des voies d'améliorations et de nouveaux développements comme la mise en œuvre d'un moteur de stratégie qui pourrait servir d'outil d'aide à la *déduction*, voire même, à la *preuve*.

Notre hypothèse de travail, à savoir appliquer les *SL* à notre *domaine de connaissances* est vérifiée. Notre objectif de faisabilité d'une *solution* informatique d'aide au traitement des *SL* est atteint.

Nous choisissons l'*ontologie* qui se fonde sur la *protothétique*, deux systèmes logiques de Leśniewski, comme moyens de *formalisation* de notre *domaine de connaissances*, à savoir : les tableaux de bord. Cette démarche explique la première partie du titre de notre ouvrage : *Théories logique de S. Leśniewski […]*. Nous verrons que ces tableaux de bord financiers et opérationnels sont soumis aux besoins d'une évolution constante, comme la mise à jour des indicateurs, l'adaptation des structures de produits, la restructuration des liens entre objets et propriétés. Parallèlement, nous soulignerons l'aspect développemental des *SL* qui se construisent petit à petit et qui permettent ainsi de s'adapter à l'évolution d'un *domaine de connaissances*. Ce besoin évolutif et cette capacité développementale expliquent la seconde partie du titre de notre ouvrage : *[…] et systèmes évolutifs de connaissances*.

Notre développement s'inscrit à l'intersection de différentes théories ou disciplines. Nous allons d'une expérience de spécialiste des tableaux de bord qui utilisent des notions comme celle d'« ensemble » qui n'a pas une définition aussi élaborée que celle que propose la théorie des ensembles.

L'*ontologie* que nous utilisons pour la *transposition* des tableaux de bord nous force à entrer dans une terminologie très précise, alors que nous partons d'une approche empirique de type taxonomique qui traite les questions des objets et de leurs propriétés.

Pour garder le niveau de formalisme que nous apportent les *SL*, nous avons fait appel à la théorie des *langages formels* pour définir notre langage *LRE*. Ce qui nous apporte une nouvelle terminologie. Finalement, notre *LRE*, spécialisé comme langage grammatical du langage des *SL*, nécessite la définition de nouvelles notions.

Nous avons donc adopté une systématique pour éviter des ambiguïtés de définitions et spécifier les différentes terminologies utilisées. Si une ambiguïté peut survenir, pour chaque entrée de notre index (en fin d'ouvrage), nous mettons entre parenthèse le domaine (*SL*, *SF*, *LRE*, etc.) pour lequel la notion a été référencée. Nous ajoutons « définition » entre parenthèse pour donner au lecteur la page où il trouvera la définition de la notion.

Après l'index, nous avons introduit une table des abréviations à laquelle on peut se référer pendant la lecture.

2. Exposé du problème

Ce chapitre a pour objectif de présenter quelques aspects du problème qui nous ont motivé à entreprendre notre recherche. Nous abordons de façon intuitive les questions que nous nous sommes posées à travers un exemple issu de la réalité quotidienne des entreprises : la malfaçon de certains tableaux de bord financiers. Les imperfections des contrôleurs financiers lors de l'élaboration de leurs rapports de gestion nous servent de fil conducteur pour notre ouvrage, sachant que le problème va bien au-delà d'une simple question financière puisqu'elle traite des objets et de leurs propriétés. Par étapes et en restant proche des représentations du domaine concerné, nous exhibons un certain nombre de difficultés auxquelles nous répondons dans les chapitres suivants.

2.1 Introduction

C'est parce que nous avons rencontré dans notre cadre professionnel et plus précisément dans la gestion des entreprises un ensemble de difficultés auxquelles il faut répondre que nous avons engagé les travaux de recherches que nous présentons dans cet ouvrage. Pourquoi les tableaux de bord, ou les rapports de gestion qui permettent de piloter une entreprise contiennent autant de malfaçons drainant des anomalies qui peuvent mettre en péril les orientations que choisissent les décideurs ? Nous nous bornons dans ce chapitre à faire le rapport de nos observations sur le sujet. Nous analysons de façon naïve les causes et les effets que nous classons en quatre niveaux : *difficultés* de *calculs*, *difficultés* de *taxonomie*, *difficultés* de *restructuration* et *difficultés du maintien d'une propriété*.

2.2 Exemple d'un tableau de bord financier

Quel que soit le domaine d'activités ou de connaissances, il nécessite une mise en forme qui permet de partager avec autrui ce que l'on peut en dire. Selon le but visé et dans de nombreux cas, la représentation que l'on peut s'en faire doit faciliter les opérations de *transformation* et de *restructuration* qui évoluent selon l'enrichissement du *domaine de connaissances*.

N'importe quelle industrie utilise des indicateurs qui lui permettent de piloter sa rentabilité financière et opérationnelle. Ils servent à établir des rapports sur l'état de l'entreprise qui offrent aux cadres le moyen de prendre les décisions stratégiques relatives à l'évolution des affaires. Très souvent, les contrôleurs de gestion n'approfondissent ou ne *formalisent* pas assez les règles de gestion nécessaires aux calculs des moyens de mesure. Ils obtiennent ainsi des rapports erronés.

Nous prenons un exemple très simplifié et véritablement issu de la réalité : l'élaboration d'un tableau de bord financier d'un commerce d'alimentation. Comme première étape, il s'agit de fournir un rapport, construit sur un tableur, qui s'intéresse au chiffre d'affaires de l'entreprise pour chaque produit et pour chaque semestre :

Résultats du commerce d'alimentation pour 2011		
Chiffre d'affaires		30 800
Whisky		10 500
1er semestre	5 000	
2nd semestre	5 500	
Vin		16 300
1er semestre	8 200	
2nd semestre	8 100	
Soda		4 000
1er semestre	1 900	
2nd semestre	2 100	

Figure 1. Tableau de bord du calcul du chiffre d'affaires

Le tableau de la *Figure 1, p. 20* calcule le chiffre d'affaires, à partir des périodes détaillées (semestre), les valeurs annuelles, consolidées sur les lignes de produits (Whisky, Vin et Soda), nouvelles valeurs qui, elles-mêmes, sont cumulées pour l'année (2011). Nous n'avons rien à signaler au sujet de ce rapport. Il correspond et répond clairement aux besoins d'informations sur l'entreprise.

Dans une deuxième étape, à des fins de comparaisons, introduisons, sans réflexion particulière, le chiffre d'affaires d'un Concurrent ventilé sur les deux semestres de 2011 :

Résultats du commerce d'alimentation pour 2011		
Chiffre d'affaires		150 800
Whisky		10 500
1er semestre	5 000	
2nd semestre	5 500	
Vin		16 300
1er semestre	8 200	
2nd semestre	8 100	
Soda		4 000
1er semestre	1 900	
2nd semestre	2 100	
Concurrent		**120 000**
1er semestre	**61 000**	
2nd semestre	**59 000**	

Figure 2. Introduction incohérente de la concurrence dans le tableau de bord

A une première lecture, il apparaît immédiatement qu'une malfaçon s'est glissée lors de l'introduction des résultats de la concurrence, puisque le chiffre d'affaires du négociant d'alimentation est consolidé avec celui de sa concurrence. Le tableau ne présente aucune anomalie d'un point vue syntaxique. Toutefois, d'un point de vue sémantique, le chiffre d'affaires de la concurrence a été introduit au même niveau que les résultats présentés par produit. Nous avons, à dessein, confondu les notions de produit et de Chiffre d'affaires. Même si une telle erreur, que nous classons dans les *difficultés de taxonomie*, paraît caricaturale, elle est assez fréquente dans la réalité des entreprises. En effet, il faut savoir que ce type de rapports est en général constitué de milliers de données qui facilitent ce genre d'anomalies.

Donnons-nous une des corrections possibles qui maintient la cohérence de notre tableau de bord :

Résultats du commerce d'alimentation pour 2011			
Notre chiffre d'affaires			30 800
Whisky		10 500	
1er semestre	5 000		
2nd semestre	5 500		
Vin		16 300	
1er semestre	8 200		
2nd semestre	8 100		
Soda		4 000	
1er semestre	1 900		
2nd semestre	2 100		
Chiffre d'affaires de la concurrence			**120 000**
Tous les produits		**120 000**	
1er semestre	**61 000**		
2nd semestre	**59 000**		

Figure 3. Introduction cohérente de la concurrence dans le tableau de bord

Dans cette correction, nous avons séparé explicitement la notion de produit de celle du Chiffre d'affaires de notre négociant.

Dans le but d'analyser les sources possibles d'anomalies, transformons la représentation sous forme de tableau de la *Figure 1, p. 20* en une structure arborescente :

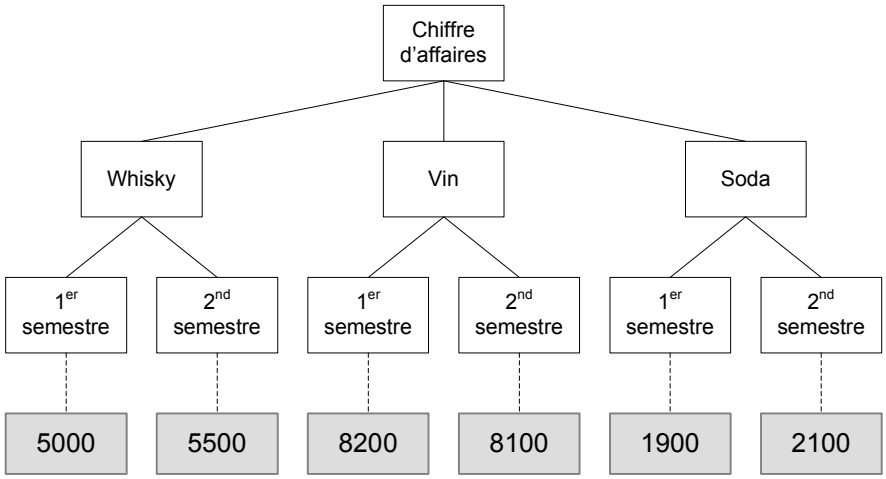


Figure 4. Une structure arborescente du tableau de bord des affaires

Cette structure introduit des nœuds et des liens entre eux qu'il reste à définir. Elle ordonne aussi des niveaux d'arborescence qui vont nous permettre d'expliciter des notions que les tableaux ne contiennent pas.

Comme troisième étape, donnons-nous un nom pour chacun de ces niveaux :

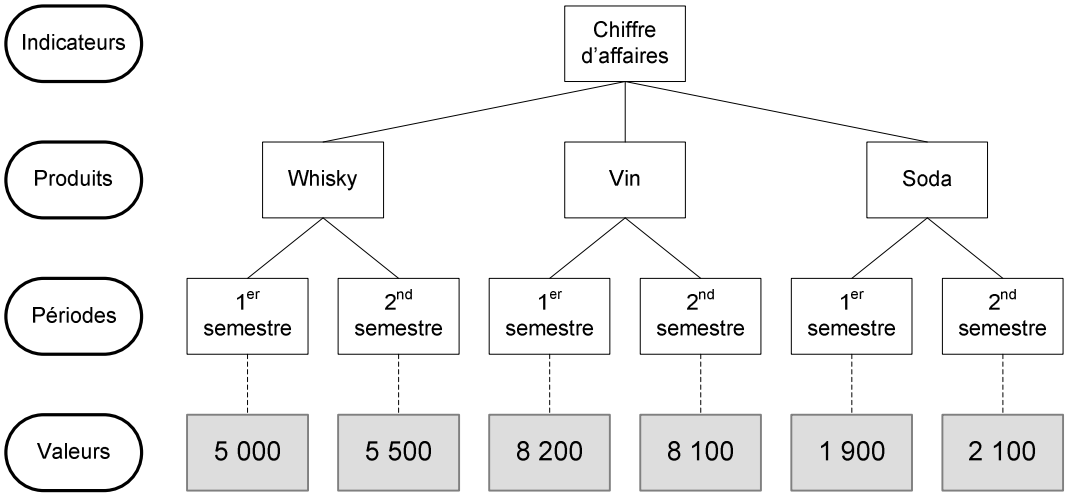


Figure 5. Une structure arborescente du tableau de bord des affaires avec mention des niveaux

De façon intuitive, nous avons nommé les niveaux de notre arborescence et donné par cet artifice une propriété commune pour tous les nœuds d'un même niveau.

Comme quatrième étape, et pour donner un reflet complet de la structure, il faut encore mentionner, quelque part, que tout le rapport est calculé pour une année précise, soit 2011. Dès lors, il s'agit d'introduire un nouveau niveau :

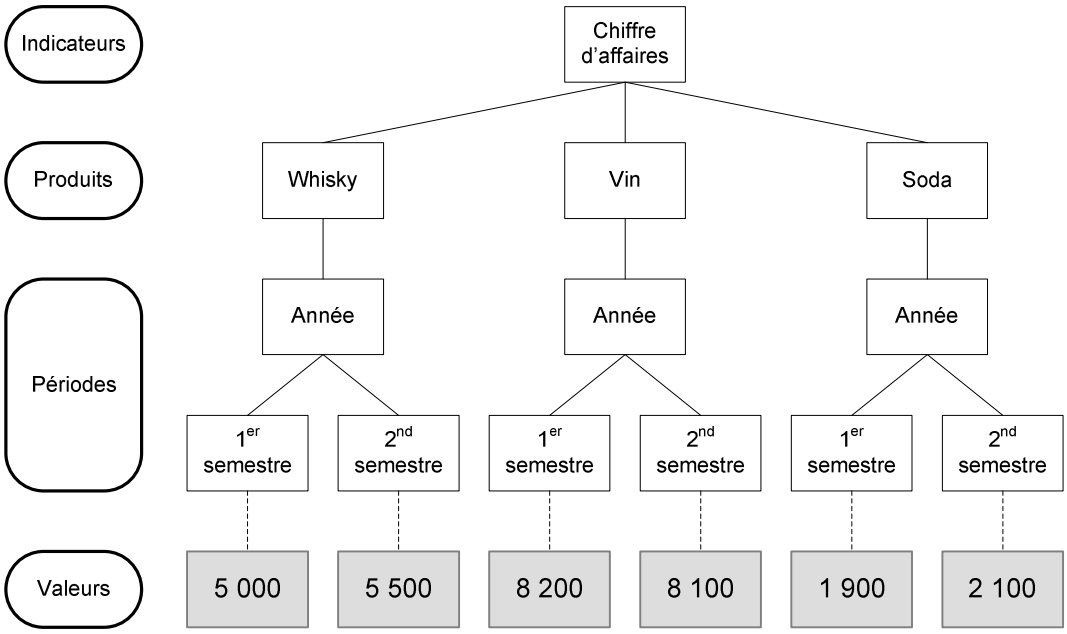


Figure 6. Une structure arborescente du tableau de bord des affaires avec introduction du niveau Année

La structure arborescente aurait pu être modifiée de différentes façons pour intégrer le niveau Année. Nous avons choisi arbitrairement de l'intégrer au-dessus des nœuds temporels, semestre, de manière à faire apparaître le cumul des semestres. Cette manière de faire a pour effet indirect que ce qui précédemment nommait un seul niveau peut en nommer deux ou plus. Par conséquent, dans cette représentation, Année et semestre prennent la même propriété : Périodes.

Pour illustrer une nouvelle malfaçon et comme cinquième étape, introduisons dans notre rapport un nouvel indicateur, le Bénéfice :

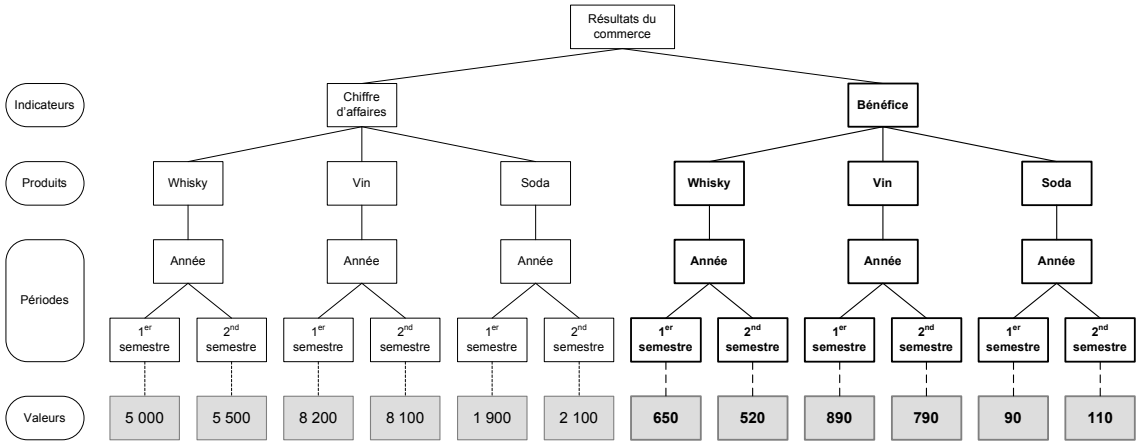


Figure 7. Introduction du sous arbre Bénéfice dans le tableau de bord

De façon intuitive, cette représentation arborescente permet de répondre sans ambiguïté à une question comme : « Quel est le bénéfice réalisé le second semestre pour les sodas ? » Avec pour réponse : 110. Nous avons procédé en partant du nœud Bénéfice et parcouru ses branches jusqu'à la dernière feuille, la valeur. Par conséquent, notre arborescence nous a servi de structure de recherche. Osons une deuxième question : « Quel est le chiffre d'affaires total ? » A savoir : 29 800. En procédant comme pour la première question, mais en cumulant toutes les valeurs, nous aboutissons à une réponse. En principe, un comptable valide que ces deux réponses sont bien celles que l'on attendait aux deux questions. Dès lors, on peut déduire qu'à partir de n'importe quel nœud, on peut parcourir ses branches et cumuler toutes les valeurs. Le même comptable validera-t-il la réponse : 33 850, à la question : « Quel sont les résultats du commerce ? » Non, il n'acceptera pas que l'on additionne un chiffre d'affaires avec un bénéfice ! En effet, le bénéfice est exprimé par la soustraction des dépenses au chiffre d'affaires. Par conséquent, si notre modèle, d'un point de vue syntaxique fonctionne parfaitement, il ne convient pas d'un point de vue sémantique. Nous nommons ce type d'erreurs : *difficultés de calculs*. En effet, aucune restriction n'est prévue pour autoriser ou non l'addition de deux nœuds quelconques. En fait, notre représentation arborescente n'est pas très heureuse. Elle ne dit rien au sujet des liens entre nœuds. Il s'agit de les spécifier formellement. Ce qui sera abordé dans les chapitres suivants.

Comme sixième étape, étudions quel impact peut avoir une nouvelle orientation des affaires en remplaçant la vente de Vin par celle de Bière. Intuitivement et pour garder la cohérence du rapport, il apparaît immédiatement que si nous nous autorisons à substituer un nom de nœud par un autre nom, nous devrons le faire partout dans le modèle. De plus, puisqu'il s'agit d'un nouveau produit, nous sommes appelé à adapter toutes les Valeurs qui dépendent des nœuds où la substitution a été réalisée[4] :

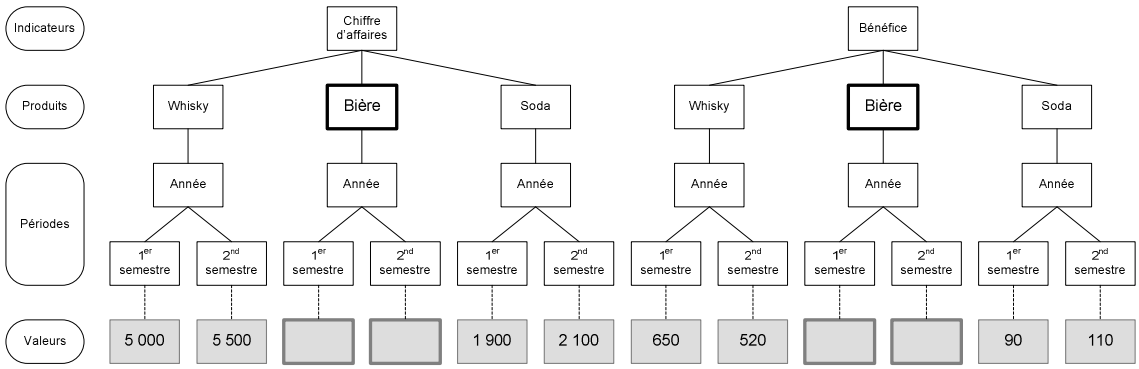


Figure 8. Substitution de l'élément Vin par Bière

[4] Pour cette nouvelle représentation, nous supprimons le niveau : Résultats du commerce avec ses deux liens qui entraînaient l'anomalie observée dans la figure précédente.

A la septième étape, tenons compte d'une exigence rencontrée fréquemment dans les affaires, par exemple : l'arrêt du négoce des sodas et le démarrage de la vente du chocolat. Prenons aussi une approche analytique plus fine des affaires en intégrant un nouveau niveau, les types de Produits (Boisson, Aliment). Pour répondre à ce nouveau besoin, une grande partie de notre arborescence nécessite une sérieuse *restructuration*. Nous nommons ce type de questions : *difficultés de restructuration*.

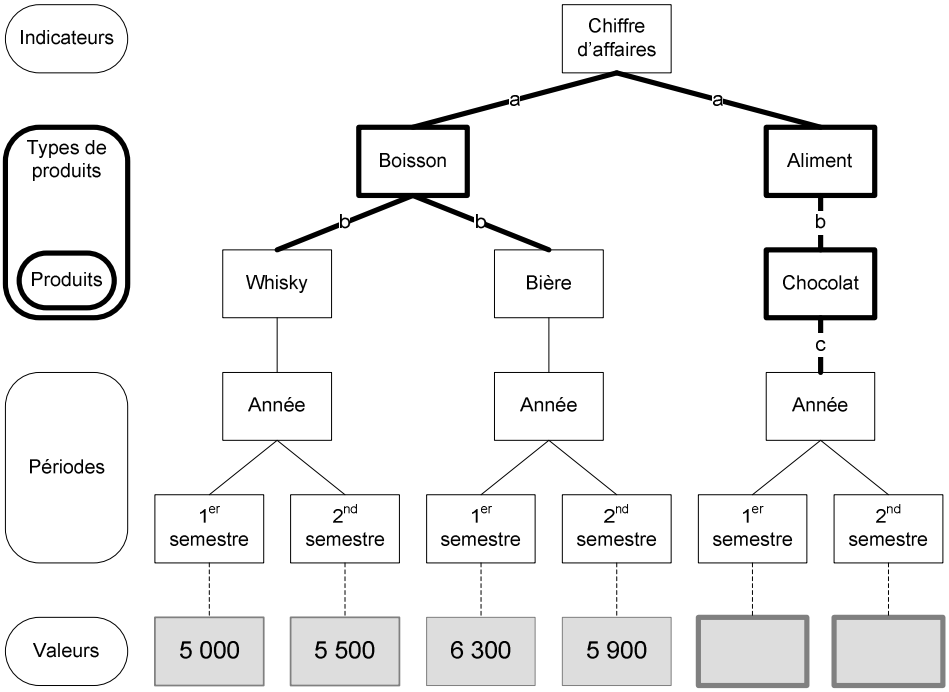


Figure 9. Remplacement de l'élément Soda par une nouvelles structure Aliment-Chocolat

Nous avons adapté notre modèle en tenant compte des observations qui ont précédé concernant la substitution et l'adaptation des valeurs, sans pour autant savoir si l'opération de *restructuration* est du type substitution, remplacement ou suppression suivi d'insertion.

Comme huitième étape, considérons que le pilotage des affaires nécessite des comparaisons d'une année fiscale sur une autre, soit de structures à structures différentes. Cela aura pour impact d'adapter, de recalculer les valeurs à ces fins de comparaisons. Etant donnée, la complexité grandissante, une pure approche intuitive ne suffit plus pour assurer la cohérence attendue de ce type de rapport.

Abordons une neuvième et dernière étape en prenant une nouvelle représentation :

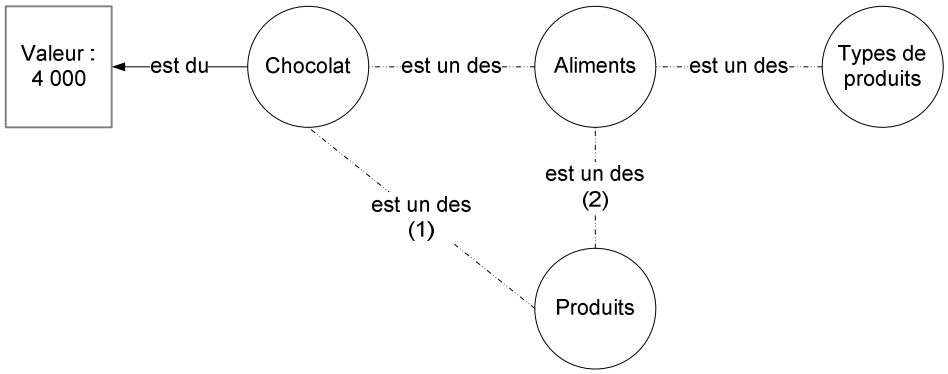


Figure 10. Taxon du chocolat, représentation intuitive

Imaginons que dans une *restructuration* quelconque nous avons à supprimer un niveau d'une arborescence, en d'autres termes, à supprimer une propriété. Posons-nous la question : « Si une *transformation* de structure devait nous conduire à supprimer la propriété Aliments,

Chocolat serait-il toujours un produit ? ». A partir de la forme ci-dessus, procédons à la suppression de la propriété Aliments :

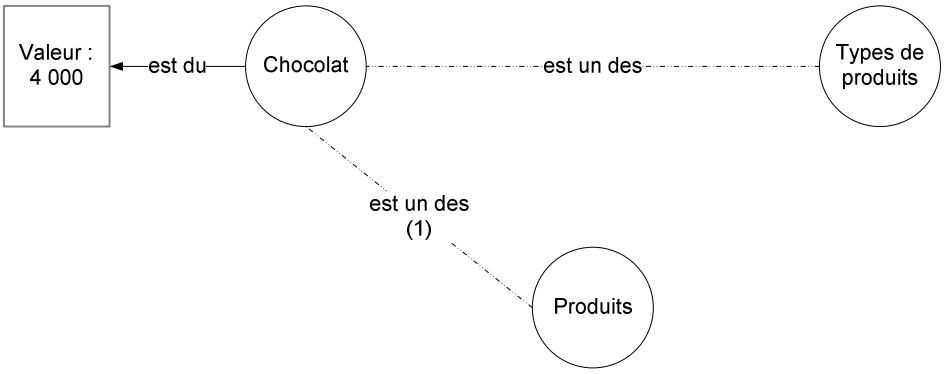


Figure 11. Suppression de la propriété Aliments dans le taxon du Chocolat

Dans le schéma résultant, nous observons indiscutablement que le lien est un des (1) nous permet d'affirmer, qu'après la *transformation* de suppression, que Chocolat est toujours un des Produits ! Evidemment, si nous n'avions pas établi le lien est un des (1) dans la *Figure 10, p. 24,* Chocolat aurait perdu sa propriété Produits. Nous appelons cette opération de suppression : *difficultés du maintien d'une propriété.*

2.3 Conclusion

Nous avons cherché les causes de malfaçons de certains tableaux de bord des affaires. Cela nous a conduit à isoler un ensemble de difficultés que nous avons classées en quatre niveaux : *difficultés de calculs*, *difficultés de taxonomie*, *difficultés de restructuration* et *difficultés du maintien d'une propriété*. Il s'agit maintenant de trouver une technique à mettre en œuvre pour éviter ces résultats erronés. Nous devons aussi nous assurer que les modèles utilisés définissent explicitement les relations entre deux entités quelconques, qu'il s'agisse d'objets ou de propriétés. De plus, le formalisme que nous recherchons doit tenir compte des deux axes syntaxique et sémantique. Pour décrire un monde de connaissances et pour nous affranchir des risques d'erreurs mis en évidence avec les tableaux de bord, nous nous sommes mis en quête d'un formalisme que nous abordons dans le chapitre suivant.

3. Recherche d'un formalisme

Après avoir exprimé les difficultés auxquelles notre développement doit répondre, nous nous mettons en quête d'un système formel qui nous permet de nous assurer d'un niveau de rigueur qui ne souffre d'aucune malfaçon de l'interprétation du domaine de connaissances que nous devons traiter. Nous examinons ce qu'apportent la théorie des ensembles, la logique des classes, la logique des prédicats et les systèmes leśniewskiens. Nous donnons pour chaque théorie ou logique, quel peut être son apport et comment nous pourrions l'utiliser pour résoudre notre problème. Nous indiquons que les logiques qui ont la capacité de quantifier les propriétés ou les foncteurs sont celles qui ont la puissance nécessaire pour résoudre l'ensemble de nos difficultés.

3.1 Introduction

Notre étude nous a amenés à considérer une théorie et plusieurs logiques, que nous nommons : *systèmes*. Nous allons évaluer chaque *système*, en reprenant les difficultés que nous avons isolées dans le chapitre précédent. Cette exploration nous conduira à utiliser deux de ces *systèmes* pour concrétiser nos besoins de *formalisation*. Reste à définir ce que nous cherchons à *formaliser*, que nous appelons *domaine de connaissances*. A noter que cette *formalisation* devra prendre en compte que le domaine évolue au cours du temps, comme nous l'avons vu dans les *restructurations* exigées par l'évolution des affaires dans les exemples du tableau de bord

3.2 Domaine de connaissances

La « réalité du monde perçu par un être humain » passe par la langue naturelle qui elle-même est représentée par un corpus linguistique. Ce corpus définit une partie de la « réalité du monde perçu par un être humain » que nous appelons le *domaine de connaissances* (*DC*). Le sujet du tableau de bord, que nous avons abordé dans le chapitre précédent, constitue un *domaine de connaissances* particulier décrit ou représenté par son corpus linguistique, schématiquement :

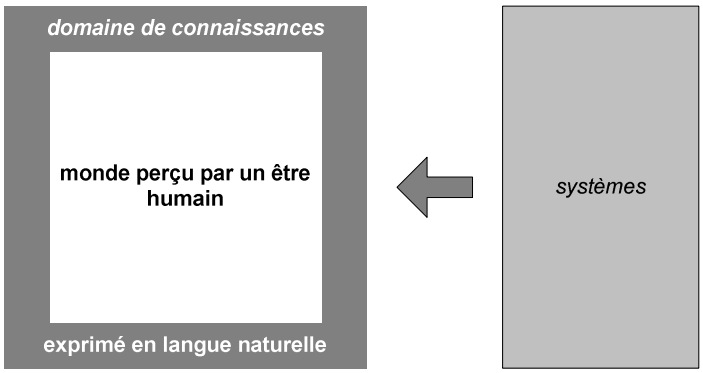


Figure 12. Première étape de notre démarche

3.3 Théorie des ensembles

Nous entendons par théorie des ensembles, la théorie dont Georg Cantor est le créateur. Issue des mathématiques, elle ne se fondait pas, initialement, sur une axiomatique. Elaborée à la fin du 19e siècle, à cause de certains paradoxes, elle a fait couler beaucoup d'encre durant le 20e siècle. Parmi la liste de ses paradoxes[5], celui qui a retenu particulièrement l'attention des

[5] Paradoxe de Berry, paradoxe de Richard, paradoxe de Russell.

mathématiciens et des logiciens est celui de Russell : quand on tente de construire l'ensemble de tous les ensembles qui n'appartiennent pas à eux-mêmes, on rencontre une antinomie. A ce niveau de notre étude, ce paradoxe ne nous empêche pas de traduire une partie de notre *domaine de connaissances* à l'aide des matériaux que nous apporte cette théorie. Sans comprendre a priori quels sont les processus qui définissent les ensembles, l'approche intuitive nous permet de définir des collections d'objets de façon extensionnelle, sans nous poser trop de questions sur le sens qui nous permet de faire appartenir un élément à un ensemble et d'utiliser ce que la théorie nous apporte comme l'appartenance, l'inclusion, l'intersection, etc. De plus, la théorie tient compte des notions du continu et de l'infini, concepts qui vont au-delà de la puissance dont nous avons besoin, puisque nos ensembles sont finis. L'approche de Cantor traite initialement d'objets mathématiques, de fonctions, de relations (d'ordre), ce qui nous arrange bien puisqu'elle nous assure un formalisme en ce qui concerne les calculs que nous mettons en œuvre dans nos structures arborescentes.

Abordons la représentation qu'utilisent les *solutions* informatiques d'aide à la décision[6]. Elles appliquent de façon intuitive ce que leur apportent les notions ensemblistes. Elles commencent par reprendre les noms de niveaux que nous nous sommes donnés dans nos arborescences comme noms d'ensembles. Par exemple, en se référant à la *Figure 8, p. 23*, pour le niveau désigné par Produits, ils définissent :

Produits = {Whisky, Bière, Soda}

En représentation graphique, nous obtenons :

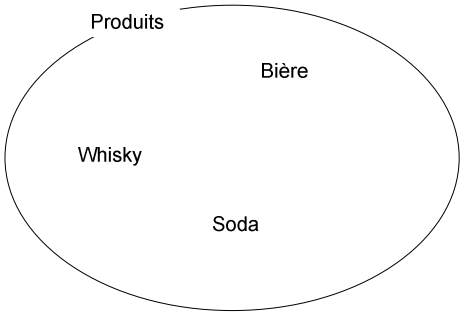


Figure 13. Ensemble des Produits

Ce que nous avons nommé dans nos arborescences des nœuds (Whisky, Bière, Soda) deviennent des éléments, les noms de niveaux deviennent des ensembles. Transformons notre représentation de l'ensemble Produits pour refléter l'arborescence de la *Figure 9, p. 24* :

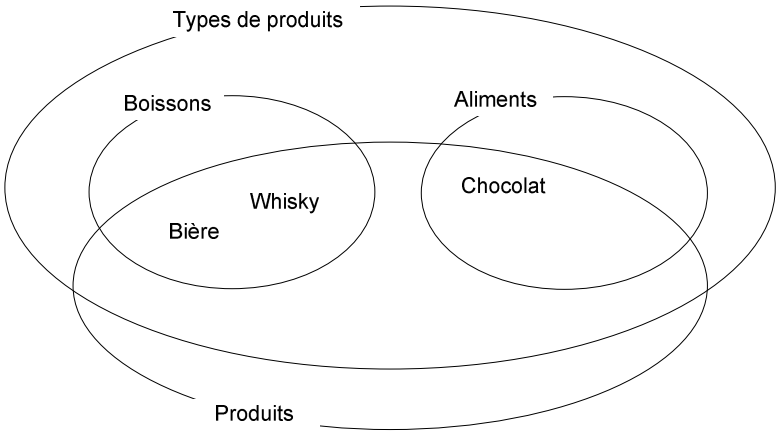


Figure 14. Ensembles des Types de produits et des Produits

[6] Pour approfondir le sujet, le lecteur pourra se tourner vers la littérature accessible à travers les mots-clés : « hyper-cube », « système multidimensionnel », « OLAP » et de façon un peu plus théorique, lire l'ouvrage de Gérard Lévy (LEVY, 1994).

Si comme auparavant les nœuds de notre arborescence Whisky, Bière et Chocolat deviennent des éléments, les nœuds Boissons et Aliments prennent le rôle de noms d'ensemble. Comme précédemment, les noms de niveau constituent aussi des noms d'ensemble. Si nous acceptons que la constitution d'un ensemble détermine une propriété sur les objets qui lui appartiennent, deux remarques intuitives s'imposent :

1) dans la représentation arborescente, on ne précise pas si un nœud désigne une propriété ou un objet. Par conséquent, si d'un point de vue syntaxique la définition d'un nœud peut rester implicite, elle peut conduire à des ambiguïtés sémantiques et par conséquent à des malfaçons. Dans le cadre de cette première remarque, il s'agit de comprendre comment le tableau de bord va calculer ses valeurs. Nous espérons obtenir respectivement une valeur pour le regroupement Boissons et pour le regroupement Aliments. Soit, deux valeurs calculées (la somme) à partir de la valeur attribuée à chaque élément. Quant au regroupement ou à l'ensemble Types de produits, c'est à partir de deux ensembles (Boissons et Aliments) que sera opérée la somme. Nous retenons ce manque d'harmonisation comme un risque pouvant conduire à des calculs erronés. Il s'agit donc de spécifier formellement le statut des entités considérées pour déterminer si nous avons à faire à des objets ou à des propriétés ;

2) dans la *Figure 14, p. 28* chaque élément est noté par trois propriétés (Types de produits, Boissons ou Aliments et Produits). L'introduction de ces propriétés n'apporte rien de significatif en ce qui concerne les calculs du tableau de bord, mais se présente comme des garde-fous en cas de *restructuration*. Il serait en effet peu cohérent de faire appartenir un nouvel élément comme le produit, Savon, à l'ensemble Aliments. Sachant qu'un « savon » n'est pas un des Aliments, il faudra mettre en œuvre des mécanismes qui interdisent d'introduire des objets qui ne respectent pas la propriété. Cette relation entre l'objet et ses propriétés doit aussi nous permettre de répondre en cas de suppression d'une propriété si les propriétés d'ordre supérieur sont maintenues. Cela revient à dire que si une *transformation* de structure devait conduire à supprimer la propriété Aliments, Chocolat resterait toujours un produit.

Tenant compte de ces deux remarques, nous postulons qu'il existe des formalismes qui peuvent nous préserver de ces malfaçons. Continuons à explorer la façon dont les *solutions* informatiques représentent les choses. A partir de :

Boissons = {Whisky, Bière}
Aliments = {Chocolat}
Type de Produits = Boisson ∪ Aliments
Produits = {Whisky, Bière, Chocolat}

Les spécialistes de l'élaboration des tableaux de bord vont construire une représentation qui s'inspire des notions ensemblistes. Ils forment un ensemble qu'ils appellent *dimension* et où ils se donnent une structure entre éléments de la *dimension*, soit, par exemple :

Dimension produits = {Tous les produits = Boissons + Aliments,
Boissons = Whisky + Bière, Aliments = Chocolat, Whisky, Bière, Chocolat}

Ce que nous pouvons traduire en représentation graphique de la façon suivante :

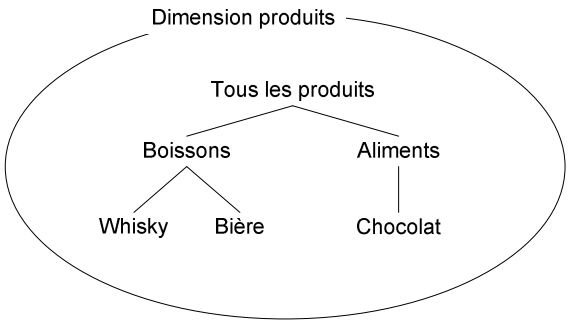


Figure 15. *Dimension* constituée d'une structure d'éléments

Tout en restant dans la théorie ensembliste, on se donne des relations entre éléments qui constituent une structure. Chaque lien exprime des notions comme : participe au calcul, participe à la consolidation, est un opérande du cumul et, de plus, hérite de propriétés d'ordre supérieur.

Nous pouvons maintenant appliquer cette représentation aux autres niveaux de notre arborescence (*Figure 9, p. 24*). Prenons la propriété, être un indicateur :

Dimension indicateurs = {Chiffre d'affaires, Dépenses,
Bénéfice = Chiffres d'affaires - Dépenses}

Cette nouvelle forme a l'avantage d'éviter la malfaçon que nous rencontrons à la *Figure 7, p. 23* où le Chiffre d'affaires est cumulé avec le Bénéfice.

Adaptons encore notre *dimension* temporelle :

Dimension périodes = {Année = 1er semestre + 2nd semestre, 1er semestre,
2nd semestre}

Souvenons-nous qu'un tableau de bord répond à des questions comme : « Quel est la valeur pour le chiffre d'affaires, pour l'année 2011 et pour les boissons ? ». Il ne nous reste plus qu'à exprimer la combinaison indiquée par le « pour » dans notre question. Les spécialistes font appel une fois encore aux notions ensemblistes en utilisant le produit cartésien des *dimensions* considérées :

Dimension indicateurs X Dimension produits X Dimension périodes

Dans notre cas, le produit cartésien génère cinquante-quatre[7] n-uplets[8] qui nous offrent l'ensemble des questions auxquelles le tableau de bord peut répondre. Voici quelques échantillons de cette liste :

<Chiffre d'affaires, Whisky, 1er semestre>
<Chiffre d'affaires, Whisky, 2nd semestre>
<Chiffre d'affaires, Whisky, Année>
<Chiffre d'affaires, Bière, 1er semestre>
<Chiffre d'affaires, Bière, 2nd semestre>
Etc.

A chacun de ces n-uplets, les *solutions* informatiques associeront soit la valeur saisie dans le logiciel, soit les valeurs qu'elles auront calculées :

<Chiffre d'affaires, Whisky, 1er semestre>> → 5 000
<Chiffre d'affaires, Whisky, 2nd semestre> → 5 500
<Chiffre d'affaires, Whisky, Année> → 10 500
Etc.

[7] Soit le produit des cardinaux des trois dimensions.

[8] Eléments de la liste de résultats d'un produit cartésien.

Un formalisme intuitif utilisant les notions ensemblistes est une démarche systématique qui permet d'éviter des calculs erronés, mais qui ne répond pas aux besoins de *restructuration* et au *maintien d'une propriété*. Nous nous sommes tourné vers une approche qui tient compte d'un raisonnement qui aborde la notion de propriété de façon plus formelle.

3.4 Logique des classes

La frontière entre théorie des ensembles et logique des classes est difficile à établir étant donnée la quantité d'ouvrages et d'articles qui se sont penchés sur les travaux de Cantor. Tout en sachant que définir cette frontière est une question d'interprétation des travaux y relatifs, nous choisissons comme différenciation que la logique des classes a pris naissance quand il s'est agi d'élaborer une axiomatique de la théorie des ensembles. Dans ce sens, certains grands noms comme John von Neumann, Kurt Gödel, Ernst Zermelo sont à l'initiative d'une axiomatisation[9] et annoncent l'arrivée de la logique des classes. La théorie des ensembles est plutôt du domaine des compétences des mathématiciens, alors que la logique des classes est plutôt du ressort des logiciens. Rappelons encore que parallèlement la logique classique, traitant des propositions, progresse vers une logique des prédicats. Théorie des ensembles et logique convergent, pour reprendre Jean-Blaise Grize (GRIZE J.-B. , 1967, p. 145) :

> On peut se demander si […] la logique des propositions ne fait pas double emploi avec celle des classes. Et nous verrons qu'en effet les deux systèmes ont une même structure abstraite. […] Les interprétations qu'on peut donner des formes syntaxiques sont aussi essentielles que ces formes elles-mêmes.

La logique des classes pour le domaine qui nous concerne, nous sert de tremplin vers la logique des prédicats.

3.5 Logique des prédicats

La logique des prédicats est une des logiques modernes standard du 1er ordre (où on ne quantifie que des *variables* d'objets). Comme dans la logique des propositions, une fonction propositionnelle de la logique des prédicats peut être vraie ou fausse. Par exemple, la *fonction logique*, construite à partir du prédicat unaire, a, être un Indicateur, et appliquée à une *variable* d'objet quelconque, x, permet de valider si la fonction propositionnelle ax est vraie ou fausse. Chaque fois que x prend pour valeur un des objets de notre domaine (Chiffre d'affaires, Dépenses, Bénéfice), la fonction est vraie. Donc, tous les x du domaine ont la propriété d'être un Indicateur. La logique des prédicats introduit un signe « tous »[10] qui porte le nom de *quantificateur* universel. La *quantification* est une condition nécessaire au traitement de nos propriétés. Elle informe une *expression* du *système* sur la portée d'un *quantificateur* et à quelles *variables* il s'applique (*variables liées*). Elle nous assure que tous les objets d'un domaine sont soumis à travers leurs propriétés à un même traitement. Par conséquent, la description (mentionnée comme *difficultés de taxonomie*) des objets et de leurs propriétés transposées en logique des prédicats, répond à notre besoin de cohérence du moins au niveau syntaxique.

La logique des prédicats ne permet pas de quantifier les nœuds des niveaux d'une arborescence (par exemple dans la *Figure 8, p. 23*). Cela nous conviendrait, pourtant, pour parler des propriétés communes à un même niveau. Mais, pour faire référence à Jean-Blaise Grize (GRIZE J.-B. , 1967, p. 224) :

> On remarquera […] que seules les variables d'objets peuvent être quantifiées. Cela signifie, si on interprète le calcul qu'il est possible de parler de « tous les objets tels que », mais pas de « toutes les propriétés telles que » […] ».

---

[9] On compte une dizaine d'axiomes. Les plus pertinents pour notre étude sont l'axiome de l'extensionalité (deux ensembles ayant les mêmes éléments sont égaux). Les schémas d'axiomes de compréhension et de remplacement (en se donnant n'importe quel ensemble E et n'importe quelle propriété P, il existe un ensemble dont les éléments de E vérifient P. En conséquence, pour n'importe quel ensemble E et n'importe quelle relation R, si R(a,b) et R(b,c) alors R(a,c), soit une proposition valide qui montre qu'il existe un ensemble qui contient les images de R à partir de E).

[10] …, tous les, pour tous, un quelconque…

Par conséquent, pour traiter la suppression d'un niveau dans nos arborescences, pour traiter le *maintien d'une propriété*, nous devons faire appel à un *système formel* plus puissant que la logique des prédicats qui soit capable de quantifier les propriétés.

3.6 Les logiques de Leśniewski

C'est la capacité d'un *système* de quantifier un *foncteur* (dans nos exemples un niveau d'arborescence constitué de propriétés) qui a retenu notre attention et a dirigé nos investigations vers les *SL*. Comme notre développement se consacre à ces logiques, nous ne nous penchons ici que sur certains aspects généraux. On compte trois *systèmes*, ordonnés dans la chronologie de leur élaboration :

1) la *méréologie* (théorie générales des ensembles) permet de traiter la relation de la partie au tout ;
2) l'*ontologie* (théorie des noms) nous offre tous le formalisme nécessaire pour traiter les aspects taxonomiques et les contrôles de cohérences lors de *restructurations* ;
3) la *protothétique* (théorie des thèses premières) apporte les fondations pour les deux *systèmes* précédents.

Grâce à l'énorme travail de compilation, d'explication, de traduction et de vulgarisation qu'ont réalisé Denis Miéville et son équipe[11], nous avons assis notre démarche sur les recherches de Leśniewski. Ce choix que doit valider notre développement explique la première partie de notre de titre : *Théories logiques de S. Leśniewski [et…].*

3.7 Choix d'un ou plusieurs systèmes

Comme préliminaire, nos investigations nous ont conduits à considérer plusieurs *systèmes* qui peuvent avoir la vocation de résoudre tout ou partie des difficultés que nous avons recensées. Voyons comment ces *systèmes* peuvent y répondre :

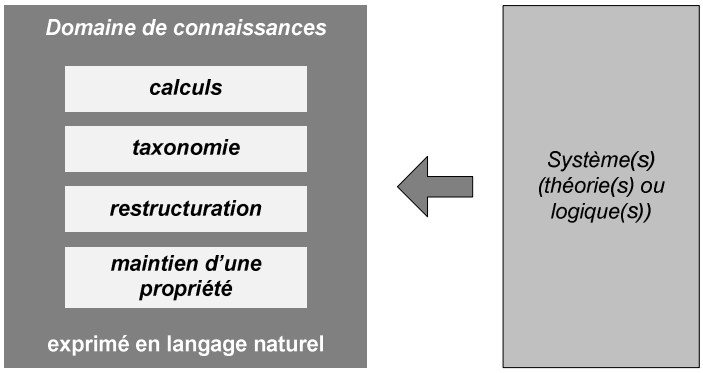


Figure 16. Classement des difficultés avec leurs solutions

Restent deux questions importantes : est-ce que certains *systèmes* en englobent d'autres ? Devons-nous traiter chacun de ces *systèmes* pour chacun de nos niveaux de questions ou pouvons-nous en choisir un seul ? En d'autres termes, quels sont les systèmes qui « parlent » d'un même *domaine de connaissances* ?

Nous avons vu que l'axiomatisation de la théorie des ensembles nous mène au niveau de la logique des classes. Nous pouvons, donc, considérer la logique des classes comme théorie qui « parle » des ensembles comme le fait la théorie des ensembles. En se référant à la citation que nous avons faite de Jean-Blaise Grize (section *3.4. Logique des classes, p. 31*), il y aurait une sorte de similitude des structures abstraites entre la logique des classes et la logique des

[11] Voir les travaux du Centre de Recherches Sémiologiques de l'Université de Neuchâtel et en particulier : (MIEVILLE, 2007), (MIEVILLE, 2004), (GESSLER, 2005) et (MIEVILLE, 2009).

propositions. En effet, elles ne sont pas constituées des mêmes connecteurs et des mêmes relations, mais leurs fonctionnement est similaire. En apportant des nouveaux *axiomes* et des nouveaux *termes* à la logique des propositions, nous obtenons la logique des prédicats en augmentant le *domaine de connaissances* et ainsi sa capacité d'en parler. Nous utilisons, plus loin dans notre raisonnement, la notion de *puissance* du système ou du langage. Nous savons aussi que la *méréologie* se fonde sur l'*ontologie* qui elle-même est construite sur la base de la *protothétique*.

Nous avons représenté dans la *Figure 17, p. 33* cette capacité des différentes théories et différents *systèmes* de parler d'un *domaine de connaissances*. Nous avons positionné les *systèmes formels* classiques du 1er ordre à la gauche des domaines qu'ils traitent et les *SL* d'ordre n à la droite.

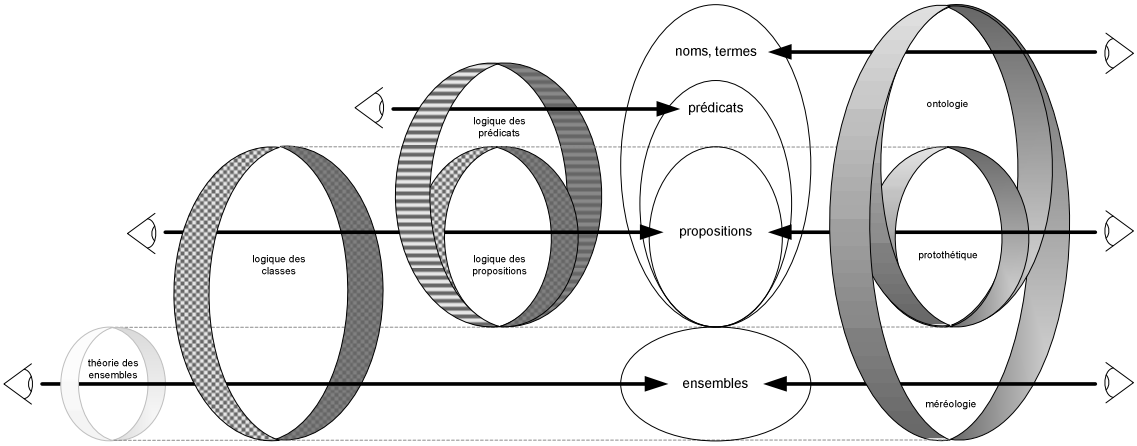


Figure 17. Représentation de la puissance des théories et des systèmes étudiés

Nous n'avons besoin que de la théorie des ensembles en ce qui concerne les questions de calculs des tableaux de bord et de l'*ontologie* pour le formalisme que nous recherchons pour traiter des *restructurations* et les *transformations* dans ce *domaine de connaissances*. Sachant que l'*ontologie* quantifie les *foncteurs* (les propriétés), nous pouvons réduire notre sélection à la théorie des ensembles et à l'*ontologie* :

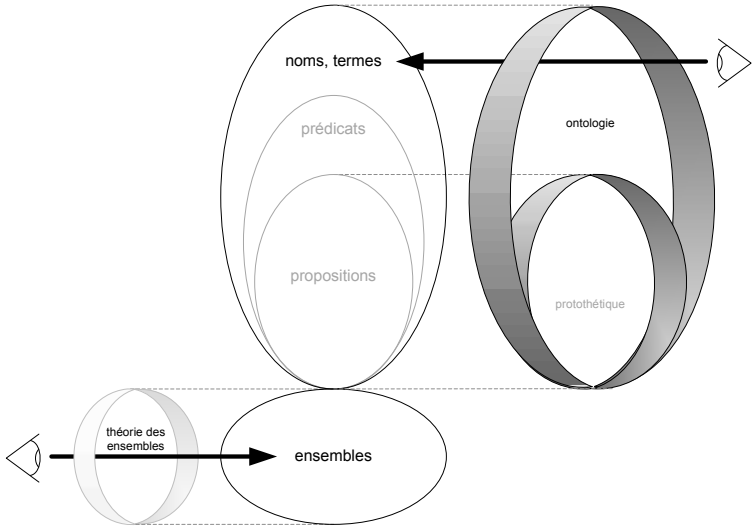


Figure 18. Sélection de la théorie des ensembles et de l'*ontologie*

Les aspects délicats des calculs que nous avons mentionnés dans le premier chapitre, comme l'addition du Chiffres d'affaires et du Bénéfice sont d'ordre sémantique. Ce que nous connaissons et qui est utilisé de la théorie des ensembles convient parfaitement à nos besoins d'un point de vue syntaxique. Par conséquent, nous ne développons pas ces aspects et transmettons la question sémantique au niveau suivant, *difficultés de taxonomie* (*Figure 16,*

p. 32). L'*ontologie* se présente comme le moyen de répondre à nos trois niveaux : *difficultés de taxonomie*, *difficultés de restructuration* et *difficultés du maintien d'une propriété*. Nous verrons aussi plus loin que les *difficultés du maintien d'une propriété*, qui nous concernent peuvent être interprétées par l'*ontologie* en insistant sur l'apport d'une *thèse-définition* concernant l'existence.

Par conséquent, notre développement va se porter sur l'*ontologie* de Leśniewski comme moyen de *formalisation* des tableaux de bord et toute extension à des *domaines de connaissances* similaires comme, par exemple, la taxonomie.

3.8 Systèmes évolutifs de connaissances

Par notre définition de *domaine de connaissances*, nous avons clarifié la notion de « *connaissances* » qui figure dans le titre de notre ouvrage : *Théories logiques de S. Leśniewski et systèmes évolutifs de connaissances*. Au cours de notre recherche d'un formalisme, nous avons précisé ce que nous entendions par *systèmes*. Reste à expliquer le terme : *évolutifs* du titre.

En reprenant notre exemple de tableau de bord, nous pouvons nous demander s'il existe un aliment « qui est une » boisson ou si une boisson « est un aliment ». La réponse à cette question est relativement difficile à donner sans obtenir plus de précisions sur la définition d'aliment et de boisson[12]. Si nous prenons pour aliment la notion de solide et pour boisson la notion de liquide, l'objet solide-liquide existe-t-il ? Le *domaine de connaissances* permet d'affirmer ou non l'existence de cet objet. Mais notre propos est ici de satisfaire d'une part à la levée d'ambiguïtés dans les tableaux de bord et, d'autre part d'accepter que l'évolution des affaires nécessite de créer, dans ce que le *système* connaît à un moment donné, des objets bizarres qui n'avaient pas d'existence (naturelle) auparavant ; par exemple, des solides-liquides du genre purée, yaourt, fromage frais. Pour cette raison le *système* doit être *évolutif*. Les *SL*, qui se construisent de façon développementale, répondent naturellement à cette exigence.

3.9 Conclusion

En explorant différents *systèmes* censés répondre à nos besoins de formalisme, nous avons montré que leurs propriétés intrinsèques en faisaient de bons candidats pour résoudre nos difficultés. Outre la théorie des ensembles, nous avons choisi, dans les *SL*, l'*ontologie*, puisqu'elle permet la *quantification* des propriétés. Comme elle trouve ses fondements dans la *protothétique*, nous exposons ces deux théories dans le chapitre suivant.

[12] Deux propriétés choisies à dessein.

4. Systèmes logiques de Leśniewski

Dans ce chapitre, nous présentons les bases qui font la spécificité des systèmes leśniewskiens comparés aux logiques formelles du 1er ordre. En évitant de paraphraser des travaux de grande envergure sur le sujet, nous insistons sur les aspects qui nous ont conduits à automatiser une partie des processus de transposition ou de traitement de notre domaine de connaissances à l'aide de ces logiques. Comme la protothétique est le fondement des autres théories, nous commencerons par la traiter pour nous arrêter ensuite sur l'ontologie.

En fin de chapitre, nous mettons en relief la frontière délicate à établir entre les actes de transpositions ou l'interprétation que l'on peut se donner du domaine de connaissances et les aspects purement théoriques des systèmes leśniewskiens. Nous terminerons sur la synthèse des transpositions et de la mise en œuvre des apports de ces logiques afin de préparer la suite de notre développement.

Stanislaw Leśniewski (1886-1939)
(repris de Wikipédia)

4.1 Préliminaire concernant les logiques formelles du 1er ordre

Pour ce qui suit, nous utilisons l'abréviation *SF* pour n'importe quel *système formel* classique, moderne, traditionnel, standard du 1er ordre, comme la logique des classes, la logique des propositions, la logique des prédicats. Comme nous l'avons mentionné dans notre introduction, nous employons *SL* pour mentionner n'importe lequel des *systèmes* logiques leśniewskiens.

Nous rappelons comment les *SF* se construisent afin de montrer l'originalité qu'apportent les *SL*. Pour comprendre le processus de *formalisation*, prenons la définition de Jean-Blaise Grize :

> Formaliser une théorie comporte deux étapes très différentes : Construire tout d'abord un système formel et lui donner ensuite une interprétation (GRIZE J.-B. , 1967, p. 168).

Nous en déduisons que la construction du *système* en tant que tel procède de la couche syntaxique et que l'interprétation apporte la dimension sémantique.

La construction de ces *systèmes* suit la procédure suivante :

1) les définitions de *signes* et de leurs natures ;
2) des *définitions inductives* qui décrivent comment ces *signes* seront manipulés pour créer des *termes* et/ou des *expressions bien formées (ebf) ;*

3) l'élaboration d'*axiomes ;*

4) l'élaboration de règles.

La mise en œuvre de ces quatre ingrédients et des processus de *déduction* ou de *preuve* permettent au *système* de s'enrichir ; c'est-à-dire de mettre en évidence des nouveaux *théorèmes*.

> Une *déduction* est une liste finie d'*ebf* telle que chacune est soit un axiome, soit résulte d'une ou de deux lignes qui la précèdent par l'application d'une des règles de conséquence immédiate.
>
> Un *théorème* est la dernière ligne d'une déduction[13]. […], tout axiome est un théorème. La déduction est alors de longueur 1 (GRIZE J.-B. , 1967, p. 172).

Eclairons de façon plus illustrée, les bases de cette construction.

4.1.1 Signes primitifs

Les *signes* sont les premiers constituants du *système*. Ils se divisent en ensembles regroupés par leurs natures, par exemple[14] :

(1) les lettres, 'p', 'q', 'r',… qui serviront de *variables propositionnelles* pour la logique des propositions ou 'a', 'b', 'c',… utilisées comme *variables de prédicats* pour la logique des prédicats ;

(2) les *signes* '∩', '∪' comme *opérateurs* dans la logique des classes, ou '∧', '⊃', '∨', '≡',… comme *opérateurs* pour la logique des propositions ;

(3) le *signe* '=' utilisé comme *foncteur*[15] dans la logique des classes ;

(4) les *signes* '(', ')', les parenthèses ou *signes* de ponctuation ;

(5) les *signes* '∀', '∃', les *quantificateurs* universel et existentiel dans la logique des prédicats.

4.1.2 Termes

Un *terme* est défini par une *définition inductive*. Par exemple, pour la logique des classes ou des propositions, en acceptant que '*' et '~' ont été défini comme *signes* interprétés comme *opérateurs* :

(1) Une *variable* est un *terme*.

(2) Si *A* est un *terme*, *~A* est un *terme*.

(3) Si *A* et *B* sont des *termes*, (*A*) * (*B*) est un *terme*.

(4) Rien n'est un *terme*, sinon par ce qui précède.

Certains *SF* ne nécessitent pas la définition de *termes* ; certains n'utilisent que celle d'*expressions bien formées (ebf)* ; d'autres utilisent les deux.

4.1.3 Expressions bien formées (ebf)

Par exemple, pour la logique des classes, la définition d'une *ebf* est aussi donnée par une *définition inductive* :

(1) Si *A* et *B* sont des *termes*, *A = B* est une *ebf*.

(2) Rien n'est une *ebf* sinon par (1).

---

[13] Un peu plus loin, nous avons adapté la définition de *déduction* de Jean-Blaise Grize en utilisant plutôt celle de *preuve* quand la dernière ligne est un *théorème.*

[14] Nous prenons quelques exemples issus des différents *systèmes formels* dans le but de couvrir un maximum de cas.

[15] Selon la littérature, on peut trouver comme synonymes : relateur, relation…

4.1.4 Axiomes

A titre d'exemple, nous reprenons la définition de Jean-Blaise Grize pour une partie de l'axiomatisation de la logique des prédicats (GRIZE J.-B. , 1967, p. 221) :

> Si *P* est un théorème du calcul des propositions et si *A* résulte de *P* en substituant une *ebf* du calcul des prédicats à chaque variable propositionnelle que contient un axiome *P, A* est un axiome.
>
> Exemples.
>
> 1. On a dans le calcul des propositions : $\vdash (p \wedge q) \supset p$. Il s'en suit que '$((\forall x)\, ax \wedge by) \supset (\forall x)\, ax$' est un axiome du calcul des prédicats.
> 2. De même, on a dans le calcul des propositions : $\vdash p \equiv \sim\sim p$. Donc, '$(\forall x)\,(ax \wedge rxy) \equiv \sim \sim (\forall x)\,(ax \wedge rxy)$' est un axiome du nouveau calcul.

Notons encore qu'un *axiome* est logiquement toujours le vrai ; quelle que soit son interprétation et quel que soit le monde sur lequel il est défini.

4.1.5 Règles

Les *règles d'inférences* participent à la cohérence des *déductions* et des *preuves*. On trouve certaines de ces règles qui contribuent à inscrire ou à construire des *déductions* et *preuves* dans différents *systèmes*, par exemple, la règle du modus ponens :

Si *A* et *B* sont des *ebf*. De *A* et $A \supset B$, on peut déduire *B*.

4.1.6 Déduction et preuve

Les constructeurs qui précèdent, soit les ensembles des *symboles*, des *termes*, des *expressions* ou formules et des *axiomes*, ainsi que des règles et des définitions permettent d'encadrer, de façon formelle, le processus intellectuel de *déduction* et de *preuve*. On trouve différentes interprétations de la notion de *déduction* et de *preuve* dans la littérature. Dans notre développement, nous utilisons les définitions suivantes :

4.1.6.1 La déduction

Une *déduction* se développe comme une liste ordonnée de lignes. La première partie est constituée d'un ensemble d'hypothèses, $\gamma = \{h_1, h_2, \ldots h_n\}$. Les lignes, qui constituent la deuxième partie, acceptent soit un *axiome*, soit une forme dérivée d'une règle et de ce qui précède, soit d'une des hypothèses ($h_1, h_2, \ldots h_n$). La dernière partie est une forme nommée : « conclusion syntaxique ». Soit la représentation :

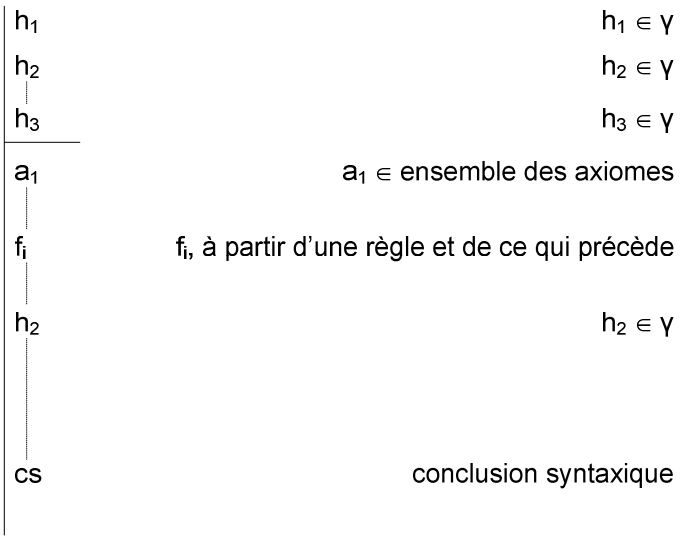


Figure 19. Schéma du processus de *déduction*

4.1.6.2 La preuve

La forme, E_1, de la première ligne d'une *preuve* appartient à l'ensemble des *axiomes*. Les lignes qui suivent peuvent prendre soit la forme, E_i, d'un *axiome*, soit une forme, E_j, résultante d'une règle et de ce qui précède. La dernière ligne, la forme E_n, est un *théorème*.

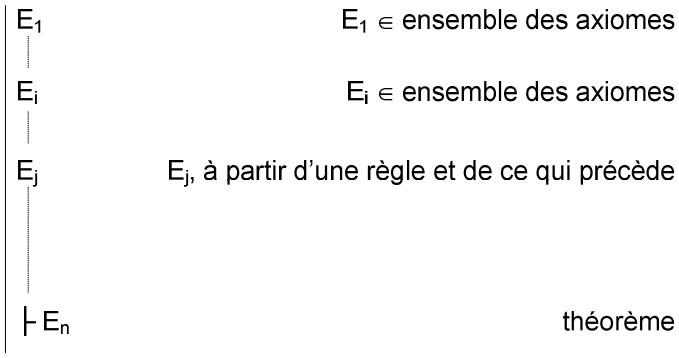


Figure 20. Schéma du processus de *preuve*

A partir de ces quelques rappels, dans une première étape, nous introduisons la notion de *transposition* et, dans une deuxième étape, nous présentons ce qui est nécessaire des *SL* pour notre démarche.

4.2 Transposition

Il convient d'aborder la notion de *formalisation*. Admettons que l'acte de *formaliser* un *domaine de connaissances* passe par la définition d'un métalangage qui permet de décrire le langage qui parle des objets et des leurs relations du *domaine de connaissances* considéré.

Soit le métalangage est construit de façon intuitive par observation des entités du domaine, soit de façon puriste, on définit, ex-nihilo, un *système formel*, un métalangage et on essaye de l'appliquer au domaine. Dans le cadre des *SF*, *formaliser* un *domaine de connaissances* passe par un procédé où l'on se donne une interprétation du *SF*. On parle dans cette approche d'un *système interprété* et quand il s'agit de considérer les logiques traditionnelles, la *transposition* se déroule en deux étapes :

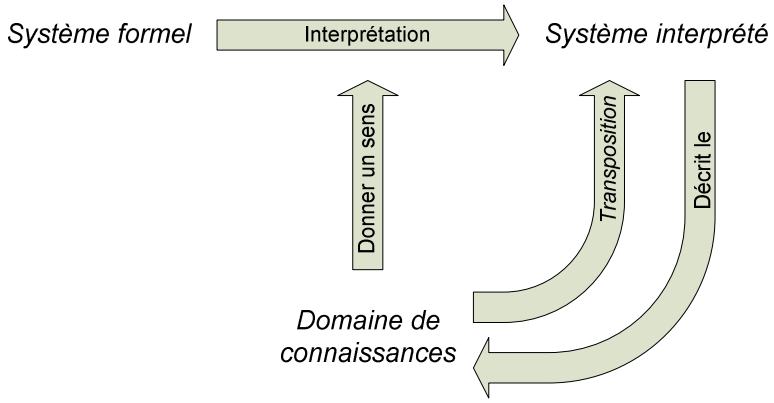


Figure 21. Transposition dans un *SF*

Notre recherche se fonde sur les *SL* qui sont des *systèmes* construits petit à petit comme des *systèmes interprétés* (qui tiennent compte des dimensions syntaxique et sémantique). Comme nous partons de *systèmes* existants dans le but d'utiliser la « mécanique » qu'ils nous offrent et qui nous assurent d'un certain nombre de propriétés comme la non-contradiction, nous devrons aussi procéder par une phase de *transposition*, de traduction, de description des « entités linguistique » constituant le *domaine de connaissances*. Pour parler de cette phase et de la liste des synonymes susmentionnés, nous utilisons le terme de *transposition*. Evidemment, ce

processus de *transposition* passe par un acte d'interprétation. Dans le cas des *SL*, comme ce sont des *systèmes interprétés*, cette phase de *transposition* procède par une seule étape :

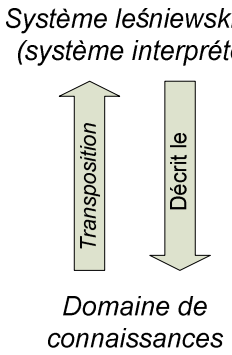


Figure 22. Transposition dans un *SL*

Nous devons affronter un certain espace de liberté quant à l'interprétation qui avec un manque de précaution peut mettre en cause les résultats obtenus. Dans notre développement, en aucun cas les *systèmes formels* sont mis en cause, si une anomalie devait apparaitre. Il s'agirait d'un manque de systématisme dans la *transposition* du *domaine de connaissances* vers le *système*. Exemple, soit une *transposition :* $\Rightarrow$ et une phrase du français : 'S'il pleut, alors on ouvre le parapluie' à transposer dans la logique des propositions :

F 4.1. 'Il pleut' $=_{df}$ p
'On ouvre le parapluie' $=_{df}$ q
'Si… alors…' $=_{df} \supset$

'S'il pleut, alors on ouvre le parapluie' $\Rightarrow$ p $\supset$ q

Le processus de *transposition* passe par une étape de définitions et une traduction dans le métalangage cible (dans le *SL*). A partir de là, on ne s'occupe plus que de la mécanique du métalangage cible, soit, dans notre cas, de la mécanique du *SL*.

4.3 Introduction aux systèmes logiques de Leśniewski

Voyons les principaux contrastes[16] entre les *SL* et les *SF :*

1) le but d'un *SL* est de fonder un calcul des propositions sur un seul *connecteur primitif*, la *biconditionnelle* '≡' ;

2) la mise en œuvre d'un *SL* n'établit pas de différence entre la construction du *système* et l'interprétation que l'on s'en donne. Elle suit un processus de construction développemental qui tient compte aussi bien de la dimension syntaxique que sémantique. Les *SL* sont des *systèmes interprétés ;*

3) à part quelques *signes* qui sont définis initialement, l'introduction de nouveaux *signes* ainsi que leurs natures se déterminent en cours de développement ;

4) si la notion de *variable* est similaire aux *SF*, les *SL* introduisent une notion très précise de *constante* (*termes constants*) ;

5) les *SL* intègrent une *bibliothèque* où seront inscrites les nouvelles *constantes* (*termes constants*) qui ont été définies, leurs *catégories syntaxico-sémantiques* et les *contextes* sur lequel elles s'appliquent durant la construction et qui sert de cadre de référence pour la formation des *inscriptions* suivantes ;

6) les *axiomes* et les *règles d'inférences* sont définis comme dans les *SF*. Les processus de *déduction* suivent eux aussi le même genre de mécanique.

[16] Un grand nombre de notions, mises dans le texte en italiques, seront définies en cours du développement et font partie de l'index en fin d'ouvrage.

En résumé, un *SL* se construit de façon développementale à partir d'un minimum de *signes*, d'*axiomes*, de *règles d'inférences* et des *thèses* que le *système* connaît à un moment donné de son développement. Les *SL* se développent selon deux axes parallèles.

1) Le premier enrichit le *système* par un processus définitoire :

   > Cette activité permet alors d'inscrire dans le système un terme constant nouveau en établissant une relation d'équivalence entre deux expressions (MIEVILLE, 2007, p. 45).

   D'un point de vue procédural, cet aspect du développement du *système*, passe par l'élaboration de *thèses-définition* à l'aide de la *règle d'inférences de définition* qui pilote la *procédure définitoire* et procède à l'inscription dans la *bibliothèque* de la nouvelle *constante*, de sa *catégorie syntaxico-sémantique* et du *contexte* auquel elle est associée. Il s'agit donc de :

   > Admettre qu'une définition doit apparaître comme une thèse (un théorème) du système (MIEVILLE, 2007, p. 46).

2) Le deuxième axe, procède de façon plus classique et comme les *SF*, à partir des *axiomes* et d'un ensemble de *thèses* ou de *thèses-définition*, les processus de *déduction* ou de *preuve* similaires à ceux que nous connaissons des *SF* s'appliquent pour engendrer des nouvelles *thèses* (ou des nouveaux *théorèmes*).

La construction et le développement d'un *système* leśniewskien n'ont pas de limite. A n'importe quel moment de son évolution, si les matériaux que le *système* connaît permettent une *déduction*, la *thèse* qui en résulte viendra enrichir le système.

A partir de ces généralités, nous allons parcourir les notions, les concepts et aspects théoriques que nous apporte la *protothétique*. Cela d'une part pour se préparer à comprendre l'*ontologie* et, d'autre part sous un angle qui nous permet d'automatiser certains aspects mécaniques de ces *systèmes*. Pour ce faire, nous proposons dans ce chapitre, de suivre une partie du cheminement que Denis Miéville (MIEVILLE, 2007) a élaboré au sujet des *SL* sans tenir compte des aspects pédagogiques.

4.4 Introduction à la protothétique

La *protothétique* de Leśniewski est un *système formel* qui peut être considéré comme la théorie des *thèses* premières. Comme nous l'avons introduit au paragraphe *1), p. 40*, une partie de la construction d'un *SL* passe par une *procédure définitoire* basée sur une équivalence entre deux *expressions*.

Leśniewski construit une définition explicite de forme :

F 4.2. $$\text{Definiendum} \leftrightarrow \text{definiens}$$

Le *Definiendum* est une *expression* qui contient le *terme* défini et le *definiens* est une *expression* qui offre la signification du *terme* défini. Pour préciser cette définition, on établit que :

F 4.3. $$\text{Definiendum} \leftrightarrow \text{ebf}$$

Le *definiens* est une *expression bien formée* (*ebf*). Et, on se donne le signe d'un *terme nouveau* '*' pour préciser le *Definiendum* :

F 4.4. $$*(\ldots) \leftrightarrow \text{ebf}$$

On introduit ensuite un jeu de *variables* :

F 4.5. $$*(v_1, v_2, \ldots, v_i) \leftrightarrow \text{ebf}(v_1, v_2, \ldots, v_i)$$

Sachant que dans le membre de gauche de l'équivalence les *signes* ne peuvent pas être répétés. Dans le membre de droite ils peuvent être répétés.

A partir de ces conditions, il faudra :

> […] structurer une procédure définitoire qui respecte ces conditions et qui se fonde sur une constante logique susceptible de fonder ce jeu entre *definiens* et *definiendum*. La réponse passe par une analyse de l'équivalence logique.
>
> Deux propositions *P* et *Q* possèdent entre elles la relation d'équivalence si et seulement si le résultat de la biconditionnelle qui leur est appliquée est une tautologie. Exprimons la chose de manière formelle (MIEVILLE, 2007, p. 46) :

F 4.6.

$$P \leftrightarrow Q =_{df} \vdash P \equiv Q$$

Avant de nous pencher sur la *procédure définitoire* et la construction d'un *SL*, il nous reste à étudier ce qui a conduit Leśniewski à s'orienter vers une syntaxe contextuelle (*version contextuelle*). En se référant à la section *4.1.1. Signes primitifs, p. 36,* essayons de comprendre pourquoi il a voulu se libérer de la nature *catégorielle* des *SF* :

> Dans un système classique, l'ensemble des signes est donné au départ, ainsi que la catégorie des entités qu'ils représentent. Ainsi, pour la logique des propositions, on aura une liste de signes pour la ponctuation, une liste de signes pour les opérateurs binaires, pour les opérateurs unaires, une liste de signes pour les variables propositionnelles. Un fois fixée, cette catégorisation est maintenue. La volonté d'attribuer à la définition un rôle inférentiel, et donc de proposer un système développemental, exige une approche qui permette l'adjonction de nouveaux signes, et cela sans contradiction ni ambiguïté. L'écriture […] qui se conformait à l'usage catégoriel des signes […] ne permet pas cet élargissement non ambigu de la syntaxe (MIEVILLE, 2007, p. 50).

Il faut noter que le processus de définition autorise la polysémie des *constantes*, soit la réutilisation d'un *signe* connu du *système*, mais associé à un nouveau *contexte*. Sans une syntaxe contextuelle (*version contextuelle*), cette polysémie ne peut pas être représentée. La notation utilisée par la syntaxe contextuelle (*version contextuelle*) est de type notation polonaise préfixée. Dès lors, l'*expression biconditionnelle primitive* prend la forme :

F 4.7.

$$\equiv (pq)$$

Cette forme nous permet d'apporter quelques définitions à des notions que nous avons déjà utilisées.

(a) La forme, constituée des *signes* de ponctuation : '(' et ')', est appelé le *contexte*. Dans la forme *F 4.7.,* le *contexte* primitif '(- -)' offre la place à deux arguments ; considération qui tient de la dimension syntaxique.

(b) Les *signes* 'p' et 'q' contenu dans le *contexte* sont deux *arguments propositionnels*, donc de la *catégorie* des propositions, *S*. La décision prise quant à la *catégorie* est d'ordre sémantique. On appelle aussi 'p' et 'q' des *termes variables* ou des *variables propositionnelles*.

(c) Le *signe* '≡' est le *terme constant* ; souvent abrégé *constante* ; ou encore :

> […] un foncteur constant formateur de proposition à deux arguments propositionnels, *S/SS* (MIEVILLE, 2007, p. 53).

$$\underbrace{\equiv (\overbrace{p}^{S}\ \overbrace{q}^{S})}_{S}$$

Figure 23. *Catégorie syntaxico-sémantique* de l'*expression biconditionnelle primitive*, *S/SS*

Ceci nous conduit à la *définition inductive* de toute *catégorie syntaxico-sémantique* en reprenant Denis Miéville (MIEVILLE, 2007, p. 65) :

F 4.8. (i) *S* est une *catégorie syntaxico-sémantique*.

F 4.9. (ii) Si C_1, C_2,…, C_n sont des *catégories* préalablement introduites, alors $C_1/C_2C_3…C_n$ est une *catégorie syntaxico-sémantique*. Des foncteurs constants d'une catégorie aussi complexes que celle-ci peuvent être introduits dans le système :

((S/SS)/S(S/S))/S(S/S) ((S/S)/S) S (S/S)

F 4.10. (iii) Rien n'est une catégorie sinon par ce qui précède.

Nous avons introduit ce qui est nécessaire pour rapporter et analyser la *procédure définitoire*. C'est une des cinq *règles d'inférences* de la *protothétique*, la *règle de définition.*

4.4.1 La règle d'inférences de définition de la protothétique

Nous reprenons cette *règle d'inférences de définition* de Denis Miéville (MIEVILLE, 2007, p. 76) :

RD 4.4-1 Une expression A est une thèse-définition si et seulement si toutes les conditions suivantes sont remplies :

RD 4.4-2 1. *A* est une généralisation $\lfloor \ldots \rfloor \lceil \ldots \rceil$ ou une fonction logique similaire à $\equiv(--)$.

RD 4.4-3 2. L'expression interne du sous-quantificateur de *A* est constituée du terme constant ≡ suivi d'un contexte primitif, $\lfloor \ldots \rfloor \lceil \equiv(--) \rceil$.

RD 4.4-4 3. Le premier argument de *A* ou du sous-quantificateur de *A* est une fonction logique, il constitue le *definiendum*. Il peut prendre la forme d'une inscription atomique, ∑. Il constituera alors une constante de la catégorie des propositions. Il peut prendre la forme d'une fonction logique régulière de la catégorie des propositions, $\sum(\ldots.)$[17]. Il peut également prendre la forme d'une fonction paramétrée de la catégorie des propositions, $\sum(\ldots.)\{\ldots.\}\ldots[\ldots.]$.

RD 4.4-5 4. Le foncteur de la fonction logique du *definiendum* est un terme constant. Ce terme constant ne saurait être équiforme à une constante logique de même catégorie syntaxico-sémantique qui appartient déjà au système.

RD 4.4-6 5. Si le foncteur constant du *definiendum* est destiné à être d'une catégorie syntaxico-sémantique qui appartient déjà au système, le contexte qui le suit doit être similaire au contexte préalablement fixé pour cette catégorie.

RD 4.4-7 6. Si le foncteur constant du *definiendum* est destiné à être d'une catégorie qui n'appartient pas encore au système, il est nécessaire d'inscrire un nouveau contexte de manière à éviter toute confusion. Cela signifie qu'on ne saurait choisir des parenthèses équiformes à un contexte préalablement inscrit et qui possède le même nombre d'arguments que le *definiendum*.

RD 4.4-8 7. Les arguments du contexte du *definiendum* sont des termes variables : $\lfloor v_1v_2\ldots v_i \rfloor \lceil \equiv(\sum(v_1v_2\ldots v_i) \text{-}) \rceil$.

RD 4.4-9 8. Aucun signe du *definiendum* n'est répété.

RD 4.4-10 9. Les termes variables libres du *definiendum* sont également inscrits dans le *definiens*, et réciproquement : $\lfloor v_1v_2\ldots v_i \rfloor \lceil \equiv(\sum(v_1v_2\ldots v_i) E_{v1v2\ldots vi}) \rceil$.

RD 4.4-11 10. Le *definiens* est une expression bien construite en fonction de ce que le système a préalablement développé. Il est soit une généralisation, soit une fonction logique.

RD 4.4-12 11. Dans la construction du *definiens* ou du *definiendum*, il est possible d'inscrire des termes variables d'une quelconque catégorie syntaxico-sémantique pour autant qu'un foncteur constant de cette catégorie ait été préalablement inscrit.

RD 4.4-13 12. L'expression *A* ne contient aucune variable libre.

[17] Les parenthèses de forme : ⦇ et ⦈ sont des métasignes.

La *règle d'inférences de définition* fournit non seulement toutes les conditions nécessaires à la construction d'une *thèse-définition*, mais aussi des notions et une terminologie propres à Leśniewski qui appellent quelques remarques :

Trois paires symétriques de parenthèses : '⌊ ⌋', '⌈ ⌉' et '( )' sont utilisées pour la mise en forme, respectivement, d'une *généralisation* et du *contexte primitif*. Le signe '≡' est introduit dans la règle et nommé : *terme constant*. Associé au *contexte primitif*, il introduit la notion de *fonction logique*. Des nouvelles paires symétriques de parenthèses peuvent être introduites dans une *thèse-définition*, par exemple : '[ ]', '{ }', ⦇⦈ et pourquoi pas : 'p q' (une forme de *contexte* construite de façon correcte). On comprend aisément que la forme symétrique de parenthèsage : '≡ ≡' n'est pas acceptable, ne permettant pas de déterminer quelle est la parenthèse ouvrante et quelle est la parenthèse fermante (dans une *expression* complexe). Ces quelques considérations déterminent deux types de *signes* : les *signes* ayant une symétrie axiale et ceux qui sont asymétriques. Dès lors, une paire symétrique de parenthèses (constituant un *contexte*) ne peut être construite qu'avec deux *signes* asymétriques (dont l'un est le reflet de l'autre). Dans la règle, on ne précise pas quelle est la forme des *signes* utilisés comme *terme constant* et *terme variable*. En ce qui concerne les *termes constants* de la *protothétique*, Lesniewski a été jusqu'à coder graphiquement les seize *termes constants* binaires (connecteurs binaires dans les *SF*). Certains des signes résultants de ce codage étant symétriques d'autres pas. En voici la table, tenant compte de l'ordre canonique usuel des tables de vérité :

○	⊙	-○	○-	○	-⊙	⊙-	⊙	-○-	-○	○-	-⊙-	⊙	⊙	-⊙	-⊙-
0	1	0	0	0	1	1	1	0	0	0	1	1	1	0	1
0	0	1	0	0	1	0	0	1	1	0	1	1	0	1	1
0	0	0	1	0	0	1	0	1	0	1	1	0	1	1	1
0	0	0	0	1	0	0	1	0	1	1	0	1	1	1	1

Figure 24. Exemple de codage graphique des *termes constants* binaires

Les quatre symboles : ⊙, ○, -○- et -⊙- présente une symétrie sur les deux axes (vertical et horizontal) ainsi qu'une table de vérité symétrique. Nous verrons dans la construction de la *protothétique* que ce sont les *termes constants* qui peuvent être définis sans passer par la quantification des *foncteurs*.

D'un point de vue de la terminologie, nous pouvons retenir que :

1) la notion de *généralisation* ne pose aucun problème en ce qui concerne la construction syntaxique d'une *thèse-définition* : ⌊…⌋ ⌈…⌉. Il est toutefois important de souligner que la condition qui accepte qu'une *thèse-définition* peut être une *fonction logique* : ≡(--) est liée la condition 12 de la règle, à savoir que tous les *termes variables* doivent être *quantifiés*. Par conséquent, les arguments d'un *contexte primitif* d'une *fonction logique* de ce type ne peuvent pas être construits n'importe comment ;

2) le *sous-quantificateur* est constitué d'une *expression biconditionnelle primitive* (entre *definiendum* et *definiens*) ;

3) le schéma syntaxique (formé des métasignes) : ◊(- -) est une *fonction logique*, un *terme constant* suivi d'un *contexte* où le *signe* ◊ est, donc, un *foncteur* ou un *terme constant* ;

4) les arguments des *contextes* sont des *termes variables*. Rien n'est spécifié quant la forme des *signes* utilisés sinon qu'ils ne peuvent pas être des *signes* préalablement utilisés par le *système* pour des *termes constants* ou des parenthèses. L'usage montre que ce sont des caractères de l'alphabet latin ou grec qui sont utilisés pour cette fonction.

Avec l'apport de la *règle d'inférences de définition*, nous avons maintenant tous les éléments nécessaires pour procéder à l'introduction de *thèses-définition* dans le *système*.

Avant de se lancer dans l'étude de la construction du *système*, il nous reste à définir les propriétés du *connecteur primitif*, de la *biconditionnelle*. Cette construction se fera en deux temps :

(1) l'introduction des *thèses-définition* où la *quantification* ne porte que sur les *variables propositionnelles* et qui ne nécessite que les deux premiers *axiomes* de la *protothétique ;*

(2) l'introduction des *thèses-définition* où la *quantification* porte non seulement sur les *variables propositionnelles*, mais aussi sur les *foncteurs* et qui nécessite un troisième *axiome.*

4.4.2 Les deux premiers axiomes de la protothétique

Voici les deux premiers *axiomes* de la *protothétique*, A1 et A2 :

F 4.11. A1 : ⌊pqr⌋⌈≡(≡(≡(pr)≡(qp))≡(rq))⌉

F 4.12. A2 : ⌊pqr⌋⌈≡(≡(p≡(qr))≡(≡(pq)r))⌉

Il convient de noter qu'un *SL* prend intérêt au moment où on a introduit :

(1) un ou plusieurs *axiomes*,

(2) dans la *bibliothèque*, ce que nous avons vu concernant l'*expression biconditionnelle primitive*, à savoir :

Catégorie	Constante	Contexte	Table de vérité
S/SS	≡	(- -)	1001[18]

(3) et que l'on dispose de la *règle d'inférences de définition*.

Par conséquent nous pouvons maintenant mettre en œuvre la *procédure définitoire*.

4.4.3 Premier temps de la construction de la protothétique

Nous n'entrerons pas dans les détails de la génération des *thèses-définition* de la *protothétique* puisque notre objectif est la *formalisation* de nos exemples de tableau de bord à l'aide de l'*ontologie*, dont nous analyserons sa construction plus en détail. Nous rappelons que la *protothétique* sert de fondement pour l'*ontologie*.

Sachant qu'à n'importe quel moment, la construction du *système* peut s'arrêter ou continuer à s'enrichir, voyons dans la figure suivante le processus d'enrichissement :

[18] La table de vérité est donnée en fonction de l'ordre canonique usuel.

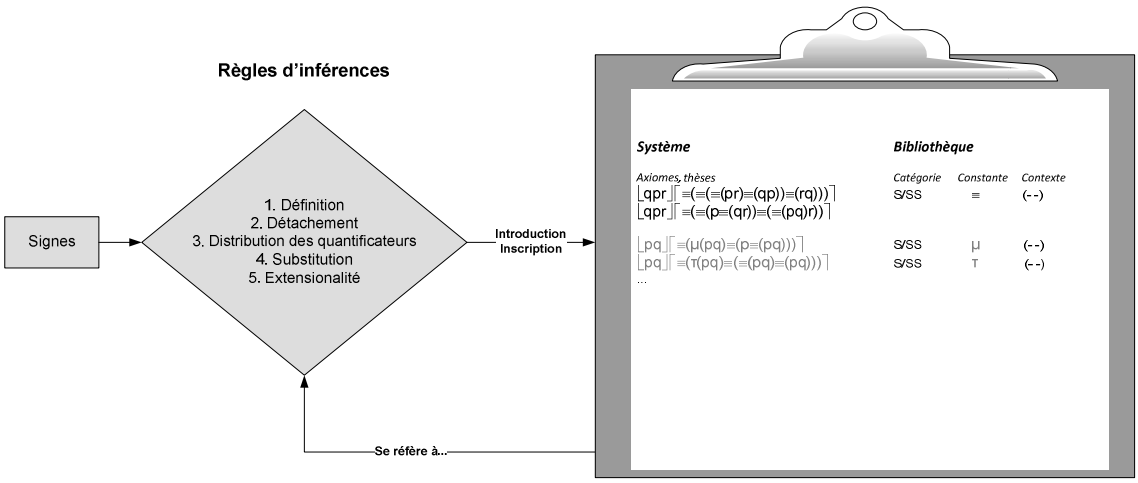


Figure 25. Processus de construction d'un *SL*

Dans le losange des *règles d'inférences*, nous avons, tout d'abord, mentionné la *règle d'inférences de définition* qui gouverne la *procédure définitoire* et permet d'introduire à tout moment une nouvelle *thèse* dans le *système*. Les quatre autres *règles d'inférences* permettent aussi d'introduire, à l'aide de ce qui précède (*axiomes* et *thèses*), des nouvelles *thèses*. L'introduction d'*inscriptions*, à l'aide de ces cinq règles, permet de procéder à des *déductions* ou à des *preuves*. Le blocs-notes illustre ce que le *système* acquiert et qui servira de référence pour une étape d'enrichissement ultérieure.

Voici la liste des *thèses-définition* introduite dans le premier temps et qui suivent strictement la *règle d'inférences de définition* :

F 4.13. D1, affirmation du conséquent :

⌊pq⌋⌈≡(μ(pq)≡(p≡(pq)))⌉

Catégorie	Constantes	Contexte	Table de vérité[19]
S/SS	≡, μ	(- -)	1010

F 4.14. D2, tautologie binaire :

⌊pq⌋⌈≡(τ(pq)≡(≡(pq)≡(pq)))⌉

Catégorie	Constantes	Contexte	Table de vérité
S/SS	≡, μ, τ	(- -)	1111

F 4.15. D3, identité / affirmation de l'antécédent :

⌊p⌋⌈≡(α(p)≡(p⌊q⌋⌈≡(qq)⌉))⌉

Catégorie	Constante	Contexte	Table de vérité
S/S	α	(-)	1100

F 4.16. D4, négation :

⌊p⌋⌈≡(~(p)≡(p⌊q⌋⌈q⌉))⌉

Catégorie	Constantes	Contexte	Table de vérité
S/S	α, ~	(-)	00

F 4.17. D5, ou exclusif :

⌊pq⌋⌈≡(ω(pq)¬(≡(pq)))⌉

[19] La table de vérité est donnée en fonction de l'ordre canonique usuel, à savoir : val(p) = 1100 et val(q)=1010.

Catégorie	Constantes	Contexte	Table de vérité
S/SS	≡, μ, τ, ω	(- -)	0110

F 4.18. D6, tautologie unaire :

⌊p⌋⌈≡(τ(p)≡(pp))⌉

La redéfinition de τ permise pour une nouvelle *catégorie* puisqu'elle est appliquée sur un *contexte* différent de *F 4.14.* C'est un exemple de polysémie du *terme constant* τ.

Catégorie	Constantes	Contexte	Table de vérité
S/S	α, ~, ¬, τ	(-)	11

F 4.19. D7, (sans interprétation) :

⌊pqr⌋⌈≡(δ(pqr)≡(r~(≡(pq)))⌉

Catégorie	Constantes	Contexte	Table de vérité
S/SSS	δ	(- - -)	01101001

F 4.20. D8, (*fonction logique* définissant la *constante logique* : faux) :

≡(F⌊p⌋⌈p⌉)

Catégorie	Constantes	Contexte	Table de vérité
S	F		0

F 4.21. D9, (*fonction logique* définissant la *constante logique* : vrai) :

≡(V~(F))

Catégorie	Constantes	Contexte	Table de vérité
S	F, V		1

F 4.22. D10, (*fonction logique* définissant la *constante logique* : vrai / tautologie) :

≡(T⌊p⌋⌈≡(pp)⌉)

Catégorie	Constantes	Contexte	Table de vérité
S	F, V, T		1

F 4.23. D11, (ou exclusif appliqué à deux *contextes*) :

⌊pqr⌋⌈≡(ω⟨pq⟩(r)≡~(≡(pq))r))⌉

La redéfinition de ω est permise pour une nouvelle *catégorie* puisqu'elle est appliquée sur des *contextes* différents de *F 4.17.* C'est un deuxième exemple de polysémie, celle du *terme constant* ω.

Catégories	Constantes	Contexte
(S/S)/SS	ω	⟨- -⟩

Cette nouvelle *thèse-définition* introduit non seulement un *terme constant* de la *catégorie* (S/S)/SS associée au *contexte* ⟨- -⟩, mais aussi la « sous-catégorie » S/S où le *terme constant* est de la forme : ω⟨- -⟩ associé au *contexte* (-) :

S/S	α, ~, ¬, τ, ω⟨- -⟩	(-)

F 4.24. D12, (ou exclusif appliqué à trois *contextes*) :

⌊pqr⌋⌈≡(ω[p]⟨q⟩(r)≡(~(≡(pq))r))⌉

La redéfinition de ω est permise pour une nouvelle *catégorie* puisqu'elle est appliquée sur des *contextes* différents de *F 4.14. et de F 4.23.* C'est un troisième exemple de polysémie, celle du *terme constant* ω.

Catégories	Constantes	Contexte
((S/S)/S)/S	ω	[-]

Comme pour la *thèse-définition* D11, en fait, deux « sous-catégories » sont introduites :

S/S	ω [-]⌊-⌋	(-)
(S/S)/S	ω [-]	⌊-⌋

Parmi les douze *inscriptions* qui viennent enrichir le *système*, trois sont des *fonctions logiques* qui définissent le « vrai » et « le faux » ; les autres sont des *thèses-définition* qui introduisent des nouveaux *termes constants*. A cet état, le *système* connaît en particulier : la négation (unaire), la disjonction exclusive (binaire), la tautologie (unaire et binaire). Il serait intéressant qu'il dispose de *termes constants* interprétés comme la conjonction, la conditionnelle et la disjonction inclusive. A ce sujet, Denis Miéville constate :

> Malgré cette richesse potentielle, un système de la logique des propositions quantifiées fondé sur l'unique foncteur constant de la biconditionnelle, et disposant notamment comme règle d'inférence la procédure définitoire […] ne donne pas accès à certains foncteurs fondamentaux tels que la conjonction propositionnelle ou la conditionnelle. Leśniewski va résoudre le problème en s'appuyant sur les travaux de Tarski (MIEVILLE, 2007, p. 65).

C'est à ce sujet qu'un des grands pivots des *SL* fait surface : la *quantification* des *foncteurs*.

4.4.4 La quantification des foncteurs

L'objectif de notre travail n'est pas ici de compiler une littérature importante dédiée à ce sujet[20]. Denis Miéville (MIEVILLE, 2007, pp. 66-67) présente les étapes qui ont conduit Tarski et Leśniewski au troisième *axiome* de la *protothétique* et à la définition de la conjonction. Voici tout d'abord le *théorème* de Tarski qui a conduit au raisonnement (MIEVILLE, 1984) :

F 4.25. Théorème de Tarski :

⌊pq⌋⌈≡(∧(pq)⌊f⌋⌈≡(p ≡(⌊r⌋⌈≡(p f(r)⌉⌊r⌋⌈≡(q f(r)⌉))⌉)⌉

Remarquons que le *contexte* du *foncteur* f est défini avec un seul *argument propositionnel*. Introduisons maintenant le troisième *axiome* de la *protothétique* « le principe d'ambivalence exprimé à l'aide de la *biconditionnelle* et des *variables de foncteurs* à deux arguments » :

F 4.26. A3 :

⌊pg⌋⌈≡(⌊f⌋⌈≡(g(pp) ≡(⌊r⌋⌈≡(f(rr) g(pp))⌉⌊r⌋⌈≡(f(rr) g(≡(p⌊q⌋⌈q⌉) p))⌉))⌉⌊q⌋⌈g(pq)⌉)⌉

Les *foncteurs* f et g sont construits avec un *contexte* à deux *arguments propositionnels* qui nous font entrer dans un *système formel* du 2ème ordre (*quantification* des *foncteurs* à deux arguments). Introduisons maintenant une *thèse-définition* qui nous prépare à la définition de la conjonction :

F 4.27. D13, préparation à la conjonction propositionnelle :

⌊pqg⌋⌈≡(Δ[gpq]≡(p~(g(pq))))⌉

Catégorie	Constantes	Contexte
S/(S/SS)SS	Δ	[- - -]

Reste à comprendre comment interpréter les *foncteurs*. Tout comme pour les *variables propositionnelles*, où l'on détermine une table de vérité, pour envisager un calcul propositionnel, nous devrons aussi nous donner les moyens d'interpréter les *foncteurs* pour

[20] Si le lecteur veut approfondir le sujet, on peut citer en particulier : TARSKI, Alfred, 1923, Sur le terme primitif de la logistique, in *Fundamenta Mathematica*, p. 196-200. Pour plus de détails, on peut se référer à la bibliographie de Denis Miéville sur Tarski.

calculer le « vrai » ou le « faux ». Nous aborderons ce sujet dans la section *4.6 Les interprétations de foncteurs et matrices de* vérités*, p. 70.*

4.4.5 Deuxième temps de la construction de la protothétique

Nous pouvons maintenant poursuivre la construction de la *protothétique* en y introduisant des nouvelles *thèses-définition* :

F 4.28. D14, conjonction propositionnelle :

⌊pq⌋⌈≡(∧(pq)⌊f⌋⌈≡(p≡(f(p)f(q)))⌉)⌉

Catégorie	Constantes	Contexte	Table de vérité
S/SS	≡, μ, τ, ω, ∧	(- -)	1000

F 4.29. D15, conditionnelle :

⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉

Catégorie	Constantes	Contexte	Table de vérité
S/SS	≡, μ, τ, ω, ∧, ⊃	(- -)	1011

F 4.30. D16, disjonction non-exclusive :

⌊pq⌋⌈≡(∨(pq)~(∧(~(p)~(q))))⌉

Catégorie	Constantes	Contexte	Table de vérité
S/SS	≡, μ, τ, ω, ∧, ⊃, ∨	(- -)	1110

Nous allons introduire maintenant quelques *thèses-définition* qui vont nous servir pour la *directive d'extensionalité* :

F 4.31. D17, introduction de la *catégorie* pour la condition 11 de la *directive d'extensionalité* :

⌊A⌋⌈≡(E<A>⌊q⌋⌈A(q)⌉)⌉

Catégorie	Constante	Contexte
S/(S/S)	E	<->

F 4.32. D18, introduction d'une deuxième *catégorie* pour la *directive d'extensionalité* :

⌊A⌋⌈≡(E⟨A⟩⌊pq⌋⌈A(pq)⌉)⌉

Catégorie	Constante	Contexte
S/(S/SS)	E	⟨-⟩

F 4.33. D19, introduction d'une troisième *catégorie* pour la *directive d'extensionalité* :

⌊A⌋⌈≡(E{A}⌊pq⌋⌈A{p}(q)⌉)⌉

Catégorie	Constante	Contexte
S/((S/S)/S)	E	{-}

La construction du *système* peut continuer. Voici l'état de la *bibliothèque* à ce niveau du développement du *système*[21] :

Catégories	Constantes	Contextes
S/SS	≡, μ, τ, ω, ∧, ⊃, ∨	(- -)
S/S	α, ~, ¬, τ, ω⟨- -⟩, ω [-]⟨-⟩	(-)
S/SSS	δ	(- - -)
S	F, V, T	

[21] La polysémie des *termes constants* apparaît de façon claire dans cette présentation de la *bibliothèque.*

(S/S)/SS	ω	⎝- -⎠
(S/S)/S	ω [-]	⎝-⎠
((S/S)/S)/S	ω	[-]
S/(S/SS)SS	Δ	[- - -]
S/(S/S)	E	<->
S/(S/SS)	E	⎝-⎠
S/((S/S)/S)	E	⎨-⎬

Ceci termine notre présentation de la construction de la *protothétique* à travers la *procédure définitoire*. Abordons maintenant les éléments nécessaires à la *déduction* ou aux *preuves*.

4.4.6 Les quatre autres règles d'inférences

A part la *règle d'inférences de définition*, on compte quatre *règles d'inférences :*

(1) la *règle d'inférences de détachement ;*

(2) la *règle d'inférences de distribution des quantificateurs ;*

(3) la *règle d'inférences de substitution ;*

(4) la *directive d'extensionalité.*

Dans la suite de notre développement, nous envisageons d'apporter des outils automatisés d'aide à la *déduction* et parce que nous nous référons à ces règles, nous les présentons dans ce qui suit.

4.4.6.1 La règle d'inférences de détachement

Elle rappelle le modus ponens[22] des *SF* (*4.1.5. Règles, p. 37*). Reprenons la règle exprimée par Denis Miéville (MIEVILLE, 2007, p. 69) :

RD 4.4-14 Soit B : ≡(- -), la fonction logique biconditionnelle primitive. Tout premier argument de B sera dorénavant appelé **la première équivalence de B**. Tout deuxième argument de B sera dorénavant appelé **la deuxième équivalence de B**.

L'inscription *A* est un théorème résultant de la règle de détachement relativement à deux thèses *B* et *C* préalablement inscrites dans le système si et seulement si :

1. *C* est équiforme à la première équivalence de *B*.

2. *A* est équiforme à la deuxième équivalence de *B*.

Illustrons cette règle par un exemple (MIEVILLE, 2007, p. 88) :

F 4.34. ⌊qr⌋⌈≡(≡(r≡(qr))≡(≡(rq)r))⌉ C

F 4.35. ≡(⌊qr⌋⌈≡(≡(r≡(qr))≡(≡(rq)r))⌉⌊qr⌋⌈≡(≡(qr))≡(rq))⌉) B

F 4.36. ⌊qr⌋⌈≡(≡(qr))≡(rq))⌉ A

4.4.6.2 La règle d'inférences de distribution des quantificateurs

Voici la règle telle qu'elle est exprimée par Denis Miéville (MIEVILLE, 2007, p. 69) :

RD 4.4-15 Tout inscription dans un quantificateur d'une généralisation est appelé **lieur**.

Soit *E* une thèse préalablement inscrite dans le système.

Si

1. L'inscription *E* est une généralisation sans variable libre, $\lfloor a\ldots z\rfloor\lceil\Sigma_{a\ldots z}\rceil$

2. Le sous-quantificateur de *E* est une fonction logique biconditionnelle primitive, $\lfloor a\ldots z\rfloor\lceil\Delta_{a\ldots m}\,K_{n\ldots z}\rceil$

[22] Voir définition chez Jean-Blaise Grize (GRIZE J.-B. , 1972, p. 13).

alors

la règle de distribution peut être appliquée. Cette transformation consiste en la distribution de tous, ou partie des lieurs du quantificateur de *E* dans un quantificateur qui précède la première équivalence du sous-quantificateur de de *E* d'une part, et dans un quantificateur qui précède la deuxième équivalence du sous-quantificateur de *E* d'autre part.

Par exemple, soit à partir de la *généralisation* :

F 4.37. ⌊pq⌋⌈≡(≡(pq)≡(qp))⌉

Et, en appliquant toutes les possibilités que nous offre la *règle* de *distribution des quantificateurs* :

F 4.38. (1) ⌊p⌋⌈≡(⌊q⌋⌈≡(pq)⌉⌊q⌋⌈≡(qp)⌉)⌉
(2) ⌊q⌋⌈≡(⌊p⌋⌈≡(pq)⌉⌊p⌋⌈≡(qp)⌉)⌉
(3) ≡(⌊pq⌋⌈≡(pq)⌉⌊pq⌋⌈≡(qp)⌉)

4.4.6.3 La règle d'inférences de substitution

Extrait des travaux de Denis Miéville (Miéville, 2007, p. 72) :

RD 4.4-16 *A* est une thèse résultant de la règle de substitution relativement à une thèse *B* préalablement inscrite dans le système, si et seulement si :

RD 4.4-17 1. *B* est une généralisation.

RD 4.4-18 2. A chaque variable équiforme de *B* qui possède un lieur dans le quantificateur de *B*, une expression *E* équiforme est substituée.

RD 4.4-19 3. L'expression *E* est de la même catégorie syntaxico-sémantique que la variable à laquelle elle est destinée à être substituée.

RD 4.4-20 4. L'expression substituée à la variable doit rester libre pour cette variable dans le sous-quantificateur de la thèse *B*.

RD 4.4-21 5. Le quantificateur de *A* contient des lieurs équiformes à toutes les variables libres de son sous-quantificateur, et à celles-là seulement.

RD 4.4-22 6. Si une inscription destinée à avoir statut de variable est substituée à une variable, cette inscription ne peut pas être équiforme à une constante préalablement inscrite dans le système, ni à un délimiteur, ni à une parenthèse préalablement inscrite.

RD 4.4-23 7. Une expression destinée à être substituée à une variable peut être de diverses natures :

7.1. un terme constant ;
7.2. un terme variable ;
7.3. une fonction logique à termes variables ou constants.

Par exemples (MIEVILLE, 2007, p. 73 et suivantes) :

D5/B : ⌊pq⌋⌈≡(ω(pq)¬(≡(pq)))⌉
E : m (qui respecte 6., m est libre pour p, *catégorie* des propositions …)
S1/A : ⌊mq⌋⌈≡(ω(mq)¬(≡(mq)))⌉

D5/B : ⌊pq⌋⌈≡(ω(pq)¬(≡(pq)))⌉
E : V (qui est un *terme constant*)
S2/A : ⌊q⌋⌈≡(ω(Vq)¬(≡(Vq)))⌉

S2/B : ⌊q⌋⌈≡(ω(Vq)¬(≡(Vq)))⌉
E : V (qui est un *terme constant*)
S3/A : ≡(ω(VV)¬(≡(VV)))

D4/B : ⌊p⌋⌈≡(¬(p)≡(p⌊q⌋⌈q⌉))⌉
E : ≡(xy) (même *catégorie* que p)
S4/A : ⌊xy⌋⌈≡(¬(≡(xy))≡(≡(xy)⌊q⌋⌈q⌉))⌉

D4/B : ⌊p⌋⌈≡(¬(p)≡(p⌊q⌋⌈q⌉))⌉
E : ≡(FF) (*fonctions logiques* dont les arguments sont des *termes constants*)
S5/A : ≡(¬(≡(FF))≡(≡(FF)⌊q⌋⌈q⌉))

D4/B : ⌊p⌋⌈≡(¬(p)≡(p⌊q⌋⌈q⌉))⌉
E : f(Vd) (de la cat. S/SS, où V est cat. S, d est un *terme variable* de la cat. S)
S6/A : ⌊fd⌋⌈≡(¬(f(Vd))≡(f(Vd)⌊q⌋⌈q⌉))⌉

4.4.6.4 La directive d'extensionalité

La *directive d'extensionalité* apporte aux *systèmes* leśniewskiens tout leur caractère. Elle se base sur l'affirmation :

> […] si deux formes propositionnelles sont formellement équivalentes, alors ce que l'on peut dire de l'une est logiquement équivalent à ce que l'on peut dire de l'autre (MIEVILLE, 2007, p. 79).

Pour la *catégorie* des propositions, *S*, Leśniewski part du schéma[23] de *thèse* :

F 4.39. ⌊gpq⌋⌈⊃(≡(pq)≡(g(p)g(q)))⌉

pour en dériver le principe d'extensionalité :

F 4.40. ⌊fg⌋⌈≡(⌊pq⌋⌈≡(f(pq) g(pq))⌉⌊ψ⌋⌈≡(ψ(f) ψ(g))⌉)⌉

Voici cette règle extraite du fascicule de Denis Miéville (MIEVILLE, 2007, p. 81) :

RD 4.4-24 L'inscription *A* est une thèse résultant de la directive d'extensionalité relativement à ce que contient actuellement le système, si et seulement si :

RD 4.4-25 1. L'inscription *A* est une généralisation. ⌊…⌋⌈…⌉

RD 4.4-26 2. Le sous-quantificateur de *A* est une fonction logique biconditionnelle. ⌊…⌋⌈≡(- -)⌉

RD 4.4-27 3. La première équivalence, respectivement la deuxième équivalence, du sous-quantificateur de *A* est une généralisation. ⌊…⌋⌈≡(⌊…⌋⌈…⌉⌊…⌋⌈…⌉)⌉

RD 4.4-28 4. Le quantificateur de *A* contient deux lieurs qui sont équiformes à des variables de *A* destinées à être de la même catégorie syntaxico-sémantique et que le système connaît actuellement. ⌊αβ⌋⌈≡(⌊…⌋⌈…⌉⌊…⌋⌈…⌉)⌉

RD 4.4-29 5. Le sous-quantificateur de la première équivalence de *A*, respectivement la deuxième équivalence de *A*, est une fonction logique biconditionnelle. ⌊αβ⌋⌈≡(⌊…⌋⌈≡(- -)⌉⌊…⌋⌈≡(- -)⌉)⌉

RD 4.4-30 6. Le premier argument, respectivement le deuxième argument, de la fonction logique biconditionnelle du sous-quantificateur de la première équivalence de *A*, respectivement de la deuxième équivalence de *A*, est une fonction logique de la catégorie des propositions. ⌊αβ⌋⌈≡(⌊…⌋⌈≡(F_1 F_2)⌉⌊…⌋⌈≡(F_3 F_4)⌉)⌉

RD 4.4-31 7. Le foncteur destiné à être celui de la fonction F_1, respectivement de la fonction F_2, est un terme variable équiforme au premier lieur du quantificateur de *A*, respectivement au deuxième lieur du quantificateur de *A*. ⌊αβ⌋⌈≡(⌊…⌋⌈≡(α(…) β(…))⌉⌊…⌋⌈≡(F_3 F_4)⌉)⌉[24]

RD 4.4-32 8. Les arguments du ou des contexte(s) de la fonction destinés à être la fonction

[23] Nous utilisons la notion de schéma, car une généralisation est bien formée si le *sous-quantificateur* est une *expression biconditionnelle*.

[24] Les parenthésages avec contours ⟪ et ⟫ […] sont à considérer comme l'expression de délimitateurs schématiques d'un ensembles de variables dont les catégories ne sont pas déterminées (MIEVILLE, 2007, p. 83).

F_1, respectivement F_2, sont des termes variables qui possèdent des lieurs équiformes dans le quantificateur contenu dans la première équivalence du sous-quantificateur de *A*. Le ou les contexte(s) destinés à être ceux de la fonction F_1, et les arguments qu'ils contiennent, sont équivalent à ceux de la fonction destinés à être la fonction F_2.
⌊αβ⌋⌈≡(⌊xy…z⌋⌈≡(α⦇xy…z⦈ β⦇xy…z⦈)⌉⌊…⌋⌈≡(F_3 F_4)⌉)⌉

RD 4.4-33 9. Le foncteur de la fonction destiné à être celui de la fonction F_3, respectivement F_4, est un terme variable formateur de proposition à un argument équiforme au premier lieur du quantificateur de *A*, respectivement équiforme au deuxième lieur du quantificateur de *A*. Ce foncteur possède un lieur équiforme dans le quantificateur contenu dans la deuxième équivalence du sous-quantificateur de *A*. ⌊αβ⌋⌈≡(⌊xy…z⌋⌈≡(α⦇xy…z⦈ β⦇xy…z⦈)⌉⌊ψ⌋⌈≡(ψ(α) ψ(β)⌉)⌉

RD 4.4-34 10. Les termes variables des fonctions logiques destinés à être ceux des fonctions F_1, F_2, F_3, et F_4, sont de catégories syntaxico-sémantiques que le système contient actuellement.

RD 4.4-35 11. Les foncteurs variables destinés à être ceux des fonctions logiques F_1 et F_2 , et le foncteur variable destiné à être celui des fonctions logiques F_3 et F_4, ont entre eux le rapport suivant :
catégorie de α et β, C1/C2…Cn et catégorie de ψ, S/(C1/C2…Cn).

Avant d'appliquer la *directive d'extensionalité* et pour respecter la règle n° 11 il faudra introduire les *catégories* nécessaires pour la mettre en œuvre :

F 4.41. D17 : ⌊A⌋⌈≡(E<A>⌊q⌋⌈A(q)⌉)⌉, soit la *catégorie* : S/(S/S), E, <->

permettant la mise en œuvre de la *règle d'extensionalité* pour :

E1 : ⌊αβ⌋⌈≡(⌊p⌋⌈≡(α(p) β(p))⌉⌊ψ⌋⌈≡ψ<α> ψ<β>⌉)⌉

F 4.42. D18 : ⌊A⌋⌈≡(E⟨A⟩⌊pq⌋⌈A(pq)⌉)⌉, soit la *catégorie* : S/(S/SS), E, ⟨-⟩

permettant la mise en œuvre de la *règle d'extensionalité* pour :

E2 : ⌊αβ⌋⌈≡(⌊pq⌋⌈≡(α(pq) β(pq))⌉⌊ψ⌋⌈≡(ψ⟨α⟩ ψ⟨β⟩)⌉)⌉

F 4.43. D19 : ⌊A⌋⌈≡(E{A}⌊pq⌋⌈A{p}(q)⌉)⌉, soit la catégorie : S/((S/S)/S), E, {-}

permettant la mise en œuvre de la *règle d'extensionalité* pour :

E3 : ⌊αβ⌋⌈≡(⌊pq⌋⌈≡(α{p}(q) β{p}(q))⌉⌊ψ⌋⌈≡(ψ{α} ψ{β})⌉)⌉

Il s'agit d'affirmer que :

> […] en formulant une règle d'extensionalité permettant d'exprimer ce principe pour toute catégorie syntaxico-sémantique que le système possède actuellement, et qui est différente de celle de proposition. […] la directive d'extensionalité ne permet d'inscrire de nouvelles thèses qu'à la condition que les foncteurs variables soient d'une catégorie préalablement introduite dans le système. De manière schématique, si la paire de catégorie suivante a déjà été introduite, C1/C2…Cn et S/(C1/C2…Cn) alors le principe d'extensionalité pour la catégorie C1/C2…Cn peut être formulé comme une thèse du système (MIEVILLE, 2007, p. 80).

4.4.7 La déduction et preuve

Avec l'apport de ces quatre nouvelles *règles d'inférences*, nous avons à notre disposition tous les éléments nécessaires pour mettre en œuvre des *déductions* logiques et des *preuves*. Denis Miéville donne des exemples dans son ouvrage (MIEVILLE, 2007, p. 87 et suivantes).

Ceci termine notre survol de la *protothétique*. Sur cette base, nous abordons l'*ontologie* de Leśniewski.

4.5 Introduction à l'ontologie

Nous avons vu dans la section *3.9 Conclusion, p. 34* que l'*ontologie* constituait notre hypothèse et notre choix comme moyen de *formaliser* les tableaux de bord et répondre aux questions y relatives. Par conséquent, nous allons approfondir cette théorie sous l'éclairage de ce qui a été dit sur la *protothétique*, qui est valide et sert de base pour ce *SL*. Au fur et mesure de la présentation, nous faisons référence à nos exemples afin de vérifier dans quelle mesure ce *système* répond au formalisme que nous recherchons.

On peut définir l'*ontologie* comme la théorie formelle des *noms*. Nous pouvons aussi la définir comme le calcul des termes[25]. Comme théorie des *noms*, elle ne fait que désigner les objets ; elle n'est pas une théorie des objets. Elle traite des *noms* d'ordre supérieur puisqu'elle permet de quantifier les propriétés. L'*ontologie* est une théorie logique qui traite du « vrai » ou du « faux ». Elle se construit à partir d'une approche intuitive de l'interprétation de la copule, « est ». Leśniewski attribue une et une seule signification à la copule. C'est celle que l'*epsilon*, ε, représente.

Prenons un exemple de notre tableau de bord :

F 4.44. $A =_{df}$ 5 000

$b =_{df}$ Chiffre d'affaires du whisky

La valeur de vérité de la proposition A ε b est le vrai parce que 5 000 dans notre *domaine de connaissances* (et dans l'exemple, *Figure 1, p. 20*) ne dénote qu'un et un seul objet. C'est le cas aussi de Chiffre d'affaires du whisky qui dénote aussi le même objet. Etendons l'exemple :

F 4.45. $A =_{df}$ Chiffre d'affaires du whisky

$b =_{df}$ Indicateurs

La valeur de vérité de la proposition A ε b est encore le vrai puisque le Chiffre d'affaires est bien un des Indicateurs.

Un *nom* dénotant un et un seul objet est qualifié de *nom singulier* (*F 4.44.*). Un *nom* qui désigne plusieurs objet, comme Indicateurs est appelé *nom général*. Dans notre *domaine de connaissances*, un *nom* comme Lapin ne désigne aucun objet, c'est un *nom vide*.

Dans l'*ontologie* de Leśniewski et afin d'affiner la signification de l'*epsilon*, la valeur de la définition suivante est le faux :

F 4.46. $a =_{df}$ Indicateurs

$b =_{df}$ Niveaux de l'arborescence du tableau de bord

Parce que ces *noms* sont des *noms généraux*.

Sous cet éclairage, donnons-nous trois objets : O1, O2 et O3 avec les définitions suivantes :

[25] Attention : le mot « terme » est pris ici dans son sens générique. Il s'agit de pas le confondre avec la notion de *terme* utilisé dans la *procédure définitoire* de la *protothétique*.

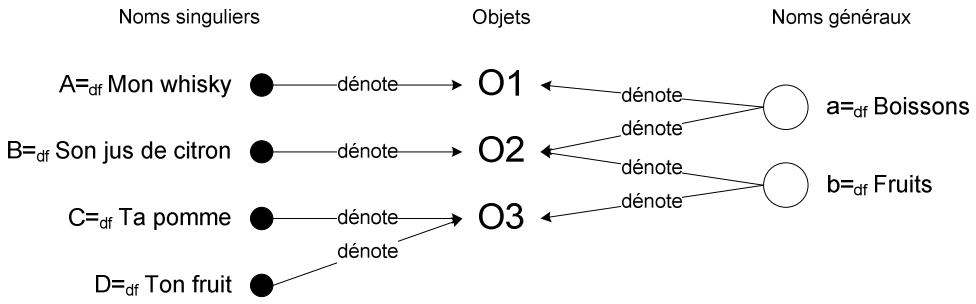


Figure 26. Exemple de dénotation

Donnons-nous la *matrice de vérités* associées aux propositions singulières de la figure qui précède :

ε	A	B	C	D	a	b
A	1	0	0	0	1	0
B	0	1	0	0	1	1
C	0	0	1	1	0	1
D	0	0	1	1	0	1
a	0	0	0	0	0	0
b	0	0	0	0	0	0

Figure 27. *Matrice de vérités*

La lecture de cette *matrice de vérités* n'appelle pas de commentaire particulier. Toutefois, elle est un exemple d'interprétation d'un *foncteur* que nous avons escamotée dans notre présentation de la *protothétique* (*4.4.4. La quantification des foncteurs, p. 47*). Nous y reviendrons plus loin (*4.6 Les interprétations de foncteurs et matrices de* vérités*, p. 70*).

La proposition singulière A ε b, peut se lire : « A est un des b ». En se référant à la *Figure 27, p. 54,* nous constatons que cette proposition n'est pas nécessairement toujours vraie. Dès lors, qu'est ce qui détermine son calcul de véracité. Denis Miéville (MIEVILLE, 2004, p. 23) résume les conditions nécessaires à cette évaluation de la façon suivante :

> Ainsi, la proposition singulière « A ε b » est vraie, si et seulement si :
>
> 1) le nom A se réfère à une certaine existence, en fait s'il n'est pas un nom vide ;
>
> 2) le nom A désigne un et un seul objet ;
>
> 3) le principe de convergence est respecté ; à savoir que si quelque nom désigne le même objet que le nom A, il exprime également le fait qu'il est l'objet ou l'un des objets désignés par le nom b.

Etudions encore la représentation graphique que rapporte Denis Miéville (MIEVILLE, 2004, p. 22) des travaux de Lejewski. l.1 est mis pour les *noms singuliers*, l.2 pour les *noms généraux* et l.3 pour les *noms vides :*

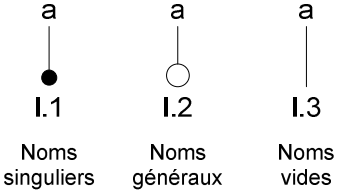


Figure 28. Représentation graphique des types de *noms*

A partir de là, Denis Miéville (MIEVILLE, 2004, p. 24) propose la représentation graphique, inspirée de Lejewski, des seize combinaisons possibles d'interprétation de l'*epsilon* :

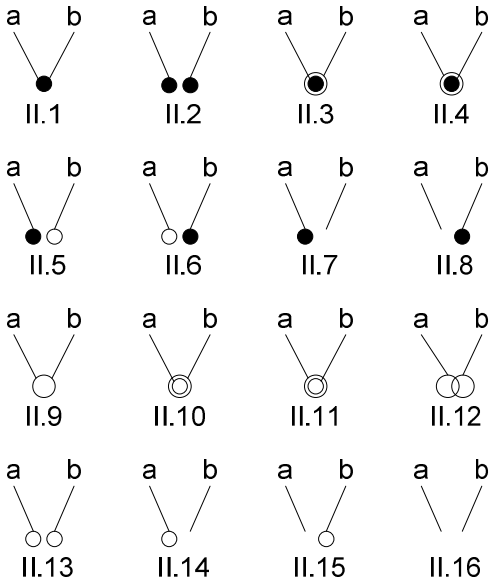


Figure 29. Les seize interprétations possibles de l'epsilon

Dans cette combinaison de représentations, la proposition A ε b est vraie que pour les diagrammes : II.1 et II.3 ; sinon elle est fausse.

Sachant que le *système* qui va se construire à partir de ces quelques éléments introductifs et de définitions de l'*epsilon*, nous pouvons déjà affirmer que les aspects statiques et descriptifs de *formalisation* peuvent être traduits par la proposition A ε b. Evidemment, un nombre important de traductions peuvent être appliquées à nos exemples, dont en voici une :

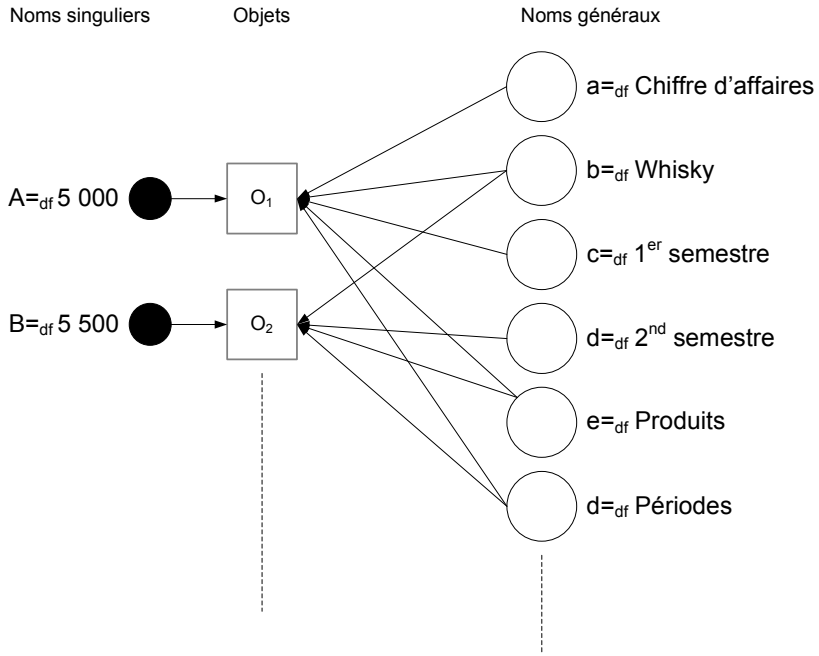


Figure 30. Application de la proposition A ε b à une partie du tableau de bord

A cause de l'introduction de la *catégorie* des *noms* à ce que connaît la *protothétique*, la *définition inductive* des *catégories syntaxico-sémantiques* est légèrement adaptée ; en reprenant Denis Miéville (MIEVILLE, 2004, p. 34) :

F 4.47. (i) *S* et *N* sont des catégories syntaxico-sémantiques ;

F 4.48. (ii) Si $C_1, C_2, \ldots, C_n$ sont des *catégories*, alors $C_1/C_2C_3 \ldots C_n$ est une catégorie syntaxico-sémantique ; il s'agit de de la catégorie formatrice de la catégorie C_1 à n-1

arguments, dont le premier est de la catégorie C_2, le deuxième de la catégorie C_3, …, le $(n-1)^{ème}$ C_n ;

F 4.49. (iii) Rien n'est une catégorie syntaxico-sémantique, sinon par ce qui précède.

L'*ontologie* s'enrichit par le biais de sept *règles d'inférences* ou de *directives inférentielles*[26] :

1) la *directive de définition ontologique de type protothétique ;*
2) la *directive de définition ontologique de type ontologique ;*
3) la *directive de distribution des quantificateurs ;*
4) la *directive de détachement ;*
5) la *directive de substitution ;*
6) la *directive d'extensionalité de type protothétique ;*
7) la *directive d'extensionalité de type ontologique.*

4.5.1 L'axiome de l'ontologie

Voici le seul et unique *axiome* de l'*ontologie :*

F 4.50. A_{ont} :

$\lfloor Ab \rfloor \ulcorner \equiv (\varepsilon\{Ab\} \wedge (\sim (\lfloor B \rfloor \ulcorner \sim (\varepsilon\{BA\}) \urcorner) \wedge (\lfloor DC \rfloor \ulcorner \supset (\wedge (\varepsilon\{DA\} \varepsilon\{CA\}) \varepsilon\{DC\}) \urcorner \lfloor D \rfloor \ulcorner \supset (\varepsilon\{DA\} \varepsilon\{Db\}) \urcorner))) \urcorner$

Voyons en *version catégorielle* quelles propriétés il porte :

F 4.51.
$$(\forall Ab)((A\ \varepsilon\ b) \equiv ((\exists B)(B\ \varepsilon\ A)) \wedge \quad \text{(existence)}$$
$$(\forall DC)(((D\ \varepsilon\ A) \wedge (C\ \varepsilon\ A)) \supset (D\ \varepsilon\ C))) \wedge \quad \text{(unicité, transitivité)}$$
$$(\forall D)((D\ \varepsilon\ A) \supset (D\ \varepsilon\ b)))) \quad \text{(convergence)}$$

Denis Miéville signale :

> Cet axiome, parce qu'il ne s'agit pas d'un schéma d'axiomes, est en fait la première thèse de l'ontologie, et c'est à partir d'elle qu'il sera possible, par le biais de règles d'inférences appropriées, d'inscrire progressivement de nouvelles thèses. (MIEVILLE, 2004, p. 25)

Nous notons que :

1) la deuxième équivalence du *sous-quantificateur* est constitué d'une conjonction ;
2) l'*epsilon* s'applique à un *contexte*, {- -}, à deux arguments de la *catégorie* des *noms* ;
3) une nouvelle *catégorie syntaxico-sémantique* (*S/NN*) est introduite ;
4) l'*axiome* de l'*ontologie* inscrit les propriétés de la signification primitive de l'*epsilon* (existence, transitivité, convergence) ;
5) il inscrit la relation entre un *nom singulier* et un *nom général* ;
6) la *biconditionnelle* du *sous-quantificateur* dominant est toujours vraie puisque c'est un *axiome*.

4.5.2 Les deux directives de définition

La *directive de définition ontologique de type protothétique* est identique à celle de la *protothétique* présentée dans la section *4.4.1 La règle d'inférences de* définition*, p. 42*.

[26] Denis Miéville utilise généralement la notion de *règle d'inférences* dans son fascicule (MIEVILLE, 2007) et la notion de *directive inférentielle* dans son fascicule (MIEVILLE, 2004). Dès lors, *règle d'inférences* et *directive* sont synonymes.

Voici la *directive de définition ontologique de type ontologique*, extraite du fascicule de Denis Miéville (MIEVILLE, 2004, p. 64 et suivantes) :

RD 4.5-1 Une inscription *T* est une thèse-définition ontologique de type ontologique en fonction de la dernière thèse *B* inscrite et issue de la base axiomatique A1-A3 contenant les catégories syntaxico-sémantiques *S* et *S/SS*, et leur contexte associé : (- -), ainsi que le foncteur constant de biconditionnelle : ≡, et l'axiome de l'ontologie A_o4 contenant les catégories nouvelles *N* et *S/NN*, et leur contexte associé : {- -}, ainsi que les foncteurs constants : ε, ~, ∧, ⊃ respectivement associé aux contextes : {- -}, (-) et (--), si et seulement si toutes les conditions suivantes sont remplies :

RD 4.5-2 1. *T* est une généralisation ⌊…⌋⌈…⌉.

RD 4.5-3 2. L'expression interne du sous-quantificateur de *T* est constitué du terme constant ≡ suivi du contexte protothétique : (- -), ⌊…⌋⌈≡(- -)⌉.

RD 4.5-4 3. Le premier argument du sous-quantificateur de *T* est une fonction singulière. Ce premier argument constitue le *definiendum* : ⌊…⌋⌈≡(ε{- -} -)⌉.

RD 4.5-5 4. Le sujet du premier argument du sous-quantificateur de *T* est une variable ; elle possède donc au moins une variable équiforme dans le deuxième argument du sous-quantificateur de *T* : ⌊…⌋⌈≡(ε{A -} …A…)⌉.

RD 4.5-6 5. Le prédicat du premier argument du sous-quantificateur de *T* est :

a. soit une fonction constante :
⌊…⌋⌈≡(ε{A∑} …A…)⌉

b. soit une fonction régulière :
⌊…⌋⌈≡(ε{A∑(…)} …A…)⌉

c. soit une fonction paramétrée :
⌊…⌋⌈≡(ε{A∑(…)⌊…⌋⟦…⟧}…A…)⌉

Dans ce contexte définitoire de la fonction constante ∑, la fonction régulière ∑(…) et la fonction paramétrée ∑(…)⌊…⌋⟦…⟧ sont toutes les deux de la catégorie des noms […][27].

RD 4.5-7 6. Le foncteur de la fonction logique du prédicat du *definiendum* est un terme constant. Ce terme constant ne saurait être équiforme à une constante logique de même catégorie syntaxico-sémantique qui appartient déjà au système.

RD 4.5-8 7. Si le foncteur constant du prédicat du *definiendum* est destiné à être d'une catégorie syntaxico-sémantique qui appartient déjà au système, le contexte qui soit doit être similaire au contexte préalablement fixé pour cette catégorie.

RD 4.5-9 8. Si le foncteur constant du prédicat du *definiendum* est destiné à être d'une catégorie qui n'appartient pas encore au système, il est nécessaire d'inscrire un nouveau contexte de manière à éviter toute confusion. Cela signifie qu'on ne saurait choisir des parenthèsage équiformes à un contexte préalablement inscrit et qui possède le même nombre d'arguments que le *definiendum*.

RD 4.5-10 9. Les arguments du ou des contextes du prédicat du *definiendum* sont des termes variables.

RD 4.5-11 10. Aucun signe du *definiendum* n'est répété.

RD 4.5-12 11. Le deuxième argument du sous-quantificateur de *T* est une conjonction ∧ de la catégorie *S/SS* ; il constitue le *definiens*.

RD 4.5-13 12. Le premier argument de cette conjonction est une fonction singulière dont le sujet et le prédicat sont équiformes au sujet du premier argument du sous-quantificateur de *T* :
⌊A…⌋⌈≡(ε{A∑(…)⟧}∧(ε {AA}-))⌉

[27] Les parenthèses avec contours sont métalinguistiques.

RD 4.5-14 13. Le deuxième argument de cette conjonction est une inscription bien construite en fonction de ce qui a été préalablement développé dans le système jusqu'à l'inscription de la dernière thèse *B*. Il est soit une généralisation soit une fonction logique.

RD 4.5-15 14. Les termes variables libres du *definiendum* sont également inscrits dans le *definiens*, et réciproquement :
$\lfloor Av_1v_2\ldots v_i\rfloor\lceil\equiv(\ \varepsilon\{A\Sigma(v_1v_2\ldots v_i)\}\wedge(\varepsilon\ \{AA\}Ev_1v_2\ldots v_i))\rceil$

RD 4.5-16 15. Dans la construction du *definiens* ou du *definiendum*, il est possible d'inscrire des termes variables d'une quelconque catégorie syntaxico-sémantique pour autant qu'un foncteur constant de cette catégorie ait été préalablement inscrit.

RD 4.5-17 16. L'expression *T* ne contient aucune variable libre.

Comme l'*ontologie* sous-tend logiquement la *protothétique*, les deux *règles de définition* peuvent être appliquée à n'importe quelle étape de construction de l'*ontologie*.

Nous avons maintenant tous les matériaux nécessaires pour mettre en œuvre la *procédure définitoire* de l'*ontologie*.

4.5.3 Construction de l'ontologie

Comme nous l'avons fait pour la *protothétique*, nous allons suivre le processus de construction de l'*ontologie* que nous indique Denis Miéville (MIEVILLE, 2004) sans pour autant tenir compte des aspects didactiques qui sont apportés. Nous ne prenons que les *thèses-définition* qui sont correctement formées. A chaque fois qu'une *thèse* répond ou peut servir à traduire un des éléments en question de notre tableau de bord, nous nous y arrêterons pour mettre à l'épreuve le formalisme.

Prenons la première *thèse-définition* :

F 4.52. D17[28], il existe au moins un a :

$\lfloor a\rfloor\lceil\equiv(!\{a\}\sim(\lfloor A\rfloor\lceil\sim(\varepsilon\{Aa\})\rceil))\rceil$

!{a}, se lit : a existe au sens de la *catégorie* des *noms*, il existe au moins un objet dénoté par a ou les a existent ou il existe au moins un a ou encore a n'est pas un *nom vide*. Cette proposition est vraie si l'une ou l'autre des situations représentées par les diagrammes I.1 et I.2 est réalisée (*Figure 29, p. 55*), sinon elle est fausse.

Catégorie	Constante	Contexte
S/N	!	{-}

Cette *thèse-définition* nous intéresse tout particulièrement puisqu'elle traite de l'existence d'un objet dénoté par son *nom* a. Elle va nous permettre de traiter la suppression d'un niveau dans nos arborescences. Prenons un extrait de notre exemple de tableau de bord et traduisons-le avec cette *thèse*. Ce n'est pas la traduction qui est simple qui retient notre attention, c'est le fait de disposer de cette *thèse* pour l'introduire dans une *déduction* plus en avant dans notre développement :

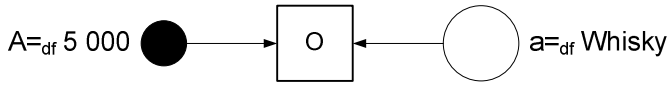


Figure 31. Application de la proposition A ε a à une partie du tableau de bord

[28] Au niveau où nous avons laissé la construction de la *protothétique*, nous avions 19 *thèses-définition*. Denis Miéville dans son fascicule sur l'ontologie ne prend pas en compte les trois thèses-définition apportées pour la mise en œuvre de la *directive d'extensionalité*. Nous avons choisi de suivre sa numérotation pour permettre au lecteur de se reporter à ses travaux sans trop de difficultés.

Qui se lit, en ce qui concerne le *definiendum* : « il existe au moins un *nom* d'objet dénoté par Whisky » et en ce qui concerne le *definiens* : « il existe une valeur, 5 000, tel que 5 000 est un des Whisky ».

Nous retenons cette *thèse* comme devant s'appliquer à la plupart des dénotations de nos objets.

Parcourons les *thèses-définition* suivantes :

F 4.53. D18, il existe au plus un a :

⌊a⌋⌈≡(¬{a}⌊BC⌋⌈⊃(∧(ε{Ba}ε{Ca})ε{BC})⌉)⌉

Lire : il existe au plus un a ou le *nom* a désigne au plus un objet. La proposition ¬{a} est vraie si et seulement si elle entre dans le champ des situations I.1 ou I.3.

Catégorie	Constante	Contexte
S/N	!, ¬	{-}

Comme celle qui précède, cette *thèse* retient notre attention puisqu'elle nous permet d'éviter, dans nos exemples, que certains nœuds ne dénotent pas un objet (qui n'ont aucun lien dans les structures arborescentes). Elle nous permet, entre autres, de tenir compte d'une synonymie de *noms singuliers* pour un même objet.

F 4.54. D19, A désigne exactement un objet :

⌊A⌋⌈≡(↓{A} ~(⌊b⌋⌈~(ε{Ab})⌉))⌉

Lire : A désigne exactement un objet (I.1) ou de manière abusive, A est unique.

Catégorie	Constante	Contexte
S/N	!, ¬, ↓	{-}

Cette *thèse* qui nous assurera d'une dénotation sur un objet unique, nous conduit à une considération délicate concernant nos exemples de tableau de bord. Quand nous dénotons un objet par une valeur, elle n'est pas unique dans notre *domaine de connaissances*. En effet, un même nombre peut désigner deux objets différents. Une même valeur 5 000 peut dénoter un « Whisky » ou un « Vin », par exemple. Dès lors, il faut affiner notre représentation. Afin d'éviter cet écueil, nous nous donnons des *noms singuliers* pour chacun des objets comme val_1, val_2, val_3, …, val_n. Nous établissons ensuite une application entre val_i et l'ensemble des nombres[29].

F 4.55. D20, A est identique à B :

⌊AB⌋⌈≡(={AB} ~(ε{AB}ε{BA}))⌉

Lire : le *nom* A désigne le même objet que le *nom* B (II.1).

Catégorie	Constantes	Contexte
S/NN	ε, =	{- -}

F 4.56. D21, A et B désignent des objets différents :

⌊AB⌋⌈≡(≠{AB} ∧(∧(ε{AA}ε{BB})~(={AB})))⌉

Lire : le *nom* A et le *nom* B désigne des individus différents (II.2).

Catégorie	Constantes	Contexte
S/NN	ε, =, ≠	{- -}

F 4.57. D22, a égale (faible) b :

⌊ab⌋⌈≡(ō{ab}⌊A⌋⌈≡(ε{Aa}ε{Ab})⌉)⌉

[29] Ce qui de façon très pratique, dans l'apport de nos outils d'automation, permet d'un point de vue informatique d'utiliser les *noms singuliers* comme identificateurs de variable de type réel.

Lire : sans prétendre à l'existence des a et des b, les *nom*s a et b désignent les mêmes objets, a égal b, le *nom* a désigne la même extension que le *nom* b. Il s'agit d'une « égalité faible » (II.1, II.9, II.16).

Nous n'avons pas considéré cette *thèse* pour nos exemples, que nous tenons comme trop permissive pour assurer la rigueur que nous recherchons.

F 4.58. D23, a égale (fortement) b :

⌊ab⌋⌈≡(⊟{ab}∧(~(⌊G⌋⌈~(ε{Ga})⌉)⌊G⌋⌈≡(ε{Ga} ε{Gb})⌉))⌉

Lire : a est « fortement égal » à b (II.1, II.9).

Catégorie	Constantes	Contexte
S/NN	ε, =, ≠, ō, ⊟	{- -}

Cette *thèse* se présente comme très utile pour nos exemples, puisqu'elle va nous permettre d'isoler, dans une structure donnée, tous les *noms généraux* identiques. Dans une même *dimension* comme Indicateurs ou Produits, nous ne pouvons pas avoir deux *noms généraux* identiques. Donc, si cette *thèse* devait d'appliquer, sous une forme ou une autre, à une *dimension* et si son calcul de véracité devait conduire à une évaluation qui est le « vrai », alors nous aurions une malfaçon.

F 4.59. D24, inclusion faible :

⌊ab⌋⌈≡(ĉ{ab}⌊A⌋⌈⊃(ε{Aa}ε{Ab})⌉)⌉

Lire : Sans prétendre à l'existence des a et des b, tout ce qui est désigné par le *nom* a est également désigné par le *nom* b, « tout a est b ». Il s'agit d'une « inclusion faible » (II.1, II.3, II.8, II.9, II.10, II.15, II.16).

Catégorie	Constantes	Contexte
S/NN	ε, =, ≠, ō, ⊟, ĉ	{- -}

F 4.60. D25, inclusion forte :

⌊ab⌋⌈≡(Ĉ{ab}∧(~(⌊G⌋⌈~(ε{Ga})⌉)⌊G⌋⌈⊃(ε{Ga} ε{Gb})⌉))⌉

Lire : chaque « a est b », « inclusion forte » (II.1, II.3, II.9, II.10).

Catégorie	Constantes	Contexte
S/NN	ε, =, ≠, ō, ⊟, ĉ, Ĉ	{- -}

Les deux *thèses* qui précèdent, sont particulièrement enrichissantes pour nos exemples. Tout d'abord, elles établissent une relation entre deux *noms généraux*. Ensuite, avec leurs apports, elles vont nous permettre de *transposer* dans l'*ontologie* et de façon descriptive pratiquement toutes les structures arborescentes de nos exemples. Prenons un extrait d'une structure de notre tableau de bord :

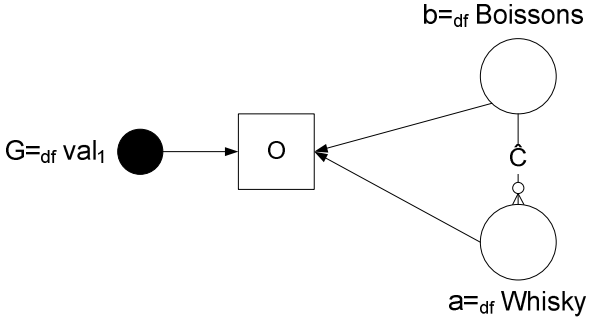


Figure 32. Application de la proposition a Ĉ b à une partie du tableau de bord

En traduction et en s'inspirant de la *version catégorielle* de la *thèse-définition 25*, nous obtenons la lecture suivante :

« Pour tout, être un des Whisky et être une des Boissons, Whisky est inclus fortement dans Boissons est équivalent à : il existe une val_i telle que val_i est une des Boissons et pour toutes les val_i si val_i est un des Whisky alors val_i est une des Boissons ».

Avec ce qui précède et qui a retenu notre attention, l'introduction de cette dernière *thèse* sert de pivot dans notre recherche. Nous sommes, dès à présent, capables de traduire l'ensemble de nos objets, de leurs propriétés et des liens entre eux qui forment nos structures. En d'autres termes, nous couvrons nos besoins de compétences descriptives et statiques[30] de nos exemples. C'est la base minimale pour nos *transpositions*. Il reste à trouver le même niveau de compétences pour nos *restructurations* et autres *transformations* élémentaires.

Nous continuons à rapporter les *thèses-définition* pour atteindre le niveau de construction que Denis Miéville nous propose.

F 4.61. D26, inclusion partielle :

⌊ab⌋⌈≡(Δ{ab}~(⌊G⌋⌈~(∧(ε{Ga}ε{Gb}))⌉))⌉

Lire : quelque a est b, « inclusion partielle » (II.1, II.3, II.4, II.10, II.11, II.12).

Catégorie	Constantes	Contexte
S/NN	ε, =, ≠, ō, ⊟, ĉ, Ĉ, Δ	{- -}

F 4.62. D27, existence :

⌊*f*⌋⌈≡(!⌈*f*⌉~(⌊a⌋⌈~(*f*{a})⌉))⌉

Soit la *quantification* d'un *foncteur*.

!{a} : « existe » associé à la *catégorie N*.
!⌈*f*⌉ : « existe » associé à la *catégorie* S/(S/N).

Catégorie	Constantes	Contexte
S/(S/N)	!	⌈-⌉

F 4.63. D28, « catégorie grammaticale des verbes » :

⌊Ab⌋⌈≡(ε⦃b⦄{A} ε{Ab})⌉

ε⦃-⦄{A} : de la *catégorie* des propositions : *S.*

ε⦃-⦄ : foncteur variable qui opère sur un unique argument nominal : *N* ; il est de la *catégorie S/N* (MIEVILLE, 2004, p. 63).

Catégorie	Constante	Contexte
(S/N)/N	ε	⦃-⦄

F 4.64. D29, définition de « constante » *nom* vide :

⌊A⌋⌈≡(ε{A**Λ**}∧(ε{AA}~(ε{AA})))⌉

Lecture de A ε **Λ** : A est un *nom* vide ; A est un *nom* qui ne dénote rien ; A est un *nom* contradictoire ; A est un *nom* qui ne détone pas quelque chose.

Catégorie	Constante	Contexte
N	**Λ**	

F 4.65. D30, définition de « constante » *nom* universel :

⌊A⌋⌈≡(ε{AV}ε{AA})⌉

Lecture de A ε V : A est un *nom* d'objet.

Catégorie	Constantes	Contexte
N	Λ, V	

[30] Ce que nous appelons aussi taxonomie.

F 4.66. D31, A est parmi les duaux de b, « le contraire » :

⌊Ab⌋⌈≡(ε{A~<b>}∧(ε{AA}~(ε{Ab}))⌉

Lecture de A~<b> : A est parmi les duaux de b ou si l'on attribue à b le *nom* « pair », A ε b peut se lire : A est pair et A ε ~<b> : A est impair.

Catégorie	Constante	Contexte
N/N	~	<->

F 4.67. D32, A parmi a et b :

⌊Aab⌋⌈≡(ε{A∩<ab>}∧(ε{AA}∧(ε{Aa} ε{Ab})))⌉

Lecture de A ε ∩ <ab> : A désigne un objet qui est parmi ce qui est désigné d'une part par le *nom* a et d'autre part par le *nom* b.

Catégorie	Constante	Contexte
N/NN	∩	<- ->

F 4.68. D33, A satisfait l'action Ψ :

⌊AΨ⌋⌈≡(ε{A S〈Ψ〉}∧(ε{AA}∧(ε{Aa}Ψ{A}))⌉

Lecture de A ε S〈Ψ〉 / ε{A S〈Ψ〉} : si Ψ portait l'action de cuisiner ; A satisfait l'acte de cuisiner, donc A est cuisinier.

Catégorie	Constante	Contexte
N/(S/N)	S	〈-〉

F 4.69. D34, « proposition subordonnée » :

⌊Abc⌋⌈≡(ε{A ε⪡b⪢<c>}∧(ε{AA}∧(ε{Ab} ε{Ac})))⌉

Voir Denis Miéville (MIEVILLE, 2004, p. 73 et suivantes).

Catégories	Constantes	Contextes
(N/N)/N	ε	⪡-⪢

Comme nous l'avons vu dans le cadre de la construction de la *protothétique*, d'une part cette *inscription* introduit la polysémie du *terme constant* ε et d'autre part il est associé à deux *contextes*. Sa définition sous-entend la *catégorie* :

N/N	~, ε⪡-⪢	<->

F 4.70. D35, connexion binaire bijective :

⌊Φ⌋⌈≡(⇆ ⦃Φ⦄ ∧(⌊abc⌋⌈⊃(∧(Φ{ab}Φ{ac})ō{bc})⌉⌊abc⌋⌈⊃(∧(Φ{ac}Φ{bc})ō{ab})⌉))⌉

Voir Denis Miéville (MIEVILLE, 2004, p. 77).

Catégorie	Constante	Contexte
S/(S/NN)	⇆	⦃-⦄

F 4.71. D36, a et b sont équinumériques :

⌊ab⌋⌈≡(∞{ab} ∧(~(⌊Φ⌋⌈~(⇆⦃Φ⦄)⌉)
≡(⌊A⌋⌈ε{Aa}~(⌊B⌋⌈~(∧(Φ{ab}ε{Bb}))⌉)⌉)
≡(⌊A⌋⌈ε{Ab}~(⌊B⌋⌈~(∧(Φ{ba}ε{Ba}))⌉)⌉)))⌉

Voir Denis Miéville (MIEVILLE, 2004, p. 77).

Catégorie	Constantes	Contexte
S/NN	ε, =, ≠, ō, ⊟, ĉ, Ĉ, Δ, ∞	{- -}

F 4.72. D37, a est la plus petite extension pour laquelle la fonction γ a du sens :

⌊ab⌋⌈≡(M⌊γ⌋{a}⊃(∧(γ{a}⌊b⌋⌈∧(ĉ{ba} γ{b})⌉)))⌉

Lecture de M(γ){a} : a est la plus petite extension pour laquelle la fonction γ a du sens, a est minimal par rapport à γ.

Catégories	Constantes	Contextes
(S/N)/(S/N)	M	(-)

Cette *inscription* introduit une sous-catégorie du *terme constant* M ; sa définition sous-entend la *catégorie* :

S/N	!, →, ↓, M(-)	{-}

F 4.73. D38, l'extension définie par a est finie :

⌊ab⌋⌈≡(F{a}⌊γb⌋⌈⊃(∧(γ{b}⌊d⌋⌈⊃(γ{d}ĉ{da})⌉)~(⌊ĉ⌋⌈~(M(γ){ĉ})⌉))⌉)⌉

Voir Denis Miéville (MIEVILLE, 2004, p. 77)

Catégorie	Constantes	Contexte
S/N	!, →, ↓, M(-), F	{-}

Cette *thèse* peut servir pour notre exemple de tableau de bord. En effet, nous rappelons que les *dimensions* qui constituent le modèle, sont des ensembles finis.

F 4.74. D39, l'extension définie par a est infinie :

⌊ab⌋⌈≡(I{a}~(F{a}))⌉

Voir Denis Miéville (MIEVILLE, 2004, p. 77).

Catégorie	Constantes	Contexte
S/N	!, →, ↓, M(-), F, I	{-}

F 4.75. D40, l'infini à la Dedekind :

⌊ab⌋⌈≡(Ⅎ{a}~(⌊b⌋⌈~(∧(∧(ĉ{ba}~(ĉ{ba}))∞{ab}))⌉))⌉

Voir Denis Miéville (MIEVILLE, 2004, p. 77).

Catégorie	Constantes	Contexte
S/N	!, →, ↓, M(-), F, I, Ⅎ	{-}

F 4.76. D41, somme logique (*ontologique*) :

⌊Aab⌋⌈≡(ε{A∪<ab>}∧(ε{AA}∨(ε{Aa}ε{Ab})))⌉

Voir Denis Miéville (MIEVILLE, 2004, p. 78).

Catégorie	Constantes	Contexte
N/NN	∩, ∪	<- ->

F 4.77. D42, somme logique (*protothétique*) :

⌊Aαβ⌋⌈≡(∪<-αβ->{A}∨(α{A}β{A}))⌉

Voir Denis Miéville (MIEVILLE, 2004, p. 78).

Catégories	Constantes	Contextes
(S/N)/(S/N)/(S/N)	∪	<- ->

Cette *inscription* introduit une sous-catégorie du *terme constant* ∪ ; sa définition sous-entend la *catégorie* :

S/N	!, →, ↓, M(-), F, I, Ⅎ, ∪<- ->	{-}

Avec cette dernière *thèse*, Denis Miéville arrête la *procédure définitoire* de l'*ontologie*. Nous obtenons la *bibliothèque* des *catégories syntaxico-sémantiques* suivante (y compris celles de la *protothétique*) :

Catégories	Constantes	Contextes
S/SS	≡, μ, τ, ω, ∧, ⊃, ∨	(- -)
S/S	α, ~, ¬, τ, ω(- -), ω [-](-)	(-)
S/SSS	δ	(- - -)
S	F, V, T	
(S/S)/SS	ω	(- -)
(S/S)/S	ω [-]	(-)
((S/S)/S)/S	ω	[-]
S/(S/SS)SS	Δ	[- - -]
S/(S/S)	E	<->
S/(S/SS)	E	(-)
S/((S/S)/S)	E	{-}
S/NN	ε, =, ≠, ō, ⊟, ĉ, Ĉ, Δ, ∞	{- -}
S/N	!, ⇁, ↓, M(-), F, I, Ⅎ, ∪<- ->	{-}
S/(S/N)	!	⌈-⌉
(S/N)/N	ε	{-}
N	**Λ**, V	
N/N	~, ε<->	<->
N/NN	∩, ∪	<- ->
N/(S/N)	S	⟨-⟩
(N/N)/N	ε	<->
S/(S/NN)	⇆	(-)
(S/N)/(S/N)	M	(-)
(S/N)/(S/N)/(S/N)	∪	<- ->

Voyons à partir de ce que connaît le *système*, comment nous pouvons *transposer* de façon formelle un exemple de structure rencontrée dans notre tableau de bord.

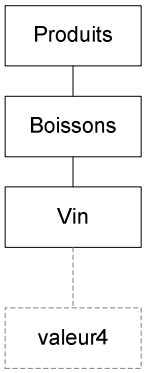


Figure 33. Extrait de l'arborescence Produits

Transposons cette figure graphiquement dans ce que nous savons de l'*ontologie :*

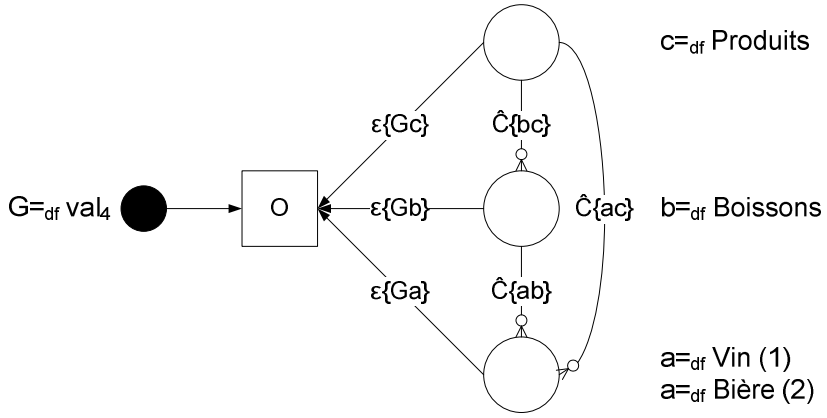


Figure 34. Transposition *ontologique* de Produits

Traduisons en définitions et dans la *thèse-définition* D25 (*F 4.60, p. 60*) la représentation graphique ci-dessus :

F 4.78. Par définitions :

$G =_{df} val_4$
$a =_{df}$ Vin
$b =_{df}$ Boissons
$c =_{df}$ Produits

F 4.79. Par la *thèse-définition* 25 :

$\lfloor ab \rfloor \lceil \equiv (\hat{C}\{ab\} \wedge (\sim(\lfloor G \rfloor \lceil \sim(\varepsilon\{Ga\})\rceil) \lfloor G \rfloor \lceil \supset(\varepsilon\{Ga\}\ \varepsilon\{Gb\})\rceil))\rceil$
$\lfloor bc \rfloor \lceil \equiv (\hat{C}\{bc\} \wedge (\sim(\lfloor G \rfloor \lceil \sim(\varepsilon\{Gb\})\rceil) \lfloor G \rfloor \lceil \supset(\varepsilon\{Gb\}\ \varepsilon\{Gc\})\rceil))\rceil$
$\lfloor ac \rfloor \lceil \equiv (\hat{C}\{ac\} \wedge (\sim(\lfloor G \rfloor \lceil \sim(\varepsilon\{Ga\})\rceil) \lfloor G \rfloor \lceil \supset(\varepsilon\{Ga\}\ \varepsilon\{Gc\})\rceil))\rceil$

Notons que les *inscriptions* comme : $\sim(\lfloor G \rfloor \lceil \sim(\varepsilon\{Ga\})\rceil)$, soit ce qui est indiqué par la *thèse-définition* 17 (*F 4.52, p. 58*) : !{a}, !{b} et !{c} assurent l'existence de la dénotation d'un *nom singulier* G à l'objet. Nous n'avons plus à en tenir compte. Ce qui est particulièrement important dans les tableaux de bord puisque le résultat de certaines malfaçons conduit dans les *restructurations* à des objets qui ne sont dénotés par aucun *nom* ou des *noms* qui ne dénotent aucun objet.

Quelle est l'*inscription* qui nous permettrait d'intégrer les trois *thèses,* soit les trois inclusions, ci-dessus ? La *transposition* intuitive : « si a est inclus dans b et que b est inclus dans c alors a est inclus dans c » trouve-t-elle une *inscription* qui tienne compte de la transitivité où nous pourrions substituer les *definiens* de ces trois *thèses* ? De façon purement exploratoire, en reprenant le *definiendum* de chaque *thèse*, nous cherchons un schéma de type :

F 4.80. $\supset(\wedge(\hat{C}\{ab\}\hat{C}\{bc\})\hat{C}\{ac\})$

Ce schéma n'existe pas dans ce qui a été construit préalablement dans le *système* (*protothétique* et *ontologie*). En considérant que $\hat{C}\{- -\}$ est de la *catégorie syntaxico-sémantique S/NN*, soit une proposition. On pourrait s'inspirer d'un schéma propositionnel comme $\supset(\wedge(pq)r)$. Mais là encore, en dans son état, le *système* nous offre pas une structure de ce type. Notre schéma tente d'exprimer la transitivité de l'inclusion. Comme nous n'avons pas trouvé de forme qui correspond à notre besoin dans ce que le s*ystème* possède, rien ne nous empêche, par exemple, de mettre en œuvre la *règle d'inférences de définition* pour générer une nouvelle *thèse-définition* et qui, par ailleurs, reprend notre schéma[31] :

[31] L'axiome de l'ontologie nous donne la propriété de transitivité de ε. Dès lors, comme le nom singulier G sert de pivot dans l'expression, il détermine aussi la propriété de transitivité de l'inclusion forte (thèse-définition 25). Par conséquent cette nouvelle thèse définition n'est qu'un exemple de schéma qui peut servir à des transformations purement syntaxiques. Elle n'apporte rien au niveau sémantique.

F 4.81. ⌊abc⌋⌈≡(Ɥ{ac} ⊃(∧(∧(~(⌊G⌋⌈~(ε{Ga})⌉)⌊G⌋⌈⊃(ε{Ga} ε{Gb})⌉)
∧(~(⌊G⌋⌈~(ε{Gb})⌉)⌊G⌋⌈⊃(ε{Gb} ε{Gc})⌉))
∧(~(⌊G⌋⌈~(ε{Ga})⌉)⌊G⌋⌈⊃(ε{Ga} ε{Gc})⌉)))⌉

Nous voilà dotés d'un nouveau *terme constant* Ɥ, de la *catégorie syntaxico-sémantique S/NN* et appliqué au *contexte* {--}.

Nous sommes maintenant armés, toujours de façon exploratoire, pour passer à la présentation des *règles d'inférences*, des *directives inférentielles* qui vont nous permettre de mettre en œuvre l'*ontologie* pour nos besoins de *restructurations* des tableaux de bord.

4.5.4 Les règles d'inférences ou directive inférentielles

Nous avons abordé les deux *règles d'inférences de définition* (*protothétique* et *ontologique*) qui nous ont permis d'appliquer la *procédure définitoire*. Abordons maintenant les cinq autres règles. Nous les présentons dans un ordre différent de celui que Denis Miéville suit, afin répondre, du plus simple au plus compliqué, à nos questions restées ouvertes pour couvrir nos besoins de *restructurations* des tableaux de bord :

1) la *directive de distribution des quantificateurs ;*
2) la *directive de substitution ;*
3) la *directive de détachement ;*
4) la *directive d'extensionalité de type protothétique ;*
5) la *directive d'extensionalité de type ontologique.*

4.5.4.1 La directive de distribution des quantificateurs

D'un point de vue des conditions qu'elle apporte, elle est identique à celle de la *protothétique*, mais exprimée de façon différente en tenant compte de l'introduction de la *catégorie* des *noms*. Nous la reprenons du fascicule de Denis Miéville (MIEVILLE, 2004, p. 81) :

RD 4.5-18 Une inscription *T* a le statut de thèse conforme à la directive de distribution des quantificateurs en fonction d'une thèse *A* préalablement inscrite et issue de la base axiomatique A1-A3 contenant les catégories syntaxico-sémantiques *S* et *S/SS*, et leur contexte associé : (- -), ainsi que le foncteur constant de biconditionnelle : ≡, et l'axiome de l'ontologie A_o4 contenant les catégories *N* et *S/NN*, et leur contexte associé : {- -}, ainsi que les foncteurs constants : ε, ~, ∧, ⊃ respectivement associé aux contextes : {- -}, (-) et (--), si et seulement si toutes les conditions suivantes sont remplies :

RD 4.5-19 1. L'inscription *A* est une thèse actuellement inscrite dans le système ; elle est une généralisation dont le sous-quantificateur est une proposition biconditionnelle. $⌊v_1…v_i⌋⌈≡(1^{er}$ arg. $2^{ème}$ arg.)⌉

RD 4.5-20 2. L'inscription *T* est soit une généralisation dont le sous-quantificateur est une proposition biconditionnelle, soit une proposition biconditionnelle.

RD 4.5-21 3. La relation formelle entre l'inscription de *A* et *T* est telle qu'elle porte l'idée d'une distribution de lieurs du quantificateur *A* dans le quantificateur du 1^{er} argument, respectivement du $2^{ème}$ argument de *T*, qu'il ait statut de généralisation ou de proposition biconditionnelle.

RD 4.5-22 4. L'inscription des lieurs distribués dans les quantificateurs appropriés n'a de sens que si elle est associée à la présence de variables équiformes :

A : $⌊v_1…v_hv_i⌋⌈≡(1^{er}$ $arg._{v1…vhvi}$ $2^{ème}$ $arg._{v1…vhvi})⌉$

T1 : $⌊v_1…v_h⌋⌈≡(⌊v_i⌋⌈1^{er}$ $arg._{v1…vhvi}⌉⌊v_i⌋⌈2^{ème}$ $arg._{v1…vhvi}⌉)⌉$

T2 : $⌊v_1…$ $⌋⌈≡(⌊v_iv_h⌋⌈1^{er}$ $arg._{v1…vhvi}⌉⌊v_iv_h⌋⌈2^{ème}$ $arg._{v1…vhvi}⌉)⌉$

…

Ti : $\equiv(\lfloor v_1 \ldots v_i v_h \rfloor \lceil 1^{er}\ arg._{v1 \ldots vhvi} \rceil \lfloor v_1 \ldots v_i v_h \rfloor \lceil 2^{ème}\ arg._{v1 \ldots vhvi} \rceil)$

4.5.4.2 La directive ontologique de substitution

Dans notre exemple de tableau de bord, nous avons relaté une question concernant la substitution d'un *nom* par un autre (Vin par Bière) dans une de nos structures (***Figure 8, p. 23***). Nous allons appliquer la *directive ontologique de substitution* pour envisager de *transposer* ce type de *transformation* élémentaire dont nous avons besoin dans notre *domaine de connaissances*. Voici cette directive (MIEVILLE, 2004, p. 87) :

RD 4.5-23 Une inscription *T* a le statut de thèse conforme à la directive de substitution en fonction d'une thèse *A* préalablement inscrite et issue de la base axiomatique A1-A3 contenant les catégories syntaxico-sémantiques *S* et *S/SS*, et leur contexte associé : (- -), ainsi que le foncteur constant de biconditionnelle : ≡, et l'axiome de l'ontologie, A_o4 contenant les catégories *N* et *S/NN*, et leur contexte associé : {- -}, ainsi que les foncteurs constants : ε, ~, ∧, ⊃ respectivement associé aux contextes : {- -}, (-) et (--), si et seulement si toutes les conditions suivantes sont remplies :

RD 4.5-24 1. *A* est une généralisation.

RD 4.5-25 2. A chaque variable équiforme de *A* qui possède un lieur équiforme dans le quantificateur de l'inscription *A* à laquelle il est effectivement lié, une inscription *E* est substituée.

RD 4.5-26 3. L'inscription *E,* qui peut être une variable, une constante ou une inscription complexe compatible avec l'état du système, est destinée à être de la même catégorie syntaxico-sémantique que la variable en voie d'être substituée.

RD 4.5-27 4. L'expression substituée à la variable doit rester libre pour cette variable dans l'essence de la généralisation *A*.

RD 4.5-28 5. Le quantificateur de *A* contient des lieurs équiformes à toute inscription destinée à être les variables libres de l'inscription interne de son sous-quantificateur et à celles-ci seulement.

RD 4.5-29 6. Si une inscription destinée à avoir statut de variable est substituée à une variable, cette inscription ne peut pas être équiforme à une constante préalablement inscrite dans le système.

Parmi les questions relatives à notre tableau de bord, nous nous sommes demandé, en tenant compte de notre *domaine de connaissances*, s'il était possible de remplacer Vin par Bière (***voir Figure 9, p. 24***). La condition 3 (***RD 4.5-24, p. 67***) de la *directive ontologique de substitution*, nous indique que *l'inscription E* à substituer peut être une *constante*, une *variable* ou une *inscription* complexe. Dans notre cas, il est évident que nous n'avons pas à faire à une *constante*, ni à une *inscription* complexe. De plus, nous ne cherchons pas à remplacer une *variable* de *nom général*, a, par une autre comme d. C'est donc dans la phase de définition des *variables*, dans l'action d'« assigner » à la *variable* a, une « constante » de *nom* que notre remplacement doit se faire :

F 4.82. La définition de 'a $=_{df}$ Vin' est remplacée par la définition a $=_{df}$ Bière

Par conséquent et quelle que soit l'*inscription* sur laquelle nous travaillons, tant que nous n'avons pas à substituer une *constante*, une *variable* ou une *inscription* complexe, nous aurons à faire nos remplacements dans les définitions « initiales ».

Nous avons aussi essayé d'utiliser la *directive ontologique de substitution* pour procéder à une insertion de structure dans une structure existante :

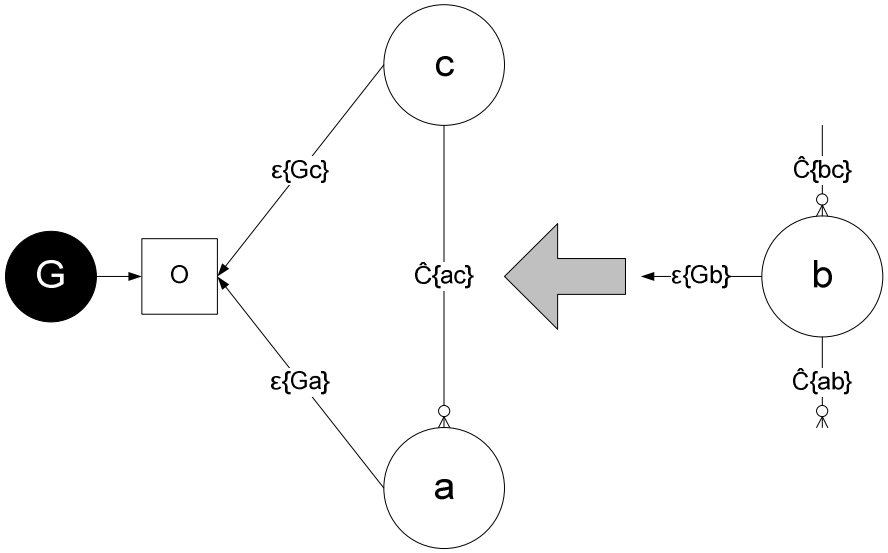


Figure 35. Tentative d'insertion

En d'autres termes de substituer à une proposition Ĉ{ac} la *transposition* d'une structure (*inscription E* mentionnée dans la condition 3 de la règle) qui pourrait se définir :

F 4.83. G=$_{df}$ non vide
b=$_{df}$ Nom à insérer
ε{Gb} : indéfini puique G est vide
Ĉ{bc} : ⌊bc⌋⌈≡(Ĉ{bc}∧(~(⌊G⌋⌈~(ε{Gb})⌉)⌊G⌋⌈⊃(ε{Gb} ε{Gc})⌉))⌉G et c vides
Ĉ{ab} : ⌊ab⌋⌈≡(Ĉ{ab}∧(~(⌊G⌋⌈~(ε{Ga})⌉)⌊G⌋⌈⊃(ε{Ga} ε{Gb})⌉))⌉G et a vides

Le monstre, ci-dessus, à moitié indéfini, ne correspond en rien à une *inscription E* de la condition 3 de la *règle de substitution*. De plus, la règle n'autorise la substitution que de *variables* (condition 2) ; ce qui n'est pas le cas de notre proposition Ĉ{ac}.

Par conséquent, cette règle nous sert plutôt de filtre qui assure qu'un monstre qui mettrait en cause tout le formalisme ne puisse pas être substitué. Continuons notre parcours des *directives inférentielles* de l'*ontologie*.

4.5.4.3 La directive de détachement

La *directive ontologique de détachement* qui suit est reprise de Denis Miéville (MIEVILLE, 2004, p. 87) ; elle est similaire à la *règle de détachement* de la *protothétique* :

RD 4.5-30 Une inscription *T* a le statut de thèse conforme à la directive de détachement en fonction de thèses *A* et *B* préalablement inscrites et issues de la base axiomatique A1-A3 contenant les catégories syntaxico-sémantiques *S* et *S/SS*, et leur contexte associé : (- -), ainsi que le foncteur constant de biconditionnelle : ≡, et de l'axiome de l'ontologie, A_o4 contenant les catégories N et S/NN, et leur contexte associé : {- -}, ainsi que les foncteurs constants : ε, ~, ∧, ⊃ respectivement associé aux contextes : {- -}, (-) et (--), si et seulement si toutes les conditions suivantes sont remplies :

RD 4.5-31 1. L'inscription *B* est une thèse actuellement inscrite dans le système et est une proposition biconditionnelle.

RD 4.5-32 2. L'inscription *A* est une thèse actuellement inscrite dans le système ; elle est équiforme au premier argument de *B*.

RD 4.5-33 3. *T* est équiforme au deuxième argument de *B*.

B : ≡(AT)
A
T

Appliquons cette règle pour la suppression d'un *nom général* dans la structure d'exemples de tableau de bord et présentée graphiquement dans la *Figure 34, p. 65*. Envisageons de supprimer le nom Boissons en maintenant l'étendue de la propriété Produits. Nous avons comme *transposition* initiale :

F 4.84. Par définitions :

$G =_{df} val_4$
$a =_{df}$ Vin
$b =_{df}$ Boissons
$c =_{df}$ Produits

Par la *thèse-définition* 25 :

$\lfloor ab \rfloor \lceil \equiv(\hat{C}\{ab\} \wedge (\sim(\lfloor G \rfloor \lceil \sim(\varepsilon\{Ga\})\rceil) \lfloor G \rfloor \lceil \supset(\varepsilon\{Ga\}\ \varepsilon\{Gb\})\rceil))\rceil$
$\lfloor bc \rfloor \lceil \equiv(\hat{C}\{bc\} \wedge (\sim(\lfloor G \rfloor \lceil \sim(\varepsilon\{Gb\})\rceil) \lfloor G \rfloor \lceil \supset(\varepsilon\{Gb\}\ \varepsilon\{Gc\})\rceil))\rceil$
$\lfloor ac \rfloor \lceil \equiv(\hat{C}\{ac\} \wedge (\sim(\lfloor G \rfloor \lceil \sim(\varepsilon\{Ga\})\rceil) \lfloor G \rfloor \lceil \supset(\varepsilon\{Ga\}\ \varepsilon\{Gc\})\rceil))\rceil$

Par notre *thèse-définition F 4.81, p. 66* :
$\lfloor abc \rfloor \lceil \equiv(\bigsqcup\{ac\} \supset (\wedge(\wedge(\sim(\lfloor G \rfloor \lceil \sim(\varepsilon\{Ga\})\rceil) \lfloor G \rfloor \lceil \supset(\varepsilon\{Ga\}\ \varepsilon\{Gb\})\rceil)$
$\wedge(\sim(\lfloor G \rfloor \lceil \sim(\varepsilon\{Gb\})\rceil) \lfloor G \rfloor \lceil \supset(\varepsilon\{Gb\}\ \varepsilon\{Gc\})\rceil))$
$\wedge(\sim(\lfloor G \rfloor \lceil \sim(\varepsilon\{Ga\})\rceil) \lfloor G \rfloor \lceil \supset(\varepsilon\{Ga\}\ \varepsilon\{Gc\})\rceil)))\rceil$

Nous acceptons pour l'instant que ce que le *système* connaît, y inclus notre nouvelle *inscription* et des *thèses* intermédiaires résultats d'une *déduction*, nous obtenons les *fonctions logiques* suivantes :

F 4.85.
$\equiv(\wedge(\wedge(\sim(\lfloor G \rfloor \lceil \sim(\varepsilon\{Ga\})\rceil) \lfloor G \rfloor \lceil \supset(\varepsilon\{Ga\}\ \varepsilon\{Gb\})\rceil)$
$\wedge(\sim(\lfloor G \rfloor \lceil \sim(\varepsilon\{Gb\})\rceil) \lfloor G \rfloor \lceil \supset(\varepsilon\{Gb\}\ \varepsilon\{Gc\})\rceil))$
$\wedge(\sim(\lfloor G \rfloor \lceil \sim(\varepsilon\{Ga\})\rceil) \lfloor G \rfloor \lceil \supset(\varepsilon\{Ga\}\ \varepsilon\{Gc\})\rceil))$ / A et description de la structure

$\wedge(\wedge(\sim(\lfloor G \rfloor \lceil \sim(\varepsilon\{Ga\})\rceil) \lfloor G \rfloor \lceil \supset(\varepsilon\{Ga\}\ \varepsilon\{Gb\})\rceil)$
$\wedge(\sim(\lfloor G \rfloor \lceil \sim(\varepsilon\{Gb\})\rceil) \lfloor G \rfloor \lceil \supset(\varepsilon\{Gb\}\ \varepsilon\{Gc\})\rceil)))$ / B et *nom général* à supprimer

$\wedge(\sim(\lfloor G \rfloor \lceil \sim(\varepsilon\{Ga\})\rceil) \lfloor G \rfloor \lceil \supset(\varepsilon\{Ga\}\ \varepsilon\{Gc\})\rceil)$/ détachement, propriété Produits maintenue

De façon résumée et schématique :

F 4.86.
$\equiv(\wedge(\hat{C}\{ab\}\hat{C}\{bc\})\hat{C}\{ac\})$ / A et description de la structure
$\wedge(\hat{C}\{ab\}\hat{C}\{bc\})$ / B et *nom général* à supprimer

$\hat{C}\{ac\}$ / détachement, propriété Produits maintenue

Nous avons pu montrer, qu'à un moment de notre raisonnement (sans tenir compte d'une *quantification* des *variables*), nous avions les éléments nécessaires pour procéder à la suppression d'un *nom général* et de ses dépendances tout en maintenant de la propriété englobante.

4.5.4.4 Les deux directives d'extensionalité de l'ontologie

La *directive d'extensionalité ontologique de type protothétique* est reprise et adaptée de Denis Miéville (MIEVILLE, 2004, p. 91) ; elle est similaire à la *directive d'extensionalité* de la *protothétique* (*4.4.6.4 La directive d'extensionalité, p. 51*) :

RD 4.5-34 Une inscription *T* a le statut de thèse conforme à la directive d'extensionalité ontologique de type protothétique en fonction de la dernière thèse inscrite du système et issue de la base axiomatique A1-A3 contenant les catégories syntaxico-sémantiques *S* et *S/SS*, et leur contexte associé : (- -), ainsi que le foncteur constant de biconditionnelle : ≡, et l'axiome de l'ontologie, A_o4 contenant les catégories *N* et *S/NN*, et leur contexte associé : {- -}, ainsi que les foncteurs constants : ε, ~, ∧, ⊃ respectivement associé aux contextes : {- -}, (-) et (--), si et seulement si toutes les conditions suivantes sont remplies :

RD 4.5-35 1. La catégorie *Cp* (catégorie d'un foncteur de type protothétique) est inscrite actuellement dans le système.

RD 4.5-36 2. La catégorie *S/Cp* est inscrite dans le système.

RD 4.5-37 3. *T* est une généralisation conforme au schéma suivant :

$$\lfloor fg \rfloor \lceil \equiv (\lfloor v_1 \ldots v_i \rfloor \lceil \equiv (f(v_1 \ldots v_i) g(v_1 \ldots v_i)) \rceil \lfloor \Theta \rfloor \lceil \equiv (\Theta[f]\Theta[g]) \rceil) \rceil$$ [32]

Cette règle s'applique aux *catégories syntaxico-sémantiques* dont le *foncteur* le plus à gauche est de la *catégorie S*. Pour introduire la *directive d'extensionalité ontologique de type ontologique*, avec le *foncteur* le plus à gauche de la *catégorie N*, Denis Miéville apporte le commentaire :

> Ainsi, si dans le système sont actuellement inscrits des foncteurs de type ontologique de la catégorie *C* et un foncteur de la catégorie *K* tel que *K* est de la catégorie *S/C*, alors quels que soient les foncteurs de la catégorie *C*, si *f* et *g* sont « équivalents » ; alors tout ce qui « se dit » de *f* est « équivalent » à ce qui « se dit » de *g*. (MIEVILLE, 2004, p. 93)

Soit la directive extraite du fascicule de Denis Miéville (MIEVILLE, 2004, p. 93) :

RD 4.5-38 Une inscription *T* a le statut de thèse conforme à la directive d'extensionalité ontologique de type ontologique en fonction de la dernière thèse inscrite du système et issue de la base axiomatique A1-A3 contenant les catégories syntaxico-sémantiques *S* et *S/SS*, et leur contexte associé : (- -), ainsi que le foncteur constant de biconditionnelle : ≡, et l'axiome de l'ontologie, A_o4 contenant les catégories *N* et *S/NN*, et leur contexte associé : {- -}, ainsi que les foncteurs constants : ε, ~, ∧ et ⊃ respectivement associé aux contextes : {- -}, (-) et (--), si et seulement si toutes les conditions suivantes sont remplies :

RD 4.5-39 1. La catégorie *Co* (catégorie d'un foncteur de type ontologique) est inscrite actuellement dans le système.

RD 4.5-40 2. La catégorie *S/Co* est également inscrite.

RD 4.5-41 3. *T* est une généralisation conforme au schéma suivant :

$$\lfloor fg \rfloor \lceil \equiv (\lfloor Av_1 \ldots v_i \rfloor \lceil \equiv (\varepsilon\{Af(v_1 \ldots v_i)\} \varepsilon\{Ag(v_1 \ldots v_i)\}) \rceil \lfloor \Theta \rfloor \lceil \equiv (\Theta[f]\Theta[g]) \rceil) \rceil$$

Introduisons encore la *thèse-définition* suivante :

F 4.87. D43, sans interprétation :

$$\lfloor \alpha \rfloor \lceil \equiv (\bullet \subset \alpha \supset \lfloor b \rfloor \lceil \varepsilon\{b\ \alpha<b>\} \rceil) \rceil$$

Voir Denis Miéville (MIEVILLE, 2004, p. 95).

Catégories	Constantes	Contextes
S/(N/N)	•	⊂-⊃

Ce qui nous permet d'exprimer la *thèse* d'extensionalité associée à la *catégorie N/N* :

F 4.88. $$\lfloor XY \rfloor \lceil \equiv (\lfloor Ab \rfloor \lceil \equiv (\varepsilon\{AX<b>\}\ \varepsilon\{AY<b>\}) \rceil \lfloor \Omega \rfloor \lceil \equiv (\Omega \subset X \supset \Omega \subset Y \supset) \rceil) \rceil$$

Avec ces dernières règles nous avons terminés notre survol de l'*ontologie*.

Dans les réponses à nos questions, il ne nous reste plus qu'à trouver une démarche nous permettant d'opérer une insertion dans nos structures. Nous étudierons cet aspect dans la section *4.7 Synthèse concernant la transposition, p. 73*.

4.6 Les interprétations de foncteurs et matrices de vérités

Dans le cadre de nos exemples, la véracité de n'importe quelle *inscription* que connaît le *système* doit pouvoir être évaluée. Si cela n'était pas réalisable, nous ne pourrions pas appliquer à nos *transpositions* les *restructurations* que nous avons citées plus haut. A quoi

[32] Les *signes* ombrés :(et) sont des métasignes.

servirait-il d'avoir supprimé un *nom général* sans savoir s'il est toujours vrai que l'étendue de la propriété englobante est maintenue ?

Le *foncteur constant* de *biconditionnelle* : ≡ et les *foncteurs constants* comme : ε, ~, ∧, ⊃… sont interprétés avec un table de vérité qui proposent un calcul propositionnel. Les *foncteurs quantifiés* n'offrent pas cette interprétation. La table doit se déduire de la forme d'un *axiome* ou d'une *thèse-définition*. Afin de poser le sujet, prenons le troisième *axiome* de la *protothétique* :

F 4.89. ⌊pg⌋⌈≡(⌊f⌋⌈≡(g(pp) ≡(⌊r⌋⌈≡(f(rr) g(pp))⌉⌊r⌋⌈≡(f(rr) g(≡(p⌊q⌋⌈q⌉) p))⌉))⌉⌊q⌋⌈g(pq)⌉)⌉

Explosons le *sous-quantificateur* entre première et deuxième équivalence :

F 4.90. ⌊f⌋⌈≡(g(pp) ≡(⌊r⌋⌈≡(f(rr) g(pp))⌉⌊r⌋⌈≡(f(rr) g(≡(p⌊q⌋⌈q⌉) p))⌉))⌉ /1ère équivalence

⌊q⌋⌈g(pq)⌉) /2nd équivalence

Trois *variables* p, q et r apparaissent dans l'*axiome*. Soit pour traiter l'ensemble des combinaisons du résultat d'un quelconque *foncteur* binaire : $2^8 = 256$ tables. Décomposons encore nos équivalences dans l'optique d'isoler chaque *expression* constituée d'un des *foncteurs* f et g, tous deux des *foncteurs* binaires de *catégorie syntaxico-sémantique S/SS* et donnons le nombre de combinaison des interprétations de leurs évaluations :

F 4.91.
(1) g(pq), soit 16 sur 256
(2') g(pp), soit 4 sur 256
(2'') g(pp), soit 4 sur 256
(3') f(rr), soit 4 sur 256
(3'') f(rr), soit 4 sur 256
(4) g(≡(p⌊q⌋⌈q⌉) p) ou val(≡(p⌊q⌋⌈q⌉))=~(p) donc g(≡(p~(p)) selon la thèse D4 (*F 4.16, p. 45*) soit 4 sur 256

Nous avons six *expressions* dont l'évaluation doit se combiner entre elles. Donc, $2^6 = 64$ valeurs par table à traiter avec le *connecteur* de la *biconditionnelle*, soit 2^{64}, ce qui fait beaucoup. Nous avons à faire à un *axiome* et sa table de vérité est toujours le « vrai », ce qui nous permet d'éliminer un nombre important de calculs ; il n'en reste pas moins qu'il n'est pas pensable d'entreprendre une analyse sans l'ordinateur qui nous aiderait à trouver des sortes de familles d'interprétations des *foncteurs*. Revenons aux travaux de Denis Miéville qui propose une approche qui pourrait être aussi traiter par ordinateur. Décrivons cette démarche très astucieuse.

Elle s'inspire de ce qui a été nommé le « modèle naturel » (RICKEY, 1985) :

> Un tel modèle est la donnée de deux extensions *N* et *O*, d'une relation de désignation entre *N* et *O* et d'une représentation ε* de l'epsilon ε dans le modèle. […] La première extension *N* est caractérisée par une liste de noms : n_1, n_2, n_3, … ; la deuxième extension *O* est constituée de signes d'objets : o_1, o_2, o_3, …. L'extension *N* doit contenir au moins un nom : quant à l'extension *O*, elle peut être vide, finie ou infinie. La relation de désignation, par rapport à un modèle particulier, lie un nom à un objet, à une extension d'objets ou à aucun objet. La relation epsilonienne entre deux noms « n_i ε* n_j » dans un modèle *M* est réalisée ssi
>
> il existe un o_i dans M tel que n_i désigne uniquement cet objet o_i
>
> ou
>
> il existe un objet dans *M* tel que n_i et n_j désigne cet objet, n_j pouvant désigner cet objet ou une extension à laquelle il appartient.
>
> Cette réalisation epsilonienne n'est pas réalisée autrement.
>
> […] à un instant donné de son développement une ontologie contient un nombre fini de thèses et donc d'inscriptions nominales N_k. Une interprétation *I* consiste alors en une application *i* entre les noms « syntaxique » dans le développement *hic et nunc* d'une ontologie *ONT*, et les « noms sémantiques » du modèle.
>
> i : Nk → N.

> L'application / est ainsi une application évolutive, une application qui s'accorde au développement d'une ontologie. L'interprétation / dispose également d'une application d'évaluation qui s'établit ainsi :
>
> VAL(A ε b) est le vrai ssi i(A) ε* i(b).
>
> (MIEVILLE, 2004, p. 134)

Cette approche et ses fondements appellent quelques remarques.

Premièrement, il est question de se donner un modèle qui parle de l'interprétation. Chose qui apparaît pour la première fois dans ce que nous avons rapporté des *SL*. La démarche établit une application entre la couche syntaxique et sémantique.

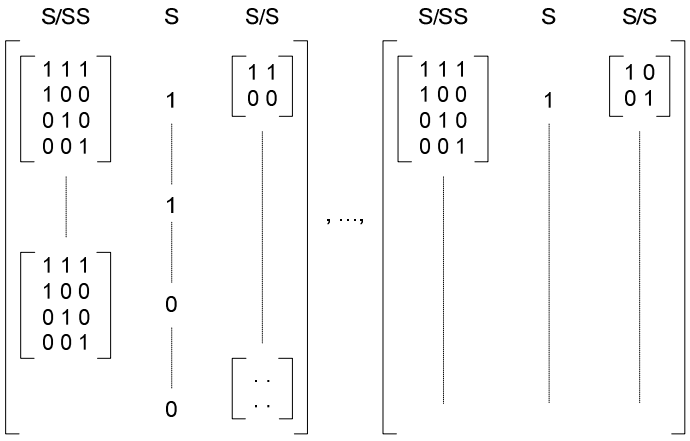
Deuxièmement, le raisonnement se base sur le développement de l'*ontologie* qui se fait petit à petit. Il tient compte de la construction développementale d'un *SL*. Il est donc évolutif. Cela nous ramène à notre titre et à sa motivation : […] *systèmes évolutifs de connaissances*.

Troisièmement, il apparaît que le modèle est une notion similaire à notre *domaine de connaissances*.

Il s'agit maintenant de comprendre[33] comment on peut attribuer une signification à un *foncteur*. Ce passage au niveau opératoire se base sur la *catégorie* d'un *foncteur*. Dès lors, le raisonnement ne cherche pas à déduire une interprétation à partir d'une *inscription* donnée du *système*, mais s'appuie sur la *catégorie*. Denis Miéville en choisissant une *catégorie* spécifique construit les bases opératoires :

> Soit la catégorie *(S/S)/(S/SS)/S*. Elle contient trois catégories constituantes : *S, S/S* et *S/SS*. Il s'agit d'attribuer à chacune d'entre elles une extension significative ES_c.
>
> Pour l'extension ES_S, il semble raisonnable d'attribuer les deux valeurs de vérité
>
> ES_S : 1 et 0 ;
>
> Pour l'extension $ES_{S/S}$, il s'agit, dans un premier temps, d'offrir une représentation des quatre opérateurs unaires classiques ;
>
> $ES_{S/S}$: as(-), vr(-), fl(-) et ~(-).
>
> Pour l'extension $ES_{S/SS}$, il convient, toujours dans un premier temps, d'offrir une représentation des seize opérateurs binaires ;
>
> $ES_{S/SS}$: ≡(--), ⊃(--), …, ↓(--).
>
> Je représenterai chacune de ces extensions significatives sous la forme d'une signature matricielle ainsi :
>
> ES_S : $\begin{bmatrix}1\end{bmatrix}$ et $\begin{bmatrix}0\end{bmatrix}$
>
> $ES_{S/S}$: $\begin{bmatrix}1&1\\0&0\end{bmatrix}, \begin{bmatrix}1&0\\0&1\end{bmatrix}, \begin{bmatrix}1&0\\0&0\end{bmatrix}, \begin{bmatrix}1&1\\0&1\end{bmatrix}$
>
> $ES_{S/SS}$: $\begin{bmatrix}1&1&1\\1&0&0\\0&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}1&1&1\\1&0&0\\0&1&1\\0&0&1\end{bmatrix}, \begin{bmatrix}1&1&1\\1&0&0\\0&1&0\\0&0&0\end{bmatrix}, \ldots, \begin{bmatrix}1&1&0\\1&0&0\\0&1&0\\0&0&1\end{bmatrix}$
>
> […] chaque signature est organisée de manière à offrir tout d'abord et dans un ordre canonique la signification des opérandes puis, en place finale, celle de la résultante.
>
> Il est également possible d'aborder maintenant, en termes de signatures l'extension significative de la catégorie *(S/S)/(S/SS)S* :

[33] Denis Miéville se restreint aux *catégories* de la *protothétique*.



(MIEVILLE, 2004, p. 136)

Si l'on considère la cardinalité de l'extension des significations combinées et l'exemple ci-dessus : $\#(c_1/c_2 \ldots c_n) = \#c_1^{\#c2,\ \#c3 \ldots cn}$ nous avons pour l'extension de $ES_{(S/S)/(S/SS)S}$ 4^{32} signatures. Denis Miéville continue le raisonnement pour les définitions paramétrées qui impliquent des signatures de signatures imbriquées (MIEVILLE, 2004, p. 138 et suivantes).

Nous avons maintenant une voie qui nous permet d'envisager l'interprétation d'un *foncteur* et cela sur la base de sa *catégorie*. Le modèle étant décidable, nous pouvons envisager de le *transposer* sur ordinateur étant donné le nombre de combinaisons à traiter[34].

4.7 Synthèse concernant la transposition

Au fur et à mesure de la présentation de l'*ontologie*, nous avons mentionné des échantillons d'apports possibles pour répondre aux questions que nous nous posions dans nos deux premiers chapitres.

Nous avons classé en quatre niveaux de difficultés que nous rencontrions :

1) les *difficultés de calculs* ou ce que nous avons mis du ressort des notions ensemblistes utilisées dans la pratique par les spécialistes des systèmes multidimensionnels destinés aux tableaux de bord. Nous avons retenu un modèle qui accepte une architecture structurée en *dimensions* et l'aide du produit cartésien ;

2) les aspects taxonomiques ou les *difficultés de taxonomie* que nous avons qualifiés de descriptifs et statiques. Ce que nous avons ensuite et durant notre parcours de l'*ontologie* désignés comme des définitions ;

3) les *difficultés de restructuration* que nous avons retenues comme exemples fréquents d'anomalies et dont nous avons besoin pour rendre opérationnel notre *domaines de connaissances*. Jusqu'à présent, nous n'avons pas apporté de moyen de *transposer* la *transformation* élémentaire d'insertion ; ce que nous élaborons plus loin ;

4) les *difficultés du maintien d'une propriété* en traitant la suppression d'un *nom général* dans une structure. Nous avons traité cet aspect à travers la *thèse-définition* de l'inclusion.

Nous commençons par un bilan des acquis et des précautions à suivre, pour ensuite se pencher sur une approche de la mise en œuvre de l'insertion.

[34] Notre logiciel prototype RE V4.4 n'intègre pas encore cette fonction de calcul et de « détermination » des matrices de vérités.

4.7.1 Un système multidimensionnel et ses calculs

En remplacement à une forme arborescente, l'architecture de systèmes multidimensionnels que nous donnent les spécialistes des tableaux de bord, nous simplifie la tâche puisque qu'elle n'établit qu'un seul « lien » de *dimension* à *dimension*, celui qui est calculé par le produit cartésien. Par conséquent, nous pouvons décrire ces *dimensions* et la structure établie entre leurs éléments de façon indépendante.

Nous n'avons pas jugé utile de présenter les aspects arithmétiques qui nous permettent d'exécuter les calculs selon les structures indiquées entre les éléments dans les *dimensions*. Nous avons néanmoins passé par un artifice en ce qui concerne les valeurs quand nous avons abordé la *thèse-définition* D19 (*F 4.54, p. 59*) en se donnant des *noms singuliers* comme val_1, val_2, … qui serviront de *variables* informatiques dans une *solution* informatique.

Nous devrions aborder le sujet sous un angle additionnel en traitant les n-uplets générés comme des *noms singuliers*. Ce qui à part le volume des données à traiter s'envisage comme étant du même genre de *transpositions* que l'on réalise dans la section qui suit.

4.7.2 Taxonomie et définitions

Nous nous sommes d'abord penché sur les aspects de dénotation ou désignation. Nous avons nommé nos objets, les valeurs, par des *noms singuliers* et nous avons repris les *noms* de chaque niveau de nos structures comme *noms généraux*. Par exemple :

F 4.92. $A =_{df} val_{12}$ / *nom singulier*

$a =_{df}$ Whisky / *nom général* repris du dernier niveau de notre arborescence

$c =_{df}$ Produits / *nom général* repris du premier niveau de notre arborescence

Nous avons ensuite établi une relation à travers la proposition construite avec l'*epsilon* entre l'objet désigné par les *noms singuliers* et *généraux* :

F 4.93. $\varepsilon(Aa)$ / A « est un des » a ou val_{12} « est un des » Whisky

$\varepsilon(Ac)$ / A « est un des » c ou val_{12} « est un des » Produits

Notre parcours sur la construction de l'*ontologie* à partir de la *procédure définitoire* nous a permis d'exploiter la *thèse-définition* D17 (*F 4.52, p. 58*) sur l'existence :

F 4.94. $\lfloor a \rfloor \lceil \equiv (!\{a\} \sim (\lfloor A \rfloor \lceil \sim (\varepsilon\{Aa\}) \rceil)) \rceil$ / il existe au moins un objet dénoté par a, a n'est pas vide

$\lfloor a \rfloor \lceil \equiv (!\{c\} \sim (\lfloor A \rfloor \lceil \sim (\varepsilon\{Ac\}) \rceil)) \rceil$ / il existe au moins un objet dénoté par c, c n'est pas vide

Un *nom général* vide entraînerait indiscutablement une anomalie dans les tableaux de bord. Dans le même ordre d'idées, nous nous sommes arrêté sur les *thèses* traitant de l'égalité ou de la différence entre objets. Nous nous servons de la *thèse-définition* D21 (*F 4.56, p. 59*) qui nous assure que dans une même *dimension* nous n'utilisons pas un même *nom* :

F 4.95. $B =_{df} val_{15}$

$\lfloor AB \rfloor \lceil \equiv (\neq\{AB\} \wedge (\wedge (\varepsilon\{AA\} \varepsilon\{BB\}) \sim (=\{AB\}))) \rceil$ / A et B désignent des objets différents

Avec la *thèse-définition* D25 (*F 4.60, 60*), l'inclusion forte, nous avons pu établir une relation entre deux *noms généraux* en notant en passant qu'elle tient compte de l'existence du *nom singulier* qui désigne l'objet :

F 4.96. $G =_{df} val_{16}$

$\lfloor ab \rfloor \lceil \equiv (\hat{C}\{ac\} \wedge (\sim (\lfloor G \rfloor \lceil \sim (\varepsilon\{Ga\}) \rceil) \lfloor G \rfloor \lceil \supset (\varepsilon\{Ga\} \varepsilon\{Gc\}) \rceil)) \rceil$ / a est inclu dans c

Il convient encore de mentionner que dans une *transposition* complète d'un tableau de bord complexe nous ferions appel à une grande majorité des *thèses-définition* de l'*ontologie* sachant qu'elle en compte quarante-six dans l'élaboration présentée par Denis Miéville (MIEVILLE, 2004).

En suivant la *procédure définitoire* et en s'inspirant de la *thèse* D25 de l'inclusion, nous avons introduit (*F 4.81, p. 66*) un nouveau *foncteur constant* (Ɥ, de la *catégorie S/SS*, avec un *contexte* {--}) qui reflète la propriété de transitivité de l'inclusion :

F 4.97. ⌊abc⌋⌈≡(Ɥ{ac} ⊃(∧(∧(~(⌊G⌋⌈~(ε{Ga})⌉)⌊G⌋⌈⊃(ε{Ga} ε{Gb})⌉)
∧(~(⌊G⌋⌈~(ε{Gb})⌉)⌊G⌋⌈⊃(ε{Gb} ε{Gc})⌉))
∧(~(⌊G⌋⌈~(ε{Ga})⌉)⌊G⌋⌈⊃(ε{Ga} ε{Gc})⌉)))⌉

Nous aimerions encore traiter l'ensemble des nœuds d'un niveau de nos structures dans une seule *inscription*. Par exemple, dire que tous les nœuds, du niveau 2, qui déterminent le type de produits Aliments et Boissons sont des Produits comestibles. Nous exploitons la *directive d'extensionalité* de l'*ontologie* pour traiter ce cas :

F 4.98. ⌊XY⌋⌈≡(⌊Ab⌋⌈≡(ε{AX<b>} ε{AY<b>}⌉⌊Ω⌋⌈≡(Ω⊂X⊃ Ω⊂Y⊃)⌉)⌉ / conforme à la directive

d =$_{df}$ niveau 2

⌊XY⌋⌈≡(⌊Ad⌋⌈≡(ε{AX<d>} ε{AY<d>}⌉⌊Ω⌋⌈≡(Ω⊂X⊃ Ω⊂Y⊃)⌉)⌉ / substitution b/d

⌊XY⌋⌈≡(⌊Gd⌋⌈≡(ε{GX<d>} ε{GY<d>}⌉⌊Ω⌋⌈≡(Ω⊂X⊃ Ω⊂Y⊃)⌉)⌉ / substitution A/G

ε{AX<d>} : A « est un des - être un niveau inférieur ou égal à 2 - »

ε{AY<d>} : A « est un des - être un niveau strictement inférieur à 1 - »

Ω : hérite du niveau 1

Ce qui vient d'être résumé dans cette section constitue un exemple de *transposition* qui nous permet de couvrir tous les aspects taxonomiques nécessaires à notre *domaine de connaissances*.

Dans une première esquisse, nous avons abordé un cas estimé du domaine de la substitution que nous avions classé dans les questions relatives aux *transformations* (*F 4.82, p. 67*). Nous avons conclu qu'il s'agissait simplement d'une redéfinition ou de l'affectation d'une « constante de nom » à une *variable de nom.*

F 4.99. a =$_{df}$ Vin

a =$_{df}$ Bière / re-définition

4.8 Les questions de restructurations et déduction

Pour aboutir et couvrir non seulement l'ensemble des *transpositions*, mais aussi des *restructurations* que nous nous sommes donnés, nous avons besoin de passer par des *déductions*. Un certain nombre de métarègles peuvent encore être retenues :

(a) la *commutation des arguments* d'une *fonction logique*, COM (MIEVILLE, 2007, p. 90) ;

(b) l'*introduction de la biconditionnelle*, INT (MIEVILLE, 2007, p. 90). La métarègle stipule que si les *inscriptions A* et *B* sont des *thèses* du *système*, alors ≡(AB) en est aussi une ;

(c) si A ≡ (B ≡ C) est une *thèse* du *système*, alors (A ≡ B) ≡ C est aussi une *thèse*, soit la métarègle, ASS (MIEVILLE, 2007, p. 92).

Nous avons déjà, à travers Jean-Blaise Grize (section *4.1 Préliminaire concernant les logiques formelles du 1er ordre, p. 35*) donné une esquisse de la définition de *déduction*. Nous avons aussi défini la *preuve* (*4.1.6 Déduction et preuve, p. 37*). Nous n'avons pas jusqu'à présent précisé la nuance entre ces raisonnements et celui de *démonstration* qui a été utilisée pour déterminer les métarègles. Si une *déduction* se développe à partir d'hypothèses à vérifier, à l'aide des règles et à partir de ce que le *SL* connaît comme les *axiomes* ou les *inscriptions* qui ont été introduits au préalable. La dernière ligne d'une *déduction* est une conclusion syntaxique et la dernière ligne d'une *preuve* est un *théorème*.

Si une *déduction* ou une *preuve* sont de l'ordre de la logique, une *démonstration* est d'ordre métalogique. Dans les efforts de *transpositions*, nous utilisons la *déduction.*

Disposant de tous les matériaux théoriques qui constituent l'*ontologie*, revenons maintenant à la question de l'insertion. Nous reproduisons ici la *Figure 35. Tentative d'insertion, p. 68* qui résume l'opération de *transformation* à laquelle nous nous attaquons :

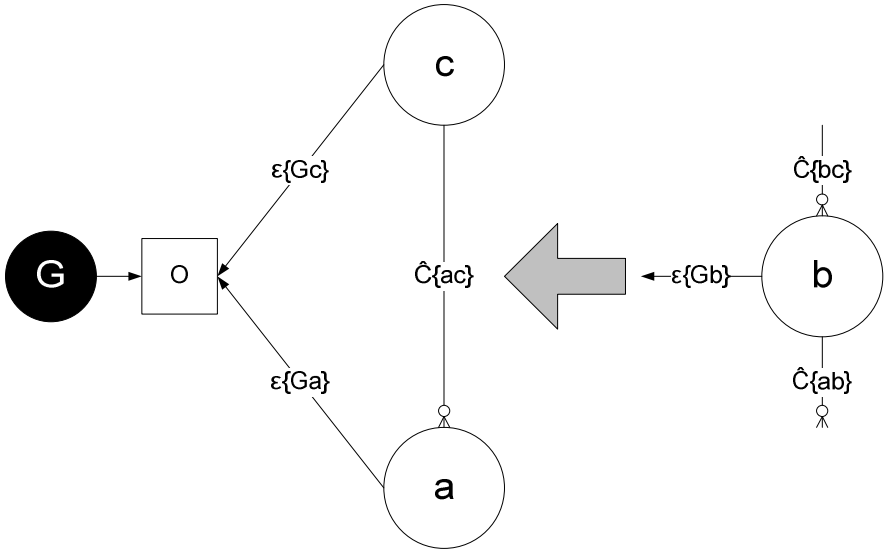


Figure 35bis.Tentative d'insertion

Comme nous l'avons vu dans la section *4.5.4.2 La directive ontologique de substitution, p. 67*, nous ne pouvons pas envisager d'insérer cette nouvelle structure par un remplacement ou une substitution conforme à la règle. En effet, la construction de l'*inscription E* dont nous avons besoin pour remplacer la proposition Ĉ{ac} ne peut pas être construite ou *transposée* de façon conforme à la *règle d'inférences de substitution*. Il ne nous reste plus qu'à décomposer cette *transformation* en opérations élémentaires. Reprenons les définitions de notre structure cible :

F 4.100. $G =_{df} val_4$ / par définition
$a =_{df}$ Vin / par définition
$c =_{df}$ Produits / par définition

D17 : ⌊a⌋⌈≡(!{a}~(⌊A⌋⌈~(ε{Aa})⌉))⌉ / thèse-définition 17
T1 : ⌊a⌋⌈≡(!{a}~(⌊G⌋⌈~(ε{Ga})⌉))⌉ / D17, sub. A/G
T2 : ⌊c⌋⌈≡(!{c}~(⌊G⌋⌈~(ε{Gc})⌉))⌉ / T1, sub. a/c

D25 : ⌊ab⌋⌈≡(Ĉ{ab}∧(~(⌊G⌋⌈~(ε{Ga})⌉)⌊G⌋⌈⊃(ε{Ga} ε{Gb})⌉))⌉ / thèse-définition 25
T3 : ⌊ac⌋⌈≡(Ĉ{ac}∧(~(⌊G⌋⌈~(ε{Ga})⌉)⌊G⌋⌈⊃(ε{Ga} ε{Gc})⌉))⌉ / D 25a, sub. b/c

Dans ce qui précède, nous nous assurons de l'existence d'au moins un désignateur G de l'objet et nous nous donnons une inclusion. Pour l'instant la *transposition* est suffisante pour décrire cette partie de notre *domaine de connaissances.*

Pour traiter la structure à insérer, dans une première phase et en gardant comme pivot l'objet désigné par le *nom singulier* G, introduisons comme définitions le *nom général* b et en appliquant la *thèse-définition* D17 (*F 4.52, p. 58*) imposons lui le fait d'« exister » (il existe au moins un a) :

F 4.101. $b =_{df}$ Boissons / par définition
T4 : ⌊b⌋⌈≡(!{b}~(⌊G⌋⌈~(ε{Gb})⌉))⌉ / D17, sub. A/G, a/b

Dans une deuxième phase, *transposons* les deux inclusions, Ĉ{ab} et Ĉ{bc} , en utilisant la *thèse-définition* 25 :

F 4.102. D25 : ⌊ab⌋⌈≡(Ĉ{ab}∧(~(⌊G⌋⌈~(ε{Ga})⌉)⌊G⌋⌈⊃(ε{Ga} ε{Gb})⌉))⌉ / thèse-définition 25
T5 : ⌊bc⌋⌈≡(Ĉ{bc}∧(~(⌊G⌋⌈~(ε{Gb})⌉)⌊G⌋⌈⊃(ε{Gb} ε{Gc})⌉))⌉ / D25, sub. a/b, b/c

La troisième phase touche, à nouveau, la frontière entre la *transposition* et l'articulation purement logique du *système*. Il s'agit d'exprimer la transitivité entre nos propositions d'inclusion. Soit de formuler « Ĉ{ab} et Ĉ{bc} » et « Ĉ{ab} et Ĉ{bc} - est équivalent à - Ĉ{ac} ». Nous avons, donc, deux options : (1) nous élaborons une *déduction* qui, à un moment donné, nous permet de construire une conjonction (un peu comme nous pouvons le faire avec la *biconditionnelle* d'une *fonction logique* à l'aide de la métarègle INT) ou (2) nous établissons une simple traduction du *domaine de connaissances* que nous donnons au *système* comme hypothèse, mais qui dans n'importe quelle forme respecte la *procédure définitoire*. Prenons la deuxième option :

F 4.103. E1 : ∧(⌊ab⌋⌈≡(Ĉ{ab}∧(~(⌊G⌋⌈~(ε{Ga})⌉)⌊G⌋⌈⊃(ε{Ga} ε{Gb})⌉))⌉
⌊bc⌋⌈≡(Ĉ{bc}∧(~(⌊G⌋⌈~(ε{Gb})⌉)⌊G⌋⌈⊃(ε{Gb} ε{Gc})⌉))⌉) / essai de transposition

Pour être conforme à la *règle d'inférences de définition*, il s'agirait d'obtenir soit une *généralisation*, soit *une fonction logique*, soit encore une *thèse* déduite qui est validée comme *thèse* ou *inscription* du *système* ; ce qui n'est pas le cas de E1. Dans une optique exploratoire, avec ce que nous connaissons jusqu'à ce niveau procédons à la distribution des *quantificateurs* des *thèses* D25 et T5 :

F 4.104. T6 : ≡ (⌊ab⌋⌈(Ĉ{ab}⌉⌊ab⌋⌈∧(~(⌊G⌋⌈~(ε{Ga})⌉)⌊G⌋⌈⊃(ε{Ga} ε{Gb})⌉) / D25, dist. a, b
T7 : ≡ (⌊bc⌋⌈(Ĉ{bc}⌉⌊bc⌋⌈∧(~(⌊G⌋⌈~(ε{Gb})⌉)⌊G⌋⌈⊃(ε{Gb} ε{Gc})⌉) / T5, dist. b, c

Comme quatrième phase, posons les trois hypothèses suivantes :

F 4.105. H1 : ⌊ab⌋⌈∧(~(⌊G⌋⌈~(ε{Ga})⌉)⌊G⌋⌈⊃(ε{Ga} ε{Gb})⌉ / hyp
H2 : ⌊bc⌋⌈∧(~(⌊G⌋⌈~(ε{Gb})⌉)⌊G⌋⌈⊃(ε{Gb} ε{Gc})⌉ / hyp
H3 : ⌊ac⌋⌈∧(~(⌊G⌋⌈~(ε{Ga})⌉)⌊G⌋⌈⊃(ε{Ga} ε{Gc})⌉ / hyp

Ceci est acceptable puisque ces hypothèses sont directement issues de notre *transposition*. Pour la cinquième phase nous utilisons la métarègle de commutativité :

F 4.106. T8 : ≡ (⌊ab⌋⌈∧(~(⌊G⌋⌈~(ε{Ga})⌉)⌊G⌋⌈⊃(ε{Ga} ε{Gb})⌉⌊ab⌋⌈(Ĉ{ab}⌉) / T6, COM
T9 : ≡ (⌊bc⌋⌈∧(~(⌊G⌋⌈~(ε{Gb})⌉)⌊G⌋⌈⊃(ε{Gb} ε{Gc})⌉⌊bc⌋⌈(Ĉ{bc}⌉) / T7, COM
T10 : ≡ (⌊ac⌋⌈(Ĉ{ac}⌉⌊bc⌋⌈∧(~(⌊G⌋⌈~(ε{Ga})⌉)⌊G⌋⌈⊃(ε{Ga} ε{Gc})⌉) / T3, dist. a, c
T11 : ≡ (⌊ac⌋⌈∧(~(⌊G⌋⌈~(ε{Ga})⌉)⌊G⌋⌈⊃(ε{Ga} ε{Gc})⌉⌊ac⌋⌈(Ĉ{ac}⌉) / T10, COM

Dans la sixième phase nous procédons au détachement du *definiendum* :

F 4.107. T12 : ⌊ab⌋⌈(Ĉ{ab}⌉ / T8, H1, dét.
T13 : ⌊bc⌋⌈(Ĉ{bc}⌉ / T9, H2, dét.
T14 : ⌊ac⌋⌈(Ĉ{ac}⌉ / T11, H3, dét.

A la phase sept, nous introduisons une nouvelle *thèse-définition* (en considérant que le *foncteur constant* Ц n'a pas encore été introduit dans le *système*) :

F 4.108. D46 : ⌊abc⌋⌈≡(Ц{ac} ⊃(∧(⌊ab⌋⌈(Ĉ{ab}⌉⌊bc⌋⌈(Ĉ{bc}⌉)⌊ac⌋⌈(Ĉ{ac}⌉)⌉
T15 : ≡(⌊ac⌋⌈Ц{ac}⌉⌊abc⌋⌈⊃(∧(⌊ab⌋⌈(Ĉ{ab}⌉⌊bc⌋⌈(Ĉ{bc}⌉)⌊ac⌋⌈(Ĉ{ac}⌉⌉) / D46, dist.

Nous pouvons encore procéder par *déduction* aux étapes qui suivent de la huitième phase :

F 4.109. H4 : ⌊ac⌋⌈Ц{ac}⌉ / hyp
T16 : ⌊abc⌋⌈⊃(∧(⌊ab⌋⌈(Ĉ{ab}⌉⌊bc⌋⌈(Ĉ{bc}⌉)⌊ac⌋⌈(Ĉ{ac}⌉⌉ / T15, H4, dét.

D'autres raisonnements peuvent être appliqués. Nous avons proposé un certain nombre de *transpositions* et de *thèses* qui en découlent. Il nous reste encore à affiner l'opération de suppression. Sous la contrainte que nous nous sommes donnée du « maintien de l'étendue d'une propriété », la suppression, principalement d'un *nom général*, s'est présentée comme une des opérations élémentaires que nous avons réalisée à l'aide de la *règle d'inférences de détachement*. Nous procédons par *déduction* :

F 4.110. H5 : ∧(⌊ab⌋⌈(Ĉ{ab}⌉⌊bc⌋⌈(Ĉ{bc}⌉) / hyp. (dérivée de nos définitions)
T17 : ⊃(∧(⌊ab⌋⌈(Ĉ{ab}⌉⌊bc⌋⌈(Ĉ{bc}⌉)⌊ac⌋⌈(Ĉ{ac}⌉ / T16, ∀e
T18 : ⌊ac⌋⌈(Ĉ{ac}⌉ / T17, ⊃e

Ceci termine nos considérations sur la *transposition* en insistant que la frontière entre la *transposition* et l'aspect purement théorique du *système* est difficile à déterminer. Les aspects théoriques, donc les *SL*, doivent nous assurer d'éviter des contradictions et nous devons nous assurer d'un maximum de systématisme et précision lors de la *transposition*. Evidemment, si nous devions durant une *déduction* obtenir une contradiction, nous pourrions en conclure deux choses : soit la *transposition* est imprécise soit la *déduction* est fausse.

4.9 Argument pour le développement d'outils d'aide et d'automation

Même avec une peu d'expérience, la manipulation des *signes*, des *expressions*, des *thèses*, des *règles d'inférences* n'est pas toujours très facile à mettre en œuvre. Si les logiciens sont familiers avec ces pratiques, ce n'est pas le cas des contrôleurs de gestion ou des spécialistes des tableaux de bord. Nous avons classé en quatre domaines possibles d'automatisation les difficultés que peuvent rencontrer les contrôleurs de gestion :

1) la *transposition* de la langue naturelle vers des définitions qui peuvent s'intégrer aux *inscriptions* des *SL* pose peu de problèmes quand il s'agit de représenter des petits exemples. Dans le cas des tableaux de bord, une société de moyenne importance peut traiter des milliers de produits. Pour faciliter un travail répétitif de traduction, une solution automatisée permettrait d'éviter de possibles erreurs ;

2) même avec un peu de pratique et en connaissant les *règles d'inférences* pratiquement par cœur, il n'est pas aisé d'assurer l'exhaustivité des filtres qu'il faut appliquer à une *expression* pour l'autoriser à entrer comme *thèse* dans le *système*. Même en parcourant à chaque nouvelle introduction la *règle d'inférences de définition*, il est très facile d'oublier ou d'insérer un *signe* de ponctuation et par conséquent d'obtenir une *expression* incorrecte. Un début d'automation pourrait traiter ces filtrages ;

3) comme les *règles d'inférences* et les *déductions* s'appliquent à ce qui a été préalablement introduit dans le *système* et dans la *bibliothèque* des *catégories*, cela nécessite à chaque étape de la construction du *système* de se référer et parcourir tout ce qui a déjà été introduit. Ce travail est consommateur de temps et pourrait faire l'objet d'une mémorisation et d'un parcours sur ordinateur ;

4) nous n'avons traité que des petites *inscriptions* s'inspirant d'exemples rudimentaires. Les *règles d'inférences transposant* un *domaine de connaissances* qui n'a pas été réduit à l'exemple peuvent générer des *inscriptions* longues et qui peuvent devenir difficiles à lire. En réponse à cette difficulté, l'ordinateur présente un intérêt certain.

Satisfait de cette étape d'exploration en faveur de l'utilisation d'un formalisme fondé sur les *SL*, nous nous sommes mis en quête d'une *solution* informatique qui réponde à ces nouveaux besoins.

4.10 Conclusion

Après avoir présenté la *protothétique*, fondement de l'*ontologie*, nous avons, au cours, de notre parcours sur les *SL*, révélé des esquisses possibles de *transposition* et *formalisation* du *domaine de connaissances* que nous nous étions donné. Nous avons, donc, abouti à notre recherche d'un *système* et nous nous appuyons sur l'*ontologie* de Leśniewski pour persévérer dans notre développement. Toutes les questions que nous nous sommes posées dans les chapitres précédents ont trouvé une réponse satisfaisante.

Toutefois, de nouvelles embuches ont fait surface comme la délimitation entre acte de *transposition* (procédure pratiquement en dehors du système) et les aspects purement théoriques du *système* étudié, comme la *déduction*. Nous avons malgré tout touché, superficiellement, une

approche qui établit un formalisme de cette frontière interprétative dans la présentation de l'interprétation des *foncteurs* (*4.6 Les interprétations de foncteurs et matrices de vérités, p. 70*) qui nous oriente vers des solutions futures. Quoiqu'il en soit, pour éviter un sujet qui constituerait à lui seul un ouvrage, nous avons pris des directions qui conviennent à nos besoins de traitement des tableaux de bord.

Parmi les nouvelles difficultés que nous avons rencontrées, nous avons aussi noté qu'il fallait une certaine pratique pour mettre en œuvre toute la mécanique leśniewskienne. Afin d'apporter une aide dans l'agilité nécessaire à déployer un formalisme comme ceux que nous avons abordés, nous nous proposons d'investiguer dans une *solution* informatique d'aide à la mise en œuvre des *SL*.

5. Une solution d'aide informatisée

Dans ce chapitre nous allons aborder une solution informatique d'aide à la mise en œuvre des SL. Nous commençons par recenser les entités constituantes de ces systèmes et de leurs processus de construction, ainsi que des éléments impliqués dans la transposition d'un domaine de connaissances vers le système afin de comprendre où et pour quelle raison une aide automatisée apporterait une valeur ajoutée.

Après avoir collecté et analysé les besoins, qui doivent nous conduire à un cahier des charges, nous élaborerons l'architecture d'une solution basée sur le principe de la « réécriture ». Nous développerons ensuite une grammaire, GRE qui définit un langage, LRE, assez général pour couvrir l'ensemble des cas à automatiser. Le LRE sert de langage grammatical aux GO pour décrire le langage, LO, utilisé par l'ontologie et le domaine de connaissances. Une fois la GRE défini, nous montrons qu'elle couvre la liste des besoins énumérés dans le cahier des charges que nous nous donnons.*

Nous présentons ensuite la solution informatique prototype, le RE, que nous avons développé qui traite la GRE et permet ainsi d'automatiser le traitement du LO. Nous abordons un exemple détaillé d'une grammaire. Nous donnons aussi dans ce cadre des recommandations quant à la programmation de grammaires GO_i.

Nous montrons, en fin de chapitre, que le LRE est la solution RE couvrent notre cahier des charges dans son ensemble.

5.1 Introduction

Nous avons commencé par classer les difficultés que nous avons rencontrées concernant les tableaux de bord (calculs, taxonomie, *restructuration*, maintien de la propriété). Nous avons ensuite sélectionné la théorie des ensembles et l'*ontologie* comme *systèmes* répondant à notre besoin de *formalisation*. Après avoir parcouru la *protothétique* et l'*ontologie*, nous avons pu vérifier que notre *domaine de connaissances* pouvait être *transposé* dans ces *systèmes*. Nous avons pu mettre en œuvre la mécanique des *SL* et monter qu'ils se présentaient comme le moyen d'éviter les anomalies rencontrées lors de la constitution des tableaux de bord. Nos intentions se dirigent maintenant vers l'élaboration d'une *solution* informatique qui a pour objectif d'aider le spécialiste des tableaux de bord à mettre en œuvre l'*ontologie* sans nécessairement être un logicien averti.

Nous allons définir un cahier des charges détaillé des compétences et des fonctions de la *solution* informatique. Mais nous savons déjà que les matériaux des *SL* : *signes*, *axiomes*, *thèses-définition*, *inscriptions*, *thèses*, *définition inductive et règles d'inférences* sont tous décrits par des langages[35]. Nous abrégeons ce langage par *LO*[36]. Qui dit langage dit grammaire ou ensemble de grammaires. Les *règles d'inférences de définition* sont des grammaires qui décrivent une *inscription* formée correctement pour être introduite dans le *système*. Nous avons non seulement l'intention de traiter les *inscriptions*, mais aussi les *règles d'inférences* qui pilotent leurs définitions, les deux niveaux appartenant au *LO*. Nous nommons l'ensemble des grammaires qui régissent tout ou partie du *LO*, *GO**.

[35] Le *système* étant lui-même un métalangage.

[36] Nous avons choisi *LO* comme abréviation mise pour *Langage Ontologique*. Il ne s'agit pas de « Le langage *ontologique* de Leśniewski » qui est plus complet et puissant que ce nous avons la prétention d'automatiser.

Pour que la *solution*[37] informatique puisse traiter les *GO**, nous devons la doter d'un langage grammatical, un langage qui parle de règles de grammaire que nous appelons le *LRE*, pour langage de réécriture, parce qu'il est basé sur le principe de réécriture que nous verrons dans ce chapitre. Comme pour ce qui précède, il faudra aussi nous donner une grammaire qui gouverne le *LRE* que nous appelons la *GRE*. Nous avons donc un modèle d'imbrications de langages qui parlent de langages chacun gouverné par sa ou ses grammaires. Soit la figure qui suit :

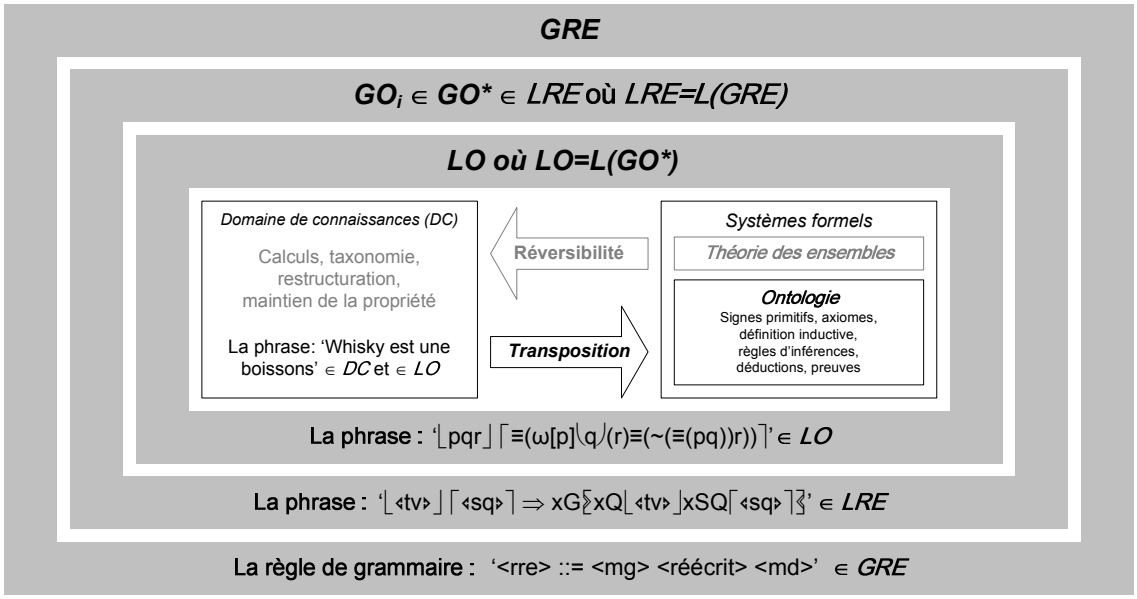


Figure 36. Positionnement du *LRE = L(GRE)*, *LO = L(GO*)* et *LO = L(L(GRE)*

Analysons cette figure. En partant du centre, nous prenons une *phrase* du *domaine de connaissances* : 'Whisky est une boisson', qui appartient au domaine et que nous voulons traiter dans le *LO* puisqu'elle est un des objets de nos *transpositions* et donc est traitée par l'*ontologie*. Tout ce qui peut être considéré comme matériaux linguistiques de l'*ontologie* constituent aussi des *phrases* du *LO* comme la *phrase* : '⌊pqr⌋⌈≡(ω[p]\q/(r)≡(~(≡(pq))r))⌉', une *thèse-définition* de la *protothétique*. La couche du *LO* est imbriquée dans la couche de la GO_i, qui est une des grammaires des *GO** et qui est elle-même une *phrase* du *LRE*. Par exemple, la *phrase* : '⌊‹tv›⌋⌈‹sq›⌉ ⇒ xG⦇xQ⌊‹v›⌋xSQ⌈‹sq›⌉⦈' qui, à partir d'une *inscription* quelconque l'analyse comme une *généralisation*. Le langage *LRE* est imbriqué dans la couche de la *GRE* qui est la grammaire qui le définit. Nous donnons comme exemple, la règle de grammaire de la *GRE* : '<rre> ::= <mg> <réécrit> <md>'.

Pour élaborer la grammaire *GRE* qui définit le langage *LRE*, nous utilisons ce que nous donne la théorie des *langages formels*. Toute la littérature à ce sujet offre un éventail important de concepts qui peuvent être mis en œuvre dans une *solution* informatique.

Cela nous conduit à poser une nouvelle hypothèse à vérifier : est-il concevable d'élaborer la *GRE* et donc le *LRE* qui couvrent les *phrases* du *domaine de connaissances*, les *phrases* qui parlent du procédé de *transposition*, les *phrases* de l'*ontologie* et de ses *règles d'inférences ?*

5.2 Analyse des besoins

En reprenant et analysant les difficultés que nous avons énumérées dans la section *4.9 Argument pour le développement d'outils d'aide et d'automation, p. 78*, nous allons établir le cahier des charges d'une *solution* informatique d'aide à la manipulation de certains aspects

[37] Afin d'éviter des confusions nous parlerons autant que possible de *solution* (informatique) et réservons la notion de *système* pour la logique.

des *SL* et d'aide à la *transposition* des *domaines de connaissances* sur lesquels nous appliquons ces *systèmes*. Ce cahier des charges se présente comme une liste de fonctions et de compétences que nous attendons d'une *solution* informatique. Deux de ces compétences générales peuvent s'exprimer en :

(1) compétences de *transposition* ;

(2) compétences de *transformation*.

A partir d'une entité α, nous devons exprimer la *transposition* ou la *transformation* qu'elle subit pour « devenir » une nouvelle entité β. Provisoirement et pour cette analyse, nous utilisons le symbole ⇒ que nous interprétons temporairement : « devient ».

F 5.1. $$\alpha \Rightarrow \beta$$

Ce qui se lit : « α devient β ».

5.2.1 Outils de transposition et réversibilité

Les besoins s'orientent selon deux axes :

1) à partir du langage du *domaine de connaissances*, être capable de *transposer* les entités linguistiques vers le *SL* ;

2) à partir du *SL*, *transposer* les entités linguistiques vers le langage du *domaine de connaissances*.

Pour traiter le cas où le *domaine de connaissances* est la source du traitement, donnons-nous quelques entités α avec leurs *transpositions* escomptées β :

F 5.2.
(1) 'val1 désigne un seul objet' ⇒ 'val1=G'
(2) 'Whisky est un nom' ⇒ 'Whisky=a'
(3) 'Boissons est un nom' ⇒ 'Boissons=b'
(4) 'val1 est un des Whisky' ⇒ 'ε(Ga)'
(5) 'val1 est un des Boissons' ⇒ 'ε(Gb)'
(6) 'Whisky est un des Boissons' ⇒ 'Ĉ{ab}'

La solution doit être capable de :

(a) attribuer des *variables de nom* (*singulier* ou *général*) aux entités (1), (2) et (3) ; *variables* qui ne sont pas encore utilisées dans le *système* et les mémoriser pour un usage futur ;

(b) traduire en *foncteurs constants* avec leurs *contextes* les propositions (4), (5) et (6).

A partir de ce qui précède, il serait utile de *transformer*, par exemple, des propositions (Ĉ{ab}) en *généralisation* de la manière suivante :

F 5.3. 'Ĉ{ab}' ⇒ '⌊ab⌋⌈≡(Ĉ{ab}∧(~(⌊G⌋⌈~(εGa})⌉)⌊G⌋⌈⊃(ε{Ga}ε{Gb})⌉))⌉'

Envisageons le deuxième cas, celui où le *SL* est la source du traitement. Il s'agit, comme illustration, à partir d'une *inscription*, de remplacer les noms des *termes variables* par une constante de nom ou la constante de nom qui a été mémorisée par la *solution*, par exemple :

F 5.4. 'Ĉ{ab}' ⇒ 'aĈb'
'aĈb' ⇒ 'a est inclus dans b'
'a est inclus dans b' ⇒ 'Whisky est inclus dans Boissons'

Nous verrons dans la suite de notre développement que ces exigences font l'objet de la *solution* que nous proposons. De plus, la *solution* utilise le même symbolisme, les mêmes *signes* que ceux qui ont été choisis par Leśniewski et Denis Miéville.

5.2.2 Traitement des règles d'inférences

Nous traitons d'abord les *règles d'inférences de définition* et, ensuite, les quatre autres *règles d'inférences*.

5.2.2.1 Les règles d'inférences de définition

La *règle d'inférences de définition* de la *protothétique* (*4.4.1 La règle d'inférences de définition, p. 42*), par exemple, permet une lecture linéaire qui facilite la construction d'une nouvelle *thèse-définition* ou de servir de filtre pour vérifier la bonne formation d'une *inscription*. Construction et filtrage n'en restent pas moins un travail qui nécessite précision et attention. Certaines notions utilisées dans la *règle d'inférences de définition*, comme celle de *catégorie syntaxico-sémantique* et de son analyse font appel à une *définition inductive* (*F 4.8 et suivantes, p. 42*) parallèle à la règle. De plus, étudions la condition 5 :

> Si le foncteur constant du *definiendum* est destiné à être d'une catégorie syntaxico-sémantique qui appartient déjà au système, le contexte qui le suit doit être similaire au contexte préalablement fixé pour cette catégorie.

Elle fait appel à « ce qui appartient déjà au *système* ». Par conséquent, mettre en œuvre une *procédure définitoire* c'est tenir compte de trois ingrédients du *système* : la *définition inductive*, la *règle d'inférences de définition* et l'état du *système*. Nous avons donc besoin d'une fonction qui nous permet de parcourir les *inscriptions* préalablement introduites dans le *système*.

Filtrer et vérifier la bonne formation d'une *inscription* doit aussi faire partie de notre cahier des charges. Voici deux exemples de vérification de la bonne formation d'*inscriptions* :

F 5.5. ⌊q⌋⌈≡(rq)⌉ ⇒ Erreur : r n'est pas quantifiée

E⟮A⟯ ⇒ Attention : le contexte ⟮-⟯ est déjà utilisé pour le foncteur E

Une troisième compétence, celle de l'apport d'informations sur une *inscription* donnée, nous paraît d'une nécessité absolue :

F 5.6. ⌊…⌋⌈≡(…)⌉ ⇒ Est une généralisation

⌊…⌋⌈≡(…)⌉ ⇒ Etre_une_généralisation⦇⌊…⌋⌈≡(…)⌉⦈

Toujours sous l'angle de l'apport d'informations aux *inscriptions* des *SL*, il nous est apparu très important de pouvoir disposer d'une compétence de calculs propositionnels :

F 5.7. ∧(pq) ⇒ Evaluation de ∧(pq) : 1000

Nous souhaitons aussi apporter une aide à la conversion des formes en *version catégorielles* (en notation classique) vers les formes en *version contextuelle* (en notation polonaise préfixée) et inversement :

F 5.8. ≡(pq) ⇒ p ≡ q
p ≡ q ⇒ ≡(pq)

Les besoins de filtrage, d'apport d'informations, de calculs propositionnels, de conversion de notations sont des compétences qu'apporte notre *solution*.

5.2.2.2 Les autres règles d'inférences

En plus des compétences déjà mentionnées, les autres *règles d'inférences* nécessitent que la *solution* apporte les capacités additionnelles suivantes :

En ce qui concerne la *règle d'inférences de distribution des quantificateurs*, nous retenons un aspect indispensable à traiter. En plus d'appliquer les conditions de la règle, la *solution* doit offrir la capacité de générer un ensemble de combinaisons, soit un ensemble d'*inscriptions* ou de nouvelles *thèses :*

F 5.9. $\lfloor pq \rfloor\lceil \equiv(\equiv(pq)\equiv(qp))\rceil \Rightarrow$
(1) $\lfloor p \rfloor\lceil \equiv(\lfloor q \rfloor\lceil \equiv(pq)\rceil\lfloor q \rfloor\lceil \equiv(qp)\rceil)\rceil$
(2) $\lfloor q \rfloor\lceil \equiv(\lfloor p \rfloor\lceil \equiv(pq)\rceil\lfloor p \rfloor\lceil \equiv(qp)\rceil)\rceil$
(3) $\equiv(\lfloor pq \rfloor\lceil \equiv(pq)\rceil\lfloor pq \rfloor\lceil \equiv(qp)\rceil)$

Pour les *règles d'inférences de substitution*, qui nécessitent, tout d'abord, une compétence de filtrage, nous devons disposer d'une fonction qui nous permet de parcourir l'état du *système* à un moment donné pour vérifier, entre autres, si une *catégorie syntaxico-sémantique* a été préalablement introduite. De plus, nous devons disposer d'un processus qui nous permette d'introduire dans la *solution* plusieurs *expressions* : l'*expression* cible de la substitution, le substitué et le substituant.

F 5.10. cible : $\lfloor pq \rfloor\lceil \equiv(\equiv(pq)\equiv(qp))\rceil$; substitué : p ; substituant : $h(r) \Rightarrow$
$\lfloor qrh \rfloor\lceil \equiv(\equiv(h(r)q)\equiv(qh(r)))\rceil$

En considérant la *règle d'inférences de détachement*, à une *inscription* donnée, la *solution* devra aussi parcourir le *système* dans la perspective de trouver une *inscription* qui serve de première équivalence qui lui permettra de « détacher » la deuxième.

Quant aux *directives d'extensionalité*, comme pour les règles précédentes, la solution devra filtrer l'*inscription* qui lui est soumise selon la règle et parcourir l'état du *système* pour la valider.

Les *règles d'inférences* et les *définitions inductives* sont exprimées en langage naturel. Nous avons dû passer par une *formalisation* à partir de cette forme linguistique par un *langage formel* facilitant une implantation automatisée des processus nécessaires à la *procédure définitoire*.

Les compétences de parcours du *système* comme référence pour le filtrage ou pour une autre opération et de génération d'une liste d'*expressions* font partie de la *solution* que nous proposons. Elles constituent, avec la section qui précède, l'ensemble du cahier des charges que nous nous sommes donné.

5.3 Les langages formels (LF)

Dans la préparation de notre cahier des charges, notre *expression*, $\alpha \Rightarrow \beta$, « α devient β » (*F 5.1, p. 83*) est une *phrase* d'un langage. Comme cette *phrase* nous permet de décrire une *transposition* ou une *transformation* en parlant des *phrases* d'un autre langage, notre langage est un métalangage. Sur cette base, nous allons définir notre *LRE* ainsi que notre *GRE* en se référant à la théorie des *langages formels*[38] que nous abrégeons *LF*. Nous rappelons, ci-après, les notions de bases de cette théorie afin de fixer une terminologie à laquelle nous nous réfèrerons dans la suite de notre développement[39].

[38] Le lecteur peut se référer aux trois ouvrages de référence (BACKHOUSE, 1979), (HOPCROFT & ULLMAN, 1969) et (HOPCROFT & ULLMANN, 1979)

[39] Nous nous inspirons de notes de cours de François Grize, *Langages formels et compilation*, Montpellier, Octobre 2007.

5.3.1 Langages

Considérons les définitions suivantes :

RD 5.3-1

1. Un ***alphabet*** A est un ensemble fini de ***symboles***.
2. Une séquence $s = a_1, a_2, \ldots, a_n$ de symboles de A s'appelle une ***chaîne*** de A.
   a. on appelle ***longueur de s***, notée |s| le nombre de symboles de s, ici : |s| = n ;
   b. la ***chaîne vide*** est notée ε telle que |ε| = 0 ;
   c. on désigne par A* = {toutes les chaînes de A}
   d. ε appartient à A* ;
   e. on désigne par $A^+ = A^* - \{\varepsilon\}$.
3. Soit u et v deux chaînes. L'opération qui consiste à écrire la chaîne v après la chaîne u, s'appelle la ***concaténation*** de v à u, notée uv (à noter que : uεv = uv).
4. Un ***vocabulaire*** V est un ensemble fini de mots, qui sont certaines chaînes de A*, donc V est inclus dans A*.
   a. |V| = n et |A*| = infini
   b. par analogie, on a V* et V^+.
   c. en théorie des langages formels, on ne fait pas la distinction entre A et V.

Exemple :

A = {	a..z,	minuscules
	A..Z,	majuscules
	≡, ⊃, ∧, …	termes constants
	⌊, ⌋, ⌈, ⌉, (,), …	ponctuation
	1, 0}	vrai, faux

A* = {toutes les chaînes possibles de A}
V = {toutes les inscriptions leśniewskiennes}
V* = {toutes les combinaisons possibles des inscriptions leśniewskiennes}

5. Un ***langage*** L défini sur V se constitue de ***phrases*** et est un sous-ensemble de V*.

   Exemple : V = {ab}

   a. $L_1 = \{\varepsilon, a, b, aa, bb, ab, ba, aaa, \ldots\}$
   b. $L_2 = \{a, ba, aab, bbb\}$
   c. $L_3 = \{a^p \mid p \text{ est premier}\}$
   d. $L_4 = \{a^n b^n \mid n \geq 1\}$
6. Le ***produit*** (pas commutatif) de deux langages L_1 et L_2, noté :
   $L_1 \bullet L_2 = \{w_1 w_2 \mid w_1 \in L_1 \text{ et } w_2 \in L_2\}$

   Exemple : $L_2 \bullet \{a\} = \{aa, baa, aaba, bbba\}$
7. La ***puissance*** d'un langage L est définie comme suit :

   $L^0 = \{\varepsilon\}$
   $L^n = L \bullet L^{n-1} \quad n \geq 1$
8. La ***clôture ou fermeture de Kleen*** L* d'un langage L est définie par :

   $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots \cup L^n \cup \ldots$
9. La ***clôture ou fermeture transitive*** L^+ d'un langage L est définie par :

   $L^+ = L^* - \{\varepsilon\}$

5.3.2 Grammaires

Retenons encore les définitions suivantes :

RD 5.3-2 Si L est fini, on peut définir L en listant ses éléments.
Si L est infini, on doit utiliser une ***grammaire*** pour le définir.

1. Une ***grammaire*** G est définie par le quadruplet :

   $$G = (N, T, P, S), \text{ où}$$

   a. N est un ensemble fini dont les éléments s'appellent des ***non-terminaux*** ou des variables ;
   b. T est un ensemble fini dont les éléments s'appellent des ***terminaux*** et où $N \cap T = \emptyset$;
   c. P est un ensemble fini dont les éléments s'appellent des ***productions*** ou des ***règles de grammaire***. Une production est toujours de la forme :
      $$\alpha \rightarrow \beta$$
      avec $\alpha \in (N \cup T)^+$ et $\beta \in (N \cup T)^*$
      le symbole $\rightarrow$ signifie « est remplacé par »[40]
   d. S est un symbole de N qu'on appelle axiome (***Start*** ou Sentence symbol) à partir duquel seront engendrées toutes les phrases du langage L défini par la grammaire G et noté :
      L(G)

Conventions d'écriture :

Type	*Elément atomique*	*Chaîne*
Terminal	a, b, c, … ∈ T	…, x, y, z ∈ T*
Non-terminal	A, B, C, … ∈ N	…, X, Y, Z ∈ N*
Arbitraire		α, β, γ, …∈ (N∪T)

Voyons ensuite la notion de *dérivation* :

RD 5.3-3 Définitions et notations :

Soit une grammaire G = (N, T, P, S)

1. La chaîne α' est une ***dérivation directe*** de la chaîne α ssi :

   $$\alpha = \beta\gamma\delta, \quad \alpha' = \beta\eta\delta \quad \text{et} \quad \gamma \rightarrow \eta \in P \text{ et qu'on note : } \alpha \rightarrow \alpha'$$

2. La chaîne α' est une ***dérivation*** de la chaîne α ssi :
   - soit α' = α
   - soit il existe une séquence de chaînes $\alpha_0, \alpha_1, \ldots, \alpha_n$ telle que $\alpha = \alpha_0$ et $\alpha' = \alpha_n$ où les α_{i+1} sont des dérivations directes de α_i $\forall i, 0 \leq i \leq n-1$ et on note : $\alpha \overset{*}{\Rightarrow} \alpha'$. Si $n \geq 1$ alors on note $\alpha \overset{+}{\Rightarrow} a'$.
3. Le langage L engendré par la grammaire G est défini comme :

   $L(G) = \{w \mid w \in T^* \text{ et } S \overset{*}{\Rightarrow} w\}$

Nous donnons encore la manière dont N. Chomsky (CHOMSKY, 1959) a classé les grammaires formelles :

[40] A ne pas confondre avec le symbole $\Rightarrow$ qui n'est pas synonyme.

RD 5.3-4 Classification des grammaires :

Type 0 ou *récursivement énumérable*

Aucune restriction sur les productions

Type 1 ou *dépendante de contexte*

Toutes les productions (règles de grammaire) ont la forme :

$\alpha \rightarrow \beta$ avec $|\alpha| \leq |\beta|$ où $\alpha, \beta \in (N \cup T)^+$

Type 2 ou *indépendante de contexte*

Toutes les productions (règles de grammaire) ont la forme :

$A \rightarrow \alpha$ avec $A \in N$ et $\alpha \in (N \cup T)^*$

Type 3 ou *régulière*

Toutes les productions (règles de grammaire) ont la forme :

- soit $A \rightarrow xB$ ou $A \rightarrow y$ (linéaire à droite)
- soit $A \rightarrow Bx$ ou $A \rightarrow y$ (linéaire à gauche)

où $A, B \in N$ et $x, y \in T^*$

A noter que si G est R ou IC alors $x \in L(G)$ est décidable.

5.3.3 Forme normale de Backus-Naur

La forme normale de Backus-Naur est une notation utilisée pour écrire les *productions* (règles de grammaire) :

RD 5.3-5 le symbole → est remplacé par ::=
les non-terminaux sont écrits entre <>
ε s'écrit <empty>
on introduit le métasymbole | qui désigne un choix

Cette notation s'est révélée trop rudimentaire et a été étendue.

5.3.4 Forme normale de Backus-Naur étendue

Des nouveaux métasymboles ont été introduits dans la forme normale de Backus-Naur afin de l'étendre. Cette forme est nommée *EBNF*.

RD 5.3-6

::=	signifie « est défini comme » ou « est composé de ». Par exemple : <phrase> ::= <sujet><verbe>
|	signifie « ou exclusivement ». Par exemple : <minuscules> ::= 'p' | 'q' | 'r'
< >	*symbole non terminal*.
' ' ou " "	*symbole terminal*.
[ ]	ce qui est contenu entre [ ] est une option qui peut apparaître 0 ou 1 fois. Par exemple : <phrase> ::= <sujet> <verbe> [<complément_objet_direct>]

{ } ou { }*	ce qui est contenu entre { } est une option qui peut apparaître 0 ou plusieurs fois. Par exemple : <phrase> ::= <sujet> <verbe> [<complément_objet_direct>] {<complément_circonstanciel>}*
{ }+	Ce qui est contenu entre { }$^{+}$ doit obligatoirement apparaître une ou plusieurs fois. Par exemple : <texte> ::= {<phrase>}$^{+}$

Nous utiliserons la notation *EBNF* pour décrire les *productions* de nos grammaires dans ce qui suit.

5.3.5 Langages formels et systèmes leśniewskiens

Après ce rapide rappel sur les *LF*, nous sommes armés pour chercher des *solutions* qui répondent à notre cahier des charges. Prenons quelques *phrases* issues d'un tout petit morceau de l'*ontologie* : Ĉ{ab}, Ĉ{aa}, Ĉ{bc}, …

Soit, le sous-langage LO_x, un sous-ensemble du *LO* et la grammaire GO_x, donc, $LO_x = L(GO_x)$[41] et en considérant les définitions de :

F 5.11.

GO_x = (NO_x, TO_x, PO_x, <inclusion>) où
NO_x = {<inclusion>, <foncteur_inclusion>, <ag>, <ad>, <nom>}
TO_x = {'a', 'b', 'c', 'Ĉ'}

PO_x :
<inclusion> ::= <foncteur_inclusion> <ag> <nom> <nom> <ad>
<foncteur_inclusion> ::= 'Ĉ'
<ag> ::= '{'
<ad> ::= '}'
<nom> ::= 'a' | 'b' | 'c'

Si la mise en forme *EBNF* convient parfaitement à la *dérivation* de *phrases* comme Ĉ{ab}, …, elle ne répond pas aux besoins fondamentaux du comportement développemental des *SL*. A la base dans les *LF*, l'ensemble des *terminaux* TO_x est connu et prédéfini. Dans les *SL*, à part quelques *signes* (primitivement définis), tous les autres sont introduits lors de la construction du *système* et lors de l'apport de nouvelles *inscriptions*. Donc, la grammaire GO_x que nous nous sommes donnée ne répond pas à la *formalisation* que nous recherchons. Nous savons que quelques *signes* sont introduits primitivement au *système* et par conséquent sont prédéfinis : l'*expression biconditionnelle primitive* et ce qui la constitue, ainsi que quelques *signes* de ponctuation. Ceci nous permet de définir une nouvelle GO_0[42] avec ce que l'on a déterminé jusqu'ici et qui respecte un peu mieux la pensée de Leśniewski :

---

[41] Nous avons donné les noms de *LO* et *GO*, pour mentionner avec 'O' que nous appliquons ce qui a été dit des *langages formels* sur des *mots* ou des *phrases* de l'*ontologie* (et/ou de la *protothétique*) de Leśniewski. Il ne s'agit pas de décrire l'*ontologie* en tant que langage, mais ses *mots* et ses *phrases*.

[42] Soit un sous-langage de *LO*, LO_0, où $LO_0 = L(GO_0)$ et $LO_0 \subset LO$.

F 5.12. $LO_0 = L(GO_0)$

GO_0 = (NO_0, TO_0, PO_0, <généralisation>) où
NO_0 = {<généralisation>, <quantification>,…}
TO_0 = { '⌊', '⌋', '⌈', '⌉', '≡', '(', ')', 'p', 'q', 'r' }

PO_0 :
<généralisation> ::= <quantification> <sous-quantification>
<quantification> ::= <dqg><terme>{<terme>}<dqd>
<sous-quantification> ::= <dsqg><fonction_logique><dsqd>
<fonction_logique> ::= <exp_bic_prim>
<exp_bic_prim> ::= <biconditionnelle><pg><terme><terme><pd>
<terme> ::= <terme_variable> | <fonction_logique>
<signes_connus> ::= <dqg> | <dqd> | <dsqg> | <dsqd> | <biconditionnelle> | <pg> | <pd> | <terme_variable>
<terme_variable> ::= <terme_variable_propositionnel> | **<autres_signes_inconnus>**
<terme_variable_propositionnel> ::= 'p' | 'q' | 'r'
<biconditionnelle> ::= '≡'
<dqg> ::= '⌊'
<dqd> ::= '⌋'
<dsqg> ::= '⌈'
<dsqd> ::= '⌉'
<pg> ::= '('
<pd> ::= ')'

Exemples de *phrases* qui appartiennent au LO_0 :
'⌊pq⌋⌈≡(pq)⌉', '⌊rq⌋⌈≡(≡(rq)q)⌉', …

Notre GO_0 remplit pratiquement toutes les exigences de la philosophie de construction développementale des *SL*, puisque l'*expression biconditionnelle primitive* et les *signes* de ponctuation sont introduits dans le *système* dès le début de sa construction. Toutefois, un *non-terminal*, <autres_signes_inconnus>, n'a pas de définition. Par conséquent, on ne peut pas assurer si une *phrase* comme : '≡(*$)', appartient ou non au LO_0. Nous avons besoin d'un artifice qui nous permet de parler des *signes* qui n'ont pas encore été introduits dans le *système*, ou en d'autres termes, de trouver un moyen de *formalisation* qui nous offre la possibilité de traiter des *non-terminaux* qui prendront une valeur lors de la construction du *SL*. Ce qui, par ailleurs, nous conduit à renforcer le terme « évolutif » dans le choix de notre titre : *[…] systèmes évolutifs de connaissances*.

A partir de maintenant, nous utiliserons *GO* pour toutes les grammaires qui traiteront des *phrases* de la *protothétique* et de l'*ontologie* de Leśniewski. Comme notre langage grammatical, *LRE*, offre la possibilité de traiter des *domaines de connaissances*, des langages, des corpus linguistiques qui vont au-delà du monde constitué par les *SL*, nous étendons l'ensemble des grammaires constituées par *GO* à *GO**. Toujours dans la perspective de répondre à notre cahier des charges et trouver une *solution* qui nous permet d'échapper à l'aspect trop déterministe des *LF*, nous allons élaborer notre *LRE*, métalangage qui nous permet de traiter la famille de l'infinité possible des grammaires *GO**.

5.4 Elaboration du langage de réécriture (LRE)

Nos deux grammaires GO_x et GO_0 appartiennent à l'ensemble des grammaires *GO* sachant que $GO \subset GO^*$. Pour satisfaire le cahier des charges, nous aurons besoin de plusieurs grammaires GO_i appartenant à l'ensemble *GO**. Par exemple : une grammaire pour satisfaire la *transposition* depuis un *domaine de connaissances* vers des définitions *ontologiques* ; une autre pour la conversion de formes en *version catégorielle* vers la forme en *version contextuelle* ; une autre qui analyse les *catégories syntaxico-sémantiques* ; etc. Une GO_i quelconque gouverne un sous-ensemble du *LO* qui parle de certains aspects des *SL* ; soit : $LO = L(GO_i)$.

L'ensemble de ces grammaires *GO** doit être défini par un langage (grammatical et de réécriture), soit le *LRE*[43] qui lui-même est défini par une métagrammaire *GRE :* ou pour utiliser le formalisme des *LF : LRE = L(GRE)*. Dit autrement, le *LRE* doit répondre point par point à l'ensemble des exigences du cahier des charges. En reprenant la théorie des *LF :*

F 5.13. $$LRE = L(GRE) \text{ et donc } LO = L_{LO}(L_{LRE}(GRE))$$

Nous allons maintenant définir dans les détails la *GRE*. A chaque étape de son élaboration nous argumenterons nos choix en fonctions des besoins et des compétences escomptées que nous avons exhibées comme cahier des charges.

Nous avons appelé le *RE* la *solution* informatique ou le logiciel qui met en œuvre les GO_i. C'est un logiciel prototype que nous avons développé afin de montrer la faisabilité de notre approche. Le *RE* traite les caractères unicodes soit une police de 64 000 caractères qui permet de répondre à une des exigences de notre cahier des charges, à savoir : respecter le choix des *signes* couramment utilisés par Leśniewski et Denis Miéville.

5.4.1 Définition des règles de réécriture (rre)

Le *LRE* est le langage « grammatical » qui permet de formuler les règles de grammaires des GO_i. Pour assurer qu'aucune ambiguïté n'apparaisse lors du processus de la compilation exécuté par le logiciel *RE*, parmi le grand nombre[44] de *signes* du *LO*, nous avons dû réserver quelques métasymboles pour discriminer les différentes entités linguistiques du *LRE*[45]. Cela alourdit le *LRE*, mais assure en même temps que toutes les *phrases* compilées par le logiciel *RE* appartiennent au *LRE ; par exemple :* '☞↳STOP↲☜'.

Dans la section *5.2 Analyse des besoins, p. 82* nous avons esquissé une définition du symbole '⇒' comme « ce qui est à gauche devient ce que l'on donne à droite ». Nous reprenons cette définition pour élaborer dans les détails notre *LRE* :

---

[43] Pour langage (L) de réécriture (RE).

[44] Plus de 64 000 caractères disponibles dans la *solution* informatique et donc dans le langage *LRE*.

[45] Dans la version 4.4 du *RE*, nous avons :

- les symboles '‹' et '›' encadrent les variables du *LRE*, par exemple : ‹variable› ;
- les mots réservés sont enveloppés par des marques distinctives : '↳' et '↲', par exemple : '↳STOP↲ ;
- certaines phrases comme les règles de grammaire du *LRE* sont enveloppées par '☞' et '☜', par exemple : ☞‹a›⇒A‹a›☜ ;
- la virgule ',' est utilisée comme séparateur d'arguments dans les fonctions du *LRE*, par exemple : ↳PLUS(‹a›, ‹b›)↲ ;
- les parenthèses '(' et ')' comme enveloppe d'arguments des fonctions du *LRE*, par exemple : ↳ALLER_A_REGLE(<entier_non_signé>)↲
- les séparateurs '╚' et '╝' délimitent des chaînes, par exemple : ↳TRIER(<portion_chaîne>, ╚<motif_quasi_alpha>╝)↲.

F 5.14. **Définition d'une règle de réécriture (*rre*) :**

Une règle de réécriture *rre* est une des *phrases* du *LRE.* Elle se construit à l'aide d'un membre de gauche *mg* qui précède le symbole '⇒' et d'un membre de droite *md* qui le suit. Le symbole '⇒' s'interprête comme : « le *mg* se réécrit en *md* » [46]. Une *rre* prend la forme suivante[47] :

$$☞ mg \Rightarrow md ☜$$

On abrège aussi une *chaîne soumise* à une *rre* : *cs* et la *chaîne résultante* après l'exécution de la *rre* : *cr*. Deux cas peuvent se présenter à l'exécution d'une *rre* : soit elle (ré)écrit la *cs* en *cr*, soit elle ne fait rien :

$$cs \stackrel{?}{\Rightarrow} cr$$

A partir d'une *cs* soumise au *mg*, si le *filtrage de motif*[48] entre la *cs* et le *mg* réussit, alors la *rre* (ré)écrit le motif du *md* donné, comme *cr*. Sinon, la *rre* ne s'applique pas.

$$\text{si } (cs \stackrel{?}{\approx} mg) \text{ alors } \Rightarrow (md \approx cr), \text{ sinon } cr = \varepsilon$$

Expression qui se lit : « si le *filtrage de motif* entre la *cs* et le *mg* est satifait alors la *rre* réécrit la *cr* selon le schéma du *md* ; sinon elle ne produit rien (ou la chaîne vide) ». Par exemple, la *rre*, la *cs* et la *cr* suivantes :

☞Il fait beau⇒On sort le chien☜
cs : 'Il fait beau'
cr : 'On sort le chien'

Par comparaison, si nous nous choisissons une *cs*, comme : 'Il pleut', le *filtrage de motif* n'aboutit pas et la *rre* ne (ré)écrit rien, donc la *cr* = ε.

Une *solution* informatique qui ne saurait faire du *filtrage de motif* que sur une *chaîne terminale* comme : 'Il fait beau' n'apporterait que très peu de chose comme outil d'aide. En effet, il faudrait autant de *rre* que de *phrases* (constituées d'une *chaîne terminale*) à traiter, soit un très grand nombre pour ne pas dire une infinité. Par conséquent, nous étendons notre définition :

F 5.15. **Construction des *mg* et des *md* d'une *rre* :**

A condition que $mg \neq \varepsilon$ et $md \neq \varepsilon$, quel que soit le membre (*mg* ou *md*), son schéma peut être construit d'une ou plusieurs *constantes* de *chaîne C*[49] et/ou d'une ou plusieurs variables de *chaîne V*[50], ssi :

(a) dans le *mg,* deux variables *V* ne se suivent jamais ;

(b) dans le *md*, les variables *V* sont mentionnées (déclarées) au moins une fois dans le *mg*.

Exemples de schémas corrects pour le *mg* :

C, V, CVC, VCV, CVCVCV, …

Exemples de schémas corrects pour un *md* :

C, V, CVC, VV, VVVCV, CVVVCVCVVV, …

---

[46] Le *mg* est similaire à α des *LF* et *md* à β des *LF*.

[47] La forme est similaire à une production des *LF*.

[48] On utilise dans la littérature : motif, schéma, correspondance, … Il existe plusieurs dénominations du processus : analyse de correspondances, recherche d'équivalences entre motifs ou en anglais : Pattern Matching. La littérature informatique spécialisée utilise la notion de *filtrage de motifs*.

[49] Nous utilisons l'abréviation *C* pour constante de *chaîne*, soit une *chaîne terminale*. Donc, *C* est similaire à une *chaîne terminale* des *LF*.

[50] Nous utilisons l'abréviation *V* pour variable de *chaîne*, soit une *chaîne non-terminale* des *LF*. Donc, *V* est similaire à une *chaîne non-terminale* des *LF*.

Sachant qu'une *variable V* est désignée par un *identificateur*[51], le codage des membres d'une *rre* nécessite une marque qui permet de discriminer une *C* d'une *V*. En effet, en admettant que var est une *variable V*, une forme comme :

Une chose var un truc

n'indique pas quelles sont les portions de la *chaîne* qui sont des *C* et lesquelles sont des *V*. De façon classique, les langages informatiques et formels traitent les *chaînes* ou les *constantes* de chaîne, *C*, en les enveloppant par des guillemets ou des apostrophes :

'Une chose 'var' un truc'

Par conséquent, nous sommes ici en présence d'un schéma de type : *CVC*. En se donnant :

☞'Une chose 'var' un truc' ⇒ 'Rien 'var' rien'☜
cs : 'Une chose est un truc'
cr : 'Rien est rien'

Prenons un exemple qui fait la mention d'un *identificateur* de *variable V :* var, dans le *mg* :

☞'ra'**var**'er' ⇒ 'se ra'var'er est nece'**varvar**'aire'☜
cs : 'raser'
cr : 'se raser est nécessaire'

Si l'*identificateur* var est bien mentionné une fois dans le *mg*, varvar n'est pas un *identificateur* mentionné dans le *mg*. Donc, nous avons à faire à une erreur de programmation du *md* !

Pour éviter ces risques d'erreurs, nous devons disposer d'une marque de différenciation. Plusieurs solutions peuvent être envisagées en utilisant des schémas comme : '*C*', *V*, *V*, '*C*' ou '*C*' – *V* – V 'C', où les caractères ',' et '-' jouent le rôle des séparateurs entre nos *C* et *V*. Nous pouvons aussi envelopper les *V* par des *symboles symétriques* : '*C*'<*V*><*V*>'*C*'. Mais dans ce cas les enveloppes constituées par les apostrophes et les symboles '<' et '>' font double emplois. Il nous reste la possibilité de n'envelopper que les *V* comme le fait le System Q (COLMERAUER, 1970) : *C*<*V*><*V*>*C*. Nous avons opté pour ce type de format pour discriminer les *C* et *V* du *LRE* :

F 5.16. **Déclaration d'une *C* et d'une *V* dans le *LRE* :**

Une variable *V* est un *identificateur* enveloppé par les *symboles symétriques* '‹' et '›', par exemple : ‹nom›. Une *constante C* n'a aucune marque qui la détermine ou la spécifie.

Les deux exemples suivants sont conformes à la définition :

☞ra‹**var**›er ⇒ se ra‹var›er est nece‹**var**›‹**var**›aire☜
cs : 'raser'
cr : 'se raser est nécessaire'

☞Le ‹**boisson**› est un liquide ⇒
Si le ‹**boisson**› est un liquide alors il est une boisson☜
cs : 'Le whisky est un liquide'
cr : 'Si le whisky est un liquide alors il est une boisson'.

Il s'agit maintenant de définir le comportement de l'exécution d'une *rre* :

[51] Une *chaîne* de signes (ou de caractères) alphanumérique qui commence toujours pas une lettre minuscule ou majuscule de l'alphabet.

F 5.17. **Exécution d'une *rre* :**

Si le *filtrage de motif* d'une *rre* réussit pour une *cs* donnée (la *chaîne soumise* à la *rre*), la *rre* produit une *cr* (la *chaîne résultante* de la *rre*) et, tant que le *filtrage de motif* réussit, la *cr* est réutilisée comme *cs* pour la même *rre*.

Par exemple :

☞‹v1›K‹v2› ⇒ ‹v1›M‹v2›☜
cs : 'aKKKz'
cr : 'aMKKz' première itération et la *cr* devient une nouvelle *cs* pour la *rre*
cr : 'aMMKz' deuxième itération et la *cr* devient une nouvelle *cs* pour la *rre*
cr : 'aMMMz' troisième itération et la *cr* devient une nouvelle *cs* pour la *rre*

Après la troisième itération, le *filtrage de motif* n'aboutit plus, donc la *rre* fournit la *chaîne cr :* 'aMMMz'. Pour reprendre les notions des *LF :*

$$\alpha = \text{'aKKKz'} \rightarrow \alpha' = \text{'aMMMz'}$$

La *dérivation* $\alpha \rightarrow \alpha'$ est constituée de trois *dérivations directes*.

Nous avons décidé, de façon arbitraire que :

RD 5.4-1 Le processus *filtrage de motif* entre une *cs* et le *mg* d'une *rre* se fait toujours de gauche à droite (du début à la fin).

Cette contrainte peut paraître surprenante. En effet, à ce niveau de définition du *LRE*, rien n'empêcherait d'avoir une approche dans l'autre sens ou même dans les deux sens. Nous verrons que ce choix nous servira pour l'analyse des *variables isoformes* imbriquées.

Avec ce qui a été défini, nous sommes capables d'exprimer, par exemple, la conversion d'*expressions* comme :

F 5.18. ☞‹definiendum›≡‹definiens›⇒≡(‹definiendum›‹definiens›)☜

Par contre nos définitions, nous interdisent la *rre* suivante, dont le *mg* est incorrect :

F 5.19. ☞≡(**‹definiendum›‹definiens›**)⇒‹definiendum›≡‹definiens›☜

En effet, deux *V* ne peuvent pas se suivre dans un *mg*. Ce qui montre, en particulier, qu'une *rre* n'est pas réversible. Nous ne pouvons pas inverser les deux membres[52]. Ce qui indique encore que '⇒' n'est pas symétrique. Arrêtons-nous ici pour expliquer ce qui nous a conduits à donner la contrainte *(a)* de la définition *F 5.15, p. 92.* Donnons-nous une *cs* et une *rre* mal formée :

F 5.20. ☞bb**‹v1›‹v2›**cc⇒v1=‹v1› et v2=‹v2›☜
cs : 'bbAAAAcc'

‹v1› et ‹v2› peuvent prendre plusieurs valeurs de *chaîne* :

‹v1›	**‹v2›**
A	AAA
AA	AA
AAA	A

Par conséquent, un schéma de *mg*, qui contient des *V* qui se succèdent, peut rencontrer, lors du *filtrage de motif*, des cas d'indécidabilité. Le schéma de type *VC*{*VC*} permet de calculer où commence et où finit une portion de *chaîne* qui peut être affectée à une *V*. Nous devrons, donc, passer par des artifices nous permettant d'exprimer la conversion de la forme en

[52] Contrairement au System Q de Colmerauer (COLMERAUER, 1970).

version catégorielle vers la forme en *version contextuelle* (*F 5.19, p. 94*). Ce qui a nécessité que l'on introduise dans le *LRE* des *attributs*.

5.4.1.1 Les attributs du langage LRE

Dans la théorie des *LF*, un *attribut* est une *fonction*, une *instruction* ou une *directive* qui vient s'ajouter dans la définition de certaines *productions*. Le *LRE* en fait usage, soit pour simplifier la programmation des *productions* et résumer un ensemble de *dérivations directes* en *dérivation*, soit pour donner des *directives* à la *solution* informatique, le logiciel *RE* ; sachant que le *RE* est piloté par la *GRE*.

Nous allons considérer un des attributs, la *fonction* du *LRE* : EXPLOSER[53], que nous avons apportée à notre langage, qui nous permet d'introduire des séparateurs de caractères. Ces séparateurs, des métasymboles, nous serviront de *C* que l'on va insérer entre deux *V*. La *fonction*, EXPLOSER, qui s'applique sur les caractères d'une *chaîne* donnée, est destinée à être utilisée dans les *md*. Une *rre* qui l'utilise prend la forme suivante :

F 5.21. **La fonction EXPLOSER :**

Elle est définie syntaxiquement de la façon suivante :

EXPLOSER(<chaîne_à_exploser>,<séparateur_gauche>,<séparateur_droit>)

où

<chaîne_à_exploser> est soit une *V* du *LRE*, soit une *C* du *LRE ;*
<séparateur_gauche> et <séparateur_droit> sont des *chaînes* plus grandes ou égales à un.

Cette *fonction* retourne (au sens informatique) une *chaîne* où chaque caractère est enveloppé par un <séparateur_gauche> et un <séparateur_droit>.

Exemple d'utilisation et de comportement de la *fonction* dans une *rre* :

☞bb‹v1›cc⇒EXPLOSER(‹v1›, '[', ']')☜
cs : 'bbAAAAcc'
cr : 'bb[A][A][A][A]cc'

Nous nous donnons une contrainte sur l'exécution de cette *fonction* : une *rre* dont le *md* contient une *fonction* EXPLOSER ne s'exécute qu'un seule fois dans le corps de la grammaire[54].

Remarque : EXPLOSER est un *terminal* de la *GRE*, soit un mot réservé par le *LRE*.

En reprenant notre illustration (*F 5.25, p. 96*) et, par exemple, pour isoler le premier 'A' :

F 5.22.

cs :	'bbAAAAcc'
	☞bb‹v1›cc⇒EXPLOSER(‹v1›, '[', ']')☜
cr qui devient *cs :*	'bb[A][A][A][A]cc'
	☞bb[‹v1›]‹v2›cc⇒v1=‹v1› et v2=‹v2›☜
cs :	v1 = A et v2 = [A][A][A]

Nous pouvons aussi reprendre la conversion de la forme en *version catégorielle* vers la forme en *version contextuelle* (*F 5.19, p. 94*) et nous donner un *mg* correct :

F 5.23. ≡([‹definiendum›][‹definiens›])⇒‹definiendum›≡‹definiens›

Par conséquent, nous avons déjà apporté, à notre *LRE*, la compétence de conversion exprimée dans le *§ F 5.8, p. 84*.

---

[53] Dans le logiciel prototype : EXPLODE.

[54] On comprendra aisément que si cette contrainte n'est pas appliquée une *rre* qui utilise EXPLOSER va s'itérer indéfiniment. Dans notre exemple, la *cr* de la deuxième itération prendrait la forme :
bb[[]A[]][[]A[]][[]A[]][[]A[]]cc !

Nous verrons plus loin dans des GO_i un peu plus complexes que lors d'un processus d'analyse morphosyntaxique, nous commençons par traiter la couche lexicale, pour ensuite se pencher sur les aspects morphosyntaxiques, puis s'arrêter sur l'axe syntaxique et terminer dans la sphère sémantique. Au niveau le plus primaire, l'analyse lexicale part du graphème (au sens linguistique et pour un corpus écrit), ce que nous avons appelé, au sens informatique, des caractères y compris les caractères non imprimables. Traiter la couche lexicale implique aussi de considérer la « granularité » des entités linguistiques traitées. Qui dit caractères non imprimables, dit aussi que certains d'entre eux servent de séparateurs, de *délimiteurs*. Par exemple, l'espace typographique détermine la séparation entre les mots. Dans un cadre plus formel, certains caractères comme les parenthèses délimiteront la fin et le début d'une *expression*. Avec la *fonction* EXPLOSER, nous nous sommes donné un outil qui nous permet de traiter ces questions d'une façon systématique et en partant de l'entité atomique : le caractère.

La *fonction* EXPLOSER du *LRE* (ainsi que toutes les autres *fonctions* que nous définirons plus loin) appelle quelques considérations :

(1) sous l'angle des *LF*, la fonction EXPLOSER prend toute ou partie de la *cs* (une *phrase* du *LRE*) pour en *dériver*[55] la *cs* (une autre *phrase* du *LRE*), nous avons donc à faire à une *production* classique de la *GRE* :

$$\alpha \rightarrow \alpha'$$

(2) en approfondissant, en fait, la *production* $\alpha \rightarrow \alpha'$ est constituée d'un ensemble de *productions*, *dérivées directement* de α :

$$\alpha \rightarrow \alpha_1,\ \alpha_1 \rightarrow \alpha_2,\ \ldots,\ \alpha_i \rightarrow \alpha_{i+1},\ \ldots,\ \alpha_n \rightarrow \alpha'$$

(3) nous pouvons en tirer la certitude que le *LRE = L(GRE)*, y inclus la *fonction* EXPLOSER, est et reste de la forme : $\alpha \rightarrow \beta$.

Ces considérations nous permettrons à la fin de la définition du *LRE = L(GRE)* de déterminer le type de grammaire à laquelle nous avons à faire.

Nous n'avons donné aucune restriction concernant l'utilisation des *identificateurs* de *variable V* dans les deux membres d'une *rre*. Ce qui nous conduit à préciser :

F 5.24. **Mentions de *variables isoformes* :**

Dans un *mg*, une variable *V* déclarée ‹V› peut être mentionnée plusieurs fois et constitue ainsi une redondance. Deux variables *V* qui utilisent le même *identificateur* sont dites *variables isoformes*[56] du *LRE*.

L'affirmation (*F 5.24*) donne au *LRE* une compétence fondamentale. L'utilisation des *variables isoformes* du *LRE*, nous permet, en particulier dans les *SL*, de traiter la mention d'un *signe*, soit un *lieur* dans un *quantificateur* avec la mention d'un *signe* équiforme dans un *sous-quantificateur*, donc une *variable liée*. Par exemple :

F 5.25. ☞⌊‹v1›q⌋⌈≡(‹v1›q)⌉ ⇒ ‹v1› est un terme variable lié☜
cs : '⌊pq⌋⌈≡(pq)⌉'
cr : 'p est un terme variable lié'

Il nous reste à relever un élément important concernant les *variables V* du *LRE* :

F 5.26. **Variable vide, V=ε :**

L'idée même d'une *constante* de *chaîne C* vide n'a pas sens. Si aucune contrainte[57] explicite n'est appliquée sur une variable *V* alors le *LRE* autorise la *chaîne vide* et procède à l'analyse du *filtrage de motif* en conséquence.

Ce qui nous permet de traiter des cas comme :

---

[55] *Dériver* et non pas *dériver directement* !

[56] « Isoforme » mis pour éviter des confusions avec les *SL*, qui utilisent la notion d'équiformité.

[57] Nous définirons plus loin un moyen d'interdire qu'une variable *V* soit vide.

F 5.27. ☞[‹v1›,‹v2›] ⇒ [] contient la(les) variable(s) propositionnelle(s) : ‹v1›‹v2›☜
cs : '[p]'
cr : '[] contient la(les) variable(s) propositionnelle(s) : p'

5.4.2 Une grammaire G comme ensemble de rre

Comme nous l'avons rappelé dans notre survol des *LF*, une grammaire G est constituée d'un ensemble de quatre ensembles et en particulier celui des *productions* (règles de grammaire). Comme notre *LRE* est un langage qui permet d'écrire des règles de grammaire, il ne prend de sens que s'il peut appliquer plusieurs *rre* à un langage donné comme *LO = L(GO*)*. Pour être conforme aux *LF :*

F 5.28. La *GRE* est constituée d'un ensemble de *rre,* soit d'un ensemble de *productions.*

F 5.29. L'exécution du *LRE* démarre toujours à la première *rre* du corps de grammaire de la *GRE* et exécute chaque *rre*, tant qu'elle satisfait au *filtrage de motif* et passe à la suivante (voir section : *5.5.3 Présentation des instructions conditionnelles du* LR*, p. 128*).

F 5.30. Dans un corps de grammaire de la *GRE,* tant que, au moins une *rre* a satisfait au *filtrage de motif*, le *LRE* recommence à la première *rre* de ce même corps de grammaire.

5.4.3 Extension du LRE pour le traitement des règles d'inférences de définition

A partir de la définition de base de la *rre*, nous allons apporter des nouvelles *fonctions* du *LRE*, nous permettant de traiter les *règles d'inférences de définition* et petit à petit de substituer le traitement à la main de la *procédure définitoire* par un traitement automatisé.

A travers notre parcours des *SL* et de notre lecture des *règles d'inférences de définition*, nous pouvons établir qu'un *SL* se donne initialement :

(a) six *signes* qui ont la fonction de *délimiteurs* qui forment des paires symétriques : '(' et ')' ; '⌊' et '⌋' ; '⌈' et '⌉'. Comme les *SL* se construisent non seulement à partir de la dimension syntaxique, mais aussi à partir de l'axe sémantique, chaque paire est définie de la manière suivante : '(…)' comme *contexte* ; '⌊…⌋' comme *quantificateur* universel, soit le *signe* '∀' dans une notation en *version catégorielle*, et '⌈…⌉' *comme sous-quantificateur* ;

(b) le *connecteur primitif* de la *biconditionnelle* : '≡' ;

(c) sinon, tous les autres *signes* sont introduits avec leurs définitions dans le *système* lors de la construction de ce dernier.

Ceci nous permet déjà de construire quelques formes :

(d) '⌊…⌋⌈…⌉', une *inscription* nommée *généralisation* ;

(e) '≡ (…)', l'*expression biconditionnelle primitive*, une *inscription* de type *fonction logique*.

Le *LRE* doit pouvoir traiter le rôle et la morphosyntaxe de ces éléments constitutifs du *système*. Nous devons disposer du moyen de déterminer, par exemple, si nous avons à faire à un *délimiteur* de *contexte* ou non ; une *fonction* qui stipule le rôle ou la propriété d'un *signe* ou de la juxtaposition de *signes*. Dans la grammaire GO_0 (*F 5.12, p. 90*) nous avons rencontré une embuche concernant la nécessité de déclarer dans une *grammaire formelle* tous les *terminaux ;* tous ceux que décrit : <autres_signes_inconnus>. Nous devons donc apporter au *LRE* un moyen de parler de quelque « chose » qui n'est pas introduit préalablement dans le *système*.

Pour répondre à ce nouveau besoin, nous enrichissons le *LRE* avec l'apport de la notion d'« ensemble » et deux *fonctions* d'appartenance : DANS[58] (appartenir à l'ensemble) et HORS[59] (ne pas appartenir à l'ensemble). Ce sont deux *fonctions*[60] conditionnelles appliquées aux *V* du *LRE*. On se reportera à la section *5.4.9 Les productions de la grammaire GRE, p. 120,* alinéa *RD 5.4-15, p. 121* pour la définition d'ensembles dans la *EBNF* de la *GRE*.

Cet apport nous offre la possibilité de définir des ensembles de *signes* ou des *chaînes* de caractères à l'aide du *LRE*. Nous utilisons aussi le symbole '⇒' parce que la définition d'ensemble suit le même principe qu'un ensemble de *productions* d'une grammaire :

F 5.31. *GO_1* :

☞bic⇒"≡"☜ symbole de la biconditionnelle
☞con_prim⇒bic☜ connecteurs primitifs

☞qg⇒"⌊"☜ délimiteur gauche de quantificateur
☞qd⇒"⌋"☜ délimiteur droit de quantificateur
☞sqg⇒"⌈"☜ délimiteur gauche de sous-quantificateur
☞sqd⇒"⌉"☜ délimiteur droit de sous-quantificateur
☞qsqg⇒qg,sqg☜ délimiteurs q et sq gauches
☞qsqd⇒qd,sqd☜ délimiteurs q et sq droits
☞qsq⇒qsqg,qsqd☜ tous les délimiteurs q et sq
☞symétriqueQ⇒"⌊⌋"☜ enveloppe quantificateur
☞symétriqueSQ⇒"⌈⌉"☜ enveloppe sous-quantificateur

☞pg⇒"("☜ parenthèse gauche
☞pd⇒")"☜ parenthèse droite
☞symétriqueP⇒"()"☜ enveloppe de contexte
☞signes_primitifs_connus⇒qsq,pg,pd,bic☜ tous les signes préalablement connus

Cette déclaration est conforme à la philosophie développementale des *SL*. Elle ne considère que les *signes* définis primitivement. C'est une forme similaire à notre grammaire *GO_0* (*F 5.12, p. 90*), mais à l'aide du *LRE (*se reporter à *5.10 Liste des sous-langages de LO et des sous-grammaires de GO*, p. 147* pour l'identification des grammaires *GO)*.

On trouve la syntaxe des *fonctions* DANS et HORS qui appliquent des conditions aux *V* du *LRE* à partir de la règle *RD 5.4-27, p. 121*. Voici un exemple qui nous permet de respecter l'aspect développemental des *SL*, chose que nous n'avons pas pu réaliser avec la *GO_0* (*F 5.12, p. 90*) :

F 5.32. *GO_2 :*

ici vient la *GO_1* (*F 5.31, p. 98*), soit les définitions d'ensembles

cs : '≡(*$)'
☞‹v› ⇒ EXPLOSER(‹v›, '●[', ']')☜ *rre* 1
cr : '●[≡]●[(]●[*]●[$]●[)]' qui devient la *cs* pour la *rre* suivante

☞●[‹v1↳DANS(con_prim)↲›]‹v› ⇒ cp[‹v1›]‹v›☜ *rre* 2
cr : 'cp[≡]●[(]●[*]●[$]●[)]' qui devient la *cs* pour la *rre* suivante

☞‹a›●[‹v1↳DANS(pg)↲›]‹z› ⇒ ‹a›pg[‹v1›]‹z›☜ *rre* 3
cr : 'cp[≡]pg[(]●[*]●[$]●[)]' qui devient la *cs* pour la *rre* suivante

☞‹a›●[‹v1↳DANS(pd)↲›]‹z› ⇒ ‹a›pd[‹v1›]‹z›☜ *rre* 4
cr : 'cp[≡]pg[(]●[*]●[$]pg[)]' qui devient la *cs* pour la *rre* suivante

☞‹a›●[‹v1↳HORS(signes_primitifs_connus)↲›]‹z› ⇒ ‹a›tv[‹v1›]‹z›☜ *rre* 5
cr : 'cp[≡]pg[(]tv[*]●[$]pg[)]' première itération
cr : 'cp[≡]pg[(]tv[*]tv[$]pg[)]' seconde itération

[58] Dans le logiciel prototype : IN.

[59] Dans le logiciel prototype : OUTOF.

[60] Les *LF* parlent d'*attributs*.

A l'aide de la cinquième *rre*, nous avons traité des *signes* que le *système* ne connaissait pas préalablement. Il reste maintenant à les mémoriser comme faisant partie du *système*. Nous verrons plus loin que nous disposerons d'autres moyens. Avec ce que nous avons défini jusqu'à présent dans le *LRE*, nous pouvons déjà donner une solution de mémorisation de l'introduction de nouveaux *signes* en modifiant la GO_2 en GO_3 :

F 5.33. *GO_3 :*

ici vient *GO_1* (*F 5.31, p. 98*), soit les définitions d'ensembles

cs : '≡(*$)'

☞‹v› ⇒ EXPLOSER(‹v›, '●[', ']')**/nouv_signes_introduits()**☜ *rre* 1

cr : '●[≡]●[(]●[*]●[$]●[)]/nouv_signes_introduits()'

☞●[‹v1↳DANS(con_prim)↲›]‹v›/nouv_signes_introduits(‹v2›) ⇒
cp[‹v1›]‹v›/nouv_signes_introduits(‹v2›)☜ *rre* 2

cr : 'cp[≡]●[(]●[*]●[$]●[)]/nouv_signes_introduits()'

☞‹a›●[‹v1↳DANS(pg)↲›]‹z›/nouv_signes_introduits(‹v2›) ⇒
‹a›pg[‹v1›]‹z›/nouv_signes_introduits(‹v2›)☜ *rre* 3

cr : 'cp[≡]pg[(]●[*]●[$]●[)]/nouv_signes_introduits()'

☞‹a›●[‹v1↳DANS(pd)↲›]‹z›/nouv_signes_introduits(‹v2›) ⇒
‹a›pd[‹v1›]‹z›/nouv_signes_introduits(‹v2›)☜ *rre* 4

cr : 'cp[≡]pg[(]●[*]●[$]pg[)]/nouv_signes_introduits()'

☞‹a›●[‹v1↳HORS(signes_primitifs_connus)↲›]‹z›
/nouv_signes_introduits(‹v2›) ⇒
‹a›tv[‹v1›]‹z›/nouv_signes_introduits(‹v2›)☜ *rre* 5

cr : 'cp[≡]pg[(]tv[*]●[$]pg[)]/nouv_signes_introduits([*])'

cr : 'cp[≡]pg[(]tv[*]tv[$]pg[)][)]/nouv_signes_introduits([*][$])'

A la suite de la GO_3 et de sa fonction de mémorisation, d'autres *rre* pourront venir tester /nouv_signes_introduits([*][$]) et ainsi permettre ou non l'introduction de nouveaux *signes*.

L'apport de ces deux *fonctions* (*attributs* au sens des *LF*), DANS et HORS appellent quelques considérations :

(1) contrairement à EXPLOSER, ces deux *fonctions* ne participent pas au processus de *dérivation*, mais peuvent appliquer des contraintes sur le processus de *filtrage de motif* des *rre* du *LRE* ;

(2) en fait dans une *production* : $\alpha \rightarrow \beta$, nous opérons une modification sur α. Dès lors, nous aurons une forme de type : $\alpha_x \rightarrow \beta$;

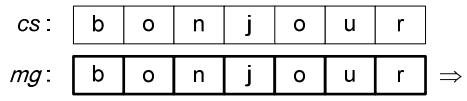
(3) ce type d'*attributs* d'un point de vue syntaxique, restent conformes aux notions des *LF*. Ils amènent néanmoins une information de type sémantique importante. En effet, DANS et HORS sont respectivement les relations $\in$ et $\notin$, donc, de façon analogique, l'application d'une propriété à un objet ; objet étant une *chaîne* dans notre cas.

5.4.3.1 Le filtrage de motif simple

Dans la perspective d'appréhender le comportement du *filtrage de motif* d'une *rre* appliquée à une *chaîne*, *cs*, voyons les différents schémas que peut prendre un *mg*, sachant qu'un schéma peut être constitué de *constantes* de *chaîne C* et de *variables* de *chaîne V*, soit les schémas suivants :

F 5.34. *C* (*mg* constitué d'une seule *C*)

Il suffit que le motif de la *cs*, corresponde parfaitement au motif du *mg* pour que le *filtrage de motif* soit satisfait. Par exemple :

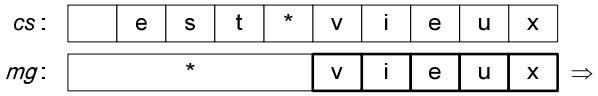


F 5.35. *C** (*mg* commençant par une *C*)

Il faut que la *solution* puisse faire correspondre parfaitement le début de la *chaîne cs* avec le motif de la première *C* du *mg*. Par exemple :

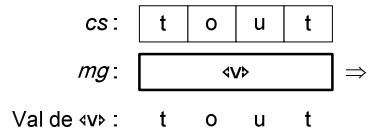
F 5.36. **C* (*mg* terminé par une *C*)

La fin de la *chaîne cs* doit parfaitement correspondre à la dernière *C* du *mg*. Voici un exemple qui satisfait au *filtrage de motif :*



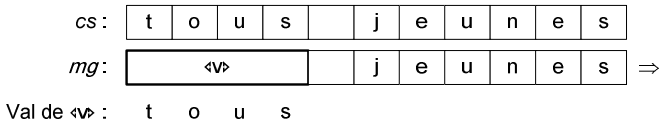
F 5.37. *V* (*mg* constitué d'une seule *V*)

Ce schéma ne présente aucune contrainte, puisque n'importe quelle *cs* peut être filtrée par l'unique *V* du *mg*[61]. Par exemple :



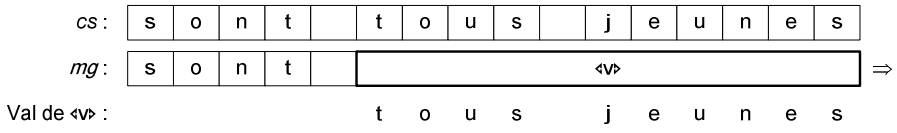
F 5.38. *V** (*mg* commençant par une *V*)

C'est une forme dont on ne peut rien dire si l'on ne connaît pas ce que contient le conteneur : *. Par définition, nous savons qu'une *V* ne peut être suivie que d'une *C*. Dès lors, si la *C* et ce qui la suit satisfont au *filtrage de motif*, nous pourrons déduire la *V*. Dans cet exemple, le *filtrage de motif* ne réussit que si le *mg* est formé de : '‹v› jeunes' :



F 5.39. **V* (*mg* terminé par une *V*)

Connaissant la condition d'alternance entre *C* et *V* dans une *chaîne* du *mg*, nous pouvons dire de cette forme qu'elle est précédée d'une *C*. Par conséquent, une *V* terminant un *mg* couvre la dernière partie de la *cs* depuis la dernière occurrence de la *C* qui a satisfait au *filtrage de motif* et jusqu'à la fin de la *cs*. Par exemple :



F 5.40. *VC* (*mg* constitué d'un schéma *VC*)

Le comportement du *filtrage de motif* de *VC* est similaire à celui de la forme **C*.

[61] Ce schéma présente certains risques quant à son utilisation. Sauf dans des conditions très précises, cet usage fait boucler le *LRE*.

F 5.41. **VC* (*mg* terminé par un schéma *VC*)

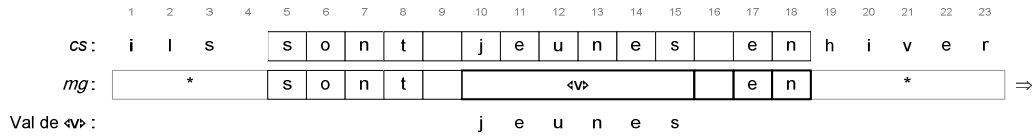
Le comportement du *filtrage de motif* de **VC* est similaire à celui de la forme **C*.

F 5.42. *VC** (*mg* commençant par un schéma *VC*)

Le comportement du *filtrage de motif* de *VC** est similaire à celui de la forme *V**.

F 5.43. **VC** (*mg* constitué d'un ou plusieurs schémas *VC*)

Quant à la forme, **VC**, nous la considérons comme la généralisation de toutes les formes. Quel que soit ce qui précède ou ce qui suit, le *filtrage de motif* recherche pour le motif donné par *VC* la première occurrence de *C* et déduit *V* de la façon suivante, en tenant compte du dernier *filtrage de motif* de la *VC* qui précède. Par exemple :



Dans cet exemple, durant le *filtrage de motif* de la forme, **VC**, qui a précédé celle qui nous intéresse ici, le *LRE* a déterminé que la *C* : 'sont '[62], se termine au caractère 9. Dans le temps d'analyse de la forme **CV** courante, il commence par rechercher une possible prochaine occurrence de la *C* : ' en'[63], qui commence à la position 15. A partir de cette nouvelle analyse, puisqu'il trouve cette occurrence, il peut calculer la *V* : 'jeunes', commençant à la position 10 et se terminant à la position 15.

Nous verrons plus loin des comportements de *filtrage de motif* un peu plus complexes concernant les *variables isoformes*.

*5.4.3.2 Filtrage de motif des schémas *VC**

Comme nous l'avons indiqué plus haut, le processus d'exécution du *filtrage de motif* d'une *rre* se fait de gauche à droite. Donc pour un schéma de type **VC**, le processus commence par rechercher la première occurrence de la première *C*, pour calculer la *V* qui la précède. Décrivons, en détail, le processus de *filtrage de motif* d'un *mg* qui permet de calculer la *V* d'un schéma : **VC**.

RD 5.4-2 En partant du premier schéma, **VC**, qui le constitue, le *mg*, pilote le processus de *filtrage de motif* de schéma **VC** en schéma **VC** jusqu'au dernier et cela tant que le *filtrage de motif* est satisfait.

RD 5.4-3 Pour que le *filtrage de motif* d'un schéma **VC** soit satisfait, il faut que *C* de **VC**, trouve une occurrence qui est équivalente à son motif en continuant l'analyse de la *cs* à partir de la position donnée par l'analyse du schéma qui précède. Un *filtrage de motif,* qui est satisfait, livre toujours la position du dernier caractère traité de la *cs* qui est mise à disposition de l'itération suivante pour calculer le début de sa *V*.

Dans la figure qui suit, nous donnons un exemple de la progression du *filtrage de motif* et de la valeur que prennent les *V* pour chaque schéma **VC** :

[62] Espace typographique y compris.

[63] Espace typographique y compris.

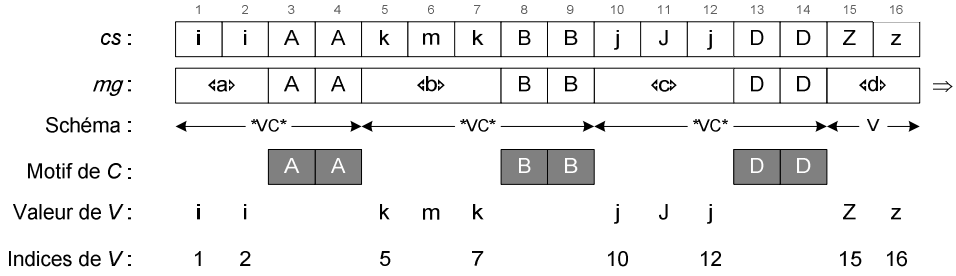


Figure 37. Exemple de la progression d'analyse des schémas**VC**

Si l'ensemble des schémas **VC** d'un *mg* satisfait au *filtrage de motif*, il est considéré que c'est toute la *rre* qui satisfait à cette analyse et elle livre une nouvelle *chaîne cr* qui servira de *cs* pour la même *rre* (ou une suivante). Donc, tant qu'une *rre* satisfait au *filtrage de motif*, elle s'exécute à nouveau sur la nouvelle *chaîne cr* produite qui devient la *cs*. Voici un exemple d'itérations d'une *rre* :

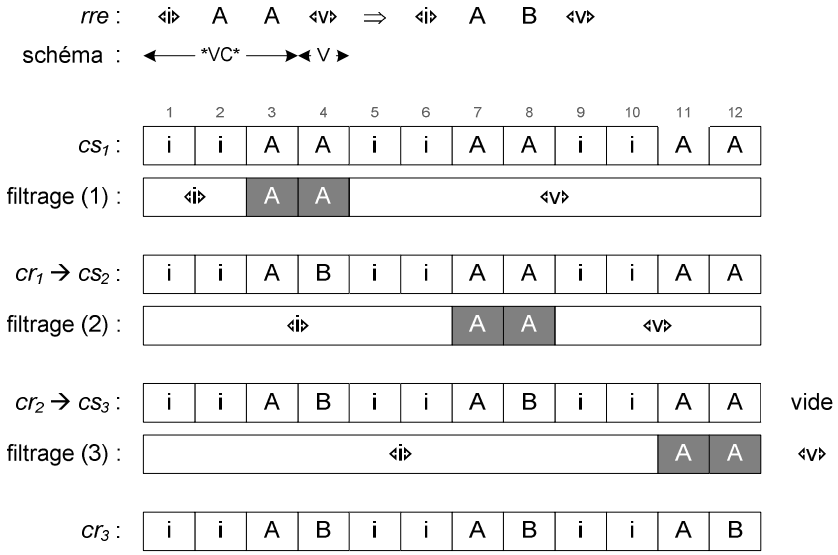


Figure 38. Exemple de trois itérations d'une *rre*

Ce processus répétitif donne de la *puissance* au *LRE*, mais a, évidemment, le désavantage d'augmenter les risques d'itérations infinies. Voici un exemple de *rre* qui ne s'arrête jamais :

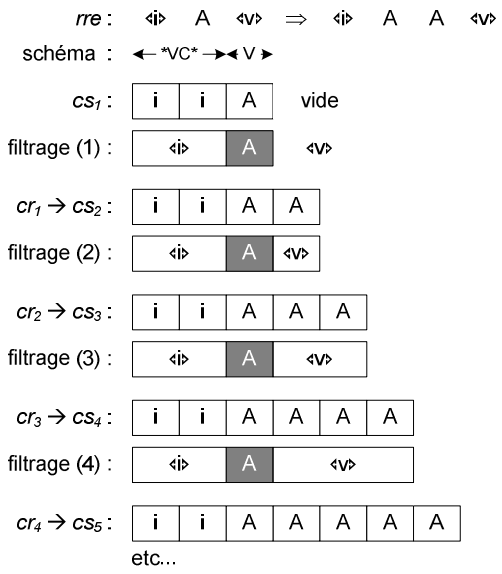


Figure 39. Exemple d'une *rre* qui ne s'arrête jamais

5.4.3.3 Filtrage de motif avec des fonctions conditionnelles appliquées aux V du LRE

L'introduction de *fonctions* conditionnelles appliquées aux *V* comme DANS ou HORS, rendent le processus de *filtrage de motif* un peu plus difficile. Voyons, à partir d'un exemple, comment il est traitée avec une seule *V* à laquelle nous avons appliqué une *fonction* conditionnelle. Soit, la définition d'ensemble, la *rre* et la *cs* suivantes :

F 5.44. Définition de l'ensemble : ☞KL⇒"K","L"☜
☞‹v1›A‹v2›DANS(KL)‹v3›B‹v3›C⇒‹v1›A‹v2›B‹v3›C/exécuter une fois☜
cs : 'xAZBAKBQC'

Suivons maintenant le rôle que doit jouer le *LRE* pour vérifier si la *cs* satisfait ou non au *filtrage de motif* donné par le *mg*. Comme nous l'avons vu, le processus se fait de schéma en schéma **VC**. Les diagrammes qui suivent, montrent chaque étape du filtrage :

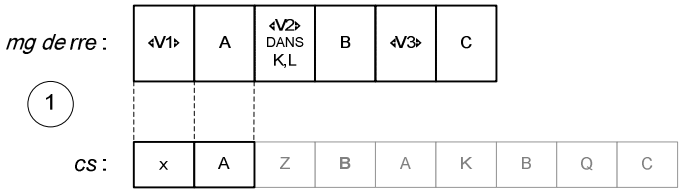


Figure 40. *Filtrage de motif* du premier schéma **VC**

La première étape de filtrage s'applique sans problème puisque la *constante*, A, est trouvée en deuxième position de la *cs* et ‹V1› se calcule sans condition. Attaquons le deuxième schéma :

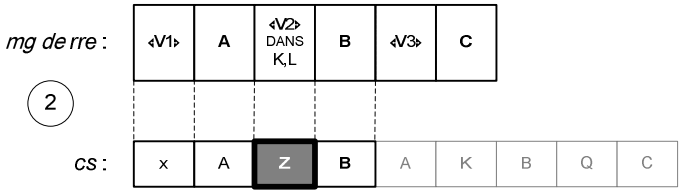


Figure 41. *Filtrage de motif* du deuxième schéma avec échec

La correspondance entre la *constante*, B et le quatrième caractère de la *cs* réussit, mais la condition sur la *variable* ‹V2›, met en échec cette partie du *filtrage de motif*. Ceci ne veut pas dire que tout le filtrage est mis en échec. Il reste donc à vérifier si un « décalage vers la droite » peut satisfaire à nos règles de filtrage :

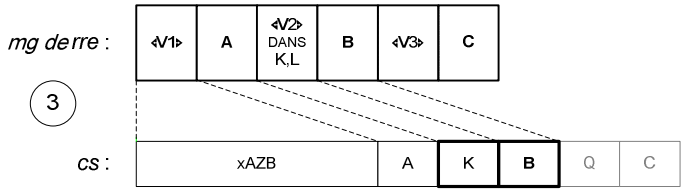


Figure 42. Décalage vers la droite et nouveau filtrage

Cette opération de décalage est parfaitement correcte. Le premier schéma et le deuxième schéma satisfont au *filtrage de motif*. Il reste donc à vérifier que notre dernier schéma correspond aussi pour valider le *filtrage de motif* de tout le *mg* :

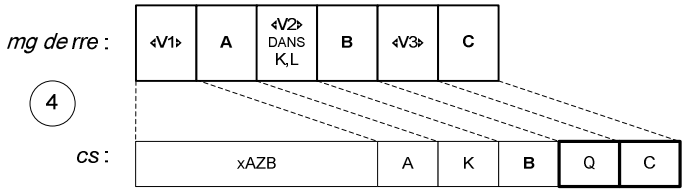


Figure 43. *Filtrage de motifs* satisfait pour tout le *mg*

Cette dernière figure, montre que le troisième schéma, tenant compte du processus qui a précédé est satisfait, ce dernier réussit aussi le *filtrage de motif*. C'est donc le cas pour tout le *mg*.

Analysons maintenant un exemple un peu plus complexe qui a le mérite de montrer le comportement du moteur de filtrage dans le cas d'une situation où plusieurs solutions peuvent se présenter. Soit, la définition d'un ensemble, la *rre* et la *cs* suivantes :

F 5.45. Définition de l'ensemble : ☞KL⇒"g","h"☜
☞‹v1›C‹v2›C‹v3›DANS(gh)›C‹v4›⇒résultat : ‹v1›C‹v2›C‹v3›C‹v4›☜
cs : 'aCbCeCfCgC'

Observons les jeux de *filtrage de motif* et de décalages nécessités par cet exemple pour aboutir :

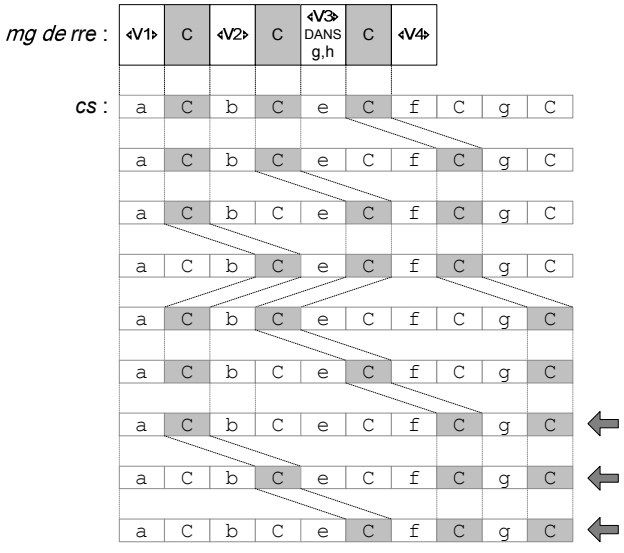


Figure 44. *Filtrages de motifs* et processus de décalage aboutissant à trois solutions

Dans cet exemple, le schéma ‹v3›C, nécessite deux décalages (vers la droite) et quatre nouveaux calculs des schémas qui précèdent avant d'aboutir à une première solution sans échec. Deux étapes supplémentaires pourraient encore calculer deux schémas possibles. Le moteur de filtrage s'arrête à la première solution qui satisfait au *filtrage de motif* du *mg*.

Illustrons encore par un troisième exemple l'utilisation de deux *variables* sur lesquelles nous avons appliqué une fonction conditionnelle. Soit, les définitions d'ensembles, la *rre* et la *cs* suivantes :

F 5.46. Définition de l'ensemble : ☞e⇒"e"☜
☞gh⇒"g","h"☜
☞‹v1›C‹v2›DANS(e)›C‹v3›DANS(gh)›C‹v4›⇒résultat : ‹v1›C‹v2›C‹v3›C‹v4›☜
cs : 'aCbCeCeCgC'

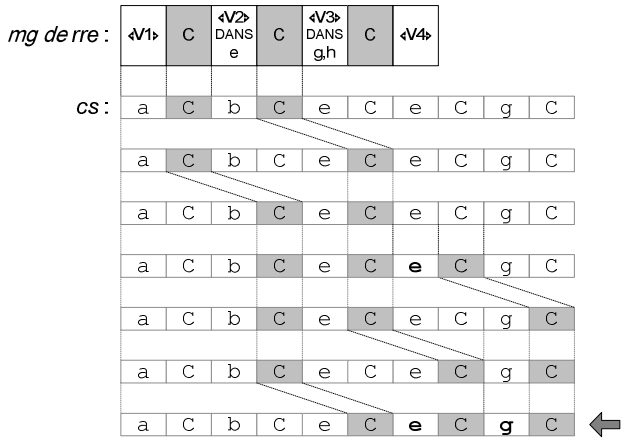


Figure 45. Décalages et calculs du filtrage à partir d'une *rre* avec deux fonctions conditionnelles

Dans cet exemple, une seule solution satisfait au *filtrage de motif*.

Ces quelques exemples nous ont permis d'aborder le principe de décalage vers la droite qu'applique le moteur de filtrage avant de déclarer forfait[64].

5.4.3.4 Filtrage de motif avec des variables isoformes du LRE

Le *filtrage de motif* qui doit tenir compte de *variables isoformes* est encore un peu plus complexe que ce que nous avons vu jusqu'ici. Nous allons décrire non seulement les difficultés que nous avons rencontrées pour l'exécution du filtrage, mais aussi à quel point l'usage de *variables isoformes* donne de la *puissance* au *LRE*.

Nous avons dessiné le *LRE* pour répondre aux besoins d'analyse d'occurrences ou de redondances. Comme nous l'avons vu, un grand nombre d'*inscriptions* leśniewskiennes générées ou dérivées, à partir de la *procédure définitoire* sont, entre autres, des *généralisations*. En d'autres termes, elles sont construites avec les marques symétriques d'un *quantificateur*. Les éléments quantifiés font l'objet de *lieurs*, donc de la gestion des *variables liées*. Dans ce domaine d'intérêt, un survol rapide de la question montre qu'il s'agit de disposer d'outils d'analyse, de comparaison et de contrôle qui permettent d'établir des relations entre deux ou plusieurs portions morphosyntaxiques d'une *expression* donnée. Quels que soient les éléments qui constituent ce qui sépare les deux entités à traiter, le *LRE* doit pouvoir en dire ce que l'on veut en obtenir. Prenons la *chaîne* de *signes* arbitraire :

F 5.47. >ab>V+*/-Vabz<

Prenons un critère d'analyse : « exhiber toutes les *chaînes* c de même forme, où $c \neq \varepsilon$ et $|c| \geq 1$ » ; soit, les trois redondances constituées de deux *chaînes* (soulignées) :

F 5.48. (1) >ab>V+*/-Vabz<
(2) >ab>V+*/-Vabz<
(3) >ab>V+*/-Vabz<

[64] La plus grande difficulté que nous avons rencontrée dans le développement du prototype a été de s'assurer que de façon exhaustive toutes les possibilités de filtrage ont bien été épuisées !

Donnons-nous un nouveau critère d'analyse : « exhiber toutes les *chaînes* constituées uniquement de lettres majuscules » ; soit une redondance de deux *chaînes* (soulignées) :

F 5.49. >ab>V+*/-Vabz<

A des fins d'analyse plus générale, désignons la sous-*chaîne* qui précède la première occurrence de 'V' : '(>ab>)', par le métasymbole '$*_p$' , la sous-*chaîne* intermédiaire qui sépare les deux occurrences de 'V' : '(+*/-)' par le métasymbole : '$*_i$' et ce qui suit la deuxième occurrence de 'V' : '(abz<)' par '$*_s$'. Répondre à notre deuxième critère d'analyse peut se traduire par : « quels que soient les $*_p$, $*_i$ et $*_s$ », nous obtenons :

F 5.50. $*_p V *_i V *_s$

Ce que nous nommons une paire d'occurrences de 'V' de même forme ou isoforme. En admettant que les métasymboles peuvent être n'importe quelle *chaîne*, y compris la *chaîne vide*, ε, nous aurons aussi :

F 5.51. VV

Prenons un nouvel exemple, un peu plus délicat à analyser et numérotons par des indices nos occurrences[65] :

F 5.52. $V_1V_2V_3V_4V_5$

Que pouvons-nous dire de cette *chaîne*, un exemple typique de *variables isoformes* que peut avoir à traiter le *LRE*, en considérant toujours notre deuxième critère d'analyse ? Nous obtenons plusieurs réponses valides : V_1-V_2, V_1-V_3, V_1-V_4, V_1-V_5, V_2-V_3, V_2-V_4, V_2-V_5, V_3-V_4, V_3-V_5, V_4-V_5, V_1V_2- V_3V_4, V_1V_2- V_4V_5, V_2V_3- V_4V_5, soit seize réponses possibles. Le *LRE* livre toujours la première solution. Si nous avions besoin d'un sélecteur nous permettant de choisir parmi ces réponses, il suffirait d'affiner notre critère d'analyse. A titre d'illustration décidons que le métasymbole de la *chaîne* : $*_i$, qui sépare deux occurrences ne peut pas être mis pour la *chaîne vide* (ε). Dès lors, utilisons ce troisième critère : « exhiber toutes les *chaînes* constituées uniquement de majuscules et séparées par une *chaîne*, $c \neq \varepsilon$ » :

F 5.53. $V_1 V_2 *_i V_3 V_4 V_5$

Cette *chaîne* a deux occurrences redondantes : V_1V_2 - V_3V_4 et V_1V_2 - V_4V_5.

Le *LRE* a la capacité de traiter les questions et réponses d'occurrences redondantes. Et, cela par la mise en œuvre des *variables isoformes*, en tenant compte de la contrainte d'alternance entre *C* et *V* du *mg* des *rre*.

Donnons-nous un premier exemple de mise en œuvre de *variables isoformes* avec une *rre* aboutissant à deux *filtrages de motif* réussis :

F 5.54. ☞⌊‹a›xT⌈‹**V**›⌉‹b›⌋⌈µ(‹c›xT⌈‹**V**›⌉‹d›)‹e›⌉⇒occurrence de "‹V›" trouvée☜
cs : '⌊xT⌈p⌉xT⌈q⌉⌋⌈≡(µ(xT⌈p⌉xT⌈q⌉)≡(xT⌈p⌉≡(xT⌈p⌉xT⌈q⌉)))⌉⌈p⌉'
cr₁ : 'occurrence de "p" trouvée'
cr₂ : 'occurrence de "q" trouvée'

Donnons-nous un deuxième exemple qui, à dessein, lève une erreur :

F 5.55. ☞⌊‹a›xT⌈‹**V**›⌉‹b›xT⌈‹**V**›⌉‹c›⌋‹z›⇒Erreur : "‹V›" est répétée dans Q☜
cs : '⌊xT⌈q⌉xT⌈q⌉⌋⌈≡(µ(xT⌈p⌉xT⌈q⌉)≡(xT⌈p⌉≡(xT⌈p⌉xT⌈q⌉)))⌉⌈p⌉'
cr : 'Erreur : "q" est répétée dans Q'

Nous avons déterminé quatre schémas principaux de *variables isoformes* dans un *mg* :

(1) le schéma simple : …*V*V*…

(2) le schéma alterné : …*V*V*…W*W*…

[65] Attention : dans cet exemple, les indices ne sont pas significatifs pour le filtrage de motif ou la reconnaissance d'occurrences !

(3) le schéma inséré : …*V*…W*W*…V*…

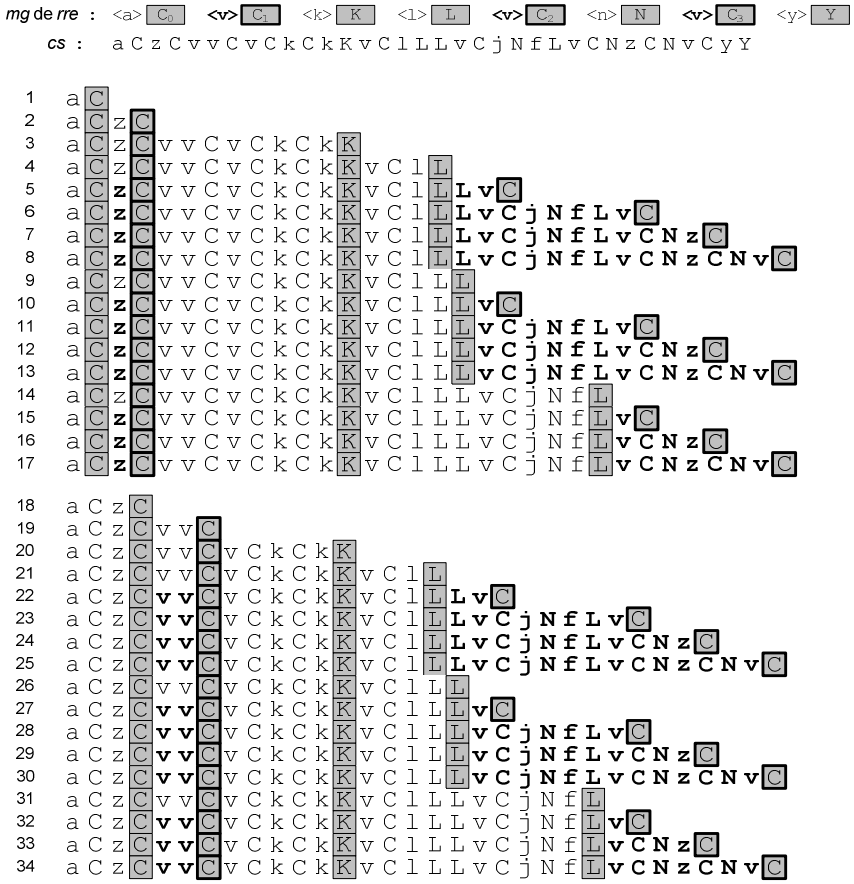
(4) le schéma imbriqué : …*V*…W*…V*…W*…

Evidemment, nous pouvons rencontrer des combinaisons de tous ces schémas, mais, même si le *LRE* permet des combinaisons complexes, afin de maîtriser la programmation de GO_i, nous préférons augmenter le nombre de *rre* plutôt que d'essayer d'utiliser toute la *puissance* des *variables isoformes* dans un minimum de *rre*. Notons en passant que nous n'avons pas trouvé de logiciel basé sur le principe de réécriture sur le marché qui traite ces questions de redondances.

Comme ce qui a été vu dans la section précédente, le traitement des *variables isoformes* utilise aussi la mécanique de décalage vers la droite. Le moteur de filtrage va chercher partout dans la *chaîne cs* si une combinaison de motifs peut ou non être satisfaite. L'exemple qui suit est beaucoup plus complexe que ce que nous avons abordé jusqu'ici et, pourtant, il ne traite que le schéma le plus simple de *variables isoformes* (voir ci-dessus*, alinéa (1) le schéma simple : …*V*V*…, p. 106*), mais il est représentatif du rôle que doit jouer le moteur de filtrage. Soit : la *rre* et la *cs* suivantes :

F 5.56. ☞‹a›C‹v›C‹k›K‹l›L‹v›C‹n›N‹v›C‹y›Y⇒rien faire☜
cs : 'aCzCvvCvCkCkKvClLvCjNfLvCNzCNvCyY'

Donc, un *mg* construit avec huit schémas **VC**, dont trois constitués d'une *variable isoforme*. Et, soit la *cs* de trente-quatre caractères. Comme le nombre d'itérations est relativement bas (84) pour ce petit exemple, nous les donnons dans son ensemble :



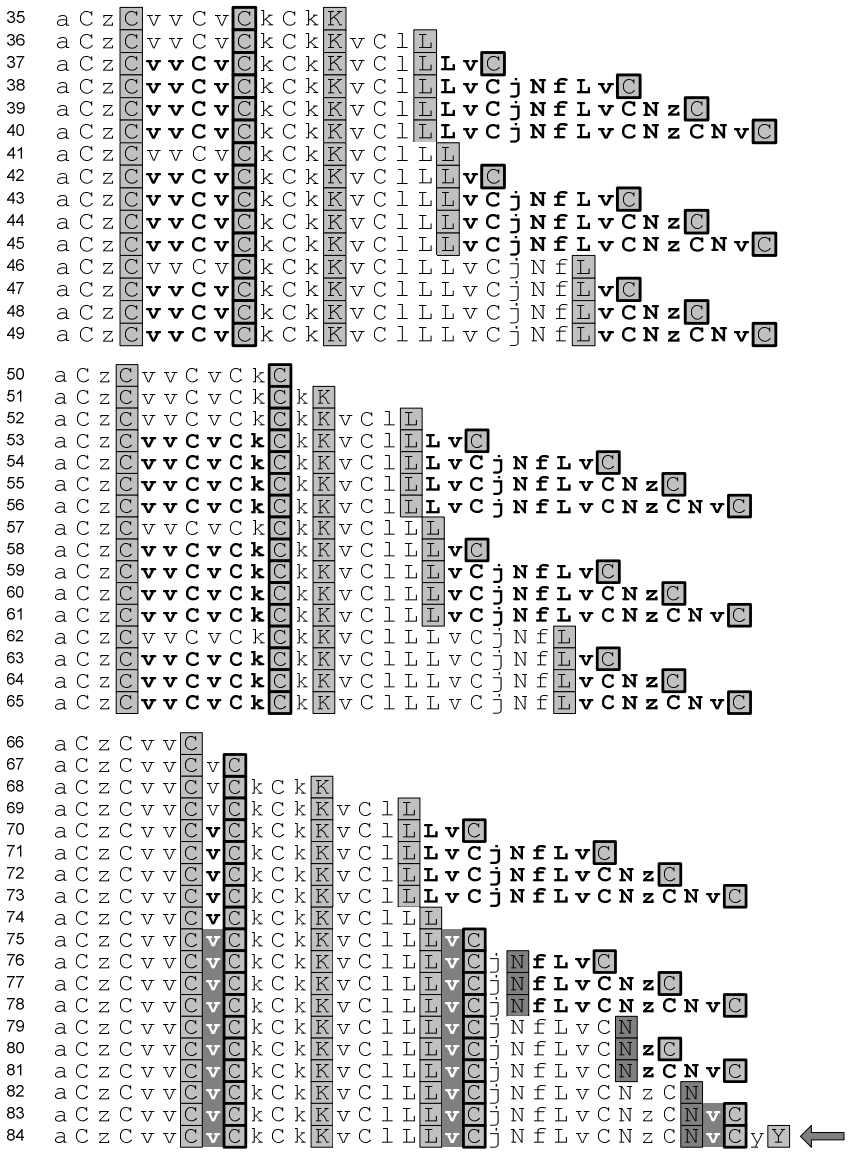


Figure 46. Exemple de *filtrage de motif* avec trois *variables isofomes*

Dans la figure ci-dessus, les *C* sont en gris, la valeur des *V* est en blanc. Quand une *variable isoforme* est calculée, elle est mise en gras pour comparaison avec la *variable isoforme* suivante. A la quatre-vingts quatrième étapes, les trois *variables isoformes* ont la même valeur et tout le reste du filtrage est réussi donc la *rre* exécute la réécriture.

5.4.4 Les fonctions du LRE qui appliquent des conditions sur des V de mg

Aux *fonctions* du *LRE*, DANS et HORS, nous avons ajouté : PASVIDE, ENTIER, LONGUEUR[66]. Ces trois nouvelles *fonctions* appliquent comme les deux premières une contrainte sur une *V*.

PASVIDE évite que le processus de *filtrage de motif* aboutisse si la *V* est vide (V=ε).

ENTIER empêche le *filtrage de motif* de réussir si la *V* concernée n'est pas un nombre entier.

Quant à LONGUEUR(<arg_ari>), cette *fonction* ne permettra au *filtrage de motif* de réussir seulement si la longueur de la *V* à laquelle elle applique la contrainte est égale à <arg_ari> (|V|=<arg_ari>).

[66] Respectivement dans le logiciel prototype : NOTEMPTY, INTEGER et FORCELENGTH.

On trouve à l'alinéa *RD 5.4-28, p. 121* la règle de la notation *EBNF* de la *GRE* qui intègre ces trois nouvelles *fonctions*.

Nous n'avons pas eu besoin d'autres *fonctions* conditionnelles appliquées aux *V* des *mg* des *rre* pour répondre à notre cahier des charges.

5.4.5 Les fonctions des md du LRE qui remplacent des dérivations

Comme pour la *fonction* EXPLOSER, nous avons défini trois autres *fonctions* qui permettent de réduire un ensemble de *dérivations directes* en une seule *dérivation* : COMPTER, TRIER et NUMEROTERDELIM[67]. Ces *fonctions* traitent des *chaînes*.

Nous nous sommes aussi donné une petite caisse à outils pour faire des calculs arithmétiques élémentaires : PLUS, MOINS, MULT, DIV et PUISSANCE[68].

Pour le traitement des calculs propositionnels, nous avons défini deux *fonctions* qui nous facilitent le traitement des tables de vérités : TABLEVERITE et CALCULVERITE[69].

Voyons l'usage de ces *fonctions*.

5.4.5.1 La fonction COMPTER du LRE

Pour traiter les occurrences de *chaînes*, *cs*, soumises à un *mg* d'une *rre*, nous nous sommes donné la *fonction* COMPTER :

↳COMPTER (<portion_chaîne>, ╚<motif_quasi_mg>╝)↲

Cette *fonction*, qui compte le nombre d'occurrences d'une *chaîne* donnée, se déclare avec deux arguments et livre un résultat de type entier positif et de format *chaîne* de caractères. Le premier argument, <portion_chaîne>, est soit une *C* entre guillemets, soit une *V* déclarée dans le *mg*.

Le deuxième argument, <motif_quasi_mg>, désigne le motif de référence qui sert au *filtrage de motif* nécessaire au comptage d'occurrences. Même s'il est mentionné dans une *fonction* dédiée aux *md*, il se comporte comme un *mg*. Par conséquent, il suit les règles de construction vues jusqu'à présent des *mg*. Les occurrences qui doivent être comptées peuvent se trouver en début et/ou en fin de portion de la *chaîne*, *cs*[70]. Etant donnée, la possible complexité du schéma d'un <motif_quasi_mg>, qui peut, par ailleurs commencer par une virgule, le séparateur d'arguments de la *fonction*, nous enveloppons ce deuxième argument par les deux nouveaux *délimiteurs* réservés du *LRE :* ╚ et ╝. Par exemple :

F 5.57. ☞(‹v›)⇒pour ‹v› ↳COMPTER(‹v›, ╚xT[‹t›]╝)↲ occurrence(s) de xT[x]☜
cs : '(xT[p]xT[q]xT[r])'
cr : 'pour xT[p]xT[q]xT[r] 3 occurences de xT[x]'

5.4.5.2 La fonction TRIER du LRE

Certains processus d'analyses comparatives entre deux portions de *chaînes* de caractères, dont nous avons besoin pour le traitement des *lieurs* et des *variables liées*, sont grandement facilités si l'on a trié, au préalable, des structures de motifs similaires et répétitives. La *fonction* :

↳TRIER (<portion_chaîne> , ╚<motif_quasi_mg>╝)↲

---

[67] Respectivement : COUNT, SORT et DELIMNUM dans le logiciel prototype.

[68] Respectivement : PLUS, MINUS, MULT, DIV et POWER dans le logiciel prototype.

[69] Respectivement : TRUTHTABLE et TRUTHCALC dans le logiciel prototype.

[70] Si le motif déclaré, le <motif_quasi_mg> commence et/ou se termine par une *C*, le *LRE* insère automatiquement une *V* en début et/ou fin de motif pour assurer que toutes les occurrences seront comptées y comprises la première et/ou la dernière (sachant que la valeur de *V* peut être la *chaîne vide*).

est destinée à ce rôle de trieuse d'occurrences. D'un point de vue de sa forme, elle se construit comme la *fonction* COMPTER avec les mêmes types d'arguments. Elle retourne une *chaîne* de caractères où les occurrences de la portion soumise sont triées. Par exemple :

F 5.58. ☞(‹v›)⇒en triant ‹v› on obtient : ↳TRIER(‹v›, ╚xT⟦‹t›⟧╝)↲☜
cs : '(xT⟦r⟧xT⟦q⟧xT⟦p⟧)'
cr : 'en triant xT⟦r⟧xT⟦q⟧xT⟦p⟧ on obtient : xT⟦p⟧xT⟦q⟧xT⟦r⟧'

5.4.5.3 La fonction NUMEROTERDELIM du LRE

Même si le *LRE* permet de numéroter les *délimiteurs* ouvrants et fermants à n'importe quel niveau d'imbrications d'*expressions*, nous avons intégré une *fonction* (récursive), qui numérote les *délimiteurs* et qui remplace plusieurs *rre* :

↳NUMEROTERDELIM (<arg_chaîne> , <liste_enveloppe>)↲

En parcourant la portion de *chaîne* déterminée par une *V* du *mg*, cette *fonction* va numéroter toutes les enveloppements rencontrés de *délimiteurs* symétriques déclarés dans le paramètre <liste_enveloppe>.

Comme pour les autres *fonctions* qui traitent des *chaînes* de caractères, le premier argument, <arg_chaîne>, est soit une *constante* de *chaîne* entre guillemets, soit une *V* déclarée dans le *mg* de la *rre*.

Le deuxième argument, <liste_enveloppe>, est un *identificateur* d'ensemble selon la définition *RD 5.4-11, p. 120 et suivantes*. Pour que la *fonction* s'exécute correctement, cet ensemble doit être constitué d'éléments de type « enveloppe » formés d'un *délimiteur* ouvrant et de son symétrique fermant. L'exemple de déclarations d'ensembles qui suit traite les enveloppes nécessaires à la numérotation des *quantificateurs*, *sous-quantificateurs* et *délimiteurs* de *contextes* de la *protothétique :*

F 5.59. ☞q_evp⇒"⌊⌋"☜
☞sq_evp⇒"⌈⌉"☜
☞ctx_evp⇒"()","[]","{}","<>","⋃","〈〉"☜
☞evp⇒q_evp,sq_evp,ctx_evp☜

L'itération principale de la *fonction* parcourt l'ensemble des enveloppes et dans une seconde itération recherche récursivement dans la portion de *chaîne* de caractères soumise, *cs*, à la *rre*, chacune des occurrences des encapsulations constituées par l'enveloppe en cours et les numérote de façon unique quels que soient leurs niveaux d'imbrications. Le format de chaque *encapsulation* numérotée et retournée par la *fonction* est réécrit avec le *délimiteur* concerné suivit par le numéro *encapsulé* lui-même par les parenthèses prédéfinie dans le *LRE :* ⟦ et ⟧[71].

Comme pour la *fonction*, EXPLOSER et pour les mêmes raisons, une *rre* où est mentionnée la *fonction* NUMEROTERDELIM est exécutée une seule fois dans un corps de grammaire GO_i donné. Prenons comme exemple, l'application de cette *fonction* à la *thèse-définition*, D1 (*F 4.13, p. 45*), en reprenant la définition de l'ensemble, evp, qui précède (*F 5.59, p. 110*) :

F 5.60. ☞‹cs›⇒⌈↳DELIMNUM(‹cs›,evp)↲⌉☜
cs : '⌊pq⌋⌈≡(μ(pq)≡(p≡(pq)))⌉'
cr : '⌊⟦1⟧pq⌋⟦1⟧⌈⟦2⟧≡(⟦3⟧μ(⟦4⟧pq)⟦4⟧≡(⟦5⟧p≡(⟦6⟧pq)⟦6⟧)⟦5⟧)⟦3⟧⌉⟦2⟧'

5.4.5.4 La fonction arithmétique PLUS du LRE

A partir de deux arguments de type *chaîne* de caractères numériques, les fonctions arithmétiques retournent une nouvelle *chaîne* inscrite dans la réécriture résultante, *cr*, selon le

[71] Dans une version améliorée du logiciel RE, on pourrait envisager de passer dans les paramètres de la fonction, les délimiteurs. Ce qui rendrait l'utilisation du logiciel plus souple.

schéma du *md*. Pour certaines d'entre elles[72], l'ordre des deux arguments n'est pas commutatif. Ces deux paramètres peuvent être soit une *constante* numérique entre guillemets, soit une *V* mentionnée dans le *mg*. La *fonction* :

↳PLUS(<arg_ari> , <arg_ari>)↲

retourne l'addition, de deux nombres entiers donnés dans les deux arguments. Voici un exemple :

F 5.61. ☞‹a›+‹b›⇒‹a›+‹b›=↳PLUS(‹a›, ‹b›)↲☜
cs : '2+3'
cr : '2+3=5'

5.4.5.5 La fonction arithmétique MOINS du LRE

Similaire, à la *fonction*, PLUS, en ce qui concerne sa construction et son utilisation, mais non commutative, la *fonction* :

↳MOINS(<arg_ari> , <arg_ari>)↲

retourne comme résultat, la soustraction des deux nombres entiers donnés en paramètres.

5.4.5.6 La fonction arithmétique MULT du LRE

Aussi similaire à ce qui précède, la *fonction* :

↳MULT(<arg_ari> , <arg_ari>)↲

retourne la multiplication des deux nombres entiers donnés en paramètres.

5.4.5.7 La fonction arithmétique DIV du LRE

La *fonction* :

↳DIV(<arg_ari> , <arg_ari>)↲

retourne la division entière sans reste des deux nombres donnés en paramètres (la division par zéro est traitée).

5.4.5.8 La fonction arithmétique PUISSANCE du LRE

Cette *fonction*, qui nous permet d'élever un nombre à une puissance donnée, est très utile pour l'affectation combinatoire de tables de vérités à des propositions dans les calculs propositionnels.

La *fonction* :

↳PUISSANCE(<arg_ari> , <arg_ari>)↲

se comporte comme les autres *fonctions* arithmétiques en livrant l'élévation du premier argument (positif et différent de zéro) à la puissance donnée dans le deuxième argument.

5.4.5.9 La fonction TABLEVERITE du LRE

Dans le but de faire du calcul propositionnel, nous devons affecter une table de vérité à chaque *proposition* (ou variable propositionnelle) entrant dans le calcul d'une *expression* donnée. La taille de la table de vérité, pu, d'une *proposition* est évidemment fonction du nombre de *propositions* considéré dans l'*expression* traitée, soit le nombre de *propositions*, np, élevé à la puissance *2* : $pu = np^2$. Chaque table est elle-même constituée d'un motif (répétitif) de uns et de zéros (mis pour vrai ou faux). Evidemment, deux *propositions* différentes doivent avoir des motifs différents. Un motif est formé d'un certain nombre de uns suivi du même nombre de

[72] Pour des raisons évidentes, c'est le cas des re-fonctions MOINS, DIV et PUISSANCE !

zéros, par exemple, pour deux *propositions* p et q, 1100 ou 1010. La table de la *proposition* p est construite sur un seul motif, m, de deux uns et deux zéros, nv, donc nous aurons, m = 1 et nv = 2. Pour la table de la *proposition* q nous avons m = 2 (soit le nombre de motifs) et nv = 1 (soit le nombre d'éléments binaires alternés).

La *fonction*, affecter une table de vérité :

↳TABLEVERITE(‹<nv>› , ‹<m>›)↵

utilise les deux paramètres <nv> (nombre d'éléments binaires) et <m> (nombre de motifs) pour retourner la table de vérité correspondante dans un format où les éléments binaires sont séparés par des virgules. Obtenir <nv> et <m> nécessite un certain nombre de *rre* qui vont, en particulier, calculer le nombre de *propositions*, np, que l'*expression* analysée contient, la taille de toutes les tables de vérité, pu, et d'affecter à toutes les *propositions* identiques, la même table. On trouve un exemple complet de GO_i qui traite ce sujet dans l'annexe : *Calculs propositionnels, p. 231*.

En acceptant que <nv> et <m> aient été préalablement calculés, voici un exemple d'utilisation de cette *fonction*, dans une *rre* qui opère deux itérations à partir de la *cs* :

F 5.62. ☞‹a›xT⦗‹nv›, ‹m›⦘‹p›⦗‹b›⇒‹a›xT⦗↳TABLEVERITE(‹nv›,‹m›)↵⦘‹p›⦗‹b›☜
cs : '(xT⦗2,1⦘⦗p⦘⊃xT⦗1,2⦘⦗q⦘)'
cr_1 *:* '(xT⦗1,1,0,0⦘⦗p⦘⊃xT⦗1,2⦘⦗q⦘)'
cr_2 *:* '(xT⦗1,1,0,0⦘⦗p⦘⊃xT⦗1,0,1,0⦘⦗q⦘)'

5.4.5.10 La fonction CALCULVERITE du LRE

D'un point de vue syntaxique, la *fonction*, calculer une table de vérité :

↳CALCULVERITE (<calc_ver> , ‹<table_verite1>› , ‹<table_verite2>›)↵

n'apporte pas de difficulté particulière. A partir de deux tables et d'un *opérateur* binaire ou unaire logique, elle calcule et retourne la table de vérité résultante. Elle est, toutefois, sujette à des contraintes. Evidemment un *opérateur* unaire ne s'appliquant que sur une seule table, c'est seulement le deuxième argument, <table_verite1>, qui est significatif. Nous n'avons prédéfini et intégré qu'un nombre limité d'*opérateurs* binaires et unaires dans le logiciel prototype[73] ; les plus fréquemment utilisés comme : la *biconditionnelle*, la conditionnelle, la conjonction, les disjonctions, les négations. Si un nouveau calcul, inconnu de la *solution*, doit être réalisé, il s'agira de donner dans le premier paramètre de la *fonction*, la table résultante du calcul entre deux propositions p et q, dont la table de vérité est prédéfinie par défaut comme : val(p) = 1100 et val(q) = 1010, soit l'ordre canonique usuel. Nous pourrions, par exemple, passer dans le premier argument, la définition de table 1,1,1,1 (forçant un calcul aboutissant à une tautologie) qui retournerait toujours vrai quelles que soient les tables soumises dans les deux autres paramètres.

Le premier argument, <calc_ver>, peut être :

- soit, une *constante* de *chaîne* de caractères de type table de vérité (constituée de 1 et de 0 séparés par des virgules) ;
- soit, une *V* mentionnée dans le *mg* et mise pour un *opérateur* connu de la solution logicielle.

Le deuxième et troisième argument <table_verite1> et <table_verite2>, dont l'ordre n'est pas commutatif, sont :

- soit, une *constante* de *chaîne* de caractères de type table de vérité (constituée de 1 et de 0 séparés par des virgules) ;

[73] Logiciel prototype *RE*, version 4.4.

- soit, une *V* mentionnée dans le *mg* et mise pour une *chaîne* de caractères de type table de vérité (constituée de 1 et de 0 séparés par des virgules).

Prenons un exemple :

F 5.63.
```
☞xE(xT⊱‹t1›⊰⟦‹p›⟧‹op›xT⊱‹t2›⊰⟦‹q›⟧)⇒
  xE⊱↳CALCULVERITE(‹op›,‹t1›,‹t2›)↲⊰(xT⟦‹p›⟧‹op›xT⟦‹q›⟧)☜
cs : 'xE(xT⊱1,1,0,0⊰⟦p⟧⊃xT⊱1,0,1,0⊰⟦q⟧)'
cr : 'xE⊱1,0,1,1⊰(xT⟦p⟧⊃xT⟦q⟧)'
```

Ces dernières *fonctions* nous permettent d'enrichir notre *GRE* dont on trouve les définitions à partir de *RD 5.4-31, p. 121*.

5.4.6 Extension du LRE pour la mémorisation et le traitement des corpus

Nous avons vu que dans le *md* d'une *rre*, nous pouvions mémoriser des éléments morphosyntaxiques (*F 5.32, p. 98*) en particulier l'introduction de nouveau *signes* dans un *SL*. Si, d'un point de vue théorique, ce moyen d'enregistrer de l'information est conforme à la philosophie du *LRE*, il est peu pratique à utiliser parce qu'il allonge très sérieusement la longueur des *cs* et *cr*. Nous nous sommes donné d'autres moyens en définissant quatre *corpus* du *LRE :*

(1) le *corpus d'entrée*, constitué d'une liste de *chaînes*, *cs*. Cette liste ne peut pas être vide[74]. Par défaut, la première *cs* de ce *corpus* est soumise au *mg* de la première *rre* d'une grammaire GO_i. Nous doterons le *LRE* des *instructions* qui permettent à une quelconque *rre* d'accéder à une quelconque *chaîne*, *cs*, du *corpus d'entrée*. Nous avons donné à ce *corpus* le nom, mot réservé du *LRE* : ↳CORPUS_ENTRE↲[75]. Avec les *instructions* d'accès aux *corpus*, nous répondons à notre cahier des charges au besoin d'accéder à plusieurs *cs*, en particulier, pour les *règles d'inférences de substitution* (*F 5.10, p. 85*). Le *LRE* ne peut accéder qu'en lecture une quelconque ligne du *corpus d'entrée*, il ne peut pas le modifier ;

(2) le *corpus librairie*, est lui aussi constitué d'une liste de *chaînes*. Il sert de référence prédéfinie dans une section dédiée d'une grammaire GO_i. Le mot réservé utilisé par *LRE* pour désigner ce *corpus* est ↳CORPUS_LIBRAIRIE↲[76]. Comme pour les autres *corpus*, nous disposerons des moyens d'accéder à une quelconque *chaîne* qui le constitue. Comme pour le *corpus d'entrée*, *LRE* ne peut accéder le *corpus librairie* qu'en lecture ;

(3) le *corpus tampon*, qui permet de mémoriser une ou plusieurs *chaînes* durant le traitement d'une GO_i et accéder à une quelconque d'entre elles. Nous utilisons le mot réservé ↳CORPUS_TAMPON↲[77] pour le mentionner. C'est ce *corpus* que nous pouvons utiliser pour mémoriser l'introduction de nouveaux *signes* dans le *SL*. Comme pour les *corpus* précédents, *LRE* peut accéder en lecture à n'importe quelle ligne, mais peut, en plus, y insérer une ligne ;

(4) le *corpus résultat*, soit l'ensemble des *chaînes*, *cr*, résultantes de la réécriture de toutes les itérations des *rre* ayant satisfaits le *filtrage de motif*[78] de toute une grammaire GO_i. De façon automatique, le *LRE* ne fait que d'écrire dans le *corpus résultat*, au fur et à mesure de l'exécution d'une GO_i.

---

[74] Un *corpus d'entrée* vide n'aurait pas de sens !

[75] Dans le logiciel prototype : ↳INPUT_CORPUS↲.

[76] Dans le logiciel prototype : ↳LIBRARY_CORPUS↲.

[77] Dans le logiciel prototype : ↳BUFFER_CORPUS↲.

[78] Il faut noter ici que le *corpus résultat* peut atteindre de très gros volumes, se montant à plusieurs milliers de lignes de réécritures !

Les *instructions* du *LRE* de manipulation des *corpus* indiquent au logiciel *RE* le comportement qu'il doit adopter à l'exécution d'une grammaire GO_i. Les *instructions* peuvent se déclarer comme un cas particulier de *rre* et dans ce cas elles n'opèrent aucune réécriture ou aucune *dérivation*. Elles peuvent aussi être utilisées comme une *fonction* d'un *md* et dans ce cas elles peuvent participer aux processus de *dérivation* en remplaçant une ou plusieurs *dérivations directes*. Nous classons ces *instructions* de manipulation des *corpus* en deux types :

1) huit *instructions* de gestion de *corpus* : ↳BASCULE_CORPUS↲, ↳CORPUS_PREMIER↲, ↳CORPUS_SUIVANT↲, ↳CORPUS_PRECEDENT↲, ↳CORPUS_DERNIER↲, ↳CORPUS_ALLER_A()↲, ↳CORPUS_PRENDRE_LIGNE↲, ↳INSERER_LIGNE↲[79]. Ces *instructions* n'influencent pas le processus de réécriture ou de *dérivation* (du *md*) ;

2) une *instruction* de concaténation ↳CONCAT()↲. Cette *instruction* participe à la réécriture et peut remplacer un ensemble de *dérivations directes*.

5.4.6.1 L'instruction BASCULE_CORPUS du LRE

L'*instruction* ↳BASCULE_CORPUS(<nom_de_corpus>)↲ sélectionne le *corpus* indiqué comme paramètre parmi ↳CORPUS_ENTRE↲, ↳CORPUS_LIBRAIRIE↲ ou ↳CORPUS_TAMPON↲. Par exemple :

↳BASCULE_CORPUS(↳CORPUS_TAMPON↲)↲

A l'exécution de cette *instruction*, le *RE* se positionne automatiquement à la première ligne (*chaîne*) du *corpus*. Par défaut, à l'initialisation du traitement d'une GO_i, la *solution* se branche sur ↳CORPUS_ENTRE↲.

5.4.6.2 Les instructions du LRE de positionnement dans les corpus

Les quatre *instructions* ↳CORPUS_PREMIER↲, ↳CORPUS_DERNIER↲, ↳CORPUS_SUIVANT↲ et ↳CORPUS_PRECEDENT↲ indiquent au *RE* comment se positionner dans la liste des *chaînes* respectivement, à la première ou dernière *chaîne*, à celle qui suit ou précède celle qui a été traitée la dernière fois dans le *corpus* sélectionné.

L'*instruction* ↳CORPUS_ALLER_A(<argument_arithmétique>)↲ donne à la *solution* informatique la ligne à laquelle elle doit se positionner dans le *corpus* sélectionné. Utilisée comme *rre*, cette *instruction* nécessite une constante d'entier non signé comme argument. Par exemple, la *rre* : ↳CORPUS_ALLER_A(14)↲. Si, elle est utilisée dans un *md* d'une *rre*, son paramètre peut être soit un entier non signé, soit une *V* mentionnée dans le *mg* de la *rre*. Par exemple :

F 5.64. ☞‹qqch›ligne=2, aller à ‹pos›⇒LRE pointe ‹pos›↳CORPUSALLERA(‹pos›)☜
cs : 'si ligne=2, aller à 16'
cr : 'LRE pointe 16'

Le *RE* renvoie une exception si le numéro de ligne n'existe pas dans le *corpus*.

5.4.6.3 L'instruction CORPUS_PRENDRE_LIGNE de gestion de corpus du LRE

Nous devons nous assurer, qu'avant d'exécuter l'*instruction*, prendre la valeur de la *chaîne* :

↳CORPUS_PRENDRE_LIGNE↲

tout d'abord le *corpus* et, ensuite, la *chaîne* pointée, sur laquelle nous désirons travailler sont bien sélectionnés par les *instructions* qui les ont précédés. A l'exécution de cette

[79] Respectivement : ↳SWITCHCORPUS↲, ↳CORPUSFIRST↲, ↳CORPUSNEXT↲, ↳CORPUSPREVIOUS↲, ↳CORPUSLAST↲, ↳CORPUSGOTO()↲, ↳GETCORPUSLINE↲, ↳APPENDBUFFERLINE↲ dans le logiciel prototype.

instruction, le *RE* transfère la *chaîne* sélectionnée dans une zone tampon, ↳ZONE_TEMPORAIRE↲[80], qui pourra être accédée par l'*instruction* CONCAT.

Il peut arriver que l'une des *instructions* ↳CORPUS_PRECEDENT↲, ↳CORPUS_SUIVANT↲ ou ↳CORPUS_ALLER_A↲ qui a pu précéder, pointe une location au-delà de la liste de *chaînes* que contient le *corpus* considéré. Dans ce cas, l'*instruction* retourne dans la zone temporaire un indicateur d'état signalant que le *RE* a débordé du *corpus*. A titre d'exemple, donnons-nous un *corpus tampon* (↳CORPUS_TAMPOM↲), de trois *chaînes* de caractères, C_{tampon} et trois *rre :*

F 5.65. $C_{tampon1}$: première ligne du corpus
$C_{tampon2}$: deuxième ligne du corpus
$C_{tampon3}$: troisième ligne du corpus

☞↳BASCULE_CORPUS(↳CORPUS_TAMPON↲)↲☜
☞↳CORPUS_ALLER_A(3)↲☜
☞↳CORPUS_PRENDRE_LIGNE↲☜

A l'exécution de la rre_3, la solution mémorise la *chaîne* de caractères : troisième ligne du corpus dans la zone tampon. Adaptons cette suite de *rre* par l'insertion d'une *rre* supplémentaire pour comprendre le comportement de l'instruction, ↳CORPUS_PRENDRE_LIGNE↲ au moment d'un débordement :

F 5.66. $C_{tampon1}$: première ligne du corpus
$C_{tampon2}$: deuxième ligne du corpus
$C_{tampon3}$: troisième ligne du corpus

☞↳BASCULE_CORPUS(↳CORPUS_TAMPON↲)↲☜
☞↳CORPUS_ALLER_A(3)↲☜
☞↳CORPUS_SUIVANT↲☜
☞↳CORPUS_PRENDRE_LIGNE↲☜

Lors de l'exécution de la quatrième *rre*, le *RE* retourne dans la zone tampon le marqueur de forme prédéfinie par la *solution* informatique : **{<eof>}**. Il s'agira donc de tester par une *rre* ou plusieurs ce marqueur d'état pour parcourir un *corpus* du début jusqu'à la fin ou inversement.

5.4.6.4 L'instruction INSERER_LIGNE du LRE

↳INSERER_LIGNE↲ (ajouter une *chaîne*) permet d'insérer à la fin du *corpus tampon*[81], ↳CORPUS_TAMPON↲, la *chaîne* produite par la dernière *rre* qui a satisfait au *filtrage de motif*. Cette *instruction* permet de mémoriser un ensemble de solutions généré par certaines *règles d'inférences*, comme la *règle d'inférences de distribution des quantificateurs*.

5.4.6.5 L'instruction CONCAT du LRE

L'*instruction* ↳CONCAT (<vers> , <depuis1> , <depuis2>)↲ concatène dans le premier paramètre <vers>, les deux valeurs sources des *chaînes* de caractères <depuis1> et <depuis2>.

Le premier argument, <vers>, la cible de la concaténation, ne peut être déclaré qu'à l'aide de deux mots réservés de nom de zone de mémorisation du *LRE* ↳ZONE_RESULTAT_MD↲[82] et ↳ZONE_TEMPORAIRE↲. Si le nom de la zone, zone de résultat d'un *md*, ↳ZONE_RESULTAT_MD↲, est utilisée, la *solution* ajoutera la valeur mémorisée dans cette zone au *corpus résultat* et la réutilisera comme *chaîne* soumise, *cs*, pour les *rre* suivantes. Au cas où c'est la zone temporaire du *RE*, ↳ZONE_TEMPORAIRE↲, qui

---

[80] Dans le logiciel prototype : ↳TEMPORARY_ZONE↲.

[81] Le *corpus tampon* est une des deux zones dans laquelle la *solution* informatique peut écrire.

[82] Dans le logiciel prototype : ↳RM_RESULT_ZONE↲.

est déclarée, alors le processus d'exécution stockera son résultat temporairement pour être réutilisé à des fins qui ne nécessitent pas d'être traité dans le *corpus résultat*.

Le deuxième (<depuis1>) et troisième argument (<depuis2>) peuvent être invoqués de deux façons : comme pour le premier en accédant aux valeurs mémorisées dans les zones : soit ↳ZONE_RESULTAT_MD↲, soit ↳ZONE_TEMPORAIRE↲ ou en donnant une *constante* de *chaîne* entre guillemets. Pour traiter un exemple, donnons-nous des valeurs pour nos deux zones et quelques *rre*, illustrant notre propos :

F 5.67. ZONE_RESULTAT_MD : des pommes
ZONE_TEMPORAIRE : il mange

☞↳CONCAT(↳ZONE_RESULTAT_MD↲,↳ZONE_TEMPORAIRE↲,"des poires")↲☜

valeur$_1$ de ZONE_RESULTAT_MD : il mange des poires

☞↳CONCAT(↳ZONE_RESULTAT_MD↲,↳ZONE_TEMPORAIRE↲,
↳ZONE_RESULTAT_MD↲)↲☜

valeur$_2$ de ZONE_RESULTAT_MD : il mange des pommes

☞↳CONCAT(↳ZONE_RESULTAT_MD↲,↳ZONE_RESULTAT_MD↲,
↳ZONE_RESULTAT_MD↲)↲☜

valeur$_3$ de ZONE_RESULTAT_MD : des pommesdes pommes

5.4.7 Les directives du LRE

Comme les *instructions*, les *directives* du *LRE* indiquent au *RE* le comportement qu'il doit adopter à l'exécution d'une grammaire GO_i. Elles n'ont pas d'influence directe sur la réécriture ou le processus de *dérivation*. A part : ↳SI_FILTRE_ALLER_REGLE()↲[83] (si le *filtrage de motif* a réussi aller à la *rre*) , les *directives* se déclarent comme un cas particulier de *rre* ou comme une *fonction* de *md*. Nous les classons en trois types[84] :

(1) une commande d'arrêt : ↳STOP↲ ;

(2) deux *directives* de branchement à une *rre* :
↳SI_FILTRE_ALLER_REGLE()↲ et ↳ALLER_A_REGLE↲[85] ;

(3) une *directive* d'appel, ↳EXECUTEGRAM↲, qui permet d'exécuter des grammaires GO_i imbriquées.

5.4.7.1 La directive STOP du LRE

La *directive* qui peut être utilisée comme un type de *rre* à un seul membre ou comme une *fonction* d'un *md :*

☞↳STOP↲☜

termine l'exécution de la grammaire GO_i à l'endroit où elle a été déclarée. Voici un exemple d'utilisation de cette *directive* comme condition d'arrêt dans un *md* :

F 5.68. ☞⌊‹a›xT[‹t↳HORS(termes)↲›]‹b›⌋⇒‹t› est un terme parasite ! ↳STOP↲☜

5.4.7.2 Les directives de branchement du LRE

La *directive* de branchement conditionnel à un numéro de *rre :*

↳SI_FILTRE_ALLER_REGLE(<entier_non_signé>)↲

doit être mise en œuvre avec beaucoup de précautions ; car son comportement peut faire facilement faire s'itérer indéfiniment une grammaire GO_i. Pour son unique paramètre, il s'agit

---

[83] Dans le logiciel prototype : ↳IFMATCH_GOTORULE()↲.

[84] Le logiciel prototype en connaît un quatrième type : une directive de nettoyage interne, ↳DIPOSERESULT↲, permettant de diminuer les volumes trop importants du *corpus résultat*.

[85] Dans le logiciel prototype : ↳GOTORULE↲.

de donner une constante de type entier, le numéro de la *rre* vers laquelle nous voulons rediriger l'exécution de la grammaire GO_i. Le *RE* se charge de tester l'existence de la *rre* cible. Si une et une seule *rre* d'une grammaire GO_i qui précède l'exécution de cette *directive* a satisfait au *filtrage de motif*, alors la condition « si filtré » est satisfaite et la *directive* ordonne au *RE* de se brancher à la *rre* indiquée par son numéro dans son unique paramètre, sinon le *RE* continue à exécuter les *rre* suivantes. Cette *directive* ne peut être invoquée que sous la forme d'une *rre*. Elle ne peut pas être mentionnée dans un *md* de *rre*.

A priori, il y a peu de sens à utiliser une *directive* inconditionnelle, aller à une *rre :*

↳ALLER_A_REGLE(<entier_non_signé>)↲

qui pourrait faire s'itérer indéfiniment une grammaire GO_i, à moins qu'une *instruction* d'arrêt (conditionnelle) soit invoquée entre la position courante de la *directive* et l'endroit où on lui demande de se brancher ou, à moins qu'elle soit destinée à se brancher à une *rre* positionnée après l'exécution de la *directive* en question.

Elle présente néanmoins un intérêt indiscutable si elle est mentionnée dans un *md*. En effet, pour faire nuance à sa sœur ↳SI_FILTRE_ALLER_REGLE()↲, sa mise en œuvre dans une *rre* standard (constituée des deux membres) ne permet d'être exécutée que si la *rre* qui l'invoque a son *mg* qui a satisfait au *filtrage de motif*. En résumé, ↳SI_FILTRE_ALLER_REGLE()↲ s'exécute, si au moins une *rre* qui l'a précédée a satisfait au *filtrage de motif* et ↳ALLER_A_REGLE()↲ s'exécute que si, et seulement si, la *rre* courante a satisfait au *filtrage de motif*.

5.4.7.3 La directive EXECUTEGRAM d'appel d'une grammaire GO_i

A part l'expérience, même si l'on connaît le domaine à couvrir, rien n'indique a priori quelle va être la taille d'une grammaire GO_i. Répondre à notre cahier des charges en ce qui concerne la *protothétique* et l'*ontologie* de Leśniewski représente plus de mille *rre*. Pour faciliter les travaux de programmation et la « réutilisation » de certaines portions de grammaires GO_i, nous avons apporté une *directive* d'appel de module de grammaire, GO_i :

↳EXECUTEGRAM (<nom_de_grammaire>)↲

est construite avec un argument, l'*identificateur* d'une grammaire GO_i qui doit être déclaré au préalable dans la section de la GO_i appelante.

5.4.8 La structure d'une grammaire GO_i

Nous allons maintenant exposer comment une grammaire GO_i se présente dans son ensemble et comment sont structurées ses différentes sections.

5.4.8.1 Les commentaires dans les grammaires GO_i

Un commentaire est un morceau de texte, que le compilateur du logiciel *RE* n'interprète pas. La *GRE* connait deux types de *symboles* pour débuter et terminer un commentaire. Ces *symboles* peuvent être utilisés n'importe où dans une grammaire GO_i.

Le premier, un *symbole* réservé par la *GRE*, '✎', ouvre un commentaire et cela jusqu'à la fin de la ligne où il a été inscrit.

Le deuxième type est composé d'un *symbole* réservé par la *GRE*, ouvrant et débutant le commentaire, '✎', et d'un symbole réservé, fermant et terminant le commentaire, '✍'.

5.4.8.2 Les sections d'une grammaire GO_i

Une grammaire GO_i est construite sur une structure de blocs dont certains obligatoires et d'autres optionnels. On trouve les définitions *EBNF* de la structure principale de la *GRE* au § *RD 5.4-4, p. 120.*

Une GO_i commence toujours par le mot réservé : ↳GRAM↲ suivi d'un *identificateur* unique de nom de grammaire, <DEFINITION_GRAM> qui ouvre le bloc et se termine par le mot réservé : ↳FINGRAM↲. Le compilateur de *RE* n'acceptera pas d'autre construction.

L'ordre des sections est prédéfini et ne peut pas être modifié.

Deux sections sont obligatoires. Une section de paramètres qui commence par le mot réservé : ↳PARAMS↲ et se termine par son dépendant : ↳FINPARAMS↲ et la section de *rre*, constituée de l'enveloppe des mots réservés : ↳REGLES↲ et ↳FINREGLES↲.

Trois blocs sont optionnels. Le premier, la déclaration des grammaires appelées, est constitué de l'enveloppe de mots réservés : ↳INCLURE↲ et ↳FININCLURE↲ ; le deuxième, les définitions d'ensembles, est encapsulé par les mots réservés : ↳ENS↲ et ↳FINENS↲ et, le troisième, la déclaration d'une *librairie*, à savoir, l'enveloppe formée par les mots réservés : ↳LIBRAIRIE↲ et ↳FINLIBRAIRIE↲.

5.4.8.3 La section obligatoire de paramètres

On trouve la définition de cette section obligatoire dans la notation *EBNF* de la *GRE* à *RD 5.4-8, p. 120*.

Exemple de déclarations pour cette section :

F 5.69.
```
↳PARAMS↲
☞MAXGRAMBOUCLE⇒50☜
☞MAXREGLEBOUCLE⇒400☜
↳ENDPARAMS↲
```

Les deux lignes de cette section sont les bornes que l'on se donne pour éviter, non seulement les possibles itérations infinies d'analyse de *filtrage de motif* réussis (MAXREGLEBOUCLE), mais aussi le nombre d'itérations de la grammaire GO_i elle-même (MAXGRAMBOUCLE).

5.4.8.4 La section optionnelle de déclaration de GO_i appelées

Dans le cas où un utilisateur veut profiter de la modularisation des grammaires GO_i et appeler une sous-grammaire dans sa grammaire principale, il devra donner dans sa GO_i les éléments minimaux de connaissance permettant l'accès à cette sous-grammaire (pour la syntaxe, se référer à *RD 5.4-6, p. 120*).

Il faut indiquer au logiciel *RE* où il trouvera les fichiers des sous-grammaires GO_i, au moment où il devra les appeler. Voici un exemple de programme de cette section :

F 5.70.
```
↳INCLURE↲
☞PREANALYSE⇒"D:\TheseL\GramXAnalyse\GX01PreAnalyse.txt"☜
☞PROCDEF⇒"D:\TheseL\GramXAnalyse\GX02ProcDef.txt"☜
☞DEPX⇒"D:\TheseL\GramXDepouille\GX05Depouille.txt"☜
☞CATEGORIE⇒"D:\TheseL\GramXCategorie\GX03Categorie.txt"☜
☞CALCULUS⇒"D:\TheseL\GramXCalculus\GX04Calculus.txt"☜
☞CONVERTXC⇒"D:\TheseL\GramXConvert\GX06Convert.txt"☜
☞DEPC⇒"D:\TheseL\GramCDepouille\GC06Depouille.txt"☜
↳FININCLURE↲
```

Les deux *identificateurs* de nom d'une grammaire GO_i appelée et de la déclaration de la section de déclaration de grammaire principale qui l'appelle doivent être identiques.

5.4.8.5 L'ensemble des caractères

La plus petite entité que traitent *LRE* et notre *RE* est le caractère. On compte parmi l'ensemble des caractères des sous-ensembles comme les majuscules ou minuscules de l'alphabet. Pour ce type de sous-ensembles, nous parlons de caractères imprimables. En informatique, l'espace typographique est aussi considéré comme un caractère, mais non

imprimable. Pour la couche linguistique *LO* nous utilisons plutôt le terme de *signe* pour désigner cette plus petite entité.

Le *LO* est très gourmand en nombre de *signes* en particulier en ce qui concerne les *délimiteurs* et les *termes*. Nous comptons qu'une bonne centaine de *signes*, de caractères supplémentaires à la base traditionnelle informatique[86], est nécessaire. Nous n'avons trouvé aucune *solution* informatique de type système expert ou de système de réécriture sur le marché qui traite le volume de caractères dont nous avons besoin.

Afin d'éviter de rendre le sujet plus compliqué qu'il ne l'était déjà initialement, nous avons décidé de rester le plus proche possible du corpus de signes utilisés par Leśniewski et Denis Miéville. Nous aurions pu nous borner à une utilisation standard des deux cent cinquante-six caractères en définissant un codage supplémentaire. Par exemple, nous pourrions nous contenter que d'une seule paire de *délimiteurs* et de la numéroter par des indices :

$$\lfloor \ldots \rfloor =_{df} (_{101} \ldots)_{101}$$

Dans le même sens, nous pourrions indicer les *termes constants*, par exemple :

$$\supset =_{df} *_3$$

A partir d'une *inscription* relativement simple :

$$\lfloor a \rfloor \lceil \equiv (\neg \{a\} \lfloor BC \rfloor \lceil \supset (\wedge (\varepsilon \{Ba\} \varepsilon \{Ca\}) \varepsilon \{BC\}) \rceil) \rceil$$

Nous obtiendrions le codage suivant :

$$(_{101}a)_{101}(_{102}*_1(_1*_{15}(_4a)_4(_{101}BC)_{101}(_{102}*_3(_1*_2(_1*_{10}(_4Ba)_4*_{10}(_4Ca)_4)_1*_{10}(_4BC)_4)_1)_{102})_1)_{102}$$

Autant oublier immédiatement cette approche, même comme structure intermédiaire, sachant que, de plus, nous chercherons à *encapsuler* avec des informations complémentaires chaque entité qui la compose.

Par conséquent, nous nous sommes orienté vers une *solution* informatique, le *RE*, capable de traiter un très grand nombre de caractères sans s'écarter du choix de signes que proposent Leśniewski et Denis Miéville. D'un point de vue informatique, il s'agit d'utiliser le codage, *unicode*[87], de 2^{16} combinaisons possibles de caractères, soit plus de 64 000 caractères. Cette approche implique la création d'une police de caractères, une table qui regroupe la forme graphique et son codage. Nous avons donc créé la police : « *Lesniewski* » à partir de la police Arial[88]. C'est en particulier cette police qui est utilisée pour pratiquement toutes les formules, *expressions*, *inscriptions*, notation *EBNF*, etc. dans le présent ouvrage[89]. Nous avons dès lors à disposition tout le corpus de *signes* nécessaires pour couvrir les besoins du *LO*.

5.4.8.6 La section optionnelle de déclaration d'une « librairie »

Nous avons présenté notre besoin de mémoriser des *inscriptions*. L'utilisation d'une *librairie* en est un des moyens. Le *LRE* et notre *solution* informatique, *RE* (version 4.4 du logiciel prototype) nécessitent que l'on déclare les éléments de cette *librairie* dans une des sections de la grammaire GO_i. Pour la construction syntaxique de cette section, on se réfèrera à *RD 5.4-15, p. 121*. Chaque ligne de la *librairie* est constituée de deux zones (la seconde étant optionnelle) séparées par les *délimiteurs* : ⌞ et ⌟. Entre chaque *délimiteur*, nous pouvons inscrire n'importe quelle *chaîne* de caractères. Comme exemple, voici un extrait de déclaration d'une *librairie :*

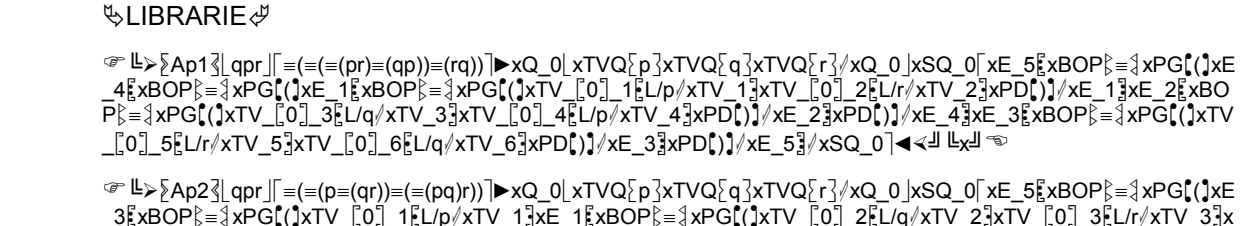
---

[86] Qui en compte 256 dans le standard (ASCII).

[87] *Unicode* : Code Universel. Codage de caractères permettant entre autres de coder les idéogrammes du mandarin.

[88] Fournie dans les différentes versions du logiciel Windows de Microsoft.

[89] Cette police « *Lesniewski* » est fournie avec notre prototype de logiciel.



Cette définition paraît bien exotique. Cet extrait fait partie des exemples de grammaires GO_i qui traitent, en particulier, la *règle d'inférences de détachement*. Seule la première zone est utilisée. Dans cet exemple, la *librairie* contient deux *inscriptions* leśnievskiennes (les deux premiers *axiomes* de la *protothétique*) transposées en *forme analysée* (*langage pivot*) que nous verrons plus loin.

5.4.9 Les productions de la grammaire GRE

La grammaire *GRE* du langage *LRE* étant complètement définie, nous donnons ci-après sa grammaire complète notation *EBNF* (les identificateurs en majuscules ont été mis pour des raisons de lisibilité - il n'y pas d'autre systématique sous-jacente) :

EBNF de la *GRE* (version française)

Structure d'une GO_i

RD 5.4-4
```
<GO> ::=  <mr_gram><DEFINITION_GRAM>
          [<mr_inclure> <LISTE_INCLUSION> <mr_fin_inclure>]
          <mr_params> <LISTE_PARAM> <mr_fin_params>
          [<mr_ens><LISTE_ENSEMBLE><mr_fin_ens>]
          [<mr_librairie> <LISTE_LIBRAIRIE> <mr_fin_librairie>]
          <mr_regle><LISTE_RRE><mr_fin_regle>
          <mr_fingram>
```

RD 5.4-5
```
<DEFINITION_GRAM> ::= <sr_maing><mr_nomgram> <sr_est>
                      <nom_gram><sr_maind>
```

Déclaration d'appels de grammaires GO_i

RD 5.4-6 `<LISTE_INCLUSION> ::= <INCLUSION> {<INCLUSION>}`

RD 5.4-7 `<INCLUSION> ::= <sr_maing> <nom_gram> <sr_est> <fichier> <sr_maind>`

Paramètres

RD 5.4-8 `<LISTE_PARAM> ::= <PARAM> {<PARAM>}`

RD 5.4-9
```
<PARAM> ::=  <sr_maing> <mr_param_ident> <sr_est> <val_param_entier>}
             <sr_maing>
```

RD 5.4-10 `<mr_param_ident> ::= <mr_gram_boucle> | <max_règle_boucle>`

Déclaration des ensembles

RD 5.4-11 `<LISTE_ENSEMBLE> ::= <ENSEMBLE> {<ENSEMBLE>}`

RD 5.4-12 `<ENSEMBLE> ::= <ident_ensemble> <sr_est> <LISTE_ELEMENT>`

RD 5.4-13 <LISTE_ELEMENT> ::= <ELEMENT> { <sr_sep_virgule> <ELEMENT>}

RD 5.4-14 <ELEMENT> ::= <sr_gime> <chaîne> <sr_gime> | <ident_ensemble>

Déclaration du corpus librairie

RD 5.4-15 <LISTE_LIBRAIRIE> ::= <LIGNE_LIBRAIRIE> {<LIGNE_LIBRAIRIE>}

RD 5.4-16 <LIGNE_LIBRAIRIE> ::= <sr_maing> <sr_sepg_entité> <chaîne>
<sr_sepd_entité>
[<sr_sepg_entité> <chaîne> <sr_sepd_entité>]
<sr_maind>

Section de règles de réécriture

RD 5.4-17 <LISTE_RRE> ::= <sr_maing><RRE_GEN> <sr_maind> {<sr_maing>
<RRE_GEN> <sr_maind>}

RD 5.4-18 <RRE_GEN> ::= <RRE> | <DIRECTIVE>

RD 5.4-19 <DIRECTIVE> ::= <INSTRUCTION> |
<branchement_conditionel>

RD 5.4-20 <INSTRUCTION> ::= <corpus_premier> | <corpus_suivant> |
<corpus_précédent> | <corpus_dernier> |
<corpus_aller> | <bascule_corpus> |
<corpus_prendre_ligne> | <aller_à_règle> |
<exécute_gram> | <insérer_ligne> | <stop> |
<concat>

Règle de réécriture (rre)

RD 5.4-21 <RRE> ::= <MG> <sr_réécrit> <MD>

Membre de gauche (mg)

RD 5.4-22 <MG> ::= <C_MG> | <V_MG>

RD 5.4-23 <C_MG> ::= <C> [<M_MG>]

RD 5.4-24 <V_MG> ::= <VG> [<M_MG>]

RD 5.4-25 <M_MG> ::= <VCM> {<VCM>} [<VG>]

RD 5.4-26 <VCM> ::= <VG> <C>

RD 5.4-27 <VG> ::= <sr_varg> <ident> [<F_MG>] <sr_vard>

RD 5.4-28 <F_MG> ::= <dans> | <hors> | <pas_vide> |
<être_type_entier> | <avoir_longueur>

Membre de doite (md)

RD 5.4-29 <MD> ::= <A_MD> {<A_MD>}

RD 5.4-30 <A_MD> ::= <C> | <V> | <F_MD>

RD 5.4-31 <F_MD> ::= <F_MD_CHAINE> | <F_MD_ARI> | <F_MD_TABLE> |
<DIRECTIVE>

RD 5.4-32 <F_MD_CHAINE> ::= <exploser> | <compter> | <trier> | <numéroter_délim>

RD 5.4-33 <F_MD_ARI> ::= <sr_maing> <opérateur_ari> <sepg_arg> <arg_ari>
<sr_sep_virgule> <arg_ari> <sepd_arg> <sr_maind>

RD 5.4-34 <opérateur_ari> ::= <mr_plus> | <mr_moins> | <mr_mult> | <mr_div> | <mr_puissance>

RD 5.4-35 <F_MD_TABLE> ::= <table_vérité> | <calcul_vérité>

Constante C (terminale) et variable V (non terminale)

RD 5.4-36 <C> ::= <chaîne>

RD 5.4-37 <V> ::= <sr_varg> <ident> <sr_vard>

Structures des directives, fonctions et instructions

RD 5.4-38 <branchement_conditionel> ::= <sr_mrg> <mr_branche_cond> <sepg_arg> <entier> <sepd_arg> <sr_mrd>

RD 5.4-39 <corpus_premier> ::= <sr_mrg> <mr_corpus_premier> <sr_mrd>

RD 5.4-40 <corpus_suivant> ::= <sr_mrg> <mr_corpus_suivant> <sr_mrd>

RD 5.4-41 <corpus_précédent> ::= <sr_mrg> <mr_corpus_précédent> <sr_mrd>

RD 5.4-42 <corpus_dernier> ::= <sr_mrg> <mr_corpus_dernier> <sr_mrd>

RD 5.4-43 <corpus_aller> ::= <sr_mrg> <mr_corpus_aller> <sepg_arg> <entier> <sepd_arg> <sr_mrd>

RD 5.4-44 <bascule_corpus> ::= <sr_mrg> <mr_bascule_corpus> <sepg_arg> <nom_corpus> <sepd_arg> <sr_mrd>

RD 5.4-45 <corpus_prendre_ligne> ::= <sr_mrg> <mr_corpus_prendre_ligne> <sr_mrd>

RD 5.4-46 <aller_à_règle> ::= <sr_mrg> <mr_aller_à_règle> <sepg_arg> <entier> <sepd_arg> <sr_mrd>

RD 5.4-47 <exécute_gram> ::= <sr_mrg> <mr_exécute_gram> <sepg_arg> <nom_gram> <sepd_arg> <sr_mrd>

RD 5.4-48 <insérer_ligne> ::= <sr_mrg> <mr_insérer_ligne> <sr_mrd>

RD 5.4-49 <stop> ::= <sr_mrg> <mr_stop> <sr_mrd>

RD 5.4-50 <concat> ::= <sr_mrg> <mr_concat> <sepg_arg> <concat_vers> <sr_sep_virgule> <concat_de> <sr_sep_virgule> <concat_de> <sepd_arg> <sr_mrd>

RD 5.4-51 <dans> ::= <sr_mrg> <mr_dans> <sepg_arg> <arg_ensemble> <sepd_arg> <sr_mrd>

RD 5.4-52 <hors> ::= <sr_mrg> <mr_hors> <sepg_arg> <arg_ensemble> <sepd_arg> <sr_mrd>

RD 5.4-53 <pas_vide> ::= <sr_mrg> <mr_pas_vide> <sr_mrd>

RD 5.4-54 <être_type_entier> ::= <sr_mrg> <mr_être_type_entier> <sr_mrd>

RD 5.4-55 <avoir_longueur> ::= <sr_mrg> <mr_avoir_longueur> <sepg_arg> <arg_ari> <sepd_arg> <sr_mrd>

RD 5.4-56 <exploser> ::= <sr_mrg> <mr_exploser> <sepg_arg> <arg_chaîne> <sr_sep_virgule> <sr_gime> <chaîne> <sr_gime> <sr_sep_virgule> <sr_gime> <chaîne> <sr_gime> <sepd_arg> <sr_mrd>

RD 5.4-57 <compter> ::= <sr_mrg> <mr_compter> <sepg_arg> <arg_chaîne> <sr_sep_virgule> <déclare_motif_quasi_mg> <sepd_arg> <sr_mrd>

RD 5.4-58 <trier> ::= <sr_mrg> <mr_trier> <sepg_arg> <arg_chaîne> <sr_sep_virgule> <déclare_motif_quasi_mg> <sepd_arg> <sr_mrd>

RD 5.4-59 <numéroter_délim> ::= <sr_mrg> <mr_numéroter_délim> <sepg_arg> <arg_chaîne> <sr_sep_virgule> <liste_evp> <sepd_arg> <sr_mrd>

RD 5.4-60 <table_vérité> ::= <sr_mrg> <mr_table_vérité> <sepg_arg> <arg_nv> <sr_sep_virgule> <arg_m> <sepd_arg> <sr_mrd>

RD 5.4-61 <calcul_vérité> ::= <sr_mrg> <mr_calcul_vérité> <sepg_arg> <arg_foncteur> <sr_sep_virgule> <var_table> <sr_sep_virgule> <var_table> <sepd_arg> <sr_mrd>

Sous structures des fonctions, instructions et directives

RD 5.4-62 <nom_corpus> ::= <mr_corpus_entrée> | <mr_corpus_librairie> | <mr_corpus_tampon>

RD 5.4-63 <concat_vers> ::= <mr_zone_résultat> | <mr_zone_temporaire>

RD 5.4-64 <concat_de> ::= <mr_zone_résultat> | <mr_zone_temporaire> | <sr_gime> <chaîne> <sr_gime>

RD 5.4-65 <déclare_motif_quasi_mg> ::= <sr_sepg_entité> <motif_quasi_mg> <sr_sepd_entité>

RD 5.4-66 <liste_evp> ::= <delimg><delimd> {<delimg><delimd>}

Structures des arguments des fonctions, instructions et directives

RD 5.4-67 <nom_gram> ::= <ident>

RD 5.4-68 <val_param_entier> ::= <entier> | <ident>

RD 5.4-69 <arg_ensemble> ::= <V> | <ident>

RD 5.4-70 <ident_ensemble> ::= <ident>

RD 5.4-71 <arg_ari> ::= <V> | <entier_signé> | <entier>

RD 5.4-72 <arg_chaîne> ::= <V> | <sr_gime> <chaîne> <sr_gime>

RD 5.4-73 <motif_quasi_mg> ::= <MG>

RD 5.4-74 <arg_nv> ::= <varg> <entier> <vard>

RD 5.4-75 <arg_m> ::= <varg> <entier> <vard>

RD 5.4-76 <arg_foncteur> ::= <sym_foncteur>| <table>

RD 5.4-77 <var_table> ::= <varg> <table> <vard>

Définitions des terminaux usuels

RD 5.4-78 <ident> ::= <lettre> {<lettre> | <chiffre>}

RD 5.4-79 <lettre> ::= 'A' | 'B' | … | 'a' | 'b' | …

RD 5.4-80 <chiffre> ::= '0' | '1' | '2' | …

RD 5.4-81 <entier> ::= <chiffre> {<chiffre>}

RD 5.4-82 <entier_signé> ::= <signe_ari> <entier>

RD 5.4-83 <signe_ari> ::= '+' | '-'

RD 5.4-84 <chaîne> ::= <car> {<car>}

RD 5.4-85 <car> ::= « tous les caractères, lettres, signes de ponctuation, chiffres, symboles, signes, que la *solution* permet »

RD 5.4-86 <delimg> ::= '(' | '[' | '{' | …
<delimd> ::= ')' | ']' | '}' | …

RD 5.4-87 <sym_foncteur> ::= 'µ' | 'Ц' | '∧' | '∨' | '⊃' |…

RD 5.4-88 <table> ::= <booléen> {<sr_sep_virgule> <booléen>}

RD 5.4-89 <booléen> ::= '1' | '0'

Définitions des symboles réservés

RD 5.4-90 <sr_réécrit> ::= '⇒'
<sr_est> ::= '⇒'

<sr_sep_virgule> ::= ','

<sr_gime> ::= '"'

<sr_sepg_entité> ::= '╚'
<sr_sepd_entité> ::= '╝'

<sr_maing> ::= '☞'
<sr_maind> ::= '☜'

<sr_mrg> ::= '↳'
<sr_mrd> ::= '↲'

<sepg_arg> ::= '('
<sepd_arg> ::= ')'

<sr_varg> ::= '⊰'
<sr_vard> ::= '⊱'

Définitions de mots réservés

RD 5.4-91 <mr_gram> ::= <sr_mrg> 'GRAM' <sr_mrd>
<mr_inclure> ::= <sr_mrg> 'INCLURE' <sr_mrd>
<mr_fin_inclure> ::= <sr_mrg> 'FININCLURE' <sr_mrd>
<mr_params> ::= <sr_mrg> 'PARAMS' <sr_mrd>
<mr_fin_params> ::= <sr_mrg> 'FINPARAMS' <sr_mrd>
<mr_ens> ::= <sr_mrg> 'ENSEMBLES' <sr_mrd>
<mr_fin_ens> ::= <sr_mrg> 'FINENSEMBLES' <sr_mrd>
<mr_librairie> ::= <sr_mrg> 'LIBRAIRIE' <sr_mrd>
<mr_fin_librairie> ::= <sr_mrg> 'FINLIBRAIRIE' <sr_mrd>
<mr_regle> ::= <sr_mrg> 'REGLES' <sr_mrd>
<mr_fin_regle> ::= <sr_mrg> 'FINREGLES' <sr_mrd>
<mr_fingram> ::= <sr_mrg> 'FINGRAM' <sr_mrd>
<mr_nomgram> ::= 'NOMGRAM'
<mr_gram_boucle> ::= 'MAXGRAMBOUCLE'
<max_règle_boucle> ::= 'MAXREGLEBOUCLE'
<mr_branche_cond> ::= 'SI_FILTRE_ALLER_REGLE'
<mr_corpus_premier> ::= 'CORPUS_PREMIER'
<mr_corpus_suivant> ::= 'CORPUS_SUIVANT'
<mr_corpus_précédent> ::= 'CORPUS_PRECEDENT'
<mr_corpus_dernier> ::= 'CORPUS_DERNIER'
<mr_corpus_aller> ::= 'ALLER_A_REGLE'
<mr_bascule_corpus> ::= 'BASCULE_CORPUS'
<mr_corpus_entrée> ::= <sr_mrg> 'CORPUS_ENTRE' <sr_mrd>
<mr_corpus_librairie> ::= <sr_mrg> 'CORPUS_LIBRAIRIE' <sr_mrd>
<mr_corpus_tampon> ::= <sr_mrg> 'CORPUS_TAMPON' <sr_mrd>
<mr_corpus_prendre_ligne> ::= 'CORPUS_PRENDRE_LIGNE'
<mr_aller_à_règle> ::= 'ALLER_A_REGLE'

<mr_exécute_gram> ::= 'EXECUTEGRAM'
<mr_insérer_ligne> ::= 'INSERER_LIGNE'
<mr_stop> ::= 'STOP'
<mr_concat> ::= 'CONCAT'
<mr_zone_résultat> ::= <sr_mrg> 'ZONE_RESULTAT_MD' <sr_mrd>
<mr_zone_temporaire> ::= <sr_mrg> 'ZONE_TEMPORAIRE' <sr_mrd>
<mr_dans> ::= 'DANS'
<mr_hors> ::= 'HORS'
<mr_pas_vide> ::= 'PASVIDE'
<mr_être_type_entier> ::='ENTIER'
<mr_avoir_longueur> ::= 'LONGUEUR'
<mr_exploser> ::= 'EXPLOSER'
<mr_compter> ::= 'COMPTER'
<mr_trier> ::= 'TRIER'
<mr_numéroter_délim> ::= 'NUMEROTERDELIM'
<mr_plus> ::= 'PLUS'
<mr_moins> ::= 'MOINS'
<mr_mult> ::= 'MULT'
<mr_div> ::= 'DIV'
<mr_puissance> ::= 'PUISSANCE'
<mr_table_vérité> ::= 'TABLEVERITE'
<mr_calcul_vérité> ::= 'CALCULVERITE'

Divers

RD 5.4-92 <fichier> ::= « format fichier Windows »

Nous avons développé le logiciel prototype *RE* V4.4 en anglais. La grammaire *GRE* a été élaborée de façon à permettre de traduire facilement les mots réservés. Voici le seul bloc de cette grammaire qui change et qui remplace ***RD 5.4-91, p. 124*** pour la version anglaise :

Définitions de mots réservés

RD 5.4-93 <mr_gram> ::= <sr_mrg> 'GRAM' <sr_mrd>
<mr_inclure> ::= <sr_mrg> 'INCLUDE' <sr_mrd>
<mr_fin_inclure> ::= <sr_mrg> 'ENDINCLUDE' <sr_mrd>
<mr_params> ::= <sr_mrg> 'PARAMS' <sr_mrd>
<mr_fin_params> ::= <sr_mrg> 'ENDPARAMS' <sr_mrd>
<mr_ens> ::= <sr_mrg> 'SETS' <sr_mrd>
<mr_fin_ens> ::= <sr_mrg> 'ENSETS' <sr_mrd>
<mr_librairie> ::= <sr_mrg> 'LIBRARY' <sr_mrd>
<mr_fin_librairie> ::= <sr_mrg> 'ENDLIBRARY' <sr_mrd>
<mr_regle> ::= <sr_mrg> 'RULES' <sr_mrd>
<mr_fin_regle> ::= <sr_mrg> 'ENDRULES' <sr_mrd>
<mr_fingram> ::= <sr_mrg> 'ENDGRAM' <sr_mrd>
<mr_nomgram> ::= 'GRAMNAME'
<mr_gram_boucle> ::= 'MAXGRAMLOOP'
<max_règle_boucle> ::= 'MAXRULELOOP'
<mr_branche_cond> ::= 'IFMATCH_GOTORULE'
<mr_corpus_premier> ::= 'CORPUSFIRST'
<mr_corpus_suivant> ::= 'CORPUSNEXT'
<mr_corpus_précédent> ::= 'CORPUSPREVIOUS'
<mr_corpus_dernier> ::= 'CORPUSLAST'
<mr_corpus_aller> ::= 'CORPUSGOTO'
<mr_bascule_corpus> ::= 'SWITHCCORPUS'
<mr_corpus_entrée> ::= <sr_mrg> 'INPUT_CORPUS' <sr_mrd>
<mr_corpus_librairie> ::= <sr_mrg> 'LIBRARY_CORPUS' <sr_mrd>
<mr_corpus_tampon> ::= <sr_mrg> 'BUFFER_CORPUS' <sr_mrd>
<mr_corpus_prendre_ligne> ::= 'GETCORPUSLINE'
<mr_aller_à_règle> ::= 'GOTORULE'

<mr_exécute_gram> ::= 'EXECUTEGRAM'
<mr_insérer_ligne> ::= 'APPENDBUFFERLINE'
<mr_stop> ::= 'STOP'
<mr_concat> ::= 'CONCAT'
<mr_zone_résultat> ::= <sr_mrg> 'RM_RESULT_ZONE' <sr_mrd>
<mr_zone_temporaire> ::= <sr_mrg> 'TEMPORARY_ZONE' <sr_mrd>
<mr_dans> ::= 'IN'
<mr_hors> ::= 'OUTOF'
<mr_pas_vide> ::= 'NOTEMPTY'
<mr_être_type_entier> ::='INTEGER'
<mr_avoir_longueur> ::= 'FORCELENGTH'
<mr_exploser> ::= 'EXPLODE'
<mr_compter> ::= 'COUNT'
<mr_trier> ::= 'TORT'
<mr_numéroter_délim> ::= 'DELIMNUM'
<mr_plus> ::= 'PLUS'
<mr_moins> ::= 'MINUS'
<mr_mult> ::= 'MULT'
<mr_div> ::= 'DIV'
<mr_puissance> ::= 'POWER'
<mr_table_vérité> ::= 'TRUTHTABLE'
<mr_calcul_vérité> ::= 'TRUTHCALC'

5.5 Logiciel prototype RE du langage LRE

5.5.1 Pourquoi un logiciel spécifique

Il existe un certain nombre de systèmes de réécriture. On parle aussi de « réécriture de termes ». Tous sont basés sur un processus conduit par une ou un ensemble de règles qui à partir d'une *expression* réécrivent une nouvelle *expression*. D'un point de vue théorique et mathématique, ce sont les travaux d'Alonzo Church qui, en 1932, ont introduit le lambda calculus[90], qui nous paraissent une fondation à toutes les approches de systèmes de réécriture. Du côté informatique, nous avons recensé les systèmes de réécriture ou ayant des fonctions similaires :

- le logiciel SNOBOL, un langage évolué de traitement des *chaînes* de caractères (GRISWOLD, 2011) ;
- le SYSTEM Q de Colmerauer (COLMERAUER, 1970), un logiciel de réécriture de *chaînes* de caractères, qui a servi de système de base à la traduction automatique dans les années soixante-dix ;
- le système ELAN, un langage et un système basé sur des processus de réécriture (LORIA-CNRS, 2011) ;
- le programme CALM, langage de programmation orienté objets traitant des données algébriques (INRIA, 2011) ;
- le système MAUDE, basé sur des processus de réécriture qui supporte le traitement de métalangages de programmation (Members of Maude Team, 2011) ;
- le logiciel ASF+SDF est un système de réécriture, de *transformation*, de traitement d'arborescences, d'analyse de langages de programmation, de génération de langages de programmation, de description syntaxique et

[90] On compte une bibliographie très importante concernant le lambda calculus. Les ouvrages du mathématicien hollandais Henk Barendregt (BARENDREGT, 1984) et (BARENDREGT, 1991) sont une bonne introduction à cette discipline.

sémantique de langages de programmation (ASF+SDF Meta-Environment contributors, 2006) ;

- le logiciel CAFEOBJ, dédié au traitement de langages algébriques et de réécriture d'expression algébriques (CafeObj Project Members, 2010).

Tous ces systèmes sont, sous certains aspects, plus sophistiqués que notre *solution* informatique, *RE*, mais, ils ne répondent pas aux deux raisons qui nous ont conduits à développer un logiciel spécifique : la gestion des caractères *unicodes* et le traitement du *filtrage de motif* des *variables isoformes*.

Nous avons aussi pris cette direction pour les raisons suivantes :

- pratiques liées au traitement des *inscriptions* leśniewskiennes. Il a été plus facile d'intégrer le calcul des tables de vérités que l'on ne trouve pas dans d'autres systèmes de réécriture ;
- de flexibilité. Nous avons pu intégrer, étape par étape, des nouvelles *fonctions* dans le logiciel comme la gestion d'une *librairie*, l'apport d'une structure de mémorisation tampon, la réinjection de résultats comme *bibliothèque*, etc.

Notre *solution* informatique, le logiciel prototype : *RE* version 4.4 a nécessité environ dix mille lignes de programme en Pascal (Delphi).

5.5.2 Dessin du logiciel prototype RE

Dans ce qui précède, nous avons vu en détail de quoi est constitué et comment est défini notre langage, *LRE*. Dans cette section, nous présentons comment nous l'avons implanté sur ordinateur. Mettre en œuvre un langage sur ordinateur, implique le développement d'un logiciel qui a pour capacité, dans un premier temps, de l'interpréter ou de le compiler et, dans un deuxième temps d'exécuter les instructions interprétées ou compilées. Notre objectif étant avant tout de vérifier la faisabilité de notre concept avant d'apporter la robustesse et la convivialité nécessaires pour une distribution sur un marché de production[91]. Voyons quels sont les grands modules et flux principaux de *RE* à travers le diagramme de blocs qui suit :

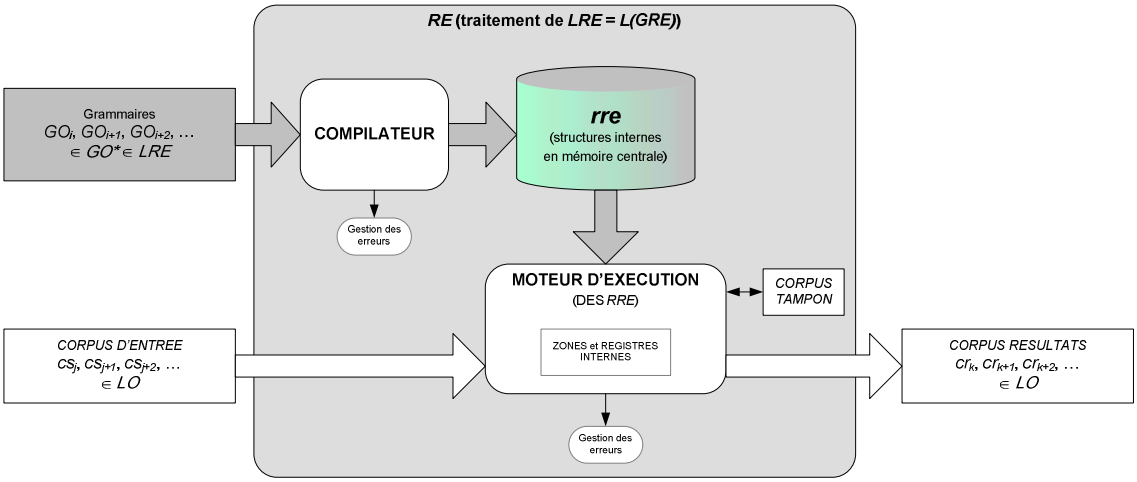


Figure 47. Diagramme de blocs du logiciel *RE*

Dans la figure ci-dessus, le bloc *RE*, est constitué de deux sous-blocs le COMPILATEUR et le MOTEUR D'EXECUTION.

[91] Il n'en reste pas moins que nous donnons dans l'annexe les éléments nécessaires pour se servir de ce prototype.

5.5.2.1 Le compilateur

Il interprète et compile tout ce que nous savons du *LRE* et en suivant les *productions* de la *GRE*. Il est très défensif et n'accepte que ce qui est défini dans la section *5.4.9 Les productions de la grammaire GRE, p. 120*. Toutes les malformations sont signalées et stoppe le compilateur avec un message d'erreur. Il est alimenté par un fichier d'une grammaire GO_i en entrée[92]. Ce fichier peut être édité, modifié, sauvegardé par le logiciel *RE* (voir l'annexe : *Manuel utilisateur du logiciel prototype RE* V4.4*, p. 283*). Seul l'ensemble des caractères *unicodes* de la police « *Lesniewski* » qui est livrée avec le logiciel constitue des caractères acceptables[93] pour le compilateur.

Tant que la grammaire GO_i soumise au compilateur est syntaxiquement correcte, le *RE* génère une structure interne et dynamique[94] dans la mémoire de l'ordinateur qui contient toutes les informations nécessaires à exécuter la GO_i compilée.

Le *RE* a été développé en langage informatique Pascal (objet). Il suit la pratique de la descente récursive de compilation présentée, entre autres, dans l'ouvrage de Niklaus Wirth (WIRTH, 1976).

5.5.2.2 Le moteur d'exécution des rre

Le moteur d'exécution est le cœur du logiciel. La structure des *rre* compilées va piloter son exécution. A partir de la première ligne d'un *corpus d'entrée*, auquel il va appliquer la première *rre* de la grammaire GO_i, il va exécuter chaque *rre* l'une après l'autre et cela tant qu'il reste une seule *rre* qui a son *mg* qui satisfait au *filtrage de motif*.

C'est, sans discussion, le traitement du *filtrage de motif*, la partie la plus complexe du moteur. L'algorithmique est simple en ce qui concerne le *filtrage de motif* d'alternances simples de schémas **VC**. Il faut par contre être attentif à l'implantation des *fonctions* conditionnelles qui s'applique aux *V* des *mg*. Quant à la gestion des *variables isoformes*, elle nécessite un niveau important d'exhaustivité et un peu de voltige dans la programmation pour garder des temps de réponse acceptables pour ce type de logiciel.

5.5.3 Présentation des instructions conditionnelles du LRE

Voyons les analogies entre nos structures déclaratives et les instructions conditionnelles des langages informatiques traditionnels de programmation :

L'expression des langages informatiques de test, si… alors… autrement, est similaire à un *filtrage de motif* réussi ou non d'un *mg* de *rre* :

---

[92] D'un point de vue implantation, une grammaire GO_i est un fichier plat de type texte.

[93] … ou sont des caractères qui sont interprétés correctement.

[94] Par « dynamique », nous entendons que théoriquement (à part les limites de la taille de la mémoire virtuelle de l'ordinateur), il n'y pas de limitation pour la taille d'une grammaire GO_i, ni de limite à la complexité des *rre*.

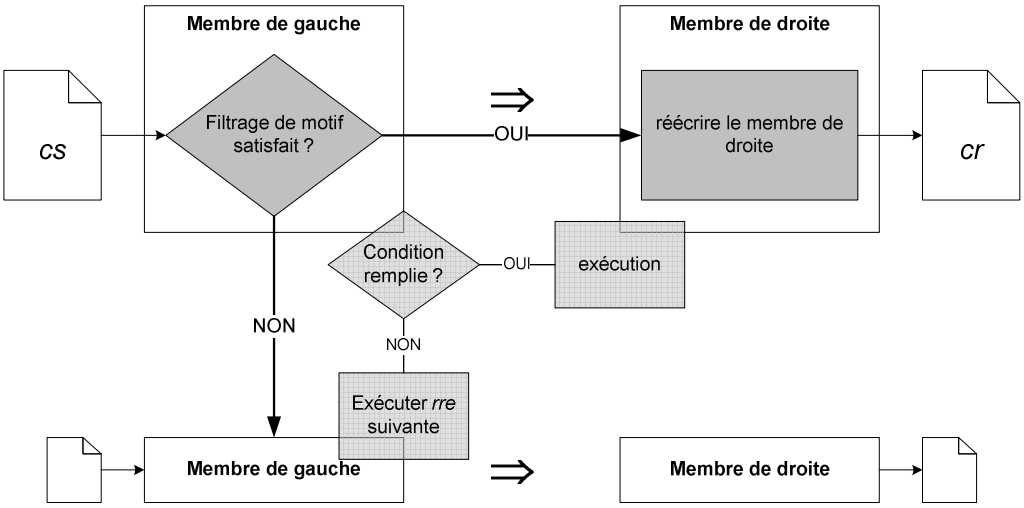


Figure 48. Analogie entre instruction conditionnelle et un *mg* de *rre*

On trouve deux analogies entre l'instruction des langages informatiques : tant…que faire et la programmation de *RE*. La première est l'itération d'une même *rre* :

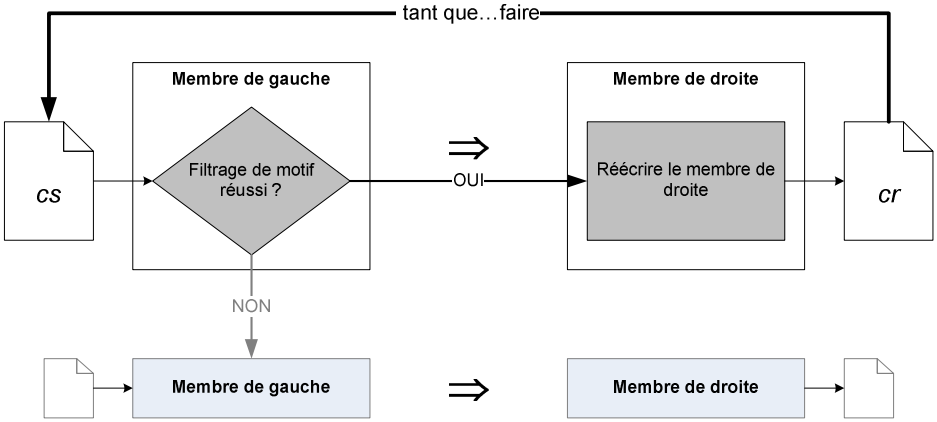


Figure 49. Analogie entre un *mg* de *rre* et l'instruction tant que… faire…

On trouve la deuxième analogie dans l'exécution de toute une grammaire GO_i :

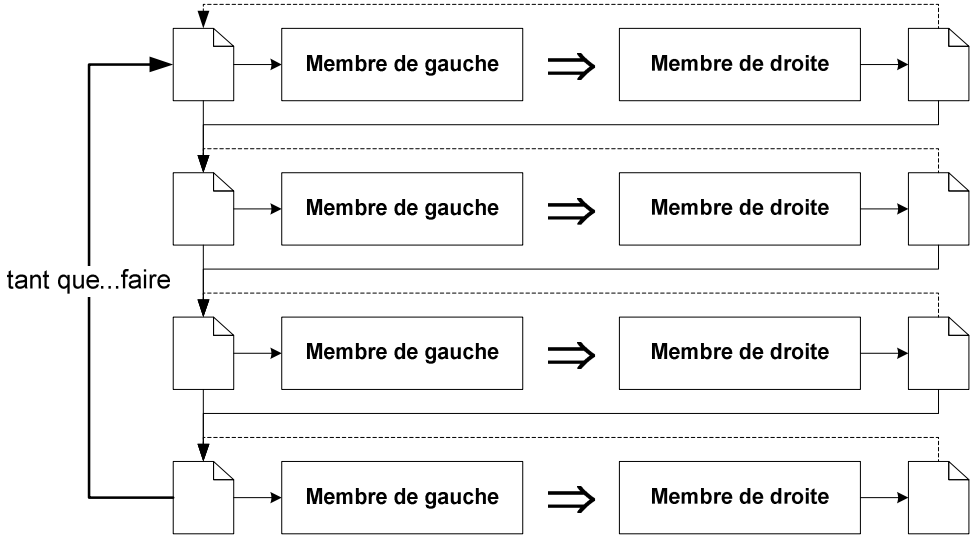


Figure 50. Analogie entre une instruction, tant que… faire… et une grammaire GO_i

5.6 Conseils et méthode de programmation des grammaires GO_i

5.6.1 Modularité et appels de sous-grammaires

Pour des traitements répétitifs de portion de GO_i, le *LRE* offre la possibilité de faire appel à des sous-grammaires GO_j (*5.4.7.3 La directive EXECUTEGRAM d'appel d'une grammaire GOi, p. 117*). Il reste à prendre garde au passage d'informations entre une GO_i principale (appelante) et une sous-grammaire GO_j (appelée) qui se fait par la dernière *chaîne, cr*, générée par la grammaire principale et qui sert de *chaîne, cs*, à la grammaire appelée. Inversement, quand une grammaire invoquée se termine, elle retourne la dernière *chaîne, cr*, comme nouvelle *chaîne, cs*, à la grammaire qui l'a appelée. Par conséquent, il faudra se donner une structure de *chaîne* qui sera respectée par les grammaires appelantes et appelées, afin de ne pas écraser une partie de l'information que l'on désire conserver. Nous utilisons, par exemple, des structures comme :

F 5.71. ☞➢‹avant_à_conserver›►‹zone_de_travail›◄‹apres_zone_résultat›≺ ⇒
➢‹avant_à_conserver›►‹zone_de_travail›◄‹apres_zone_résultat›≺☜

5.6.2 Itérations infinies

Il est assez facile de programmer une GO_i qui itère indéfiniment. Nous avons apporté au *RE* une contrainte qui permet de stopper une exécution après un certain nombre d'itérations (*5.4.8.3 La section obligatoire de paramètres, p. 118*). Le *RE* compte le nombre d'itérations y compris celles qui sont nécessaires pour le traitement. Il s'agit donc d'opérer un réglage adéquat.

Plusieurs cas peuvent entraîner des itérations infinies.

Le premier cas, le plus évident est l'itération sans fin d'une *rre*[95]. Exemple qui s'itérera indéfiniment quelle que soit la *cs :*

F 5.72. *rre :* ‹toute_la_chaine›⇒n'importe quoi

Voici un deuxième exemple qui fait s'itérer indéfiniment une même *rre :*

F 5.73. *rre :* ‹a› suivi par ‹b›⇒‹a› est suivi par ‹b›
cs : A suivi par B
cr_1 : A est suivi par B
cr_2 : AA est est suivi par BB
cr_3 : AAA est est est suivi par BBB
etc.

Pour prendre un troisième cas d'une grammaire GO_i qui ne s'arrête pas. Donnons-nous une petite grammaire GO_i :

F 5.74.
```
↳GRAM↲
☞GRAMNAME⇒BOUCLE☜

↳PARAMS↲
☞MAXGRAMBOUCLE⇒10☜
☞MAXREGLEBOUCLE⇒10☜
↳FINPARAMS↲

↳REGLES↲
☞il_‹etre›_‹qualificatif›⇒on_dit_qu'il_est_beau☜ ✎[1]
☞‹debut›_est_‹qualificatif›⇒‹debut›_était_encore_plus_beau☜ ✎[2]
☞on_dit_qu'‹pronom›_‹fin›⇒il_va_bien☜ ✎[3]
↳FINREGLES↲
↳FINGRAM↲
```

[95] Deux exceptions sont à considérer : l'utilisation des deux *fonctions* du *md* : EXPLOSER et NUMEROTERDELIM qui inhibent la *rre* après une première exécution et pour le reste de la grammaire GO_i.

En se donnant la *cs :* il_est_très_vieux nous obtenons les résultats suivants[96] :

F 5.75.
```
T: F, -34: GRAMMAR IS LOOPING - STOP !, Grammar: BOUCLE is looping, with: 11 iterations
T:   In modules: Bottom.Execute(Matching).ExecuteGrammar.GetNextRule.

RESULTS LOG AFTER EXECUTION:

BOUCLE, 1, 1, 1: on_dit_qu'il_est_beau
BOUCLE, 1, 2, 1: on_dit_qu'il_était_encore_plus_beau
BOUCLE, 1, 3, 1: il_va_bien
BOUCLE, 2, 1, 2: on_dit_qu'il_est_beau
BOUCLE, 2, 2, 2: on_dit_qu'il_était_encore_plus_beau
BOUCLE, 2, 3, 2: il_va_bien
BOUCLE, 3, 1, 3: on_dit_qu'il_est_beau
BOUCLE, 3, 2, 3: on_dit_qu'il_était_encore_plus_beau
BOUCLE, 3, 3, 3: il_va_bien
…
BOUCLE, 9, 1, 9: on_dit_qu'il_est_beau
BOUCLE, 9, 2, 9: on_dit_qu'il_était_encore_plus_beau
BOUCLE, 9, 3, 9: il_va_bien
BOUCLE, 10, 1, 10: on_dit_qu'il_est_beau
BOUCLE, 10, 2, 10: on_dit_qu'il_était_encore_plus_beau
BOUCLE, 10, 3, 10: il_va_bien
T: Execution Thread Termination with error
```

L'instruction : ALLER_A_REGLE utilisée comme *rre* et se branchant à une *rre* qui précède fait évidemment s'itérer indéfiniment la grammaire GO_i comme dans notre troisième cas.

Abordons comme quatrième cas, l'utilisation de l'*instruction* de branchement conditionnelle SI_FILTRE_ALLER_REGLE (). Selon la complexité de la grammaire GO_i, des itérations infinies sont difficiles à dépister. Pour que le branchement ait lieu, il faut qu'au moins une *rre* qui précède ait satisfait au *filtrage de motif*. Voici un exemple très simple qui a été créé pour illustrer ce cas. Il se termine avec une *expression cs* donnée et s'itère indéfiniment avec une autre pratiquement identique !

F 5.76.
```
↳GRAM↲
☞GRAMNAME⇒BOUCLECOND☜

↳PARAMS↲
☞MAXGRAMBOUCLE⇒10☜
☞MAXRULEBOUCLE⇒10☜
↳FINPARAMS↲

↳ENS↲
☞vp⇒"p","q","r","s"☜ ✎ variables propositionnelles
↳FINENS↲

↳REGLES↲
✎ envelopper les termes
☞▶‹a›[‹p↳IN(vp)↲›]‹b›◀⇒▶‹a›aT[●‹p›]‹b›◀☜ ✎[1]

✎ envelopper les expressions constituées de deux termes identiques
☞▶‹a›*(aT[●‹p›]aT[●‹p›])‹b›◀⇒▶‹a›xE[≡(aT[●‹p›]aT[●‹p›])]‹b›◀☜ ✎[2]

✎ envelopper les expressions constituées de deux termes différents
☞▶‹a›*(aT[●‹p›]aT[●‹q›])‹b›◀⇒▶‹a›xE[≡(aT[●‹p›]aT[●‹q›])]‹b›◀☜ ✎[3]

✎ suppression des marqueurs : ●
☞▶‹a›aT[●‹p›]‹b›◀⇒▶‹a›xT[‹p›]‹b›◀☜ ✎[4]

✎ encapsuler les expressions constituées d'expressions
☞▶‹a›*(xE[‹t1›]xE[‹t2›])‹b›◀⇒▶‹a›xE[≡(xE[‹t1›]xE[‹t2›])]‹b›◀☜ ✎[5]
```

[96] L'échantillon d'itérations donné ici est le résultat issu de *RE* pour la GO_i donnée. De gauche à droite, BOUCLE est le nom de la GO_i ; le premier chiffre est le nombre d'itération de la GO_i ; le deuxième est le numéro de la *rre* ; le troisième donne le nombre d'itérations de la *rre*, suivi de la *cr*.

✎ remplacement de termes identiques par q et q… (provocant la boucle) !
☞►‹a›xE⟦≡(xT⟦‹p›⟧xT⟦‹p›⟧)⟧‹b›◄⇒►‹a›xE⟦≡(xT⟦q⟧xT⟦q⟧)⟧‹b›◄☜ ✎[6]

☞↳SI_FILTRE_ALLER_REGLE(4)↲☜ ✎[7]

✎ on a mis ici une instruction STOP pour s'assurer que ce n'est pas la mécanique de
✎ la grammaire qui la fait boucler !
☞↳STOP↲☜

↳FINREGLES↲
↳FINGRAM↲

Soumettons une première *cs :* ►*(*(⟦p⟧⟦q⟧))*(*(⟦q⟧⟦r⟧)*(⟦p⟧⟦r⟧))◄, au *RE* et voyons le résultat :

F 5.77.
```
BOUCLECOND, 1, 1, 1: ►*(*(aT⟦•p⟧⟦q⟧))*(*(⟦q⟧⟦r⟧)*(⟦p⟧⟦r⟧))◄
BOUCLECOND, 1, 1, 2: ►*(*(aT⟦•p⟧aT⟦•q⟧))*(*(⟦q⟧⟦r⟧)*(⟦p⟧⟦r⟧))◄
BOUCLECOND, 1, 1, 3: ►*(*(aT⟦•p⟧aT⟦•q⟧))*(*(aT⟦•q⟧⟦r⟧)*(⟦p⟧⟦r⟧))◄
BOUCLECOND, 1, 1, 4: ►*(*(aT⟦•p⟧aT⟦•q⟧))*(*(aT⟦•q⟧aT⟦•r⟧)*(⟦p⟧⟦r⟧))◄
BOUCLECOND, 1, 1, 5: ►*(*(aT⟦•p⟧aT⟦•q⟧))*(*(aT⟦•q⟧aT⟦•r⟧)*(aT⟦•p⟧⟦r⟧))◄
BOUCLECOND, 1, 1, 6: ►*(*(aT⟦•p⟧aT⟦•q⟧))*(*(aT⟦•q⟧aT⟦•r⟧)*(aT⟦•p⟧aT⟦•r⟧))◄
BOUCLECOND, 1, 3, 1: ►*(xE⟦≡(aT⟦•p⟧aT⟦•q⟧)⟧)*(*(aT⟦•q⟧aT⟦•r⟧)*(aT⟦•p⟧aT⟦•r⟧))◄
BOUCLECOND, 1, 3, 2: ►*(xE⟦≡(aT⟦•p⟧aT⟦•q⟧)⟧)*(xE⟦≡(aT⟦•q⟧aT⟦•r⟧)⟧*(aT⟦•p⟧aT⟦•r⟧))◄
BOUCLECOND, 1, 3, 3: ►*(xE⟦≡(aT⟦•p⟧aT⟦•q⟧)⟧)*(xE⟦≡(aT⟦•q⟧aT⟦•r⟧)⟧xE⟦≡(aT⟦•p⟧aT⟦•r⟧)⟧)◄
BOUCLECOND, 1, 4, 1: ►*(xE⟦≡(xT⟦p⟧aT⟦•q⟧)⟧)*(xE⟦≡(aT⟦•q⟧aT⟦•r⟧)⟧xE⟦≡(aT⟦•p⟧aT⟦•r⟧)⟧)◄
BOUCLECOND, 1, 4, 2: ►*(xE⟦≡(xT⟦p⟧xT⟦q⟧)⟧)*(xE⟦≡(aT⟦•q⟧aT⟦•r⟧)⟧xE⟦≡(aT⟦•p⟧aT⟦•r⟧)⟧)◄
BOUCLECOND, 1, 4, 3: ►*(xE⟦≡(xT⟦p⟧xT⟦q⟧)⟧)*(xE⟦≡(xT⟦q⟧aT⟦•r⟧)⟧xE⟦≡(aT⟦•p⟧aT⟦•r⟧)⟧)◄
BOUCLECOND, 1, 4, 4: ►*(xE⟦≡(xT⟦p⟧xT⟦q⟧)⟧)*(xE⟦≡(xT⟦q⟧xT⟦r⟧)⟧xE⟦≡(aT⟦•p⟧aT⟦•r⟧)⟧)◄
BOUCLECOND, 1, 4, 5: ►*(xE⟦≡(xT⟦p⟧xT⟦q⟧)⟧)*(xE⟦≡(xT⟦q⟧xT⟦r⟧)⟧xE⟦≡(xT⟦p⟧aT⟦•r⟧)⟧)◄
BOUCLECOND, 1, 4, 6: ►*(xE⟦≡(xT⟦p⟧xT⟦q⟧)⟧)*(xE⟦≡(xT⟦q⟧xT⟦r⟧)⟧xE⟦≡(xT⟦p⟧xT⟦r⟧)⟧)◄
BOUCLECOND, 1, 5, 1: ►xE⟦≡(xE⟦≡(xT⟦p⟧xT⟦q⟧)⟧)*(xE⟦≡(xT⟦q⟧xT⟦r⟧)⟧xE⟦≡(xT⟦p⟧xT⟦r⟧)⟧)⟧◄
BOUCLECOND, 1, 5, 2: ►xE⟦≡(xE⟦≡(xT⟦p⟧xT⟦q⟧)⟧)xE⟦≡(xE⟦≡(xT⟦q⟧xT⟦r⟧)⟧xE⟦≡(xT⟦p⟧xT⟦r⟧)⟧)⟧⟧◄
```

Il n'y a rien à signaler de particulier ! La grammaire GO_i produit ce que nous attendions. Soumettons, maintenant, une deuxième *cs :* ►*(*(⟦p⟧⟦p⟧))*(*(⟦q⟧⟦r⟧)*(⟦p⟧⟦r⟧))◄ pour obtenir les résultats qui s'itèrent indéfiniment :

F 5.78.
```
T: F, -33: RULE IS LOOPING - STOP !,  Rule N° : 6 is looping in grammar: BOUCLECOND, with: 11
iterations
T:    In modules: Bottom.Execute(Matching).ExecuteGrammar.ExecuteRule.
T:    In Rule: 6, Pass: 11

RESULTS LOG AFTER EXECUTION:
BOUCLECOND, 1, 1, 1: ►*(*(aT⟦•p⟧⟦p⟧))*(*(⟦q⟧⟦r⟧)*(⟦p⟧⟦r⟧))◄
BOUCLECOND, 1, 1, 2: ►*(*(aT⟦•p⟧aT⟦•p⟧))*(*(⟦q⟧⟦r⟧)*(⟦p⟧⟦r⟧))◄
BOUCLECOND, 1, 1, 3: ►*(*(aT⟦•p⟧aT⟦•p⟧))*(*(aT⟦•q⟧⟦r⟧)*(⟦p⟧⟦r⟧))◄
BOUCLECOND, 1, 1, 4: ►*(*(aT⟦•p⟧aT⟦•p⟧))*(*(aT⟦•q⟧aT⟦•r⟧)*(⟦p⟧⟦r⟧))◄
BOUCLECOND, 1, 1, 5: ►*(*(aT⟦•p⟧aT⟦•p⟧))*(*(aT⟦•q⟧aT⟦•r⟧)*(aT⟦•p⟧⟦r⟧))◄
BOUCLECOND, 1, 1, 6: ►*(*(aT⟦•p⟧aT⟦•p⟧))*(*(aT⟦•q⟧aT⟦•r⟧)*(aT⟦•p⟧aT⟦•r⟧))◄
BOUCLECOND, 1, 2, 1: ►*(xE⟦≡(aT⟦•p⟧aT⟦•p⟧)⟧)*(*(aT⟦•q⟧aT⟦•r⟧)*(aT⟦•p⟧aT⟦•r⟧))◄
BOUCLECOND, 1, 3, 1: ►*(xE⟦≡(aT⟦•p⟧aT⟦•p⟧)⟧)*(xE⟦≡(aT⟦•q⟧aT⟦•r⟧)⟧*(aT⟦•p⟧aT⟦•r⟧))◄
BOUCLECOND, 1, 3, 2: ►*(xE⟦≡(aT⟦•p⟧aT⟦•p⟧)⟧)*(xE⟦≡(aT⟦•q⟧aT⟦•r⟧)⟧xE⟦≡(aT⟦•p⟧aT⟦•r⟧)⟧)◄
BOUCLECOND, 1, 4, 1: ►*(xE⟦≡(xT⟦p⟧aT⟦•p⟧)⟧)*(xE⟦≡(aT⟦•q⟧aT⟦•r⟧)⟧xE⟦≡(aT⟦•p⟧aT⟦•r⟧)⟧)◄
BOUCLECOND, 1, 4, 2: ►*(xE⟦≡(xT⟦p⟧xT⟦p⟧)⟧)*(xE⟦≡(aT⟦•q⟧aT⟦•r⟧)⟧xE⟦≡(aT⟦•p⟧aT⟦•r⟧)⟧)◄
BOUCLECOND, 1, 4, 3: ►*(xE⟦≡(xT⟦p⟧xT⟦p⟧)⟧)*(xE⟦≡(xT⟦q⟧aT⟦•r⟧)⟧xE⟦≡(aT⟦•p⟧aT⟦•r⟧)⟧)◄
BOUCLECOND, 1, 4, 4: ►*(xE⟦≡(xT⟦p⟧xT⟦p⟧)⟧)*(xE⟦≡(xT⟦q⟧xT⟦r⟧)⟧xE⟦≡(aT⟦•p⟧aT⟦•r⟧)⟧)◄
BOUCLECOND, 1, 4, 5: ►*(xE⟦≡(xT⟦p⟧xT⟦p⟧)⟧)*(xE⟦≡(xT⟦q⟧xT⟦r⟧)⟧xE⟦≡(xT⟦p⟧aT⟦•r⟧)⟧)◄
BOUCLECOND, 1, 4, 6: ►*(xE⟦≡(xT⟦p⟧xT⟦p⟧)⟧)*(xE⟦≡(xT⟦q⟧xT⟦r⟧)⟧xE⟦≡(xT⟦p⟧xT⟦r⟧)⟧)◄
BOUCLECOND, 1, 5, 1: ►xE⟦≡(xE⟦≡(xT⟦p⟧xT⟦p⟧)⟧)*(xE⟦≡(xT⟦q⟧xT⟦r⟧)⟧xE⟦≡(xT⟦p⟧xT⟦r⟧)⟧)⟧◄
BOUCLECOND, 1, 5, 2: ►xE⟦≡(xE⟦≡(xT⟦p⟧xT⟦p⟧)⟧)xE⟦≡(xE⟦≡(xT⟦q⟧xT⟦r⟧)⟧xE⟦≡(xT⟦p⟧xT⟦r⟧)⟧)⟧⟧◄
BOUCLECOND, 1, 6, 1: ►xE⟦≡(xE⟦≡(xT⟦q⟧xT⟦q⟧)⟧)xE⟦≡(xE⟦≡(xT⟦q⟧xT⟦r⟧)⟧xE⟦≡(xT⟦p⟧xT⟦r⟧)⟧)⟧⟧◄
BOUCLECOND, 1, 6, 2: ►xE⟦≡(xE⟦≡(xT⟦q⟧xT⟦q⟧)⟧)xE⟦≡(xE⟦≡(xT⟦q⟧xT⟦r⟧)⟧xE⟦≡(xT⟦p⟧xT⟦r⟧)⟧)⟧⟧◄
…
BOUCLECOND, 1, 6, 9: ►xE⟦≡(xE⟦≡(xT⟦q⟧xT⟦q⟧)⟧)xE⟦≡(xE⟦≡(xT⟦q⟧xT⟦r⟧)⟧xE⟦≡(xT⟦p⟧xT⟦r⟧)⟧)⟧⟧◄
BOUCLECOND, 1, 6, 10: ►xE⟦≡(xE⟦≡(xT⟦q⟧xT⟦q⟧)⟧)xE⟦≡(xE⟦≡(xT⟦q⟧xT⟦r⟧)⟧xE⟦≡(xT⟦p⟧xT⟦r⟧)⟧)⟧⟧◄
T: Execution Thread Termination with error
```

Evidemment, c'est la *rre*[97] 6 qui fait s'itérer indéfiniment l'*instruction* conditionnelle de branchement (*rre* 7). Parce que cette *rre* satisfait et satisfera toujours au *filtrage de motif* à la

[97] On signale dans les résultats livrés par *RE* que c'est la *rre* n° 6 qui boucle et non pas la grammaire elle-même. C'est une décision arbitraire que nous avons prise dans la version 4.4 du prototype. L'exécution d'une grammaire

présentation d'une *cs* avec deux enveloppements xT identiques et donc au moins une fois dans ce qui précède l'*instruction* SI_FILTRE_ALLER_REGLE.

5.6.3 Dépannage à l'aide de l'instruction STOP

Comme nous l'avons vu, une *instruction* STOP, en tant que *rre* ou dans un *md*, arrête la grammaire GO_i. Elle peut servir de point d'arrêt n'importe où dans une grammaire GO_i. Il peut aussi être utile de connaître, à un point de la grammaire, le contenu complet de la dernière réécriture générée par la dernière *rre* qui a satisfait au *filtrage de motif*. Nous pouvons utiliser la *rre* de dépannage suivante qui affichera le dernier résultat et stoppera la grammaire :

F 5.79. ☞‹a›⇒contenu de la dernière réécriture : ‹a›↳STOP↲☜

5.6.4 Les marqueurs

Un marqueur peut se définir comme l'introduction d'un caractère ou d'une *chaîne* de caractères insérée dans la *chaîne* réécrite à des fins de blocage, de reconnaissance ou de transport d'informations. Si l'on ne veut pas qu'une *rre* s'itère plus d'un certain nombre de fois, nous pouvons marquer une partie de la *chaîne*. Dans l'exemple qui suit et pour éviter que la *rre* numéro 1 ne s'itère indéfiniment :

F 5.80.
```
↳GRAM↲
☞GRAMNAME⇒MARK1☜

↳PARAMS↲
☞MAXGRAMBOUCLE⇒10☜
☞MAXRULEBOUCLE⇒10☜
↳FINPARAMS↲

↳ENS↲
☞vp⇒"p","q","r","s"☜ ✎ variables propositionnelles
↳FINENS↲

↳REGLES↲
✎ envelopper les termes et introduire un marqueur : ●
☞►‹a›[‹p↳DANS (vp)↲›]‹b›◄⇒►‹a›aT[●‹p›]‹b›◄☜ ✎[1]

✎ …

↳FINREGLES↲
↳FINGRAM↲
```

En soumettant la *cs* : ►*(*([p][q]))*(*([q][r])*([p][r]))◄, nous obtenons :

F 5.81.
```
BOUCLECOND, 1, 1, 1: ►*(*(aT[●p][q]))*(*([q][r])*([p][r]))◄
BOUCLECOND, 1, 1, 2: ►*(*(aT[●p]aT[●q]))*(*([q][r])*([p][r]))◄
BOUCLECOND, 1, 1, 3: ►*(*(aT[●p]aT[●q]))*(*(aT[●q][r])*([p][r]))◄
BOUCLECOND, 1, 1, 4: ►*(*(aT[●p]aT[●q]))*(*(aT[●q]aT[●r])*([p][r]))◄
BOUCLECOND, 1, 1, 5: ►*(*(aT[●p]aT[●q]))*(*(aT[●q]aT[●r])*(aT[●p][r]))◄
BOUCLECOND, 1, 1, 6: ►*(*(aT[●p]aT[●q]))*(*(aT[●q]aT[●r])*(aT[●p]aT[●r]))◄
```

Par l'introduction du marqueur : '●', on évite que la *rre* n'itère indéfiniment et que les enveloppes de xT soient traitées plus d'une fois. Remplaçons cette *rre* 1, par une nouvelle *rre* :

F 5.82.
```
✎ envelopper le premier terme et introduction d'un marqueur : '/une_fois'
☞►‹a›[‹p↳DANS(vp)↲›]‹b›◄⇒►‹a›aT[‹p›]‹b›◄/une_fois☜ ✎[1]
```

Soumettons une fois encore la *cs :* ►*(*([p][q]))*(*([q][r])*([p][r]))◄ :

F 5.83. BOUCLECOND, 1, 1, 1: ►*(*(**aT[p]**[q]))*(*([q][r])*([p][r]))◄**/une_fois**

L'introduction du marqueur, soit une nouvelle forme du dernier schéma de la *rre*, qui change la *C* de '◄' à '◄/une_fois' met en défaut un deuxième *filtrage de motif*.

---

n'incrémente le compteur d'exécution de la grammaire seulement quand cette dernière a atteint la dernière *rre*. Dans notre exemple, l'instruction de branchement conditionnel empêche la grammaire d'aboutir ou de passer par la dernière *rre* !

L'utilisation de marqueurs est aussi très pratique pour le traitement d'erreurs. Par exemple :

F 5.84. ✎ TESTER LA REPETITION DE TERMES DANS LE QUANTIFICATEUR
☞►‹a›xQ⌊‹b›xT[‹v›]‹c›xT[‹v›]‹d›⌋‹z›◄⇒
►‹a›xQ⌊‹b›_erreur_xT[‹v›]‹c›_erreur_xT[‹v›]‹d›⌋‹z›◄/répétition_trouvée☜

✎ DEPISTER LE DEFINIENDUM
✎ …

✎ TRAITER LES ERREURS
☞►‹a›◄/répétition_trouvée⇒Erreur dans : ►‹a›◄, trouvé une répétition de termes dans le quantificateur↳STOP↲☜

✎ …

Par l'utilisation de marqueurs, nous avons vu des fonctions de blocage et de reconnaissance. Approchons encore un dernier exemple dédié au transport d'information d'une *rre* à une autre. Ce principe est très utile pour conserver une information dont nous aurons besoin dans une étape ultérieure. Par exemple, un extrait de notre grammaire GO_i qui traite des tables de vérité dans la *protothétique* :

F 5.85. ✎ CALCULER LES VALEURS POUR LES TV DU SOUS-QUANTIFICATEUR DOMINANT
✎ …
✎ détecter les variables propositionnelles et les copier dans les paramètres temporaires
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›●→‹b›xTV_[‹lien↳ENTIER↲›]_‹i↳ENTIER↲›
[L/‹p›/xTV_‹i›]‹c›←●‹z›◁/‹te›/TMP⟩‹s›tvListe=⟨‹t›⟩/‹u›⟨⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›‹b›xTV_[‹lien›]_‹i›[L/‹p›/xTV_‹i›]
●→‹c›←●‹z›◁/‹te›/TMP⟩‹s›tvListe=⟨‹t›‹p›⟩/‹u›⟨☜ ✎[18]
✎ …

Dans cette *rre* nous avons réservé une zone (soulignée) de paramètres temporaires qui nous permet de mémoriser au fur à mesure de notre analyse, les *variables propositionnelles* rencontrées, dans le but de leur affecter une table de vérité.

5.6.5 L'utilisation de curseurs

Les curseurs, une sorte d'enveloppement temporaire qui permet de cerner et de délimiter une portion de *chaîne* de caractères simplifie le *filtrage de motif* dans des *rre* complexes. Par exemple, un schéma de type : ●→‹v›←● permet de ne traiter que la portion de *chaîne* : ‹v›.

5.6.6 L'encapsulation préfixée

Nous avons déjà abordé à quelques reprises le principe « d'enveloppement » d'une entité morphosyntaxique à l'aide de métasymboles ou de métadélimiteurs symétriques comme : [x]. Cela nous sert, entre autres, à faciliter le *filtrage de motif*, tenant compte de l'alternance de *C* et *V* dans les *mg* des *rre*. Nous allons préfixer ce type d'enveloppe par une information sur son contenu : est_un[x]. Nous appelons ce type de forme : *encapsulation préfixée*[98].

Donnons-nous une *expression biconditionnelle primitive* dans la notation en *version catégorielle* et dans la notation en *version contextuelle*. Prenons comme objectif l'analyse de conversion d'une notation vers l'autre et réciproquement :

$$\equiv(pq) \Rightarrow p\equiv q$$

Nous avons, ici, trois types d'objets : les *variables propositionnelles* 'p' et 'q', la *biconditionnelle* '≡' et les deux *délimiteurs* '(' et ')'. Nous isolons l'ensemble des *signes* à l'aide de la fonction EXPLOSER (*F 5.21, p. 95*) :

[98] A ne pas confondre avec la notion d'encapsulation informatique en programmation objet !

p≡q ⇒ [p][≡][q]

≡(pq) ⇒ [≡][([p][q])]

Par un ensemble de *rre* d'une GO_i, nous pouvons amener de l'information à chacune des *encapsulations* en les préfixant. Par exemple, en préfixant les *termes* par T :

[p][≡][q] ⇒ T[p][≡]T[q]

[≡][([p][q])] ⇒ [≡][(T[p]T[q])]

En préfixant les *délimiteurs symétriques* '(' et ')' par, respectivement PG et PD :

[≡][(T[p]T[q])] ⇒ [≡]PG[(T[p]T[q]PD)]

En préfixant la *biconditionnelle* '≡' par BIC :

T[p][≡]T[q] ⇒ T[p]BIC[≡]T[q]

[≡][(T[p]T[q])] ⇒ BIC[≡]PG[(T[p]T[q]PD)]

Pour amener encore plus d'information, celle d'*expression* : 'cE' et 'xE', nous pouvons procéder par *encapsulation préfixée* d'*encapsulations préfixées :*

T[p]BIC[≡]T[q] ⇒ cE[T[p]BIC[≡]T[q]]

[≡][(T[p]T[q])] ⇒ xE[BIC[≡]PG[(T[p]T[q]PD)]]

Ce nouvel apport, nous permet d'envisager une conversion d'une notation à l'autre :

cE[T[p]BIC[≡]T[q]] ⇒ xE[BIC[≡]PG[(T[p]T[q]PD)]]

xE[BIC[≡]PG[(T[p]T[q]PD)]] ⇒ cE[T[p]BIC[≡]T[q]]

Nous voilà dotés d'informations supplémentaires sur les *signes*. Nous avons donc apporté, par des *rre* de l'information à α en apportant, par *encapsulation préfixée*, dans β de l'information jusque là inconnue de α. Nous avons :

$\alpha \rightarrow \beta$ où $|\alpha| < |\beta|$

Pour revenir à la classification et la définition des *LF* (*RD 5.3-4, p. 88*), *GRE* se présente comme une grammaire *dépendante de contexte*. Nous verrons plus loin l'impact que peut avoir cette observation.

La plupart des *inscriptions* leśniewskiennes sont construites avec des *expressions* contenant d'autres *expressions*. Dès lors, nos grammaires GO_i doivent tenir compte de la profondeur du parenthèsage de ces *expressions*. Nous utilisons pour ce traitement des *encapsulations préfixées* auxquelles nous assignons un même numéro de début et de fin à notre enveloppe[99]. Nous obtenons ainsi les niveaux de profondeur des structures de parenthèses. Nous pouvons reporter ce niveau aux préfixes d'*expression* et fermer cette dernière avec un suffixe indiquant le même niveau ; par exemple :

F 5.86. **xE_1**[tc[≡]pg[(]xE_2[tc[≡]T[p]T[q]pd[)]/xE_2]
xE_3[tc[≡]T[p]T[q]pd[)]/xE_3[pd[)]/**xE_1**]

Ce type d'*encapsulations préfixées* permet à tout moment de cerner, sans ambiguïté, le contenu d'une *expression* ; par exemple, de cerner le contenu de l'*expression* de niveau, i :

F 5.87. ►‹a›xE_‹i›[‹contenu›/xE_‹i›]‹z›◄⇒faire quelque chose avec ‹contenu›

Ce principe est assez puissant pour traiter la *protothétique* et l'*ontologie*. On trouve un exemple complet de l'utilisation de ce principe dans la GO_i, PREANALYSE, dans l'annexe *Préanalyse, analyse et application de la procédure définitoire, p. 209*.

[99] Pour simplifier une programmation complexe à ce sujet, nous avons introduit dans notre *RE*, la *fonction* de *md* du *LRE* : NUMEROTERDELIM.

Plus loin, nous élaborerons une systématique de l'apport de cette information (apporté par *encapsulation*) à chaque entité linguistique pour en faire des « entités analysées » et qui sont des *phrases* de *LO* et que nous appelons *langage pivot*, $LO_p = L(GO_p)$. Dans ce *langage pivot*, une *encapsulation préfixée* qui est destinée à ne contenir qu'une entité morphosyntaxique ne sera que préfixée, dans le cas où l'enveloppe devrait contenir plusieurs entités, nous ajoutons un suffixe auquel on intègre son indice de niveau.

Le Langage *LRE = L(GRE)* et son utilisation étant défini, nous pouvons aborder un exemple d'une GO_i complète comme illustration de ce qui a été présenté jusqu'ici.

5.7 Programmation et analyse détaillée de la GO_c

Nous savons que le *LRE* est un langage grammatical, donc ses *phrases* décrivent ou sont véritablement des règles de grammaire. Il a été élaboré pour traiter des sous-ensembles de *phrases* appartenant à *LO*. Le *LO* a été défini pour décrire le corpus linguistique des *SL*. Nous allons présenter, dans les sections qui suivent, un nombre important d'exemples de GO_i dédiées à ces logiques. Pour le lecteur qui ne serait pas familier avec les *SL*, nous présentons, ici, un exemple détaillé de programmation d'une grammaire qui s'inspire d'un *domaine de connaissances* complémentaire à la logique, un corpus rudimentaire extrait de la langue naturelle, constitué de deux *phrases :* w_1 et w_2 appartenant au $LO_c \in LO$,*où* $LO_c = L(GO_c)$:

F 5.88. w_1 *:* 'le chat boit le lait dans la coupe'
et
w_2 : 'dans la coupe, le chat boit le lait'[100]

Dans les grammaires traditionnelles et normatives, on se donne comme modèle de structure de ces deux phrases :

F 5.89. m_1 : <sujet> <verbe> <complément d'objet> <complément de lieu>
et
m_2 : <complément de lieu>, <sujet> <verbe> <complément d'objet>

Le modèle m_1 est construit avec quatre *non-terminaux* et m_2, est constitué d'un *terminal* : ',' et de quatre *non-terminaux.* Notre but est de développer une grammaire GO_c telle que :

F 5.90. $$m_1 \Rightarrow m_2$$

Il nous faut envisager une analyse morphosyntaxique qui traite les unités syntaxiques jusqu'au niveau de la phrase. Elle passe indiscutablement par l'analyse des graphèmes, des mots, des syntagmes (nominal, verbal) et de la phrase.

Donnons-nous pour commencer les grands blocs de notre grammaire GO_c, à laquelle nous avons donné le nom de LECHAT, sachant, par expérience[101], qu'il n'y a aucune raison qu'une *rre* ou GO_c ne s'itèrent plus que dix fois :

[100] Sans chercher à savoir où se trouve le chat pendant l'action !

[101] A noter que certaines *rre* vont s'itérer plusieurs fois et que la grammaire ne passe que par une seule itération.

F 5.91. Squelette de GO_c (LECHAT)

```
↳GRAM↲
☞NOMGRAM⇒LECHAT☜
✎ But de GOc : analyser une phrase rudimentaire, construire sa structure
✎ avec vérification des genres,
✎  et lui faire subir une transformation en préfixant le complément circonstanciel de lieu, soit :
✎  cs : "le chat boit le lait dans la coupe"
✎  cr : "dans la coupe, le chat boit le lait"

↳PARAMS↲
☞MAXGRAMBOUCLE⇒10☜
☞MAXREGLEBOUCLE⇒10☜
↳ENDPARAMS↲

↳ENS↲
✎…
↳FINENS↲

↳REGLES↲
✎…
↳FINREGLES↲

↳FINGRAM↲
```

Pour l'analyse lexicale et le début de l'analyse morphosyntaxique, nous programmons une section de définitions d'ensembles. Sans rentrer dans une analyse complexe de catégories grammaticales, ni de structures de radicaux avec préfixes et suffixes ou autres accords, ni de formes verbales avec conjugaisons avancées, etc. aspects que le *LRE* serait apte à traiter avec une grammaire GO_i un peu plus complexe. Donnons-nous les *terminaux* les plus simples dont nous avons besoin :

F 5.92. Section de définitions d'ensembles[102] de GO_c (LECHAT)

```
↳ENS↲
☞verbetrans3persdupresent⇒"boit", "mange"☜
☞articlemasculin⇒"le"☜
☞articlefeminin⇒"la"☜
☞substantifmasculin⇒"chat","lait"☜ ✎ version 2
☞substantiffeminin⇒"coupe"☜ ✎ version 2
☞substantif⇒substantifmasculin,substantiffeminin☜
☞preplieu⇒"dans"☜
↳FINENS↲
```

Nous décidons que la *cs* : "le chat boit le lait dans la coupe", est enveloppée par des guillemets. Cela facilite l'analyse en début et fin de *chaîne*. Les mots séparés par des espaces typographiques complexifient un peu l'analyse. Dès lors, programmons les *rre* 1 à 3 qui séparent les mots par une *encapsulation :* '{' et '}'. Cette partie d'analyse ne fait que remplacer des graphèmes, comme l'espace typographique, par d'autres :

[102] Cette section de déclarations d'ensembles vient s'insérer dans le bloc qui lui correspond dans le squelette de la grammaire GO_c, LECHAT, qui précède.

F 5.93. Traitement des séparateurs de mots de *GO$_c$* (LECHAT)

```
↳REGLES↲

✎ REMPLACER LES ESPACES TYPOGRAPHIQUES PAR UNE ENCAPSULATION : {}

✎ Traiter le premier mot de la phrase
✎ Schéma du mg de la rre 1 : C1VaC2Vb C1, où C1 : " et C2 : espace typographique
✎ Attention : l'espace se voit mal dans une rre suivant la typographie

☞"‹a› ‹b›"⇒{‹a›}"‹b›"☜ ✎ rre 1

✎ Traiter tous les autres mots sauf le dernier
✎ Schéma du mg de la rre 2 : C1VaC2VbC3VcC4 où C1 : {, C2 : }", C3 : espace, C4 : "

☞{‹a›}"‹b› ‹c›"⇒{‹a›}{‹b›}"‹c›"☜ ✎ rre 2

✎ Traiter le dernier mot
✎ Schéma de α de la rre 1 : VaC2VbC2 où C2 : "

☞‹a›"‹b›"⇒‹a›{‹b›}☜ ✎ rre 3

✎…
↳FINREGLES↲
```

Ces trois *rre* constituent le début de la *GO$_c$*, si on l'exécute avec comme *cs :* "le chat boit le lait dans la coupe", elle suit les itérations et les *cr* suivantes[103] :

F 5.94. Résultat du traitement des séparateurs de mots de *GO$_c$* (LECHAT)

```
LECHAT, 1, 1,1: {le}"chat boit le lait dans la coupe"
LECHAT, 1, 2,1: {le}{chat}"boit le lait dans la coupe"
LECHAT, 1, 2,2: {le}{chat}{boit}"le lait dans la coupe"
LECHAT, 1, 2,3: {le}{chat}{boit}{le}"lait dans la coupe"
LECHAT, 1, 2,4: {le}{chat}{boit}{le}{lait}"dans la coupe"
LECHAT, 1, 2,5: {le}{chat}{boit}{le}{lait}{dans}"la coupe"
LECHAT, 1, 2,6: {le}{chat}{boit}{le}{lait}{dans}{la}"coupe"
LECHAT, 1, 3,1: {le}{chat}{boit}{le}{lait}{dans}{la}{coupe}
```

Ci-dessus, à la ligne 1,3,1 du *corpus de résultat*, tous les mots sont *encapsulés*. Grâce aux définitions d'ensembles qui ont précédé, nous pouvons apporter à chacune des *encapsulations* des informations par la méthode d'*encapsulation préfixée*. Nous utilisons la fonction, DANS, destinée aux *mg* qui nous permet d'appliquer une condition à une *V* en se référant à un ensemble :

F 5.95. Encapsulation préfixée des mots dans *GO$_c$* (LECHAT)

```
✎ RECHERCHER ET ENCAPSULER LES ARTICLES MASCULINS
✎ si l'on trouve un article masculin, on lui apporte 2 niveaux d'information :
✎ déterminant et genre (on ne traite pas le nombre).
✎ La première couche d'encapsulation préfixe le genre,
✎ la deuxième la catégorie grammaticale, soit : <catégorie>[<genre>[…]]

☞‹x›{‹a↳DANS(articlemasculin)↲›}‹r›⇒‹x›(art[m["‹a›"]])‹r›☜ ✎ rre 4

✎ RECHERCHER ET ENCAPSULER LES ARTICLES FEMININS
✎ on procède comme pour l'article masculin

☞‹x›{‹a↳DANS(articlefeminin)↲›}‹r›⇒‹x›(art[f["‹a›"]])‹r›☜ ✎ rre 5

✎ RECHERCHER ET ENCAPSULER LES NOMS MASCULINS
✎ avec leur genre et se donner un marqueur : ● qui va servir au traitement des erreurs.

☞‹x›{‹a↳DANS(substantifmasculin)↲›}‹r›⇒‹x›(●subs[m["‹a›"]])‹r›☜ ✎ rre 6

✎ RECHERCHER ET ENCAPSULER LES NOMS FEMININS
```

[103] Le *RE* identifie ou numérote les lignes résultantes, à savoir les *cr*, de la façon suivante : en début de lignes, on trouve, tout d'abord : l'identificateur de la grammaire qui est exécutée ; en deuxième position : le nombre d'itérations de la grammaire ; en troisième position : le numéro de la *rre* et avant le résultat lui-même : le nombre d'itérations de la *rre*. Ce premier échantillon de résultats est issu du logiciel prototype *RE*, version 4.4.

✎ avec leur genre et se donner un marqueur : ● qui va servir au traitement des erreurs.

☞‹x›{‹a↳DANS(substantiffeminin)↲›}‹r›⇒‹x›(●subs[f["‹a›"]])‹r›☜ ✎ *rre* 7

✎ TESTER SI POUR TOUTES LES FORMES : ARTICLE / SUBSTANTIF
✎ QUE LE GENRE EST IDENTIQUE
✎ Utilisation de la *variable isoforme* : ‹genre›.
✎ A chaque *filtrage de motif* réussi les marqueurs : ● sont éffacés.

☞‹a›(●art[‹genre›[‹art›]])(●subs[‹genre›[‹nom›]])‹z›⇒
‹a›(art[‹genre›[‹art›]])(subs[‹genre›[‹nom›]])‹z›☜ ✎ *rre* 8

✎ TESTER S'IL Y A UNE ERREUR SUR LES GENRES
✎ Une erreur se repère s'il reste un ou des marqueurs : ●.

☞‹a›(●art[‹genre1›[‹art›]])(●subs[‹genre2›[‹nom›]])‹z›⇒
l'article : ‹art› n'a pas le même genre que le nom : ‹nom› !↳STOP↲☜ ✎ *rre* 9

✎ RECHERCHER LES PREPOSITIONS DE LIEU

☞‹x›{‹l↳DANS(preplieu)↲›}‹r›⇒‹x›(plieu["‹l›"])‹r›☜ ✎ *rre* 10

On notera la « sur-encapsulation » de l'unité traitée par une paire de parenthèses '(' et ')'. Elle nous sert de marqueur qui nous indique que nous n'avons pas terminé l'analyse de l'unité en question. Ce procédé nous sera très utile pour vérifier en fin de traitement si toutes les unités ont été traitées. Voici ce que génère le *RE* à l'exécution des *rre* de 4 à 5 :

F 5.96. Résultats du traitement des *encapsulations préfixées* des articles dans GO_c (LECHAT)

LECHAT, 1, 4,1: **(art[m["le"]])**{chat}{boit}{le}{lait}{dans}{la}{coupe}
LECHAT, 1, 4,2: (art[m["le"]]){chat}{boit}**(art[m["le"]])**{lait}{dans}{la}{coupe}
LECHAT, 1, 5,1: (art[m["le"]]){chat}{boit}(art[m["le"]]){lait}{dans}**(art[f["la"]])**{coupe}

Voyons, comment se comporte cette grammaire à partir de la *rre* 6, si nous lui soumettons une *cs* correctement construite : "le chat boit le lait dans la coupe" :

F 5.97. Résultats de GO_c pour une *cs* correcte (LECHAT)

LECHAT, 1, 6,1:
(●art[m["le"]])**(●subs[m["chat"]])**{boit}(●art[m["le"]]){lait}{dans}(●art[f["la"]]){coupe}
LECHAT, 1, 6,2:
(●art[m["le"]])(●subs[m["chat"]]){boit}(●art[m["le"]])**(●subs[m["lait"]])**{dans}(●art[f["la"]]){coupe}
LECHAT, 1, 7,1:
(●art[m["le"]])(●subs[m["chat"]]){boit}(●art[m["le"]])(●subs[m["lait"]]){dans}(●art[f["la"]])
(●subs[f["coupe"]])
LECHAT, 1, 8,1:
(art[m["le"]])(subs[m["chat"]]){boit}(●art[m["le"]])(●subs[m["lait"]]){dans}(●art[f["la"]])
(●subs[f["coupe"]])
LECHAT, 1, 8,2:
(art[m["le"]])(subs[m["chat"]]){boit}**(art[m["le"]])(subs[m["lait"]])**{dans}(●art[f["la"]])
(●subs[f["coupe"]])
LECHAT, 1, 8,3: (art[m["le"]])(subs[m["chat"]]){boit}(art[m["le"]])(subs[m["lait"]]){dans}
(art[f["la"]])(subs[f["coupe"]])

LECHAT, 1, 10,1: (art[m["le"]])(subs["chat"]){boit}(art[m["le"]])(subs["lait"])**(plieu["dans"])**
(art[f["la"]])(subs["coupe"])

Afin de bien comprendre un exemple de traitement d'erreurs, observons le comportement de notre grammaire, GO_c, si l'on introduit une erreur dans la *cs* : "le chat boit le lait dans **le** coupe" :

F 5.98. Résultats avec erreur sur les genres de GO_c (LECHAT)

LECHAT, 1, 4,1: (●art[m["le"]]){chat}{boit}{le}{lait}{dans}**{le}{coupe}**
LECHAT, 1, 4,2: (●art[m["le"]]){chat}{boit}(●art[m["le"]]){lait}{dans}**{le}{coupe}**
LECHAT, 1, 4,3: (●art[m["le"]]){chat}{boit}(●art[m["le"]]){lait}{dans}**(●art[m["le"]]){coupe}**
LECHAT, 1, 6,1:
(●art[m["le"]])(●subs[m["chat"]]){boit}(●art[m["le"]]){lait}{dans}**(●art[m["le"]]){coupe}**
LECHAT, 1, 6,2:
(●art[m["le"]])(●subs[m["chat"]]){boit}(●art[m["le"]])(●subs[m["lait"]]){dans}
(●art[m["le"]]){coupe}
LECHAT, 1, 7,1:
(●art[m["le"]])(●subs[m["chat"]]){boit}(●art[m["le"]])(●subs[m["lait"]]){dans}
(●art[m["le"]])(●subs[f["coupe"]])
LECHAT, 1, 8,1:
(art[m["le"]])(subs[m["chat"]]){boit}(●art[m["le"]])(●subs[m["lait"]]){dans}
(●art[m["le"]])(●subs[f["coupe"]])
LECHATDEUX, 1, 8,2:
(art[m["le"]])(subs[m["chat"]]){boit}(art[m["le"]])(subs[m["lait"]]){dans}
(●art[m["le"]])(●subs[f["coupe"]])

LECHAT, 1, 9,1: l'article : "le" n'a pas le même genre que le nom : "coupe"

Traitons maintenant les syntagmes. Toujours par notre procédé d'*encapsulation préfixée*, nous augmentons encore l'information sur nos objets linguistiques :

F 5.99. Traitement des syntagmes nominaux par *encapsulations préfixées* dans GO_c (LECHAT)

✎ RECHERCHER LES SYNTAGMES NOMINAUX

☞‹x›(art[‹m›[‹a›]])(subs[‹s›])‹r›⇒‹x›sn[art[‹m›[‹a›]]subs[‹s›]]‹r›☜ ✎ *rre* 11

Nous avons trois syntagmes nominaux dans notre phrase, donc, trois itérations du *filtrage de motif* (en gras) :

F 5.100. Résultats du traitement des syntagmes nominaux avec le procédé d'*encapsulation préfixée* dans GO_c (LECHAT)

LECHAT, 1, 11,1:
sn[art[m["le"]]subs["chat"]**]{**boit}(art[m["le"]])(subs["lait"])(plieu["dans"])(art[f["la"]])
(subs["coupe"])

LECHAT, 1, 11,2:
sn[art[m["le"]]subs["chat"]]{boit}**sn[**art[m["le"]]subs["lait"]**]**(plieu["dans"])(art[f["la"]])
(subs["coupe"])

LECHAT, 1, 11,3: sn[art[m["le"]]subs["chat"]]{boit}sn[art[m["le"]]subs["lait"]](plieu["dans"])
sn[art[f["la"]]subs["coupe"]]

Traitons maintenant le syntagme verbal et son complément d'objet direct :

F 5.101. Traitement des syntagmes verbaux dans GO_c (LECHAT)

✎ RECHERCHER UN VERBE TRANSITIF

☞‹x›sn[‹a›]{‹v›}sn[‹b›]‹r›⇒‹x›sn[‹a›](verbt3pp["‹v›"])sn[‹b›]‹r›☜ ✎ *rre* 12

✎ CONSTRUCTION DU SYNTAGME VERBAL ET DU COMPLEMENT D'OBJET DIRECT

☞‹x›sn[‹a›](verbt3pp[‹v›])sn[‹b›](‹r›⇒
sv[sn[‹a›]verbt3pp[‹v›]cd[sn[‹b›]]](‹r›☜ ✎ *rre* 13

L'exécution de ces deux nouvelles *rre*, nous conduit aux résultats suivants (nous mettons en gras les changements pour faciliter la lecture) :

F 5.102. Résultats du traitement des syntagmes verbaux dans la GO_c (LECHAT)

```
LECHAT, 1, 12,1:
sn[art[m["le"]]subs["chat"]](verbt3pp["boit"])sn[art[m["le"]]subs["lait"]](plieu["dans"])
sn[art[f["la"]]subs["coupe"]]
LECHAT, 1, 13,1:
sv[sn[art[m["le"]]subs["chat"]]verbt3pp["boit"]cd[sn[art[m["le"]]subs["lait"]]]](plieu["dans"])
sn[art[f["la"]]subs["coupe"]]
```

Traitons encore le complément circonstanciel de lieu.

F 5.103. Traitement du complément de lieu (LECHAT)

```
✎ CONSTRUCTION D'UN COMPLEMENT CIRCONSTANCIEL DE LIEU

☞sv[‹v›](plieu[‹l›])sn[art[‹m›[‹a›]]subs[‹s›]]‹r›⇒
sv[‹v›]cc[plieu[‹l›]sn[art[‹m›[‹a›]]subs[‹s›]]]‹r›☜ ✎ rre 14
```

Avec comme résultat :

F 5.104. Résultat du traitement du complément de lieu dans la GO_c (LECHAT)

```
LECHAT, 1, 11,1:
sv[sn[art[m["le"]]subs["chat"]]verbt3pp["boit"]cd[sn[art[m["le"]]subs["lait"]]]]
cc[plieu["dans"]sn[art[f["la"]]subs["coupe"]]]
```

Procédons à une vérification. Plus aucune accolade ne devrait être inscrite dans notre dernier résultat. Comme nous l'avons remarqué, dans cette première partie de grammaire, nous nous sommes donné des marqueurs permettant de valider l'exhaustivité de notre traitement. Donc, plus aucune parenthèse ne devrait faire partie de notre dernière *chaîne* de caractères. Donnons-nous des *rre* destinées à vérifier de possibles erreurs :

F 5.105. Traitement des erreurs dans GO_c (LECHAT)

```
✎ EXEMPLES DE TRAITEMENT D'ERREUR

☞‹x›{‹y›⇒Erreur : traitement incomplet↳STOP↲☜ ✎ rre 15
☞‹x›}‹y›⇒Erreur : traitement incomplet↳STOP↲☜ ✎ rre 16
☞‹x›(‹y›⇒Erreur : traitement incomplet↳STOP↲☜ ✎ rre 17
☞‹x›)‹y›⇒Erreur : traitement incomplet↳STOP↲☜ ✎ rre 18
```

Avec une *cs* correctement construite, aucune de ces *rre* ne sera exécutée. Passons au niveau de la phrase :

F 5.106. *Encapsulation préfixée* de la phrase dans la GO_c (LECHAT)

```
✎ ENCAPSULER LE TOUT COMME ETANT UNE PHRASE

☞sv[‹v›]cc[‹c›]⇒ph[sv[‹v›]cc[‹c›]]☜ ✎ rre 19
```

Avec comme résultat :

F 5.107. Résultat de l'*encapsulation préfixée* de la phrase dans la GO_c (LECHAT)

```
LECHAT, 1, 19,1:
ph[sv[sn[art[m["le"]]subs["chat"]]verbt3pp["boit"]cd[sn[art[m["le"]]subs["lait"]]]]
cc[plieu["dans"]sn[art[f["la"]]subs["coupe"]]]]
```

Ayant atteint la fin de notre analyse. Arrêtons-nous quelques instants pour comprendre l'intérêt du procédé.

Nous avons passé par une analyse lexicale, suivie d'une analyse morphosyntaxique par étapes successives. A chacune de ces étapes, nous avons chargé d'informations nos objets linguistiques par la méthode d'*encapsulation préfixée*. De plus, nous avons intégré des marqueurs destinés à la vérification du traitement. Nous avons abouti à un résultat qui n'est qu'une simple *chaîne,* mais lourde d'informations. Cette méthode nous conduit vers la représentation arborescente :

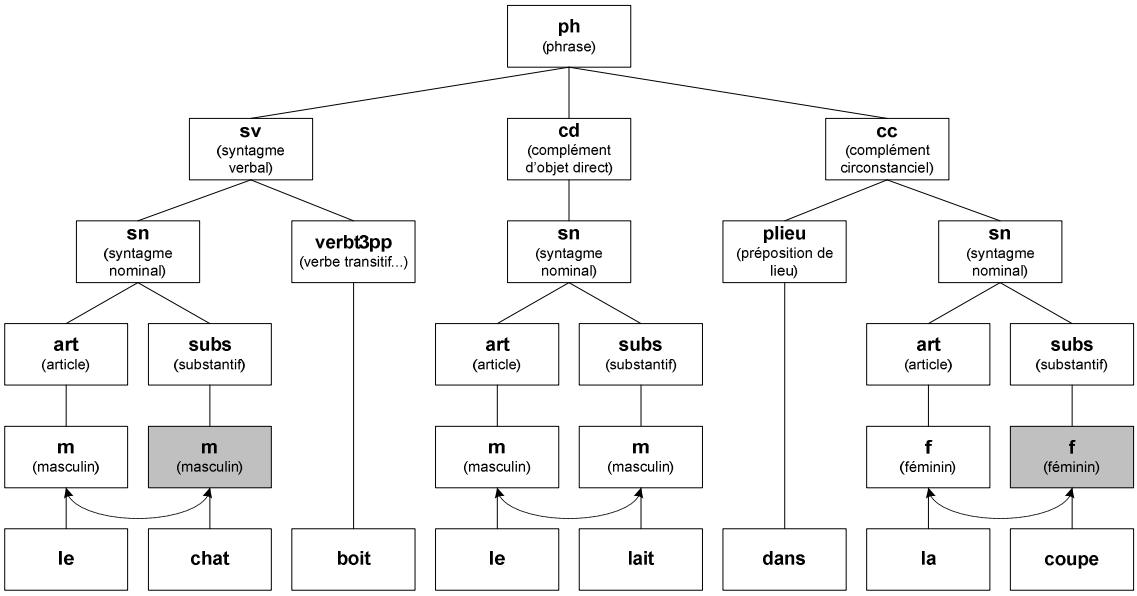


Figure 51. Représentation arborescente ou générative de l'exemple de grammaire GO_c (LECHAT)

Cette représentation rappelle les arborescences des grammaires génératives et transformationnelles de N. Chomsky.

Avec les moyens que nous nous sommes donnés et en se bornant aux définitions des ensembles de référence lexicale que nous avons déclarées, il est difficile d'analyser plus en détail cette phrase. Nous désignons le résultat obtenu comme la *forme analysée*. La plupart des grammaires GO_i qui traitera des *inscriptions* leśniewskiennes passera par un processus d'analyse similaire. Nous utilisons aussi pour déterminer la *forme analysée* un *langage pivot* que nous définissons plus loin.

Il s'agit, maintenant et à partir de cette *forme analysée*, de transformer notre phrase en préfixant le complément de lieu. Continuons la programmation de notre GO_c :

F 5.108. *Transformation* de la phrase dans la GO_c (LECHAT)

```
✎ PREFIXER LA PHRASE AVEC LE COMPLEMENT CIRCONSTANCIEL
✎ suivi d'une virgule

☞ph[sv[‹v›]cc[‹c›]]⇒ph[cc[‹c›], sv[‹v›]]☜ ✎ rre 20
```

Qui nous génère le résultat :

F 5.109. Résultat de la *transformation* de la phrase dans la GO_c(LECHAT)

```
LECHAT, 1, 20,1:
ph[cc[plieu["dans"]sn[art[f["la"]]subs["coupe"]]],
sv[sn[art[m["le"]]subs["chat"]]verbt3pp["boit"]cd[sn[art[m["le"]]subs["lait"]]]]]
```

A partir de la *forme analysée* et en une seule *rre*, nous avons préfixé le complément circonstanciel de lieu. Ce simple exemple montre comment le *LRE* peut représenter et opérer une *transformation* (dans notre cas, grammaticale). Dès lors, le *LRE* a la capacité de mettre en œuvre les formes nécessaires au traitement de certains aspects des grammaires transformationnelles de N. Chomsky.

Evidemment, notre dernier résultat est surchargé d'informations qui rendent sa lisibilité difficile. Il faut à cette étape supprimer toutes les informations inutiles. Nous procédons, pour ce cas, par la méthode du *dépouillement*. Nous allons passer de la *forme analysée* à la *forme dépouillée* :

F 5.110. Dépouillement dans la GO_c (LECHAT)

```
✎ DEPOUILLER LA PHRASE DE SES ENCAPSULATIONS

☞ph[‹p›]⇒‹p›☜ ✎ rre 21✎
☞cc[plieu["‹p›"]sn[art[f["‹a›"]]subs["‹s›"]]]‹z›⇒‹p› ‹a› ‹s›‹z›☜rre 22
☞‹a›, sv[‹b›cd[‹c›]]⇒‹a›, ‹b›‹c›☜ ✎ rre 23
☞‹a›, ‹x›sn[art[‹b›]subs[‹c›]]‹y›⇒‹a›, ‹x›art[‹b›]subs[‹c›]‹y›☜ ✎ rre 24
☞‹a›, ‹b›art[‹c›["‹d›"]]subs["‹e›"]‹f›⇒‹a›, ‹b› ‹d› ‹e› ‹f›☜ ✎ rre 25
☞‹a›, ‹b› verb‹c›["‹d›"] ‹f›⇒‹a›, ‹b› ‹d› ‹f›☜ ✎ rre 26
```

Nous obtenons comme résultats :

F 5.111. Résultats du dépouillement dans la GO_c (LECHAT)

```
LECHAT, 1, 21,1:
cc[plieu["dans"]sn[art[f["la"]]subs["coupe"]]],
sv[sn[art[m["le"]]subs["chat"]]verbt3pp["boit"]cd[sn[art[m["le"]]subs["lait"]]]]
LECHAT, 1, 22,1:
dans la coupe, sv[sn[art[m["le"]]subs["chat"]]verbt3pp["boit"]cd[sn[art[m["le"]]subs["lait"]]]]
LECHAT, 1, 23,1:
dans la coupe, sn[art[m["le"]]subs["chat"]]verbt3pp["boit"]sn[art[m["le"]]subs["lait"]]
LECHAT, 1, 24,1:
dans la coupe, art[m["le"]]subs["chat"]verbt3pp["boit"]sn[art[m["le"]]subs["lait"]]
LECHAT, 1, 24,2: dans la coupe, art[m["le"]]subs["chat"]verbt3pp["boit"]art[m["le"]]subs["lait"]
LECHAT, 1, 25,1: dans la coupe,  le chat verbt3pp["boit"]art[m["le"]]subs["lait"]
LECHAT, 1, 25,2: dans la coupe,  le chat verbt3pp["boit"] le lait
LECHAT, 1, 26,1: dans la coupe,  le chat boit le lait
```

Avec vingt-six *rre*, nous avons abouti à notre objectif. On trouve la grammaire GO_c (dans sa version exécutable) au complet dans l'annexe *Le chat, version 2, p. 208.*

5.8 La puissance du LRE

Dans le développement qui précède, nous avons passé par trois formalismes : les *LF*, la notation *EBNF* et le *LRE*. Donnons-nous une *production* de chacune des grammaires de ces formalismes et considérons l'isomorphisme au quel nous avons à faire :

F 5.112.

LF :	$A \rightarrow Aa$
EBNF :	<A> ::= <A>'a'
LRE :	‹A›⇒‹A›a

Quelle que soit la *chaîne* à analyser ou à générer, les trois formalismes donneront les mêmes résultats, soit des *mots* comme : 'a', 'aa', 'aaa', 'aaaa', … Voyons ce que l'on peut en dire concernant la *puissance* du *LRE*.

Jusqu'à présent, nous avons abordé deux notions de « contexte » : celle des *SL* et celle de la théorie des *LF*.

Revenons sur la théorie des *LF* pour affiner le type du *LRE*. Dans la section *5.6.6 L'encapsulation préfixée, p. 134*, nous avons déterminé que le *LRE* était du type : *dépendant de contexte* puisque nos *rre* peuvent prendre la forme :

$$\alpha \Rightarrow \beta \text{ est de forme : } \alpha \rightarrow \beta \text{ où } |\alpha| \leq |\beta|$$

Nous avons aussi vu dans notre procédé de *dépouillement* dans notre GO_c, (*F 5.110, p. 143*) que les *rre* pouvaient prendre la forme :

$$\alpha \Rightarrow \beta \text{ est de forme : } \alpha \rightarrow \beta \text{ où } |\alpha| \geq |\beta|$$

Par conséquent, nous n'avons aucune restriction sur les *rre* du *LRE* et dès lors, le *LRE* est du type : *récursivement énumérable*. Ce qui s'interprète comme $w \in LRE = L(GRE)$ n'est pas décidable. Ce qui se traduit, aussi, dans le jargon informatique du *RE* par : « une *phrase cs* quelconque soumise à une GO_i quelconque peut faire s'itérer indéfiniment le *RE* ». Nous

l'avons déjà vu et c'est ce qui nous a conduit à donner des *directives* de limitation des itérations d'une *rre* ou d'une GO_i. Cette considération ne nous empêche pas de répondre à notre cahier des charges. La notion de *contexte* dans la théorie des *LF* est très bien définie. Il n'en reste pas moins qu'un aspect nous intéresse particulièrement :

Le compilateur du *RE* peut traiter la *GRE* et l'enchaînement de ses *symboles* avec une longueur d'avance[104]. Après un rapide coup d'œil sur l'*EBNF* de la grammaire *GRE* du *LRE* et pour reprendre le formalisme des *LF*, la *GRE* est construit sur la forme :

$$A \rightarrow \alpha \text{ avec } A \in N \text{ et } \alpha \in (N \cup T)^*$$

Donc, la *GRE* est *indépendante de contexte* ! Par conséquent :

la *GRE* est de type : *indépendante de contexte* ;

le *LRE* et les GO_i que *GRE* traite sont de type : *récursivement énumérable*.

Ce qui se traduit en : « une grammaire formelle (*GRE*) *indépendante de contexte* peut traiter un *LF* (grammatical, le *LRE* et donc les GO_i) *dépendant de contexte* et même *récursivement énumérable* »[105]. Nous pouvons en déduire que le compilateur du *RE* s'arrête toujours. En fait, le compilateur n'a ni besoin de connaître le ou les *symboles* qui précèdent celui qu'il analyse à un moment donné, ni de connaître le ou les *symboles* qui suivront. Nous pouvons aussi en tirer que la notion de contexte au sens des *LF* est une notion issue de la dimension syntaxique.

Dans le cas des *SL*, le *contexte* est défini avec la même rigueur syntaxique que celle que nous avons rencontrée ci-dessus, mais font appel à un cadre référentiel. De façon formelle (en reprenant *la règle d'inférences de définition* de la *protothétique*, section *4.4.1 La règle d'inférences de* définition*, p. 42*) :

> 4. Le foncteur de la fonction logique du *definiendum* est un terme constant. Ce terme constant ne saurait être équiforme à une constante logique de même catégorie syntaxico-sémantique qui appartient déjà au système.
>
> 5. Si le foncteur constant du *definiendum* est destiné à être d'une catégorie syntaxico-sémantique qui appartient déjà au système, le contexte qui le suit doit être similaire au contexte préalablement fixé pour cette catégorie.
>
> 6. Si le foncteur constant du *definiendum* est destiné à être d'une catégorie qui n'appartient pas encore au système, il est nécessaire d'inscrire un nouveau contexte de manière à éviter toute confusion. Cela signifie qu'on ne saurait choisir des parenthèses équiformes à un contexte préalablement inscrit et qui possède le même nombre d'arguments que le *definiendum*.
>
> 7. Les arguments du contexte du *definiendum* sont des termes variables : $\lfloor v_1 v_2 \ldots v_i \rfloor \lceil \equiv(\sum(v_1 v_2 \ldots v_i)\ \text{-}) \rceil$.

Traduit d'une façon syntaxique avec le *LRE* comme une GO_i et pour un *terme constant* qui n'a pas été préalablement introduit dans le *système*, mais dont la *catégorie syntaxico-sémantique* a déjà été introduite et avec un nouveau *contexte* :

F 5.113.
✎ *cs :* *[◙♠]/CAT⟦⟦1/2⟧⟧CST⟦⟦≡⟧⟦μ⟧⟧CTX⟦⟦(--)⟧⟧
✎ La fonction logique du *definiendum* : *[◙♠]
✎ Zone de référence de l'état du système avant la définition de la nouvelle *fonction logique* :
✎ /CAT⟦⟦1/2⟧⟧CST⟦⟦≡⟧⟦μ⟧⟧CTX⟦⟦(--)⟧⟧

✎ *Encapsuler* les *signes* :
☞‹v›/‹mem›⇒↳EXPLOSER(‹v›, '●⟦', '⟧')↲/‹mem›☜ ✎ *rre* 1
✎ *cr :* ●⟦*⟧●⟦⟦⟧●⟦◙⟧●⟦♠⟧●⟦⟧⟧/CAT⟦⟦1/2⟧⟧CST⟦⟦≡⟧⟦μ⟧⟧CTX⟦⟦(--)⟧⟧

✎ Avec des *variables isoformes*, vérifier si *le terme constant* existe déjà dans le système :
☞●⟦‹tc›⟧‹v›/CAT⟦‹cat›⟧CST⟦‹c1›⟦‹tc›⟧‹c2›⟧CTX⟦‹ctx›⟧⇒
‹tc› a déjà été défini dans la système↳STOP↲☜✎ *rre* 2

✎ Procéder à l'*encapsulation préfixée* du *terme constant* et l'inscrire dans le système :

---

[104] C'est ce qui explique certains métasymboles exotiques comme la main de début et de fin de *rre* !

[105] Cela, sachant que *GRE* est doté d'*attributs* comme les *fonctions*, les *instructions* et les *directives*.

☞●[‹tc›]‹v›⁄CAT[‹cat›]CST[‹cst›]CTX[‹ctx›]⇒
TC[‹tc›]‹v›⁄CAT[‹cat›]CST[‹cst›[‹tc›]]CTX[‹ctx›]☜ ✎ *rre* 3
✎ *cr :* TC[*]●[[]●[◙]●[♠]●[]]⁄CAT[[1/2]]CST[[≡][μ][*]]CTX[[(--)]]

✎ Avec des *variables isoformes*, vérifier si la parenthèse gauche de *contexte*
✎ existe déjà dans le système :
☞TC[‹tc›]●[**‹pg›**]‹v›⁄CAT[‹cat›]CST[‹cst›]CTX[‹c1›[**‹pg›**-‹c›]‹c2›]⇒
Le délimiteur gauche du contexte existe déjà dans le système↳STOP↲☜ ✎ *rre* 4

✎ Procéder à l'*encapsulation préfixée* de la parenthèse gauche :
☞TC[‹tc›]●[‹pg›]‹v›⁄‹mem›⇒TC[‹tc›]PG[‹pg›]‹v›⁄‹mem›☜✎ *rre* 5
✎ *cr :* TC[*]PG[[]●[◙]●[♠]●[]]⁄CAT[[1/2]]CST[[≡][μ][*]]CTX[[(--)]]

✎ Avec des *variables isoformes*, vérifier si la parenthèse droite de *contexte*
✎ existe déjà dans le système :
☞‹v›●[**‹pd›**]⁄‹cat›]CST[‹cst›]CTX[‹c1›[‹c›-**‹pd›**]‹c2›]⇒
Le délimiteur droite du contexte existe déjà dans le système↳STOP↲☜ ✎ *rre* 6

✎ Procéder à l'*encapsulation préfixée* de la parenthèse gauche :
☞‹v›●[‹pd›]⁄‹mem›⇒‹v›PD[‹pd›]⁄‹mem›☜ ✎ *rre* 7
✎ *cr :* TC[*]PG[[]●[◙]●[♠]PD[]]⁄CAT[[1/2]]CST[[≡][μ][*]]CTX[[(--)]]

✎ Procéder à l'*encapsulation préfixée* des *termes variables* :
☞‹v1›PG[‹pg›]●[‹tv›]‹v2›PD[‹pd›]⁄‹mem›⇒
‹v1›PG[‹pg›]TV[‹tv›]‹v2›PD[‹pd›]⁄‹mem›☜ ✎ *rre* 8
✎ *cr :* TC[*]PG[[]TV[◙]●[♠]PD[]]⁄CAT[[1/2]]CST[[≡][μ][*]]CTX[[(--)]]
✎ *cr :* TC[*]PG[[]TV[◙]TV[♠]PD[]]⁄CAT[[1/2]]CST[[≡][μ][*]]CTX[[(--)]]

✎ Compter le nombre d'arguments du *contexte* :
☞‹v1›PG[‹pg›]‹v2›PD[‹pd›]⁄‹mem›⇒
‹v1›PG[‹pg›]‹v2›PD[‹pd›]⁄‹mem›
/arg=↳COMPTER(‹v2›,╚TV[‹tv›]╝)↲☜ ✎ *rre* 9
✎ *cr :* TC[*]PG[[]TV[◙]TV[♠]PD[]]⁄CAT[[1/2]]CST[[≡][μ][*]]CTX[[(--)]]/arg=2

✎ Vérifier, à l'aide de *variables isoformes* et 'un marqueur ok,
✎ que la *catégorie syntaxico-sémantique* existe dans le système :
☞‹v›⁄CAT[‹v1›[‹v2›/**‹nb_arg›**]‹v3›]CST[‹cst›]CTX[[‹ctx›]]/arg=**‹nb_arg›**⇒
‹v›/⁄CAT[‹v1›[‹v2›/‹nb_arg›]‹v3›]CST[‹cst›]CTX[[‹ctx›]]/ok☜ ✎ *rre* 10
✎ *cr :* TC[*]PG[[]TV[◙]TV[♠]PD[]]⁄CAT[[1/2]]CST[[≡][μ][*]]CTX[[(--)]]/ok

✎ Si la *catégorie syntaxico-sémantique* n'est pas encore introduite
✎ dans le système, traiter l'erreur :
☞‹v›⁄‹mem›/arg‹v1›⇒
La catégorie n'existe pas dans le système↳STOP↲☜ ✎ *rre* 11

✎ Procéder à l'*encapsulation préfixée* du *contexte*
✎ et à l'introduction du nouveau contexte dans le système :
☞‹tc›PG[‹pg›]‹v›PD[‹pd›]⁄CAT[‹cat›]CST[‹cst›]CTX[‹ctx›]/ok⇒
‹tc›CTX⟩PG[‹pg›]‹v›PD[‹pd›]⟨⁄
CAT[‹cat›]CST[‹cst›]CTX[‹ctx›[‹pg›--‹pd›]] ✎ *rre* 12
✎ *cr :* TC[*]CTX⟩PG[[]TV[◙]TV[♠]PD[]]⟨
✎ ⁄CAT[[1/2]]CST[[≡][μ][*]]CTX[[(--)][--]]

Cette GO_i filtre, d'une part, la construction syntaxique de la *fonction logique* et, d'autre part, suit deux procédés :

(1) aller vérifier dans une « zone de référence » (ce qui a été préalablement introduit dans le *système*) si oui ou non un nouveau *signe* doit être introduit préalablement ou non dans le *système* ;

(2) si cette vérification est validée, apporter de l'information au *signe* par la méthode d'*encapsulation préfixée*.

Le premier procédé assure que le « sens » qu'apporte le deuxième à un *signe* suit la *procédure définitoire* indiquée par la *règle d'inférences de définition*. Sans visiter la référence,

la construction des nouvelles *inscriptions* amènerait des ambiguïtés au *système*[106]. Soulignons que la notion de *contexte* (*rre* 12 : CTX⦃PG⦃[⦄TV⦃◙⦄TV⦃♠⦄PD⦃]⦄⦄) chez Leśniewski, est une partie de la *fonction logique* qui détermine une *catégorie syntaxico-sémantique* et qui suit le *terme constant* (TC⦃..⦄CTX⦃…⦄)[107]. La mise en œuvre des dimensions syntaxique et sémantique et le processus d'accès à une référence, l'état du *système* à un moment donné, sont très puissants.

Cette considération est d'autant plus remarquable, comme nous venons de le montrer, qu'une GO_i, donc des *phrases* du *LRE* ont la *puissance*[108], de traiter cet aspect du *LO*, en répétant encore que la grammaire du *LRE*, soit la *GRE* est une grammaire classée dans les grammaires *indépendantes de contexte*.

5.9 Synthèse sur les réponses au cahier des charges

Comme le *LRE* est complètement défini, nous pouvons maintenant confronter ses compétences à notre cahier des charges (*5.2 Analyse des besoins, p. 82*). Nous retenons deux aspects, les *primitives du LRE*, qui en font sa spécificité de base et qui sont utilisées dans toutes les GO_i *:*

(1) le principe de réécriture ;

(2) la mise en œuvre des *variables isoformes*.

En ce qui concerne : *5.2.1 Outils de transposition et réversibilité, p. 83*, nous sommes capables d'opérer des *transpositions* et inversement à l'aide des *primitives du LRE*, de la déclaration d'ensembles et des *fonctions* HORS et DANS, d'accéder à un cadre référentiel pour déterminer des *identificateurs* de *variables de nom* qui ne sont pas encore attribués. Nous sommes aptes à apporter de l'information supplémentaire par le procédé d'*encapsulation préfixée*. De plus, par le procédé de réécriture et l'utilisation de *corpus*, nous pouvons construire des *inscriptions* du type : Ĉ{ab} ⇒ ⌊ab⌋⌈≡(Ĉ{ab}∧(~(⌊G⌋⌈~(εGa})⌉)⌊G⌋⌈⊃(ε{Ga}ε{Gb})⌉))⌉.

Pour répondre au filtrage attendu par *5.2.2.1 Les règles d'inférences de définition, p. 84*, nous avons déjà vu quelques exemples de filtrage : la GO_c, LECHAT (*5.7 Programmation et analyse détaillée de la* GOc*, p. 136*) et une grammaire GO_i traitant du *contexte* (*F 5.113, p. 144*). L'accès au *corpus* ou aux autres formes de mémorisation, nous permettent de traiter les aspects référentiels liés à l'introduction préalable des *inscriptions* dans le *système*. Les *fonctions* du *LRE* traitant du calcul propositionnel ont aussi été apportées à notre *solution*, le *RE*. Par la primitive de réécriture, nous sommes capables de faire de la conversion de notation.

Afin de générer plusieurs *inscriptions* satisfaisantes à la *règle d'inférences de distribution des quantificateurs*, nous avons défini un processus de gestion d'un *corpus tampon.*

Quant à l'entrée de plusieurs *cs* nécessaire pour les *règles d'inférences de substitution*, nous nous sommes donné un système de gestion d'accès à n'importe quelle ligne du *corpus d'entrée*.

La mise en œuvre de toutes les *fonctions* définies dans le *LRE* va nous permettre d'élaborer un ensemble de GO_i qui montre concrètement la faisabilité de notre approche. Ces grammaires sont présentées dans le chapitre suivant.

---

[106] Par analogie, l'accès à la zone de référence procède comme une relation anaphorique au sens linguistique ; la zone de référence fonctionne comme moyen d'éviter des ambiguïtés.

[107] Par contraste et en se donnant comme définition littéraire de contexte : « les entités linguistiques auxquelles il faut faire appel pour lever de possibles ambiguïtés dans le texte », nous pouvons conclure, pour un discours linéaire d'une langue naturelle, que le contexte précède le texte du moins à un moment précis d'une lecture. Evidemment, une phrase de type : « comme nous verrons plus loin… » fait appel à un aspect contextuel qui suit le texte.

[108] Nous disons aussi, au sens linguistique, qu'elle a la performance pour traiter le corpus considéré.

*5.10 Liste des sous-langages de LO et des sous-grammaires de GO**

Nous avons vu et allons encore aborder un ensemble de sous-langages de *LO* avec leurs sous-grammaires correspondantes de *GO**. Prenons cette liste à titre de résumé :

F 5.114. $LO_c = L(GO_c)$

Un exemple, issu d'un *domaine de connaissances* différent de celui des *SL*, de la grammaire générative et transformationnelle du « chat » ; où $LO_c \subset LO$ et $GO_c \subset GO^*$ (*5.7 Programmation et analyse détaillée de la GOc, p.136*).

F 5.115. $LO_x = L(GO_x)$

Un exemple, pris dans les *SL*, donnant un aperçu de l'utilisation de la notation *EBNF* pour décrire le *definiendum* de la *thèse-définition* de l'inclusion ; où $LO_x \subset LO$ et $GO_x \subset GO_{ot} \subset GO \subset GO^*$ (*F 5.11, p. 89*).

F 5.116. $LO_0 = L(GO_0)$

Un exemple, pris dans les *SL*, montrant les limites de la notation *EBNF* pour répondre aux exigences de la philosophie développementale des *SL* ; où $LO_0 \subset LO$ et $GO_0 \subset GO_{ot} \subset GO \subset GO^*$ (*F 5.12, p. 90*).

F 5.117. $LO_1 = L(GO_1)$ et $LO_2 = L(GO_2)$

Un exemple, pris dans les *SL*, utilisant *LRE* permettant de créer des *non-terminaux* répondant aux exigences de la philosophie développementale des *SL* ; où $LO_1 \subset LO$ et $GO_1 \subset GO_{ot} \subset GO \subset GO^*$ et où $LO_2 \subset LO$ et $GO_2 \subset GO_{ot} \subset GO \subset GO^*$ (*F 5.31, p. 98 et F 5.32, p. 98*).

F 5.118. $LO_3 = L(GO_3)$

Un exemple, pris dans les *SL*, utilisant *LRE* permettant de créer des *non-terminaux* répondant aux exigences de la philosophie développementale des *SL* et qui mémorise les *signes* utilisés ; où $LO_3 \subset LO$ et $GO_3 \subset GO_{ot} \subset GO \subset GO^*$ (*F 5.33, p. 99*).

F 5.119. $LO_p = L(GO_p)$

Le *langage pivot* des *inscriptions* analysées des *SL*, servant de base de traitement à toutes les autres GO_i en charge d'une partie du langage *LO* ; où $LO_p \subset LO$ et $GO_p \subset GO \subset GO^*$ (*6.2 Forme analysée et langage pivot, p. 151*).

F 5.120. Les sous-langages des *SL*

Nous avons divisé les sous-langages des *SL* en cinq ensembles (*8. Bilan et conclusion de la recherche, p. 185*) :

$LO_{ot} = L(GO_{ot})$: l'ensemble des *axiomes*, des *thèses*, des *fonctions logiques* des *SL* ;

$LO_{or} = L(GO_{or})$: tous les langages et grammaires qui traitent des *règles d'inférences* des *SL* ;

$LO_{od} = L(GO_{od})$: la grammaire qui traite des *définitions inductives* des *SL* ;

$LO_{oc} = L(GO_{oc})$: la grammaire qui permet d'analyser les *catégories syntaxico-sémantiques* ;

$LO_{cp} = L(GO_{cp})$: la grammaire qui exécute du calcul propositionnel.

où les cinq LO_j qui précèdent : $LO_j \subset LO$ et $GO_j \subset GO \subset GO^*$.

F 5.121. Les sous-grammaires GO_i

Une GO_i est une quelconque grammaire qui décrit un sous-langage LO_i où $LO_i = L(GO_i)$. Voici quelques exemples traités dans notre recherche :

GO_{i+1}, **gestion des *contextes* ;** où $LO_{i+1} \subset LO_{ot} \subset LO$ et $GO_{i+1} \subset GO_{ot} \subset GO \subset GO^*$ (*F 5.113, p. 144*) ;

GO_{i+2}, préanalyse, procédure définitoire, **analyse et mise au format pivot** ; où $LO_{i+2} \subset LO_{ot} \subset LO$ et $GO_{i+2} \subset GO_{ot} \subset GO \subset GO^*$ (*6.3 La GOi d'analyse et de mise au format LOp, p. 155*) ;

GO_{i+3}, détermination des ***catégories syntaxico-sémantiques*** ; où $LO_{i+3} \subset LO_{oc} \subset LO$ et $GO_{i+3} \subset GO_{oc} \subset GO \subset GO^*$ (*6.5 Détermination des catégories syntaxico-sémantiques, p. 160*) ;

GO_{i+4}, **calcul propositionnel** ; où $LO_{i+4} \subset LO_{cp} \subset LO$ et $GO_{i+4} \subset GO_{cp} \subset GO \subset GO^*$ (*6.6 Les calculs propositionnels, p. 161*) ;

GO_{i+5}, **conversion de notation** ; où $LO_{i+5} \subset LO_{ot} \subset LO$ et $GO_{i+5} \subset GO_{ot} \subset GO \subset GO^*$ (*6.7 Les conversions de notations, p. 164*) ;

GO_{i+6}, ***règle d'inférences de détachement*** ; où $LO_{i+6} \subset LO_{or} \subset LO$ et $GO_{i+6} \subset GO_{or} \subset GO \subset GO^*$ (*6.8 Traitement de la règle d'inférences de détachement, p. 164*) ;

GO_{i+7}, ***règle d'inférences de distribution des quantificateurs*** ; où $LO_{i+7} \subset LO_{or} \subset LO$ et $GO_{i+7} \subset GO_{or} \subset GO \subset GO^*$ (*6.9 La règle d'inférences de distribution des quantificateurs, p. 166*) ;

GO_{i+8}, ***règle d'inférences de substitution*** ; où $LO_{i+8} \subset LO_{or} \subset LO$ et $GO_{i+8} \subset GO_{or} \subset GO \subset GO^*$ (*6.10 Traitement de la règle d'inférences de substitution (protothétique), p. 170*) ;

GO_{i+9}, ***directive d'extensionalité*** ; où $LO_i \subset LO_{or} \subset LO$ et $GO_{i+9} \subset GO_{or} \subset GO \subset GO^*$ (*6.11 La directive d'extensionalité et traitement des filtres syntaxiques et sémantiques, p. 171*) ;

GO_{i+10}, ***directive d'extensionalité*** ; où $LO_{i+10} \subset LO_{or} \subset LO$ et $GO_{i+10} \subset GO_{or} \subset GO \subset GO^*$ (*6.11 La directive d'extensionalité et traitement des filtres syntaxiques et sémantiques, p. 171*) ;

GO_{i+11}, **réversibilité de la transposition** ; où $LO_{i+11} \subset LO_{ot} \subset LO$ et $GO_{i+11} \subset GO_{ot} \subset GO \subset GO^*$ (*6.14 Remplacement des termes variables et des termes constants par des entités linguistiques en langue naturelle, p. 173*) ;

A noter que :

$GO_{ot} = GO_x \cup GO_0 \cup GO_1 \cup GO_2 \cup GO_3 \cup GO_{i+1} \cup GO_{i+2} \cup GO_{i+5} \cup GO_{i+11} \cup GO_i \ldots$

$GO_{or} = GO_{i+6} \cup GO_{i+7} \cup GO_{i+8} \cup GO_{i+9} \cup GO_i \ldots$

$GO_{oc} = GO_{i+3} \cup GO_i \ldots$

$GO_{cp} = GO_{i+4} \cup GO_i \ldots$

$GO = GO_p \cup GO_{ot} \cup GO_{or} \cup GO_{oc} \cup GO_{cp} \cup GO_i \ldots$

$GO^* = GO_c \cup \ldots \cup GO_k \cup GO_{k+1} \cup \ldots GO_i \ldots$

5.11 Conclusion

Après avoir défini un cahier des charges dans la perspective d'automatiser certains aspects destinés aux non-logiciens, nous avons défini formellement un langage, *LRE* dans la perspective de traiter des grammaires, GO_i, qui elles-mêmes gouvernent les phrases du *LO*. Nous nous sommes assuré du formalisme du *LRE* en nous basant sur la théorie des *LF*.

Nous avons fait quelques considérations sur la *puissance* du *LRE*, un langage grammatical destiné aux GO_i, des grammaires *récursivement énumérables*, alors que la grammaire du *LRE*, *GRE*, se limite à une grammaire *indépendante de contexte*. Avec la grammaire *GRE*, nous pouvons décider si une *phrase* appartient ou non au *LRE* qui traite de grammaires GO_i, mais avec les GO_i, nous ne pouvons pas décider si une *phrase* appartient à *LO* !

Les quelques exemples de GO_i, qui ont illustré notre propos, nous ont déjà donné un avant-goût de la réponse à notre cahier des charges, chose que nous développons plus en détail dans le chapitre suivant.

A ce niveau de notre développement, nous savons que nos travaux nous permettent de de décrire le *LO* à l'aide de grammaires GO_i. Ces grammaires sont programmées à l'aide de notre *LRE* qui est défini comme : $LRE = L(GRE)$ et par conséquent $LO = L(L(GRE))$. L'élaboration de $LRE = L(GRE)$ a permis le développement s'une solution informatique qui traite des *phrases* de *LO*. Nous avons donc atteint notre objectif.

6. GO_i destinées aux logiques de Leśniewski

Dans ce chapitre nous allons aborder en détail des exemples de GO_i qui mettent en œuvre les SL et les processus de transposition à partir et vers le domaine de connaissances. Chaque règle d'inférences des SL est couverte par un exemple. La détermination de la catégorie syntaxico-sémantique et du calcul propositionnel sont aussi traités.

Nous montrons par cette démarche que notre approche répond à notre objectif de faisabilité : le LRE a la puissance nécessaire pour traiter l'ensemble des corpus du LO que nous avons retenu dans notre cahier des charges du chapitre précédant.

6.1 Introduction

Nos travaux sont destinés à montrer la faisabilité de l'approche et non pas à livrer une solution clé en main, prête à être mise en œuvre. Toutes les GO_i que nous présentons dans les sections qui suivent sont des exemples. Même si elles s'exécutent correctement, elles ne traitent pas toujours tous les cas. Elles donnent des orientations pour des travaux qui permettraient une implantation dans un environnement de production. Nous nous positionnons comme une couche de facilitation dans un processus qui traite le *LO*.

Nous nous sommes donné les cinq intentions suivantes, à savoir de traiter :

1) la vérification de la bonne formation d'une quelconque *inscription* leśniewskienne de la *protothétique* ou de l'*ontologie* en suivant la *procédure définitoire* ;
2) la conversion d'*inscriptions* notées en *version contextuelle* vers la notation en *version catégorielle* ;
3) la détermination de la *catégorie syntaxico-sémantique* d'une *thèse-définition* ;
4) quand il est possible, le calcul propositionnel du *sous-quantificateur* dominant d'une *inscription* de type *thèse-définition, axiome* ou *fonction logique* ;
5) les *règles d'inférences :* de *détachement*, de la *distribution des quantificateurs*, de *substitution* et de la *directive d'extensionalité* de la *protothétique* ou de l'*ontologie*.

Dans nos exemples de GO_i, nous n'avons pas jugé nécessaire de distinguer si elles s'adressaient plutôt à la *protothétique* qu'à l'*ontologie*. Les objectifs décrits ci-dessus sont interdépendants. Par exemple, avant de calculer une table de vérité, nous devrons nous assurer que l'*inscription* considérée est bien formée. Chacun des exemples d'extraits des GO_i, présentés dans les sections qui suivent, est pris dans un exemple complet recensé dans l'annexe *Exemples de grammaires GOi, p. 207*.

6.2 Forme analysée et langage pivot

Une *inscription* leśniewskienne ne peut pas être traitée dans sa « version brute ». Elle présenterait trop de complexité et de risques d'ambiguïtés d'analyse dans des traitements aussi délicats que les *règles d'inférences*, par exemple. Nous avons dû nous donner une *forme analysée* qui contienne les informations nécessaires pour tous les traitements escomptés. Nous avons défini une *forme analysée* qui constitue un *langage pivot*, LO_p. Le LO_p porte l'analyse et les informations nécessaires qui servent à toutes nos GO_i qui traitent d'*inscriptions* leśniewskiennes. Les *phrases* du LO_p forment un sous-ensemble du *LO*. Le LO_p est défini par la

grammaire GO_p qui appartient aux grammaires *GO**. Cette mise en forme est basée sur le principe de l'*encapsulation préfixée* et postfixée. Voici les règles de grammaires de ce *langage pivot* :

EBNF du LO_p

RD 6.2-1 *Encapsulation des généralisations, des quantificateurs et sous-quantificateurs*

<encap_xG> ::= <pref_xG> <i> <ms_xGg> <encap_xQ> <encap_xSQ> <ms_post> <pref_xG> <i> <ms_xGd>

<encap_xQ> ::= <pref_xQ> <i> <signePrim_Qg> <encap_xTQ> { <encap_xTQ> } <ms_post> <pref_xQ_> <i> <signePrim_Qd>

<encap_xSQ> ::= <pref_xSQ> <i> <signePrim_SQg> <encap_xE> <ms_post> <pref_xSQ> <i> <signePrim_SQd>

<encap_xFL> ::= <pref_xFL> <i> <ms_xFLg> <encap_xTC> <encap_xE> <ms_post> <pref_xFL> <i> <ms_xFLd>

RD 6.2-2 *Encapsulation des expressions*

<encap_xE> ::= <pref_xE> <i> <ms_xEg> <xEXP> <ms_post> <pref_xE> <i> <ms_xEd>

<xEXP> ::= <xT> | <xE>

<xE> ::= <encap_xTC> <encap_xDELIMg> <xInE> { <xInE> } <encap_xDELIMd>

<xInE> ::= <xT> <xT> | <xT> <xE> | <xE> <xT> | <xE> <xE> | <xT>

<xT> ::= <encap_xTV> | <xG> | <encap_xDUM>

<xG> ::= <encap_xG>

<encap_xDUM> ::= <pref_xDUM> <ms_xDUMg> <xDUM> <ms_xDUMd>
<xDUM> ::= <encap_xTC> <encap_xCTX> { <encap_xCTX> }

RD 6.2-3 *Encapsulation des termes le ℒO*

Termes constants ou foncteur

<encap_xTC> ::= <pref_xTC> <ms_xTCg> <xTCF> <ms_xTCd>
<xTCF> ::= <xTC> | <xF>
<xTC> ::= <signePrim_BIC> | <signeTC_hors>
<xF> ::= <signeF_hors>

Termes variables

<encap_xTV> ::= <pref_xTV> <i> <ms_xTVg> <TYPE_TV> <ms_post> <pref_xTV> <i> <ms_xTVd>
<TYPE_TV> ::= <TV_typeDL> <TV> | <TV_typeCP> <CP>
<TV_typeDL> ::= 'D/' | 'L/'
<TV_typeCP> ::= 'C/'
<CP> ::= <signeCP_hors>
<TV> ::= <signeTV_hors>

Termes quantifiés

<encap_xTQ> ::= <pref_xTQ> <ms_xTQg> <xTQ> <ms_xTQd>
<xTQ> ::= <signeTV_hors> | <signeF_hors>

RD 6.2-4 *Encapsulation des contextes*

<encap_xCTX> ::= <pref_xCTX> <i> <ms_xCTXg> <encap_xDELIMg> <encap_xTV> { <encap_xTV> } <encap_xDELIMd> <ms_post> <pref_xCTX> <i> <ms_xCTXd>

RD 6.2-5 *Encapsulations des délimiteurs de ℒO*

<encap_xDELIMg> ::=<pref_xPG> <ms_xPG> <signePG_equi_hors> <ms_xPD>
<encap_xDELIMd> ::=<pref_xPD> <ms_xPG> <signePD_equi_hors> <ms_xPD>

RD 6.2-6 *Terminaux de ℒ0*

Termes variables

<signeTV_hors> ::= <signeVP_hors> | <signeVN_hors>
<signeVP_hors> ::= ‘p’ | ‘q’ | ‘r’ | ‘s’ | …
<signeVN_hors> ::= <signeVS_hors> | <signeVG_hors>
<signeVS_hors> ::= ‘A’ | ‘B’ | ‘C’ | ‘D’ | …
<signeVG_hors> ::= ‘a’ | ‘b’ | ‘c’ | ‘e’ | …

Termes constants

<signeTC_hors> ::= <signeTC1_hors> | <signeTC2_hors>| <signeTCn_hors> | <signeTCmc_hors>
<signeCP_hors> ::= ‘V’ | ‘F’ | ‘T’
<signePrim_BIC> ::= ‘≡’
<signeTC1_hors> ::= ‘~’ | ‘¬’ | ‘α’ | ‘!’ | ‘↛’ | ‘↓’ |…
<signeTC2_hors> ::= ‘μ’ | ‘ε’ | ‘Ц’ | ‘∧’ | ‘∨’ | ‘⊃’ |…
<signeTCn_hors> ::= ‘Δ’ | ‘δ’ |…
<signeTCmc_hors> ::= ‘τ’ | ‘ω’ | …

Foncteurs propositionnels

<signeF_hors> ::= <signeF1_hors> | <signF2_hors>
<signF2_hors> ::= ‘f’ | ‘g’ | …
<signeF1_hors> ::= ‘h’ | …

RD 6.2-7 *Délimiteurs de ℒO*

<signePrim_Qg> ::= ‘⌊’
<signePrim_Qd> ::= ‘⌋’
<evp_Q> ::= <signePrim_Qg> <signePrim_Qd>
<signePrim_SQg> ::= ‘⌈’
<signePrim_SQd> ::= ‘⌉’
<evp_SQ> ::= <signePrim_SQg> <signePrim_SQd>
<signePG_equi_hors> ::= <pgA_sy> | <pgB_sy> | <pgC_sy> | <pgD_sy> | <pgE_sy> | …
<pgA_sy> ::= ‘(’
<pgB_sy> ::= ‘[’
<pgC_sy> ::= ‘{’
<pgD_sy> ::= ‘<’
<pgE_sy> ::= ‘⟨’
…
<signePD_equi_hors> ::= <pdA_sy> | <pdB_sy> | <pdC_sy> | <pdD_sy> | <pdE_sy> | …
<pdA_sy> ::= ‘)’
<pdB_sy> ::= ‘]’
<pdC_sy> ::= ‘}’
<pdD_sy> ::= ‘>’
<pdE_sy> ::= ‘⟩’
…

<evp_DELIM> ::= <pgA_sy><pdA_sy> | <pgB_sy><pdB_sy> |
<pgC_sy><pdC_sy> | …

RD 6.2-8 *Signes primitifs*

<signePrim_dans> ::= <signePrim_Qg> | <signePrim_Qd> | <signePrim_SQg> |
<signePrim_SQd> | <signePrim_BIC> | <pgA_sy> |
<pdA_sy>

RD 6.2-9 *Métasymboles de $\mathcal{L}O_p$*

<metaMarqueurs> ::= <ms_mi>| <ms_decog> | …

<ms_mi> ::= '•' Marque d'invalidité
<ms_decog> ::= '  ' Marque de decomposition gauche
<ms_decod> ::= '  ' Marque de decomposition droite
<ms_tmg> ::= '  ' Encapsulation gauche des terminaux
<ms_tmd> ::= '  ' Encapsulation droite des terminaux
<ms_meg> ::= '  ' Métaencapsulation générale gauche
<ms_med> ::= '  ' Métaencapsulation générale droite
<ms_numg> ::= '  ' Métasigne gauche des numéroteurs de niveau
<ms_numd> ::= '  ' Métasymbole droit des numéroteurs de niveau
<ms_post> ::= '  ' Fin d'encapsulation
<ms_xGg> ::= '  '
<ms_xGd> ::= '  '
<ms_xEg> ::= '  ' Métasymbole gauche des expressions
<ms_xEd> ::= '  ' Métasymbole droit des expressions
<ms_xTCg> ::= '  '
<ms_xTCd> ::= '  '
<ms_xTVg> ::= '  ' Métasymbole gauche des termes
<ms_xTVd> ::= '  ' Métasymbole droit des termes
<ms_xTQg> ::= '  '
<ms_xTQd> ::= '  '
<ms_xFLg> ::= '  '
<ms_xFLd> ::= '  '
<ms_xDUMg> ::= '  '
<ms_xDUMd> ::= '  '
<ms_xCTXg> ::= '  '
<ms_xCTXg> ::= '  '
<ms_xPG> ::= '  ' Métasymbole gauche des délimiteurs
<ms_xPD> ::= '  ' Métasymbole droit des délimiteurs

Prefixes

<pref_xG> ::= 'xG_'
<pref_xQ> ::= 'xQ_'
<pref_xSQ> ::= 'xSQ_'
<pref_xE> ::= 'xE_'
<pref_xTC> ::= 'xTC'
<pref_xTV> ::= 'xTV_'
<pref_xTQ> ::= 'xTQ'
<pref_xPG> ::= 'xPG'
<pref_xPD> ::= 'xPD'
<pref_xFL> ::= 'xFL_'
<pref_xDUM> ::= 'xDUM'
<pref_xCTX> ::= 'xCTX_'
<pref_xTVQ> ::= 'xTVQ'

Dans le LO_p et pour respecter la philosophie développementale des *SL*, nous avons déterminé deux types de *signes :* les *signes primitifs* et les *signes* (_hors) que le *SL* ne connaît

pas au moment de la construction d'une nouvelle *thèse*. Sachant que le nombre de caractères n'est pas infini dans le *RE*, il n'en reste pas moins que nous avons « prédéterminé » les *signes* utilisés pour les *termes constants*, les *termes variables* et les *délimiteurs*. Même si nous avons le moyen de mémoriser les nouveaux *signes* introduits, ceci facilite la partie filtrage de nos analyses des *inscriptions*.

Voici en représentation graphique la *forme analysée* et « informée » que nous obtenons, par exemple, de la *thèse-définition* D12 (*F 4.24, p. 46*), soit : ⌊pqr⌋⌈≡(ω[p]\q/(r)≡(~(≡(pq))r))⌉ :

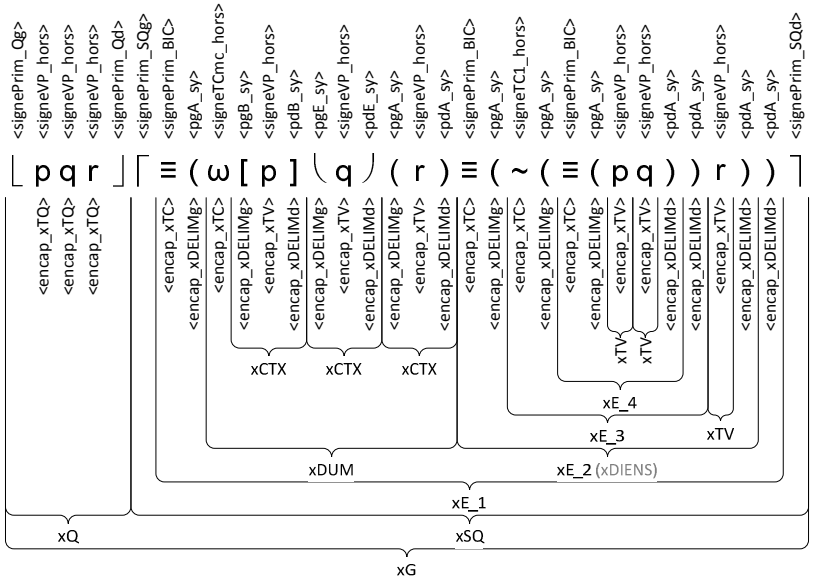


Figure 52. Pré-analyse de la *thèse* D12

En traduction comme *chaîne cr* de la grammaire GO_p du *langage pivot* LO_p, on obtient :

F 6.1.

```
xG_0≺xQ_0⌊xTQ{p}xTQ{q}xTQ{r}/xQ_0⌋xSQ⌈xE_1{xTC}≡{
xDUM{xTC}ω{xCTX_1[xPG{(}xTV_1{D/p/xTV_1}xPD{)}]/xCTX_1]
xCTX_2[xPG{\}xTV_2{D/q/xTV_2}xPD{/}]/xCTX_2]
xCTX_3[xPG{(}xTV_3{D/r/xTV_3}xPD{)}]/xCTX_3]}
xE_2{xTC}≡{xPG{(}xE_3{xTC}~{xPG{(}
xE_4{xTC}≡{xPG{(}xTV_4{L/p/xTV_4}xTV_5{L/q/xTV_5}xPD{)}]/xE_4}
xPD{)}]/xE_3}xTV_6{L/p/xTV_6}xPD{)}xPD{)}]/xE_3}/xE_2}
/xE_1}/xSQ_0⌉/xG_0≻
```

Evidemment cette *cr* est illisible. La *forme analysée* en *langage pivot* ne sert qu'à un usage interne, mais elle apporte les informations nécessaires aux traitements de toutes les autres GO_i.

Pour passer la *forme analysée* de GO_i en GO_i, nous nous donnons comme format standard de *cs*, traitées et transmises, l'*encapsulation* principale :

≻‹avant›►‹forme_analysée›◄‹après›≺

La zone ‹avant› est destinée à conserver une *inscription* dans son format brut (non analysé). La zone ‹forme_analysée› est réservée pour passer la forme à traiter, donc, la *forme analysée*. Le morceau de *chaîne* ‹après› sert à enregistrer des nouveaux résultats ou des nouvelles informations.

6.3 La GO_i d'analyse et de mise au format LO_p

Nous avons décidé qu'avant toute chose, une nouvelle *inscription* doit passer, dans un premier temps, par un filtre qui assure sa bonne formation et, dans un deuxième temps sa mise

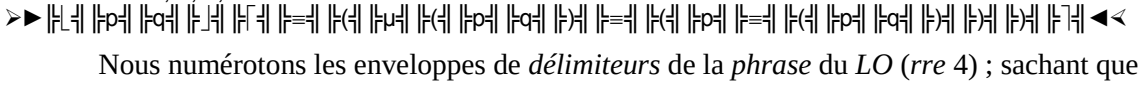
en forme dans le standard analysé, soit la *forme analysée* en *langage pivot*. Nous avons divisé ce traitement en deux GO_i : PREANALYSE et PROCDEF. La première, PREANALYSE, traite les éléments de base des *règles d'inférences de définition.* Elle s'assure que les *expressions* sont bien formées. Elle procède à l'*encapsulation* des éléments morphosyntaxiques en évitant une quelconque ambiguïté qui pourrait mettre en échec des traitements futurs. La deuxième, PROCDEF, se charge d'une analyse un peu plus profonde. Elle vérifie, par exemple, qu'il n'y a pas de répétition de *termes* dans les *quantificateurs*. Elle établit les relations entre *lieurs* et *variables liées*, etc. Pour mettre en œuvre ces deux grammaires, nous les avons chaînées dans la grammaire ANALYSE, qui est donnée dans sa version complète dans l'annexe *Analyse, p. 210*.

6.3.1 La GO_i Préanalyse

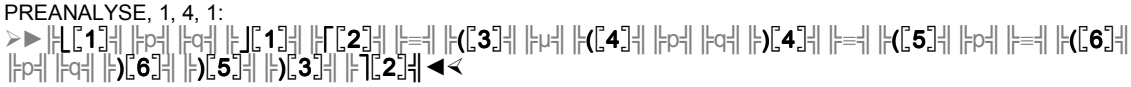
Une des vocations de la GO_i, PREANALYSE (annexe *Préanalyse, p. 210*), est le traitement de l'ensemble des *signes terminaux*. Nous avons défini, dans le bloc « ensemble » de cette dernière, les *signes terminaux* (y compris les métasignes de la GO_p) qui nous serviront à apporter l'information nécessaire à leurs *encapsulations*.

Comme dans pratiquement toutes nos GO_i, nous testons le format de la *cs* dans les *rre* 1 et 2. L'utilisation de la forme atomique des *signes* que fait Leśniewski pour ses objets morphosyntaxiques, nous facilite la tâche. Il nous suffit de décomposer la *chaîne* en *signes* isolés pour le traitement qui suit, soit la *rre* 3, avec la *cs* :

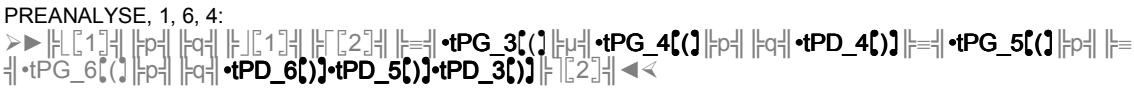
cs : ➢►⌊pq⌋⌈≡(μ(pq)≡(p≡(pq)))⌉◄≺

PREANALYSE, 1, 3, 1:


Nous numérotons les enveloppes de *délimiteurs* de la *phrase* du *LO* (*rre* 4) ; sachant que les enveloppes considérées sont définies dans le bloc ensemble pour l'*identificateur* evp. Soit :

PREANALYSE, 1, 4, 1:


Dans les *rre* 5 et 6, nous *encapsulons* les *délimiteurs* par des préfixes temporaires qui devront être validés plus loin. Avec comme résultat :

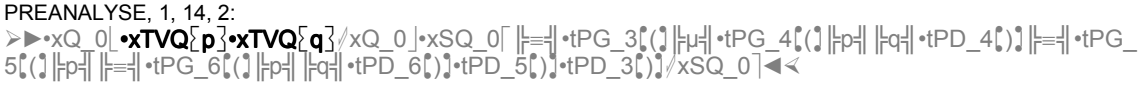
PREANALYSE, 1, 6, 4:


Dans cet exemple de GO_i, nous cherchons d'abord à dépister les éléments qui sont les plus faciles à détecter. Ce qui nous permet de diminuer de possibles ambiguïtés pour des éléments qui restent à cerner. Dans les *rre* 7 à 10, nous *encapsulons* les *délimiteurs*, indiscutablement univoques, de *quantificateurs* et de *sous-quantificateurs*. Ceci nous permet de traiter les *encapsulations* des *quantificateurs* et des *sous-quantificateurs* dans les *rre* 11 à 13. Soit :

PREANALYSE, 1, 10, 1:


Nous notons en passant que ces *rre* conviennent au traitement de toutes les *généralisations*, dominantes et inscrites dans une *expression*. Nous utilisons t (pour temporaire) dans le préfixe des *encapsulations* pour marquer que ce morceau de *chaîne* devra encore être validé par la suite et remplacé par x, mis pour *version contextuelle*. Cette partie de la GO_i répond à la première partie des *règles d'inférences de définition.* Elle vérifie la construction d'une *généralisation*, du *quantificateur* et du *sous-quantificateur*.

Traitons maintenant le contenu du (des) *quantificateur(s)* ; soit les *rre* 14 et 15 :

PREANALYSE, 1, 14, 2:


Ce qui répond au traitement des *termes variables* dans le *quantificateur* des *règles d'inférences de définition*. Ensuite, dans les *rre* 17 à 24, nous traitons le *definiendum* d'une *thèse-définition* et son ou ses *context*es. Avec pour *cr* :

PREANALYSE, 1, 24, 1:
➢►xQ_0⌊•xTVQ⁅p⁆•xTVQ⁅q⁆/xQ_0⌋xSQ_0⌈╞═╡•tPG_3❴❵•provFLTN⁅xTCAC⁅μ⁆xCTX_1⁅xPG❴❵•xTV_0⁅D/p/xTV_0⁆•xTV_0⁅D/q/xTV_0⁆xPD❴❵⁆/xCTX_1⁆/provFLTN⁆╞═╡•tPG_5❴❵╞p╡╞═╡•tPG_6❴❵╞p╡╞q╡•tPD_6❴❵❵•tPD_5❴❵❵•tPD_3❴❵❵/xSQ_0⌉◄≺

Le bloc de *rre* 25 à 28 *encapsule* les *termes constants*. Soit comme *cr* :

PREANALYSE, 1, 25, 3:
➢►xQ_0⌊•xTVQ⁅p⁆•xTVQ⁅q⁆/xQ_0⌋xSQ_0⌈•xBOP⁅≡⁆•tPG_3❴❵•provFLTN⁅xTCAC⁅μ⁆xCTX_1⁅xPG❴❵•xTV_0⁅D/p/xTV_0⁆•xTV_0⁅D/q/xTV_0⁆xPD❴❵⁆/xCTX_1⁆/provFLTN⁆•xBOP⁅≡⁆•tPG_5❴❵╞p╡•xBOP⁅≡⁆•tPG_6❴❵╞p╡╞q╡•tPD_6❴❵❵•tPD_5❴❵❵•tPD_3❴❵❵/xSQ_0⌉◄≺

La *rre* 32 *encapsule* les *termes variables :*

PREANALYSE, 1, 32, 3:
➢►xQ_0⌊•xTVQ⁅p⁆•xTVQ⁅q⁆/xQ_0⌋xSQ_0⌈•xBOP⁅≡⁆•tPG_3❴❵•xE_1⁅xTCAC⁅μ⁆xCTX_1⁅xPG❴❵•xTV_0⁅D/p/xTV_0⁆•xTV_0⁅D/q/xTV_0⁆xPD❴❵❵/xCTX_1⁆/xE_1⁆•xBOP⁅≡⁆•tPG_5❴❵•pTV_1⁅L/p/pTV_1⁆•xBOP⁅≡⁆•tPG_6❴❵•pTV_2⁅L/p/pTV_2⁆•pTV_3⁅L/q/pTV_3⁆•tPD_6❴❵❵•tPD_5❴❵❵•tPD_3❴❵❵/xSQ_0⌉◄≺◊/cnttrm=4/cntexp=2◊

Traitons toutes les *expressions* (voir *RD 6.2-2, p. 152*) une à une et pour n'importe quelle couche de parenthèsage. Les *rre* 35 à 41 traitent une *expression* constituée d'un *terme constant* unaire appliqué à un *terme* (voir *RD 6.2-3, p. 152*). Les *rre* 42 à 46 se chargent d'*encapsuler* les couches d'*expressions* les plus profondes constituées de deux *termes*. Nous obtenons pour notre exemple :

PREANALYSE, 1, 45, 1:
➢►xQ_0⌊•xTVQ⁅p⁆•xTVQ⁅q⁆/xQ_0⌋xSQ_0⌈•xBOP⁅≡⁆•tPG_3❴❵•xE_1⁅xTCAC⁅μ⁆xCTX_1⁅xPG❴❵•xTV_0⁅D/p/xTV_0⁆•xTV_0⁅D/q/xTV_0⁆xPD❴❵❵/xCTX_1⁆/xE_1⁆•xBOP⁅≡⁆•tPG_5❴❵•pTV_1⁅L/p/pTV_1⁆•xE_2⁅xBOP⁅≡⁆xPG❴❵•xTV_2⁅L/p/xTV_2⁆•xTV_3⁅L/q/xTV_3⁆xPD❴❵❵/xE_2⁆•tPD_5❴❵❵•tPD_3❴❵❵/xSQ_0⌉◄≺◊/cnttrm=4/cntexp=3◊

Quant aux *rre* allant de 47 à 59, elles vont traiter les *expressions* construites, soit d'un *terme* et d'une *expression*, soit d'une *expression* et d'un *terme*, soit de deux *expressions*. Evidemment, nous nous sommes doté de branchements conditionnels qui vont exécuter à nouveau cette partie de la GO_i jusqu'à épuisement. Ce qui nous donne à cette étape intermédiaire :

PREANALYSE, 1, 58, 2:
➢►xQ_0⌊•xTVQ⁅p⁆•xTVQ⁅q⁆/xQ_0⌋xSQ_0⌈•xE_4⁅xBOP⁅≡⁆xPG❴❵xE_1⁅xTCAC⁅μ⁆xCTX_1⁅xPG❴❵•xTV_0⁅D/p/xTV_0⁆•xTV_0⁅D/q/xTV_0⁆xPD❴❵❵/xCTX_1⁆/xE_1⁆xE_3⁅xBOP⁅≡⁆xPG❴❵•xTV_1⁅L/p/xTV_1⁆xE_2⁅xBOP⁅≡⁆xPG❴❵•xTV_2⁅L/p/xTV_2⁆•xTV_3⁅L/q/xTV_3⁆xPD❴❵❵/xE_2⁆xPD❴❵❵/xE_3⁆xPD❴❵❵/xE_4⁆/xSQ_0⌉◄≺◊/cnttrm=4/cntexp=5◊/mark_suivant

A part quelques cas particuliers qui ne sont pas concernés par notre exemple, la GO_i a couvert sa fonction de vérification de la construction correcte de la *thèse*. Il reste quelques tests d'erreurs et la forme livrée par PREANALYSE est :

PREANALYSE, 1, 78, 1:
➢►xQ_0⌊•xTVQ⁅p⁆•xTVQ⁅q⁆/xQ_0⌋xSQ_0⌈xE_4⁅xBOP⁅≡⁆xPG❴❵xE_1⁅xTCAC⁅μ⁆xCTX_1⁅xPG❴❵•xTV_0⁅D/p/xTV_0⁆•xTV_0⁅D/q/xTV_0⁆xPD❴❵❵/xCTX_1⁆/xE_1⁆xE_3⁅xBOP⁅≡⁆xPG❴❵•xTV_1⁅L/p/xTV_1⁆xE_2⁅xBOP⁅≡⁆xPG❴❵•xTV_2⁅L/p/xTV_2⁆•xTV_3⁅L/q/xTV_3⁆xPD❴❵❵/xE_2⁆xPD❴❵❵/xE_3⁆xPD❴❵❵/xE_4⁆/xSQ_0⌉◄≺

Cette forme respecte le premier niveau des contraintes que nous nous sommes donné pour notre *langage pivot, LO_p,* et le filtrage de la construction d'une *inscription*.

6.3.2 La GO_i d'une partie plus fine de la procédure définitoire

Maintenant, nous pouvons passer la *thèse* par un filtre un peu plus fin pour tester d'éventuelles redondances interdites, la validité des relations entre *lieurs* et *variables liées.* C'est la responsabilité de la GO_i, PROCDEF, que nous présentons en version complète dans l'annexe *Procédure définitoire, p. 217*.

A part les tests initiaux, des *rre* 6 à 13, cette GO_i commence par vérifier si aucun *terme variable* ou *foncteur* ne sont répétés dans le *quantificateur*. Nous nous donnons une *cs*, résultante de la GO_i, PREANALYSE, mais avec une erreur :

cs : ⌊**pp**⌋⌈≡(μ(pq)≡(p≡(pq)))⌉⦃T1⦄

```
PREANALYSE, 1, 78, 1:
➢⦃T1⦄⌊pp⌋⌈≡(μ(pq)≡(p≡(pq)))⌉▶xQ_0⌊•xTVQ⦃p⦄•xTVQ⦃p⦄⁄xQ_0⌋xSQ_0⌈xE_4⦃xBOP⦄≡⦃xPG⦅(⦆xE_1⦃
xTCAC⦃μ⦄xCTX_1⦅xPG⦅(⦆•xTV_0⦃D/p⁄xTV_0⦄•xTV_0⦃D/q⁄xTV_0⦄xPD⦅)⦆⁄xCTX_1⦆⁄xE_1⦄xE_3⦃xBO
P⦄≡⦃xPG⦅(⦆•xTV_1⦃L/p⁄xTV_1⦄xE_2⦃xBOP⦄≡⦃xPG⦅(⦆•xTV_2⦃L/p⁄xTV_2⦄•xTV_3⦃L/q⁄xTV_3⦄xPD⦅)⦆
⁄xE_2⦄xPD⦅)⦆⁄xE_3⦄xPD⦅)⦆⁄xE_4⦄⁄xSQ_0⌉◀≺

PROCDEF, 1, 8, 1:
ERREUR :⦃T1⦄⌊pp⌋⌈≡(μ(pq)≡(p≡(pq)))⌉▶mark_xQ_0⌊●→•xTVQ_mark⦃_erreur_p⦄•xTVQ⦃_erreur_p⦄←●
⁄mark_xQ_0⌋xSQ_0⌈xE_4⦃xBOP⦄≡⦃xPG⦅(⦆xE_1⦃xTCAC⦃μ⦄xCTX_1⦅xPG⦅(⦆•xTV_0⦃D/p⁄xTV_0⦄
•xTV_0⦃D/q⁄xTV_0⦄xPD⦅)⦆⁄xCTX_1⦆⁄xE_1⦄xE_3⦃xBOP⦄≡⦃xPG⦅(⦆•xTV_1⦃L/p⁄xTV_1⦄
xE_2⦃xBOP⦄≡⦃xPG⦅(⦆•xTV_2⦃L/p⁄xTV_2⦄•xTV_3⦃L/q⁄xTV_3⦄xPD⦅)⦆⁄xE_2⦄xPD⦅)⦆⁄xE_3⦄xPD⦅)⦆
⁄xE_4⦄⁄xSQ_0⌉◀≺/mark1, trouvé une répétition de termes variables dans un quantificateur !
```

Dans les *rre* 21 à 23, on notera l'utilisation des marqueurs pour le traitement d'erreurs. Nous vérifions la conformité avec les *règles d'inférences de définition* qui n'acceptent pas les *contextes* vides. Les *rre* 34 à 50 traitent de la relation entre *lieurs* du *quantificateur* et *variables liées* du *definiendum*.

La *rre* 43 illustre la mise en œuvre des *variables isoformes*. La *rre* 45, quant à elle montre un exemple de traitement d'erreurs filtrant une malformation possible de l'*inscription*. Nous trouvons aussi un branchement conditionnel à la *rre* 47, qui permet de traiter le cas d'un *definiendum* construit avec plusieurs *contextes*. Soumettons, une *cs* à cette partie de la GO_i en introduisant une erreur :

cs : ⌊pq⌋⌈≡(μ(q**r**)≡(p≡(pq)))⌉⦃T1⦄

```
PREANALYSE, 1, 78, 1:
➢⦃T1⦄⌊pq⌋⌈≡(μ(qr)≡(p≡(pq)))⌉▶xQ_0⌊•xTVQ⦃p⦄•xTVQ⦃q⦄⁄xQ_0⌋xSQ_0⌈xE_4⦃xBOP⦄≡⦃xPG⦅(⦆xE_1⦃
xTCAC⦃μ⦄xCTX_1⦅xPG⦅(⦆•xTV_0⦃D/q⁄xTV_0⦄•xTV_0⦃D/r⁄xTV_0⦄xPD⦅)⦆⁄xCTX_1⦆⁄xE_1⦄xE_3⦃xBOP
⦄≡⦃xPG⦅(⦆•xTV_1⦃L/p⁄xTV_1⦄xE_2⦃xBOP⦄≡⦃xPG⦅(⦆•xTV_2⦃L/p⁄xTV_2⦄•xTV_3⦃L/q⁄xTV_3⦄xPD⦅)⦆⁄x
E_2⦄xPD⦅)⦆⁄xE_3⦄xPD⦅)⦆⁄xE_4⦄⁄xSQ_0⌉◀≺

PROCDEF, 1, 45, 1: ERREUR :
➢⦃T1⦄⌊pq⌋⌈≡(μ(qr)≡(p≡(pq)))⌉▶xQ_0⌊○→xTVQ⦃p⦄xTVQ⦃q⦄←○⁄xQ_0⌋xSQ_0⌈xE_4⦃xBOP⦄≡⦃xPG⦅(⦆x
E_1⦃xTCAC⦃μ⦄xctx_mark_1⦅xPG⦅(⦆•xTV_0⦃D/q⁄xTV_0⦄●→_erreur_xTC_0⦃D/r⁄xTC_0⦄←●xPD⦅)⦆⁄xctx
_mark_1⦆⁄xE_1⦄xE_3⦃xBOP⦄≡⦃xPG⦅(⦆•xTV_1⦃L/p⁄xTV_1⦄xE_2⦃xBOP⦄≡⦃xPG⦅(⦆•xTV_2⦃L/p⁄xTV_2⦄
•xTV_3⦃L/q⁄xTV_3⦄xPD⦅)⦆⁄xE_2⦄xPD⦅)⦆⁄xE_3⦄xPD⦅)⦆⁄xE_4⦄⁄xSQ_0⌉◀≺/these/mark7, trouvé un
terme constant dans le Definiendum !
```

Dans le bloc qui suit, de la *rre* 51 à 56, nous vérifions qu'aucun *signe* n'est répété dans le *definiendum*. Testons cette partie en soumettant une *cs* contenant une répétition de *termes variables* dans le *definiendum* :

cs : ⌊pq⌋⌈≡(μ(**pp**)≡(p≡(pq)))⌉⦃T1⦄

```
PREANALYSE, 1, 78, 1:
➢⦃T1⦄⌊pq⌋⌈≡(μ(pp)≡(p≡(pq)))⌉▶xQ_0⌊•xTVQ⦃p⦄•xTVQ⦃q⦄⁄xQ_0⌋xSQ_0⌈xE_4⦃xBOP⦄≡⦃xPG⦅(⦆xE_1⦃
xTCAC⦃μ⦄xCTX_1⦅xPG⦅(⦆•xTV_0⦃D/p⁄xTV_0⦄•xTV_0⦃D/p⁄xTV_0⦄xPD⦅)⦆⁄xCTX_1⦆⁄xE_1⦄xE_3⦃xBOP
⦄≡⦃xPG⦅(⦆•xTV_1⦃L/p⁄xTV_1⦄xE_2⦃xBOP⦄≡⦃xPG⦅(⦆•xTV_2⦃L/p⁄xTV_2⦄•xTV_3⦃L/q⁄xTV_3⦄xPD⦅)⦆⁄x
E_2⦄xPD⦅)⦆⁄xE_3⦄xPD⦅)⦆⁄xE_4⦄⁄xSQ_0⌉◀≺

PROCDEF, 1, 55, 1: ERREUR :
➢⦃T1⦄⌊pq⌋⌈≡(μ(pp)≡(p≡(pq)))⌉▶xQ_0⌊xTVQ⦃p⦄xTVQ⦃q⦄⁄xQ_0⌋xSQ_0⌈xE_4⦃xBOP⦄≡⦃xPG⦅(⦆xE_1⦃x
TCAC⦃μ⦄●→xCTX_1⦅xPG⦅(⦆•xTV_0⦃_erreur_D/p⁄xTV_0⦄•xTV_0⦃_erreur_D/p⁄xTV_0⦄xPD⦅)⦆⁄xCTX_1⦆
←●⁄xE_1⦄xE_3⦃xBOP⦄≡⦃xPG⦅(⦆•xTV_1⦃L/p⁄xTV_1⦄xE_2⦃xBOP⦄≡⦃xPG⦅(⦆•xTV_2⦃L/p⁄xTV_2⦄•xTV_
3⦃L/q⁄xTV_3⦄xPD⦅)⦆⁄xE_2⦄xPD⦅)⦆⁄xE_3⦄xPD⦅)⦆⁄xE_4⦄⁄xSQ_0⌉◀≺/these/mark4, trouvé des termes
(D/p) répétés dans le Definiendum !
```

La section de la *rre* 57 à 83 vérifie que, dans le *sous-quantificateur*, tous les *termes variables* sont liés par un des *quantificateurs* associés ou dominant. Nous avons apporté un maximum de commentaires dans la GO_i pour décrire ce modèle de programmation. Le schéma de programmation est proche d'un traitement procédural de trois itérations imbriquées, à savoir :

Tant que nous n'avons pas traité tous les *sous-quantificateurs*, faire…

> Tant que nous n'avons pas traité tous les *termes variables* du *sous-quantificateur*, faire…
>
> > Tant que nous n'avons pas traité tous les *quantificateurs* pour trouver celui qui est associé au *terme variable* traité, faire…

Le lecteur notera au passage que nous introduisons dans le préfixe de tous les *termes variables* analysés un numéro de « liaison » qui se réfère au *quantificateur* associé.

Voyons un exemple avec une *cs* à laquelle nous avons introduit comme erreur, une *variable libre* :

C_s : ⌊pq⌋⌈≡(μ(pq)≡(p≡(rq)))⌉⦗T1⦘

PREANALYSE, 1, 78, 1:
➢⦗T1⦘⌊pq⌋⌈≡(μ(qp)≡(p≡(rq)))⌉▶xQ_0⌊•xTVQ⦗p⦘•xTVQ⦗q⦘⁄xQ_0⌋xSQ_0⌈xE_4⦗xBOP⦘≡⦗xPG⦗❩xE_1⦗xTCAC⦗μ⦘xCTX_1⦗xPG⦗❩•xTV_0⦗D/q⁄xTV_0⦘•xTV_0⦗D/p⁄xTV_0⦘xPD⦗❩⦘⁄xCTX_1⦘⁄xE_1⦘xE_3⦗xBOP⦘≡⦗xPG⦗❩•xTV_1⦗L/p⁄xTV_1⦘xE_2⦗xBOP⦘≡⦗xPG⦗❩**•xTV_2⦗L/r⁄xTV_2⦘**•xTV_3⦗L/q⁄xTV_3⦘xPD⦗❩⦘⁄xE_2⦘xPD⦗❩⦘⁄xE_3⦘xPD⦗❩⦘⁄xE_4⦘⁄xSQ_0⌉◀≺

PROCDEF, 1, 119, 1: **ERREUR :**
➢⦗T1⦘⌊pq⌋⌈≡(μ(qp)≡(p≡(rq)))⌉▶xQ_0⌊xTVQ⦗p⦘xTVQ⦗q⦘⁄xQ_0⌋xSQ_0⌈xE_4⦗xBOP⦘≡⦗xPG⦗❩xE_1⦗xTCAC⦗μ⦘xCTX_1⦗xPG⦗❩xTV_⦗0⦘_0⦗D/p⁄xTV_0⦘xTV_⦗0⦘_0⦗D/q⁄xTV_0⦘xPD⦗❩⦘⁄xCTX_1⦘⁄xE_1⦘xE_3⦗xBOP⦘≡⦗xPG⦗❩xTV_⦗0⦘_1⦗L/p⁄xTV_1⦘xE_2⦗xBOP⦘≡⦗xPG⦗❩**•xTV_2⦗L/r⁄xTV_2⦘**•xTV_3⦗L/q⁄xTV_3⦘xPD⦗❩⦘⁄xE_2⦘xPD⦗❩⦘⁄xE_3⦘xPD⦗❩⦘⁄xE_4⦘⇦**_erreur_**⊂⁄xSQ_0⌉◀≺, **trouvé un curseur parasite : ⊂ !**

Même si la GO_i se comporte correctement, nous pourrions encore affiner le traitement d'erreurs. Si le traitement se termine sans erreur, la *cr* est de la forme escomptée par le *langage pivot, LO_p,* et pourra servir à toutes les autres GO_i. Comme nous l'avons déjà mentionné, la *forme analysée* issue du *langage pivot, LO_p,* est illisible. Toutefois, ce que nous avons survolé montre que nos GO_i fonctionnent correctement et répondent à une bonne partie de notre cahier des charges.

6.4 Le dépouillement des inscriptions analysées en version contextuelle

Comme le format *langage pivot* est peu lisible, il est nécessaire de retrouver à partir d'une *inscription* en *forme analysée* sa version brute (initiale). C'est ce que produit la GO_i : DEPX. Le lecteur trouvera sa version complète dans l'annexe *DEPX, p. 225*. Pour la tester, nous utilisons l'enchaînement de GO_i qui met au format *langage pivot*, une *inscription* qui doit être dépouillée, que l'on trouve dans l'annexe *Enchaînement de GOi de tests pour DEPX, p. 224*. Vérifions par l'exemple, le bon fonctionnement de cette GO_i. Nous nous donnons comme *cs* : l'*axiome* 1 de la *protothétique* :

cs : ⌊pqr⌋⌈≡(≡(≡(pr)≡(qp))≡(rq))⌉

TESTDEPX, 1, 1, 1: ➢**⌊pqr⌋⌈≡(≡(≡(pr)≡(qp))≡(rq))⌉**▶⌊pqr⌋⌈≡(≡(≡(pr)≡(qp))≡(rq))⌉◀≺

PREANALYSE, 1, 78, 1:
➢⌊pqr⌋⌈≡(≡(≡(pr)≡(qp))≡(rq))⌉**▶xQ_0⌊•xTVQ⦗p⦘•xTVQ⦗p⦘•xTVQ⦗q⦘⁄xQ_0⌋xSQ_0⌈xE_5⦗xBOP⦘≡⦗xPG⦗❩xE_4⦗xBOP⦘≡⦗xPG⦗❩xE_1⦗xBOP⦘≡⦗xPG⦗❩•xTV_1⦗L/p⁄xTV_1⦘•xTV_2⦗L/r⁄xTV_2⦘xPD⦗❩⦘⁄xE_1⦘xE_2⦗xBOP⦘≡⦗xPG⦗❩•xTV_3⦗L/q⁄xTV_3⦘•xTV_4⦗L/p⁄xTV_4⦘xPD⦗❩⦘⁄xE_2⦘xPD⦗❩⦘⁄xE_4⦘xE_3⦗xBOP⦘≡⦗xPG⦗❩•xTV_5⦗L/r⁄xTV_5⦘•xTV_6⦗L/q⁄xTV_6⦘xPD⦗❩⦘⁄xE_3⦘xPD⦗❩⦘⁄xE_5⦘⁄xSQ_0⌉◀≺**

PROCDEF, 1, 109, 1:
➢⌊pqr⌋⌈≡(≡(≡(pr)≡(qp))≡(rq))⌉▶xQ_0⌊xTVQ⦗p⦘xTVQ⦗q⦘xTVQ⦗r⦘⁄xQ_0⌋xSQ_0⌈xE_5⦗xBOP⦘≡⦗xPG⦗❩xE_4⦗xBOP⦘≡⦗xPG⦗❩xE_1⦗xBOP⦘≡⦗xPG⦗❩xTV_⦗0⦘_1⦗L/p⁄xTV_1⦘xTV_⦗0⦘_2⦗L/r⁄xTV_2⦘xPD⦗❩⦘⁄xE_1⦘xE_2⦗xBOP⦘≡⦗xPG⦗❩xTV_⦗0⦘_3⦗L/q⁄xTV_3⦘xTV_⦗0⦘_4⦗L/p⁄xTV_4⦘xPD⦗❩⦘⁄xE_2⦘xPD⦗❩⦘⁄xE_4⦘xE_3⦗xBOP⦘≡⦗xPG⦗❩xTV_⦗0⦘_5⦗L/r⁄xTV_5⦘xTV_⦗0⦘_6⦗L/q⁄xTV_6⦘xPD⦗❩⦘⁄xE_3⦘xPD⦗❩⦘⁄xE_5⦘⁄xSQ_0⌉◀≺

DEPX, 1, 23, 1:
➢⌊pqr⌋⌈≡(≡(≡(pr)≡(qp))≡(rq))⌉▶xQ_0⌊xTVQ⦗p⦘xTVQ⦗q⦘xTVQ⦗r⦘⁄xQ_0⌋xSQ_0⌈xE_5⦗xBOP⦘≡⦗xPG⦗❩xE_4⦗xBOP⦘≡⦗xPG⦗❩xE_1⦗xBOP⦘≡⦗xPG⦗❩xTV_⦗0⦘_1⦗L/p⁄xTV_1⦘xTV_⦗0⦘_2⦗L/r⁄xTV_2⦘xPD⦗❩⦘⁄xE_1⦘xE_2⦗xBOP⦘≡⦗xPG⦗❩xTV_⦗0⦘_3⦗L/q⁄xTV_3⦘xTV_⦗0⦘_4⦗L/p⁄xTV_4⦘xPD⦗❩⦘⁄xE_2⦘xPD⦗❩⦘⁄xE_4⦘xE_3⦗xBOP⦘≡⦗xPG⦗❩xTV_⦗0⦘_5⦗L/r⁄xTV_5⦘xTV_⦗0⦘_6⦗L/q⁄xTV_6⦘xPD⦗❩⦘⁄xE_3⦘xPD⦗❩⦘⁄xE_5⦘⁄xSQ_0⌉◀≺▷**⌊pqr⌋⌈≡(≡(≡(pr)≡(qp))≡(rq))⌉**◁

Nous retrouvons nos différentes zones : l'*inscription* brute préservée au début de la *chaîne*, la conversion en *forme analysée* (*langage pivot*) entre ► et ◄ et l'*inscription* brute dépouillée ▷⌊pqr⌋⌈≡(≡(≡(pr)≡(qp))≡(rq))⌉◁.

6.5 Détermination des catégories syntaxico-sémantiques

Une *catégorie syntaxico-sémantique* est une sorte de « métadescription » construite à partir de l'analyse d'une *inscription* introduite dans le *système*. Elle se détermine à partir de la *fonction logique* du *definiendum* et plus précisément à partir du *terme constant* et de son ou de ses *contextes*. Pour des raisons de commodité, le résultat de cette détermination est inscrit dans la *bibliothèque* en parallèle avec l'introduction de la nouvelle *inscription* obtenue à partir d'une des *règles d'inférences de définition.* Elle se présente en trois volets :

1) la *catégorie syntaxico-sémantique*, que nous détaillons ci-après ;
2) le *terme constant*, un *signe* participant à la construction de la *fonction logique* du *definiendum*, introduit dans le *système* et qui respecte la *procédure définitoire* ;
3) le *contexte*, introduit en même temps que le *terme constant* auquel il est associé.

On trouve la *définition inductive* des *catégories syntaxico-sémantiques* (de la *protothétique*) à l'alinéa *F 4.8, p. 42* et suivants.

Notre GO_i, CATEGORIE, est destinée à déterminer la *catégorie syntaxico-sémantique* d'une *inscription* de type *thèse-définition*. Ce petit exemple d'une trentaine de *rre* ne traite que les *catégories syntaxico-sémantiques* concernant les *termes constants* ; il ne traite pas les *fonctions propositionnelles ;* traitement additionnel qui serait similaire à celui que nous donnons dans l'annexe *CATEGORIE, p. 227*. La GO_i CATEGORIE va procéder sur la base d'un schéma comme celui qui est donné ici à titre d'exemple :

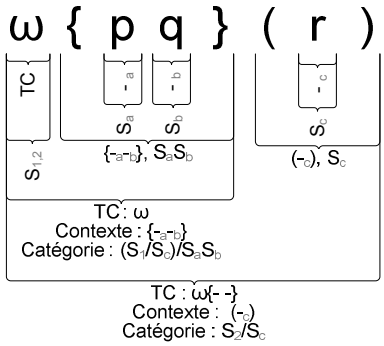


Figure 53. *Catégories syntaxico-sémantiques* de la thèse D11, (S/S)/SS et S/S

Nous introduisons ces nouvelles *catégories syntaxico-sémantiques* dans le *système* (sachant que la *catégorie* S/S est une sous-*catégorie* introduite lors la définition du *terme constant* ω):

Catégories	Constantes	Contextes
S/S	ω{- -}	(-)[109]
(S/S)/SS	ω	{- -}

La GO_i, après avoir cerné la *fonction logique* du *definiendum* et le *terme constant* va analyser les *contextes* et le nombre d'arguments qu'ils peuvent contenir. La grammaire a été construite de façon à démarrer son analyse du *contexte* le plus « profond » à celui qui se trouve en « surface » ; dans notre exemple, en partant de : {-a-b} et se terminer à (-c).

Nous créons une zone de paramètres en fin de *chaîne* et que nous mettons petit à petit à jour. Le but est de retourner, ici, une série d'informations de forme :

[109] Introduction de la sous-catégorie S/S lors de la définition de ω.

/CAT=⟦1:cat⟩S/SS⟨cst⟩≡⟨ctx⟩(--)⟨⟧⟧

Pour tester cette GO_i, nous utilisons TESTCAT qui enchaîne l'analyse nécessaire à créer la structure de format *langage pivot*. Donnons-nous un exemple, avec plusieurs *contextes*, pour observer le comportement de cette GO_i :

cs : ⌊qpr⌋⌈≡(ω[p]\q/(r)≡(~(≡(pq))r))⌉

CATEGORIE, 1, 35, 1:
➢⌊qpr⌋⌈≡(ω[p]\q/(r)≡(~(≡(pq))r))⌉►xQ_0⌊xTVQ⟦p⟧xTVQ⟦q⟧xTVQ⟦r⟧/xQ_0⌋xSQ_0⌈xE_5⟦xBOP⟩≡⟨xPG⟦(⟧xE_1⟦xTCAC⟦ω⟧xCTX_1⟦xPG⟦[⟧xTV_⟦0⟧_0⟦D/p/xTV_0⟧xPD⟦]⟧/xCTX_1⟧xCTX_2⟦xPG⟦\⟧xTV_⟦0⟧_0⟦D/q/xTV_0⟧xPD⟦/⟧/xCTX_2⟧xCTX_3⟦xPG⟦(⟧xTV_⟦0⟧_0⟦D/r/xTV_0⟧xPD⟦)⟧/xCTX_3⟧/xE_1⟧xE_4⟦xBOP⟩≡⟨xPG⟦(⟧xE_3⟦xMOP⟩~⟨xPG⟦(⟧xE_2⟦xBOP⟩≡⟨xPG⟦(⟧xTV_⟦0⟧_1⟦L/p/xTV_1⟧xTV_⟦0⟧_2⟦L/q/xTV_2⟧xPD⟦)⟧/xE_2⟧xPD⟦)⟧/xE_3⟧xTV_⟦0⟧_3⟦L/r/xTV_3⟧xPD⟦)⟧/xE_4⟧xPD⟦)⟧/xE_5⟧/xSQ_0⌉◄
/CAT=⟦1:cat⟩((S/S)/S)/S⟨cst⟩ω⟨ctx⟩[-]⟨⟧⟧⟦2:cat⟩(S/S)/S⟨cst⟩ω[-]⟨ctx⟩\-/⟨⟧
⟦3:cat⟩S/S⟨cst⟩ω[-]\-/⟨ctx⟩(-)⟨⟧⟧≺

Nous obtenons bien les trois *catégories syntaxico-sémantiques* définies par cette *thèse-définition*. Prenons encore un exemple basé sur la *catégorie* des *noms* :

cs : ⌊a⌋⌈≡(!{a}~(⌊A⌋⌈~(ε{Aa})⌉))⌉

CATEGORIE, 1, 35, 1:
➢⌊a⌋⌈≡(!{a}~(⌊A⌋⌈~(ε{Aa})⌉))⌉►xQ_0⌊xTVQ⟦a⟧/xQ_0⌋xSQ_0⌈xE_6⟦xBOP⟩≡⟨xPG⟦(⟧xE_1⟦xTCAC⟦!⟧xCTX_1⟦xPG⟦{⟧xTV_⟦0⟧_0⟦D/a/xTV_0⟧xPD⟦}⟧/xCTX_1⟧/xE_1⟧xE_5⟦xMOP⟩~⟨xPG⟦(⟧xE_4⟦xQ_1⌊xTVQ⟦A⟧/xQ_1⌋xSQ_1⌈xE_3⟦xMOP⟩~⟨xPG⟦(⟧xE_2⟦xBOP⟩ε⟨xPG⟦{⟧xTV_⟦1⟧_1⟦L/A/xTV_1⟧xTV_⟦0⟧_2⟦L/a/xTV_2⟧xPD⟦}⟧/xE_2⟧xPD⟦)⟧/xE_3⟧/xSQ_1⌉/xE_4⟧xPD⟦)⟧/xE_5⟧xPD⟦)⟧/xE_6⟧/xSQ_0⌉◄**/CAT**
=⟦1:cat⟩S/N⟨cst⟩!⟨ctx⟩{-}⟨⟧⟧≺

Nous disposons maintenant d'une GO_i permettant de déterminer une *catégorie syntaxico-sémantique* qui montre concrètement comment nous répondons à notre cahier des charges.

6.6 Les calculs propositionnels

Le calcul propositionnel des *expressions* construites avec des *termes constants* de la *protothétique* (de la *catégorie* des propositions) ne pose pas de problème et nous le montrons avec notre exemple de GO_i que l'on trouve dans l'annexe Calculs propositionnels, *p. 231*. Le calcul de l'*epsilon* de l'*ontologie* est plus délicat. Même si l'*axiome* de l'*ontologie* en définit toutes les propriétés, il n'en reste pas moins que sans l'interprétation des *variables de noms* d'une *expression* nous ne pouvons pas aboutir à un calcul décidable. Même si nous n'avons pas apporté à notre *RE* et son *LRE*, les éléments nécessaires à des calculs nécessitant des interprétations un peu plus difficiles, cela ne met pas en cause la faisabilité du *LRE*.

La GO_i, CALCULUS, qui traite des calculs spécifiques aux *expressions* formées de *termes constants*, a nécessité l'implantation dans le *RE*, des *fonctions* spécifiques de *md*, dont on trouve la description dans les sections *5.4.5.9 La fonction TABLEVERITE du LRE, p. 111* et *5.4.5.10 La fonction CALCULVERITE du LRE, p. 112*. Analysons cette GO_i qui diffère des autres par le fait que, même si elle a besoin de passer par le dépistage d'éléments morphosyntaxiques, n'est destinée qu'à faire du calcul. Comme nous l'avons vu dans ce qui précède, nous contrôlons par les *rre* 1 à 5 la bonne formation de la *chaîne*, *cs*. Ensuite, dans les *rre* 6 à 8 nous éliminons les cas qui traitent de la valeur par définition des *constantes propositionnelles*.

Le bloc de *rre* 10 à 14 dépiste à quel type d'*inscription* nous avons à faire (*axiome*, *thèse-définition* ou *fonction logique*). Quant à la longue section, qui montre que s'il est possible de faire de l'arithmétique avec le *LRE*, cela nécessite néanmoins beaucoup de *rre*, à savoir celles qui vont de 15 à 33. Cette partie de la GO_i ne fait que de calculer la combinatoire et que d'affecter dans le préfixe d'*encapsulation* des *termes* une table de vérité unique. Le reste de la GO_i fouille les *expressions* du niveau le plus « profond » à la « surface » pour calculer petit à petit la table de vérité de chacune d'entre elles. Elle s'arrête quand elle a calculé la dernière valeur, celle du *sous-quantificateur* dominant. Cette GO_i offre une sorte de schéma standard que

l'on retrouve dans la majorité de nos GO_i. Elle montre notre façon de découper en modules (de GO_i) nos traitements. En voici le schéma, en représentation graphique :

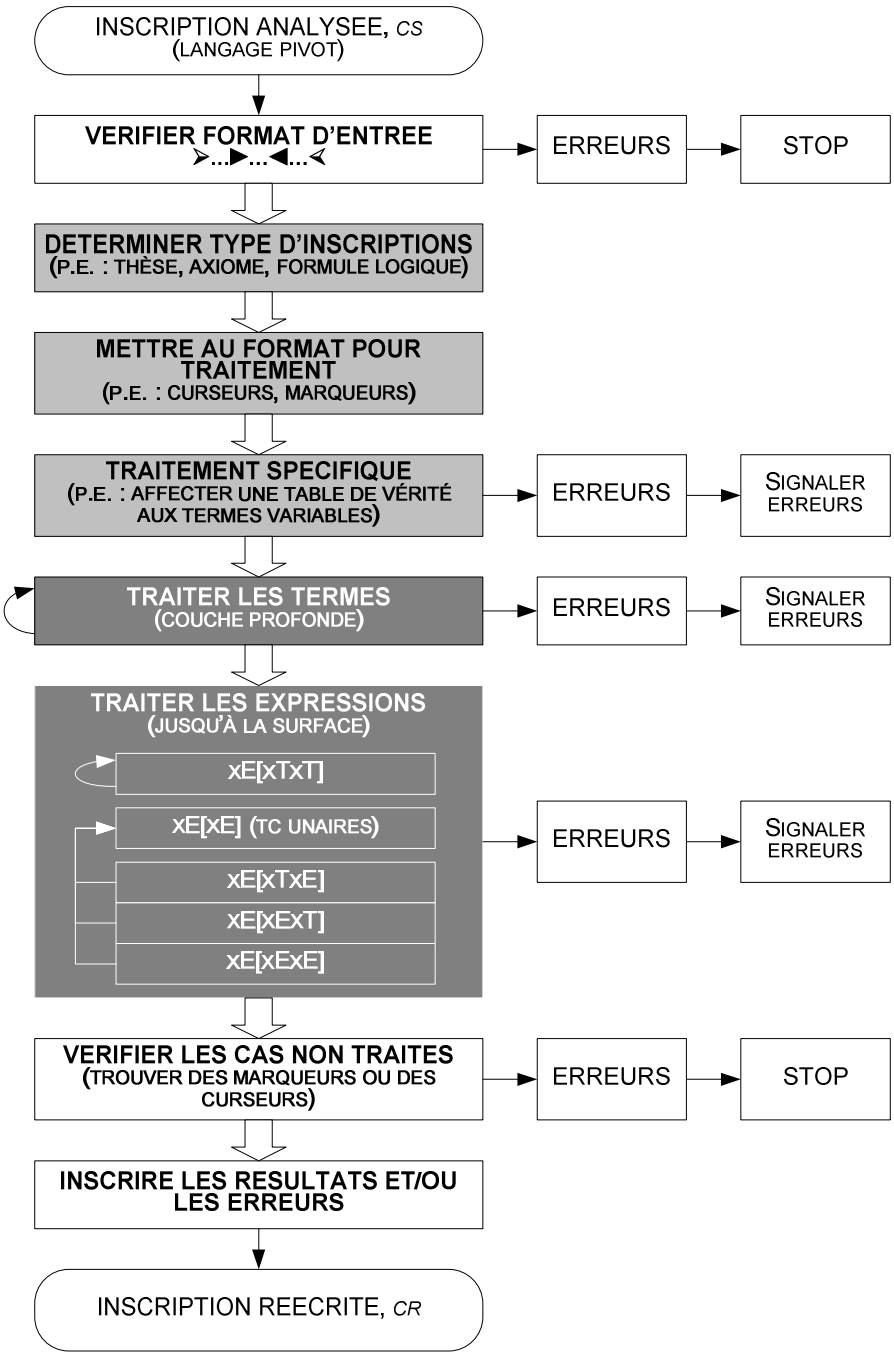


Figure 54. Division en blocs d'une GO_i

Nous pouvons y observer les motifs de traitement des *termes* et des *expressions*[110] qui reviennent dans la plupart de nos GO_i.

Prenons un exemple de l'exécution de la GO_i, CALCULUS, pour calculer la table du *definiens* de la *thèse-définition* de la conditionnelle :

cs : ⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉

[110] *Terme* et *expression* sont pris dans le sens que nous leur avons donné dans le *langage pivot* LO_p.

Nous donnons ici quelques pas illustratifs des résultats de tout l'enchaînement du traitement de la GO_i qui teste CALCULUS (annexe *GOi de tests pour CALCULUS, p. 231*). A la fin de l'exécution de PROCDEF, voici le format en *langage pivot* de l'*inscription* :

```
PROCDEF, 1, 109,1:
➢⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉►xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈xE_5[xBOP}≡{xPG[(]xE_1[xTC
AC{⊃}xCTX_1[xPG[(]xTV_[0]_0[D/p/xTV_0]xTV_[0]_0[D/q/xTV_0]xPD[)]/xCTX_1]/xE_1]xE_4[
xMOP}~{xPG[(]xE_3[xBOP}∧{xPG[(]xTV_[0]_1[L/p/xTV_1]xE_2[xMOP}~{xPG[(]xTV_[0]_2[L/q
/xTV_2]xPD[)]/xE_2]xPD[)]/xE_3]xPD[)]/xE_4]xPD[)]/xE_5]/xSQ_0⌉◄≺
```

Arrêtons-nous aux étapes intéressantes de CALCULUS en prenant d'abord le résultat de l'affectation des tables de vérité aux *termes variables* :

```
CALCULUS, 1, 33,1:
➢⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉►xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈xE_5[xBOP}≡{xPG[(]xE_1[xTC
AC{⊃}xCTX_1[xPG[(]xTV_[0]_0[D/p/xTV_0]xTV_[0]_0[D/q/xTV_0]xPD[)]/xCTX_1]/xE_1]xE_4[
xMOP}~{xPG[(]xE_3[xBOP}∧{xPG[(]xTV_[0]_1[L/p/xTV_1]xE_2[xMOP}~{xPG[(]xTV_[0]_2[L/q
/xTV_2]xPD[)]/xE_2]xPD[)]/xE_3]xPD[)]/xE_4]xPD[)]/xE_5]/xSQ_0⌉◄/CAT=}[1:cat}S/SS{cst}⊃
{ctx}(--){]}≺
▷xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈xE_5[xBOP}≡{xPG[(]xE_1[xTCAC{⊃}xCTX_1[xPG[(]xTV_
[0]_0[D/p/xTV_0]xTV_[0]_0[D/q/xTV_0]xPD[)]/xCTX_1]/xE_1]xE_4[xMOP}~{xPG[(]xE_3[xBOP
}∧{xPG[(]xTV_[0]_1[1,1,0,0]L/p/xTV_1]xE_2[xMOP}~{xPG[(]xTV_[0]_2[1,0,1,0]L/q/xTV_2]xPD
[)]/xE_2]xPD[)]/xE_3]xPD[)]/xE_4]xPD[)]/xE_5]/xSQ_0⌉◁/t/Puis={4}/mark0
```

Prenons ensuite un extrait du calcul de l'*expression* ~(q) :

```
CALCULUS, 1, 69,1:
➢⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉►xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈xE_5[xBOP}≡{xPG[(]xE_1[xTC
AC{⊃}xCTX_1[xPG[(]xTV_[0]_0[D/p/xTV_0]xTV_[0]_0[D/q/xTV_0]xPD[)]/xCTX_1]/xE_1]xE_4[
xMOP}~{xPG[(]xE_3[xBOP}∧{xPG[(]xTV_[0]_1[L/p/xTV_1]xE_2[xMOP}~{xPG[(]xTV_[0]_2[L/q
/xTV_2]xPD[)]/xE_2]xPD[)]/xE_3]xPD[)]/xE_4]xPD[)]/xE_5]/xSQ_0⌉◄/CAT=}[1:cat}S/SS{cst}⊃
{ctx}(--){]}≺
▷xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈xE_5[xBOP}≡{xPG[(]xE_DUM_1[xTCAC{⊃}xCTX_1[xPG[(
]xTV_[0]_0[D/p/xTV_0]xTV_[0]_0[D/q/xTV_0]xPD[)]/xCTX_1]/xE_1]xE_4[xMOP}~{xPG[(]xE_3
[xBOP}∧{xPG[(]xTV_[0]_1[1,1,0,0]L/p/xTV_1]xE_calc_2[0,1,0,1][xMOP_calc}~{xPG[(]xTV_[0]_
2[L/q/xTV_2]xPD[)]/xE_2]xPD[)]/xE_3]xPD[)]/xE_4]xPD[)]/xE_5]/xSQ_0⌉◁/t/mark0
```

Voici, maintenant, l'étape après le calcul de l'*expression* du *definiens* :

```
CALCULUS, 1, 99,1:
➢⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉►xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈xE_5[xBOP}≡{xPG[(]xE_1[xTC
AC{⊃}xCTX_1[xPG[(]xTV_[0]_0[D/p/xTV_0]xTV_[0]_0[D/q/xTV_0]xPD[)]/xCTX_1]/xE_1]xE_4[
xMOP}~{xPG[(]xE_3[xBOP}∧{xPG[(]xTV_[0]_1[L/p/xTV_1]xE_2[xMOP}~{xPG[(]xTV_[0]_2[L/q
/xTV_2]xPD[)]/xE_2]xPD[)]/xE_3]xPD[)]/xE_4]xPD[)]/xE_5]/xSQ_0⌉◄/CAT=}[1:cat}S/SS{cst}⊃
{ctx}(--){]}≺
▷xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈xE_5[1,0,1,1]xBOP}≡{xPG[(]xE_1[xTCAC{⊃}xCTX_1[xP
G[(]xTV_[0]_0[D/p/xTV_0]xTV_[0]_0[D/q/xTV_0]xPD[)]/xCTX_1]/xE_1]xE_4[xMOP}~{xPG[(]x
E_3[xBOP}∧{xPG[(]xTV_[0]_1[L/p/xTV_1]xE_2[xMOP_calc}~{xPG[(]xTV_[0]_2[L/q/xTV_2]xPD[
)]/xE_2]xPD[)]/xE_3]xPD[)]/xE_4]xPD[)]/xE_5]/xSQ_0⌉◁
```

Et, nous présentons, ici, la dernière étape où l'on trouve le résultat inscrit (en gras) :

```
CALCULUS, 1, 110,1:
➢⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉►xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈xE_5[xBOP}≡{xPG[(]xE_1[xTC
AC{⊃}xCTX_1[xPG[(]xTV_[0]_0[D/p/xTV_0]xTV_[0]_0[D/q/xTV_0]xPD[)]/xCTX_1]/xE_1]xE_4[
xMOP}~{xPG[(]xE_3[xBOP}∧{xPG[(]xTV_[0]_1[L/p/xTV_1]xE_2[xMOP}~{xPG[(]xTV_[0]_2[L/q
/xTV_2]xPD[)]/xE_2]xPD[)]/xE_3]xPD[)]/xE_4]xPD[)]/xE_5]/xSQ_0⌉◄/CAT=}[1:cat}S/SS{cst}⊃
{ctx}(--){]}≺/TABLE={1,0,1,1}≺
```

En conclusion, même si cet exemple de GO_i n'est pas exhaustif, il montre que par améliorations successives, nous pourrions traiter pratiquement tous les cas de calcul propositionnel y compris les *fonctions propositionnelles*. Cela nécessitera l'introduction d'un système de calculs des *matrices de vérités* des *foncteurs* (y compris l'*epsilon* de l'ontologie) (chapitre 7. *Perspectives et développements futurs, p. 175*).

6.7 Les conversions de notations

La conversion d'*inscriptions* formées avec la notation en *version contextuelle* vers la notation en *version catégorielle* est réalisée par la GO_i, CONVERTXC, qui, à partir d'une *inscription* analysée dans le format *langage pivot*, va la convertir en *forme analysée* en *version catégorielle*. Comme pour les autres GO_i, nous en avons une qui enchaîne les traitements nécessaires aux tests de CONVERTXC, ainsi que la grammaire qui dépouille la *version catégorielle* analysée (format *langage pivot*) d'une *inscription*, DEPC ; trois modules que l'on trouve dans l'annexe *Conversions d'inscriptions de la version contextuelle à la version catégorielle, p. 241*.

Cette GO_i suit le modèle structurel de celle du calcul des tables de vérité (section *6.6. Les calculs propositionnels, p. 161*). Voici un exemple du résultat d'un traitement opéré sur le deuxième *axiome* de la *protothétique* :

cs : ⌊qpr⌋⌈≡(≡(p≡(qr))≡(≡(pq)r))⌉

DEPC, 1, 29, 1:
➢⌊qpr⌋⌈≡(≡(p≡(qr))≡(≡(pq)r))⌉▶cQ_0❴(∀cTVQ❴p❵cTVQ❴q❵cTVQ❴r❵)/cQ_0❵cSQ_0❵cE_5❴cPG❴(❵cE_3❴cPG❴(❵cTV_❴0❵_1❴L/p/cTV_1❵cBOP❵≡❮cE_1❴cPG❴(❵cTV_❴0❵_2❴L/q/cTV_2❵cBOP❵≡❮cTV_❴0❵_3❴L/r/cTV_3❵cPD❴)❵/cE_1❵cPD❴)❵/cE_3❵cBOP❵≡❮cE_4❴cPG❴(❵cE_2❴cPG❴(❵cTV_❴0❵_4❴L/p/cTV_4❵cBOP❵≡❮cTV_❴0❵_5❴L/q/cTV_5❵cPD❴)❵/cE_2❵cBOP❵≡❮cTV_❴0❵_6❴L/r/cTV_6❵cPD❴)❵/cE_4❵cPD❴)❵/cE_5❵/cSQ_0❵◀≺▷**(∀pqr)((p≡(q≡r))≡((p≡q)≡r))**◁

Nous n'avons pas jugé nécessaire d'apporter ici la GO_i qui convertit une *inscription* notée en *version catégorielle* vers la *version contextuelle*. Elle serait, en effet, un simple miroir de notre grammaire CONVERTXC. Il reste encore à noter que cette GO_i ne convertit pas la négation du *quantificateur* universel en *quantificateur* existentiel.

6.8 Traitement de la règle d'inférences de détachement

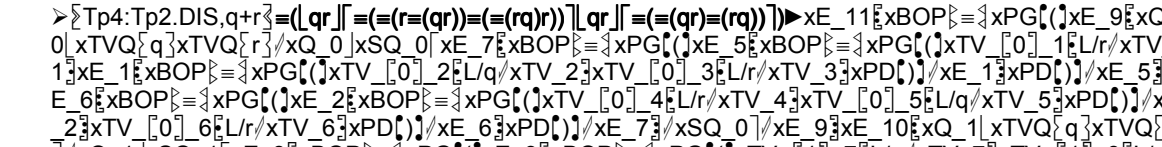
Nous abordons maintenant un sujet qui montre que le *RE* peut être un véritable outil d'aide pour la *déduction* dans le cadre des *SL*. Nous allons traiter la *règle d'inférences de détachement*. Ceci nous conduit à utiliser un corpus comme référence pour opérer les comparaisons nécessaires entre la première équivalence de l'*inscription*, *cs*, et une *inscription* enregistrée dans la *librairie* ou dans le *corpus d'entrée*. Pour ce faire, nous avons créé deux GO_i. La principale, DETACH, va boucler sur le *corpus* de référence, dans notre cas la *librairie* de la GO_i dans laquelle nous avons enregistré des *inscriptions* analysées (en LO_p). La seconde, COMPARE, est appelée par DETACH après chaque recherche d'une nouvelle *inscription* à comparer dans la *librairie*. Nous trouvons ces deux GO_i dans les annexes *GOi de la directive de détachement, p. 251* et *GOi de comparaison, p. 255*.

La section, LIBRAIRIE, de la GO_i DETACH est un peu lourde puisque nous y avons introduit une liste d'*inscriptions* en *forme analysée*, donc longue à cause de la surcharge apportée par les *encapsulations*. D'un point de vue structure, cette grammaire se présente comme une boucle qui parcourt la *librairie* jusqu'à ce qu'elle trouve un schéma de la deuxième équivalence qui soit identique à celle de la *cs*. Même si les *expressions* comparées peuvent être parfaitement équivalentes en version brute, elles ne le seront pas nécessairement en version *langage pivot*, parce que les numéroteurs des préfixes vont être différents. C'est ce qui rend la GO_i appelée, COMPARE, un peu plus complexe qu'une simple comparaison qui pourrait se résumer à une seule *rre*.

Pour illustrer le fonctionnement de ces deux GO_i nous prenons une partie de la démonstration présentée par Denis Miéville (MIEVILLE, 2007, p. 88) dont le but est d'obtenir l'*inscription* T5, résultat du détachement entre T4 et T3. Dans notre exemple, T4 sera soumise comme *cs* et T3 devrait être trouvées dans la *librairie* de la GO_i, DETACH. Nous rappelons que les *chaînes* traitées doivent être en format *langage pivot* et notées en *version contextuelle*. Soit, en format brut :

cs / T4 : ≡(⌊qr⌋⌈≡(≡(r≡(qr))≡(≡(rq)r))⌉⌊qr⌋⌈≡(≡(qr)≡(rq))⌉)

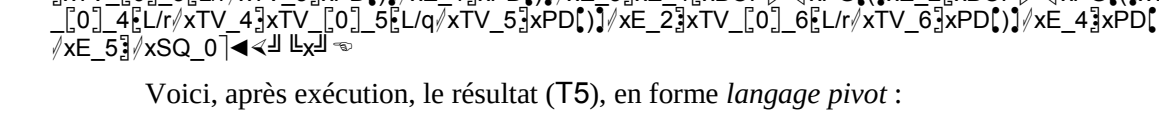
et en version d'entrée en *langage pivot* :



Quant à T3, en version brute :

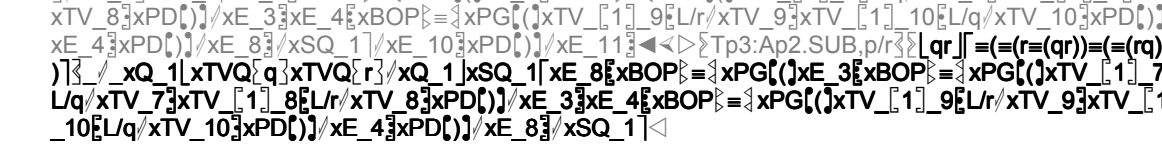
Librairie / T3 : ⌊qr⌋⌈≡(≡(r≡(qr))≡(≡(rq)r))⌉

et en format *langage pivot* dans la *librairie* de la grammaire :



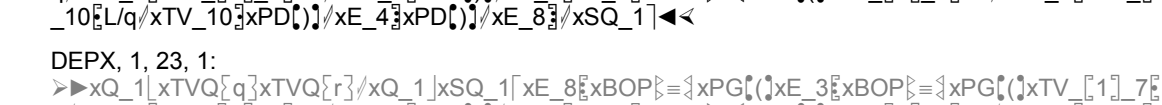
Voici, après exécution, le résultat (T5), en forme *langage pivot* :

DETACH, 1, 26, 2:

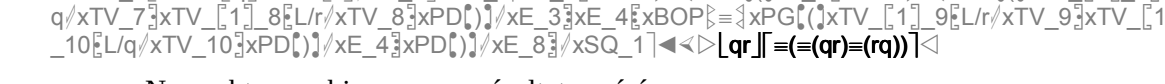


qui, soumis comme nouvelle *cs* à la GO_i de dépouillement, DEPX, nous donne :

cs / T5 :



DEPX, 1, 23, 1:



Nous obtenons bien comme résultat espéré :

cr / T5 : ⌊qr⌋⌈≡(≡(qr)≡(rq))⌉

Cet exemple de GO_i montre des horizons intéressants pour l'aide à la *déduction*. Comme nous n'avons pas de modèle global de stratégie ou de scénario à suivre pour la *déduction* (chapitre 7. *Perspectives et développements futurs, p. 175*) à ce niveau de notre étude, nous stoppons cet exemple de GO_i dès qu'elle trouve une comparaison satisfaisante. Nous pourrions parfaitement imaginer continuer l'exécution et mémoriser toutes les solutions de détachement trouvées pour enchaîner la suite d'une *déduction*. On trouve un exemple de mémorisation de solutions multiples dans la GO_i, DISTRIB, dédiée au traitement de la *règle d'inférences de distribution des quantificateurs* (section *6.9. La règle d'inférences de distribution des* quantificateurs*, p. 166*).

6.9 La règle d'inférences de distribution des quantificateurs

L'exemple de GO_i, DISTRIB, que nous avons développé peut être considéré comme un modèle de structure de traitement qui met en œuvre combinatoire et analyse morphosyntaxique. La GO_i est donnée en version complète[111] dans l'annexe *Traitement de la règle d'inférences de distribution des* quantificateurs*, p. 258*. A sa lecture, on pourra noter que le *LRE* et son *RE* acceptent des *rre* assez complexes comme c'est le cas pour les *rre* 25 et 26.

Nous choisissons une *inscription* simple : ⌊pq⌋⌈≡(≡(pq)≡(qp))⌉

Pour se remémorer la *règle* d'inférences *de distribution des quantificateurs*, on peut se reporter à la section *4.4.6.2 La règle d'inférences de distribution des quantificateurs, p. 49*. Etudions pas à pas cette GO_i. Prenons cette *inscription* que nous soumettons à notre *RE :*

cs : ⌊pq⌋⌈≡(≡(pq)≡(qp))⌉

Comme nous allons traiter toutes les distributions possibles ; nous devons obtenir :

cr : (1) ⌊p⌋⌈≡(⌊q⌋⌈≡(pq)⌉⌊q⌋⌈≡(qp)⌉)⌉

cr : (2) ⌊q⌋⌈≡(⌊p⌋⌈≡(pq)⌉⌊p⌋⌈≡(qp)⌉)⌉

cr : (3) ≡(⌊pq⌋⌈≡(pq)⌉⌊pq⌋⌈≡(qp)⌉)

Evidemment, comme pour les autres GO_i, nous devons soumettre une *cs* dans le format du *langage pivot :*

➢⟦test⟧⌊pq⌋⌈≡(≡(pq)≡(qp))⌉►xQ_0⌊xTVQ⟦p⟧xTVQ⟦q⟧/xQ_0⌋xSQ_0⌈xE_3⟦xBOP⟧≡⟦xPG⟦(⟧xE_1⟦xBOP⟧≡⟦xPG⟦(⟧xTV_⟦0⟧_1⟦L/p/xTV_1⟧xTV_⟦0⟧_2⟦L/q/xTV_2⟧xPD⟦)⟧/xE_1⟧xE_2⟦xBOP⟧≡⟦xPG⟦(⟧xTV_⟦0⟧_3⟦L/q/xTV_3⟧xTV_⟦0⟧_4⟦L/p/xTV_4⟧xPD⟦)⟧/xE_2⟧xPD⟦)⟧/xE_3⟧/xSQ_0⌉◄≺

6.9.1 Vérification du format de la cs et mise en forme analysée

La GO_i part du principe que l'*inscription* soumise est correctement formée. Si ce n'était pas le cas, la première *rre* ne marquerait pas la *chaîne* avec un marqueur (/verifier_gen) qui permettrait la suite du traitement. Cette même *rre*, si elle satisfait au *filtrage de motif*, sans toucher la zone standard d'entrée (➢‹avant›►‹formule›◄‹apres›≺) prépare deux nouvelles zones (▷◊→‹formule›←◊◁➲➲) et trois nouveaux paramètres (/GEN=[]/COMBITABLE=⟦⟧/TV=[x]). Donc, cette petite *rre* cumule trois fonctions (introduction d'un marqueur, définition de zones et préparation de paramètres) ; dont voici le résultat de son exécution sur notre exemple :

DISTRIB, 1, 1,1:
➢⟦test⟧⌊pq⌋⌈≡(≡(pq)≡(qp))⌉►xQ_0⌊xTVQ⟦p⟧xTVQ⟦q⟧/xQ_0⌋xSQ_0⌈xE_3⟦xBOP⟧≡⟦xPG⟦(⟧xE_1⟦xBOP⟧≡⟦xPG⟦(⟧xTV_⟦0⟧_1⟦L/p/xTV_1⟧xTV_⟦0⟧_2⟦L/q/xTV_2⟧xPD⟦)⟧/xE_1⟧xE_2⟦xBOP⟧≡⟦xPG⟦(⟧xTV_⟦0⟧_3⟦L/q/xTV_3⟧xTV_⟦0⟧_4⟦L/p/xTV_4⟧xPD⟦)⟧/xE_2⟧xPD⟦)⟧/xE_3⟧/xSQ_0⌉◄≺▷◊→xQ_0⌊xTVQ⟦p⟧xTVQ⟦q⟧/xQ_0⌋xSQ_0⌈xE_3⟦xBOP⟧≡⟦xPG⟦(⟧xE_1⟦xBOP⟧≡⟦xPG⟦(⟧xTV_⟦0⟧_1⟦L/p/xTV_1⟧xTV_⟦0⟧_2⟦L/q/xTV_2⟧xPD⟦)⟧/xE_1⟧xE_2⟦xBOP⟧≡⟦xPG⟦(⟧xTV_⟦0⟧_3⟦L/q/xTV_3⟧xTV_⟦0⟧_4⟦L/p/xTV_4⟧xPD⟦)⟧/xE_2⟧xPD⟦)⟧/xE_3⟧/xSQ_0⌉←◊◁➲➲/GEN=[]/COMBITABLE=⟦⟧/TV=[x]/verifier_gen

La zone ▷◊→‹formule›←◊◁ est la copie, sur laquelle nous allons travailler. Nous y avons introduit deux curseurs ◊→ et ←◊ qui nous serviront plus loin.

La *rre* 2 sélectionne le corpus dans lequel nous mémoriserons nos diverses distributions résultantes.

6.9.2 Vérification de l'inscription

Il n'y a de sens à traiter la *règle d'inférences de distribution des quantificateurs* que si nous avons à faire à une *généralisation*. Les *rre* 3 et 4 se chargent de cette vérification. Deux cas acceptables peuvent se présenter : la *chaîne* de caractères est une *expression* qui contient

[111] Afin de simplifier sa lecture et malgré les commentaires, nous avons souligné en gras les réécritures principales de chaque *membre de droite*.

une *généralisation* ou elle est directement une *généralisation*. L'exemple que nous traitons, ici, passe ce filtre.

6.9.3 Création d'une table de combinaisons

Tout d'abord, aux *rre* 5 et 6, la GO_i va compter le nombre de *termes variables* à considérer dans le *quantificateur* dominant. C'est ce qui nous permet de sélectionner le nombre possible de combinaisons de distribution que nous pouvons envisager. Le résultat est mémorisé dans un paramètre tot[…] inscrit dans la zone COMBITABLE. Nous nous donnons une table de combinaisons allant des *rre* 7 à 10. Cette table tient compte, au maximum, de quatre *termes variables* quantifiés qui peuvent être distribués. COMBITABLE est construite selon la convention suivante : tot[…] est le nombre de *termes* quantifiés à distribuer ; combi[…] décrit la combinatoire ; nbc[…] indique le nombre d'*expressions* résultantes généré par la combinaison des distributions possibles (dans notre exemple : 3). Sachant que notre *inscription* est constituée de deux *termes variables*, notre GO_i devra sélectionner ultérieurement la table définie dans la *rre* 8. Le paramètre combi[…] est, selon le nombre de *termes* à distribuer, composé de zones : le nombre d'*expressions* résultantes constituées (*) du nombre de *termes* distribués suivi d'un parenthèsage qui décrit comment ces *termes* sont distribués dans les nouvelles sous *généralisations*. L'enveloppe formée de '(' et ')' mentionne, et cela entre '{' et '}', le(s) *terme(s) variable(s)* qui constitue(nt) le nouveau sous *quantificateur*. Comme, dans notre exemple, nous avons précompté le nombre de *variables*, à savoir deux, c'est la ligne de la table de la *rre* 8 qui sera sélectionnée. Vérifions-le :

DISTRIB, 1, 8,1:
➢{test{|pq|[≡(≡(pq)≡(qp))]►xQ_0|xTVQ{p}xTVQ{q}/xQ_0|xSQ_0[xE_3[xBOP}≡{xPG[(]xE_1[xBOP}≡{xPG[(]xTV_[0]_1[L/p/xTV_1]xTV_[0]_2[L/q/xTV_2]xPD[)]]/xE_1]xE_2[xBOP}≡{xPG[(]xTV_[0]_3[L/q/xTV_3]xTV_[0]_4[L/p/xTV_4]xPD[)]]/xE_2]xPD[)]]/xE_3]/xSQ_0]◄≺▷◊→xQ_0|◙→xTVQ{p}xTVQ{q}←◙/xQ_0|xSQ_0[xE_3[xBOP}≡{xPG[(]xE_1[xBOP}≡{xPG[(]xTV_[0]_1[L/p/xTV_1]xTV_[0]_2[L/q/xTV_2]xPD[)]]/xE_1]xE_2[xBOP}≡{xPG[(]xTV_[0]_3[L/q/xTV_3]xTV_[0]_4[L/p/xTV_4]xPD[)]]/xE_2]xPD[)]]/xE_3]/xSQ_0]←◊◁➲⊂/GEN=[0]/**COMBITABLE=[tot[2]combi[2*1=[({1})({2})]1*2=[({1}{2})]]nbc[3]]**/TV=[x]/faire4_table

Ce qui est le cas. Nous allons, évidemment, garder ces informations tout au long du traitement.

6.9.4 Associer la position des termes à la table combinatoire

Nous numérotons les *termes* du *quantificateur* (*rre* 11 à 13) dans la perspective de les associer à la table combinatoire et nous obtenons le résultat :

DISTRIB, 1, 13,1:
➢{test{|pq|[≡(≡(pq)≡(qp))]►xQ_0|xTVQ{p}xTVQ{q}/xQ_0|xSQ_0[xE_3[xBOP}≡{xPG[(]xE_1[xBOP}≡{xPG[(]xTV_[0]_1[L/p/xTV_1]xTV_[0]_2[L/q/xTV_2]xPD[)]]/xE_1]xE_2[xBOP}≡{xPG[(]xTV_[0]_3[L/q/xTV_3]xTV_[0]_4[L/p/xTV_4]xPD[)]]/xE_2]xPD[)]]/xE_3]/xSQ_0]◄≺▷◊→xQ_0|◙→xTVQ_**[1]**_{p}xTVQ_**[2]**_{q}←◙/xQ_0|xSQ_0[xE_3[xBOP}≡{xPG[(]xE_1[xBOP}≡{xPG[(]xTV_[0]_1[L/p/xTV_1]xTV_[0]_2[L/q/xTV_2]xPD[)]]/xE_1]xE_2[xBOP}≡{xPG[(]xTV_[0]_3[L/q/xTV_3]xTV_[0]_4[L/p/xTV_4]xPD[)]]/xE_2]xPD[)]]/xE_3]/xSQ_0]←◊◁➲⊂/GEN=[0]/COMBITABLE=[tot[2]combi[2*1=[({1})({2})]1*2=[({1}{2})]]nbc[3]]/TV=[x]/faire6_table

Nous reportons ensuite cette numérotation dans notre table combinatoire (*rre* 14 à 18). Voici le nouveau résultat obtenu :

DISTRIB, 1, 19,1:
…/COMBITABLE=[tot[2]combi[2*1=[({•p})({•q})]1*2=◙→[({•p}{•q})]←◙]nbc[3]]/TV=[q]/init1_table

Nous avons bien remplacé les positions par les *termes* eux-mêmes (avec un marqueur de validation à faire sauter dans les étapes finales).

6.9.5 Décomposition de la biconditionnelle et préparation des nouveaux quantificateurs

Cette partie est une véritable manipulation de portions de chaînes de caractères. Nous n'opérons que dans les zones de travail entre ▷…◁ et ➲…➲. A l'issue de cette manipulation les deux zones sont prêtes à subir la distribution :

DISTRIB, 1, 26,1:
➢test⌊pq⌈≡(≡(pq)≡(qp))⌉▶xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈xE_3xBOP⊧≡⌋xPG(⌋xE_1xBOP⊧≡⌋xPG(⌋xTV_[0]_1L/p/xTV_1xTV_[0]_2L/q/xTV_2xPD(⌋)⌋/xE_1xE_2xBOP⊧≡⌋xPG(⌋xTV_[0]_3L/q/xTV_3xTV_[0]_4L/p/xTV_4xPD(⌋)⌋/xE_2xPD(⌋)⌋/xE_3/xSQ_0⌉◀≮▷xQ_0⌊◊→xTVQ{p}xTVQ{q}←◊/xQ_0⌋xSQ_0⌈xE_3xBOP⊧≡⌋xPG(⌋xE_1**●→xBOP⊧≡⌋xPG(⌋xTV_[0]_1L/p/xTV_1xTV_[0]_2L/q/xTV_2xPD(⌋)⌋←●**/xE_1xE_2**○→xBOP⊧≡⌋xPG(⌋xTV_[0]_3L/q/xTV_3xTV_[0]_4L/p/xTV_4xPD(⌋)⌋**←○/xE_2xPD(⌋)⌋/xE_3/xSQ_0⌉◁➲◊→xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋←◊xSQ_0⌈xE_3xBOP⊧≡⌋xPG(⌋xE_201xQ_201⌊●→←●/xQ_201⌋xSQ_201⌈xE_1xBOP⊧≡⌋xPG(⌋xTV_[0]_1L/p/xTV_1xTV_[0]_2L/q/xTV_2xPD(⌋)⌋/xE_1/xSQ_201⌉/xE_201xE_202xQ_202⌊○→←○/xQ_202⌋xSQ_202⌈xE_2xBOP⊧≡⌋xPG(⌋xTV_[0]_3L/q/xTV_3xTV_[0]_4L/p/xTV_4xPD(⌋)⌋/xE_2/xSQ_202⌉/xE_202xPD(⌋)⌋/xE_3/xSQ_0⌉➲/GEN=[0]/COMBITABLE=[tot[2]combi(2*1=[({•p})({•q})]1*2=◙→[↕→({p}{•q})←↕]←◙(nbc[3]/TV=[p]/ok

6.9.6 Processus principal de distribution des quantificateurs

Nous avons divisé cette partie de notre traitement en deux blocs. Ils traitent respectivement de la première et de la seconde *expression* où se distribuent les *quantificateurs*. Le processus boucle sur ces deux structures et recherche si oui ou non le *terme* considéré doit être quantifié dans l'*expression* traitée. Les *rre* 28 à 37 parcourent la première *expression* et celle de 38 à 48 la deuxième. A partir de là (*rre* 49 à 51), cette section va boucler jusqu'à ce que tous les *termes* soient traités. Voyons, à la première itération des deux blocs, le résultat :

DISTRIB, 1, 49,1:
➢test⌊pq⌈≡(≡(pq)≡(qp))⌉▶xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈xE_3xBOP⊧≡⌋xPG(⌋xE_1xBOP⊧≡⌋xPG(⌋xTV_[0]_1L/p/xTV_1xTV_[0]_2L/q/xTV_2xPD(⌋)⌋/xE_1xE_2xBOP⊧≡⌋xPG(⌋xTV_[0]_3L/q/xTV_3xTV_[0]_4L/p/xTV_4xPD(⌋)⌋/xE_2xPD(⌋)⌋/xE_3/xSQ_0⌉◀≮▷xQ_0⌊◊→xTVQ_fait{p}xTVQ{q}←◊/xQ_0⌋xSQ_0⌈xE_3xBOP⊧≡⌋xPG(⌋xE_1●→xBOP⊧≡⌋xPG(⌋xTV_fait_[0]_1L/p/xTV_1xTV_[0]_2L/q/xTV_2xPD(⌋)⌋←●/xE_1xE_2○→xBOP⊧≡⌋xPG(⌋xTV_[0]_3L/q/xTV_3xTV_fait_[0]_4L/p/xTV_4xPD(⌋)⌋←○/xE_2xPD(⌋)⌋/xE_3/xSQ_0⌉◁➲◊→**xQ_0⌊xTVQ{q}/xQ_0⌋**←◊xSQ_0⌈xE_3xBOP⊧≡⌋xPG(⌋xE_201**xQ_201⌊●→xTVQ{p}←●/xQ_201⌋**xSQ_201⌈xE_1xBOP⊧≡⌋xPG(⌋xTV_[0]_1L/p/xTV_1xTV_[0]_2L/q/xTV_2xPD(⌋)⌋/xE_1/xSQ_201⌉/xE_201**xE_202⌊xQ_202⌊○→xTVQ{p}←○/xQ_202⌋**xSQ_202⌈xE_2xBOP⊧≡⌋xPG(⌋xTV_[0]_3L/q/xTV_3xTV_[0]_4L/p/xTV_4xPD(⌋)⌋/xE_2/xSQ_202⌉/xE_202xPD(⌋)⌋/xE_3/xSQ_0⌉➲/GEN=[0]/COMBITABLE=[tot[2]combi(2*1=[({•p})({•q})]1*2=◙→[↕→({p}{•q})←↕]←◙(nbc[3]/TV=[p]/mark_combitable

Nous voyons (en gras) l'évolution de nos trois *quantificateurs* qui par ailleurs sont renumérotés ainsi que leur *sous-quantificateur* associé. Observons le résultat à la deuxième itération :

DISTRIB, 1, 49,2:
➢test⌊pq⌈≡(≡(pq)≡(qp))⌉▶xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈xE_3xBOP⊧≡⌋xPG(⌋xE_1xBOP⊧≡⌋xPG(⌋xTV_[0]_1L/p/xTV_1xTV_[0]_2L/q/xTV_2xPD(⌋)⌋/xE_1xE_2xBOP⊧≡⌋xPG(⌋xTV_[0]_3L/q/xTV_3xTV_[0]_4L/p/xTV_4xPD(⌋)⌋/xE_2xPD(⌋)⌋/xE_3/xSQ_0⌉◀≮▷xQ_0⌊◊→xTVQ_fait{p}xTVQ_fait{q}←◊/xQ_0⌋xSQ_0⌈xE_3xBOP⊧≡⌋xPG(⌋xE_1●→xBOP⊧≡⌋xPG(⌋xTV_fait_[0]_1L/p/xTV_1xTV_fait_[0]_2L/q/xTV_2xPD(⌋)⌋←●/xE_1xE_2○→xBOP⊧≡⌋xPG(⌋xTV_fait_[0]_3L/q/xTV_3xTV_fait_[0]_4L/p/xTV_4xPD(⌋)⌋←○/xE_2xPD(⌋)⌋/xE_3/xSQ_0⌉◁➲◊→**xQ_0⌊/xQ_0⌋←◊xSQ_0⌈**xE_3xBOP⊧≡⌋xPG(⌋xE_201xQ_201⌊●→**xTVQ{p}xTVQ{q}←●/xQ_201⌋**xSQ_201⌈xE_1xBOP⊧≡⌋xPG(⌋xTV_[0]_1L/p/xTV_1xTV_[0]_2L/q/xTV_2xPD(⌋)⌋/xE_1/xSQ_201⌉/xE_201xE_202**xQ_202⌊○→xTVQ{p}xTVQ{q}←○/xQ_202⌋**xSQ_202⌈xE_2xBOP⊧≡⌋xPG(⌋xTV_[0]_3L/q/xTV_3xTV_[0]_4L/p/xTV_4xPD(⌋)⌋/xE_2/xSQ_202⌉/xE_202xPD(⌋)⌋/xE_3/xSQ_0⌉➲/GEN=[0]/COMBITABLE=[tot[2]combi(2*1=[({•p})({•q})]1*2=◙→[↕→({p}{q})←↕]←◙(nbc[3]/TV=[q]/mark_combitable

Nous pouvons constater que le *quantificateur* dominant a été dépouillé de ses *termes* et les deux nouveaux *sous-quantificateurs* sont pourvus des deux *termes* p et q. Dans les paramètres, la COMBITABLE a été mise à jour pour cette combinaison (les deux marqueurs • ont été supprimés).

6.9.7 Suppression des quantificateurs et de la généralisation

Si un *quantificateur* reste vide ou est vidé en ce qui concerne le *quantificateur* dominant, nous les supprimons. Nous procédons, de la même façon, avec la *généralisation* dominante si son *quantificateur* devait être vide. C'est ce que font les *rre* 52 à 70. Et, nous trouvons le résultat suivant :

DISTRIB, 1, 70,1:
➢test pq ≡(≡(pq)≡(qp)) ►xQ_0 xTVQ p xTVQ q /xQ_0 xSQ_0 xE_3 xBOP ≡ xPG (xE_1 xBOP ≡ xPG (xTV_ 0 _1 L/p /xTV_1 xTV_ 0 _2 L/q /xTV_2 xPD) /xE_1 xE_2 xBOP ≡ xPG (xTV_ 0 _3 L/q /xTV_3 xTV_ 0 _4 L/p /xTV_4 xPD) /xE_2 xPD) /xE_3 /xSQ_0 ◄≺▷xQ_0 ◊→xTVQ_fait p xTVQ_fait q ←◊/xQ_0 xSQ_0 xE_3 xBOP ≡ xPG (xE_1 •→xBOP ≡ xPG (xTV_fait_ 0 _1 L/p /xTV_1 xTV_fait_ 0 _2 L/q /xTV_2 xPD) ←•/xE_1 xE_2 ○→xBOP ≡ xPG (xTV_fait_ 0 _3 L/q /xTV_3 xTV_fait_ 0 _4 L/p /xTV_4 xPD) ←○/xE_2 xPD) /xE_3 /xSQ_0 ◁➲**xE_3 xBOP ≡ xPG (xE_201 xQ_201 xTVQ p xTVQ q /xQ_201 xSQ_201 xE_1 xBOP ≡ xPG (xTV_ 0 _1 L/p /xTV_1 xTV_ 0 _2 L/q /xTV_2 xPD) /xE_1 /xSQ_201 ⎤/xE_201 xE_202 xQ_202 xTVQ p xTVQ q /xQ_202 xSQ_202 xE_2 xBOP ≡ xPG (xTV_ 0 _3 L/q /xTV_3 xTV_ 0 _4 L/p /xTV_4 xPD) /xE_2 /xSQ_202 ⎤/xE_202 xPD) /xE_3** ➲/GEN=[0]/COMBITABLE=tot[1]combi 2*1=◙→[↕→({p})←↕({•q}))←◙1*2=[({p}{q}))] nbc[3] /TV=[p]/nettoyer

De plus la table combinatoire est mise à jour pour la solution de distribution suivante. Dans notre cas, c'est q qui devra être traité pour lui seul.

6.9.8 Mémorisation du résultat et mise à jour de la structure combinatoire

Comme notre GO_i doit encore traiter des cas possibles de distribution, nous devons mémoriser le résultat obtenu. Les *rre* 71 à 83 se chargent de cette tâche et ajustent les pointeurs dans la *zone tampon*. En passant, la GO_i appelle DEPX pour dépouiller la version en format *langage pivot* du résultat.

Quant aux *rre* 84 à 90, elles mettent à jour les paramètres de la table combinatoire et recommencent le processus jusqu'à ce que toutes les possibilités de distribution soient produites (trois, dans notre exemple).

6.9.9 Lecture de la zone tampon

La dernière partie du traitement (*rre* 91 à 99) va relire les données résultantes qui ont été enregistrées dans la *zone tampon*. Nous avons comme résultat final, la version en *forme analysée* et la version dépouillée pour chaque combinaison :

DISTRIB, 1, 96,1:
DET:➢test pq ≡(≡(pq)≡(qp)) ►xQ_0 xTVQ p xTVQ q /xQ_0 xSQ_0 xE_3 xBOP ≡ xPG (xE_1 xBOP ≡ xPG (xTV_ 0 _1 L/p /xTV_1 xTV_ 0 _2 L/q /xTV_2 xPD) /xE_1 xE_2 xBOP ≡ xPG (xTV_ 0 _3 L/q /xTV_3 xTV_ 0 _4 L/p /xTV_4 xPD) /xE_2 xPD) /xE_3 /xSQ_0 ◄≺➲[3]xE_3 xBOP ≡ xPG (xE_201 xQ_201 xTVQ p xTVQ q /xQ_201 xSQ_201 xE_1 xBOP ≡ xPG (xTV_ 0 _1 L/p /xTV_1 xTV_ 0 _2 L/q /xTV_2 xPD) /xE_1 /xSQ_201 ⎤/xE_201 xE_202 xQ_202 xTVQ p xTVQ q /xQ_202 xSQ_202 xE_2 xBOP ≡ xPG (xTV_ 0 _3 L/q /xTV_3 xTV_ 0 _4 L/p /xTV_4 xPD) /xE_2 /xSQ_202 ⎤/xE_202 xPD) /xE_3 ➲

DISTRIB, 1, 97,1: DEP:▷[3]**≡(pq ≡(pq) pq ≡(qp))**◁

DISTRIB, 1, 96,2:
DET:➢test pq ≡(≡(pq)≡(qp)) ►xQ_0 xTVQ p xTVQ q /xQ_0 xSQ_0 xE_3 xBOP ≡ xPG (xE_1 xBOP ≡ xPG (xTV_ 0 _1 L/p /xTV_1 xTV_ 0 _2 L/q /xTV_2 xPD) /xE_1 xE_2 xBOP ≡ xPG (xTV_ 0 _3 L/q /xTV_3 xTV_ 0 _4 L/p /xTV_4 xPD) /xE_2 xPD) /xE_3 /xSQ_0 ◄≺➲[2]xQ_0 xTVQ q /xQ_0 xSQ_0 xE_3 xBOP ≡ xPG (xE_201 xQ_201 xTVQ p /xQ_201 xSQ_201 xE_1 xBOP ≡ xPG (xTV_ 0 _1 L/p /xTV_1 xTV_ 0 _2 L/q /xTV_2 xPD) /xE_1 /xSQ_201 ⎤/xE_201 xE_202 xQ_202 xTVQ p /xQ_202 xSQ_202 xE_2 xBOP ≡ xPG (xTV_ 0 _3 L/q /xTV_3 xTV_ 0 _4 L/p /xTV_4 xPD) /xE_2 /xSQ_202 ⎤/xE_202 xPD) /xE_3 /xSQ_0 ⎤➲

DISTRIB, 1, 97,2: DEP:▷[2][q][≡([p][≡(pq)][p][≡(qp)])]◁

DISTRIB, 1, 96,3:
DET:➢{test{[pq][≡(≡(pq)≡(qp))]►xQ_0[xTVQ{p}xTVQ{q}/xQ_0]xSQ_0⌈xE_3{xBOP}≡⦃xPG{(}xE_1{xBOP}≡⦃xPG{(}xTV_[0]_1{L/p/xTV_1}xTV_[0]_2{L/q/xTV_2}xPD{)}]/xE_1}xE_2{xBOP}≡⦃xPG{(}xTV_[0]_3{L/q/xTV_3}xTV_[0]_4{L/p/xTV_4}xPD{)}]/xE_2}xPD{)}]/xE_3}/xSQ_0⌉◄≺➲[1]xQ_0[xTVQ{p}/xQ_0]xSQ_0⌈xE_3{xBOP}≡⦃xPG{(}xE_201{xQ_201[xTVQ{q}/xQ_201]xSQ_201⌈xE_1{xBOP}≡⦃xPG{(}xTV_[0]_1{L/p/xTV_1}xTV_[0]_2{L/q/xTV_2}xPD{)}]/xE_1}/xSQ_201⌉/xE_201}xE_202{xQ_202[xTVQ{q}/xQ_202]xSQ_202⌈xE_2{xBOP}≡⦃xPG{(}xTV_[0]_3{L/q/xTV_3}xTV_[0]_4{L/p/xTV_4}xPD{)}]/xE_2}/xSQ_202⌉/xE_202}xPD{)}]/xE_3}/xSQ_0⌉➲

DISTRIB, 1, 97,3: DEP:▷[1][p][≡([q][≡(pq)][q][≡(qp)])]◁

Nous retrouvons bien les trois distributions escomptées.

En conclusion, nous sommes capables de générer toutes les combinaisons de la *distribution des quantificateurs*. Sans disposer d'une sorte de script de stratégie qui nous indiquerait quelle *expression* choisir, nous ne sommes pas capables d'aller plus loin (chapitre 7. *Perspectives et développements futurs, p. 175*).

6.10 Traitement de la règle d'inférences de substitution (protothétique)

Tout comme pour les autres GO_i qui traitent des *règles d'inférences*, dans le cas de celle de substitution, sans un scénario de *déduction*, nous ne pouvons pas décider a priori quels sont les éléments morphosyntaxiques substitués et substituants. En revanche, en connaissant ces données, nous pouvons parfaitement mettre en œuvre la mécanique de substitution et les contrôles qu'elle nécessite. C'est ce que fait notre exemple de GO_i, SUBSTITU, sachant, que l'on peut trouver sa version complète avec la GO_i de test qui l'appelle dans l'annexe *Traitement de la règle de substitution (protothétique), p. 265*. Sa structure est aussi un modèle que le programmeur de GO_i rencontrera souvent : un bloc de filtres qui teste des conditions, suivi d'un traitement ; dont voici une illustration graphique pour le traitement de la *règle d'inférences de substitution :*

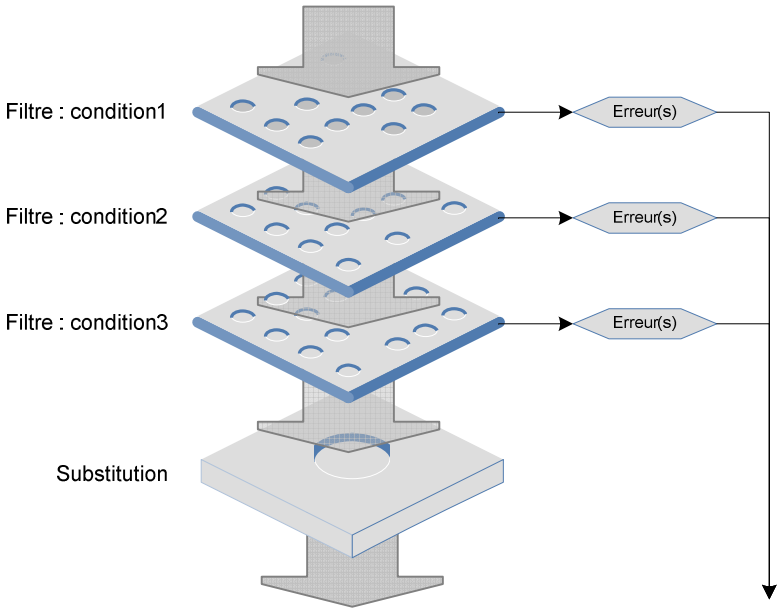


Figure 55. Modèle d'un bloc de filtres suivi d'un traitement

La section de filtres correspond aux *rre* 2 à 41 et la mécanique de substitution en tant que telle débute à la *rre* 42 et se termine à la *rre* 82. Même si cette GO_i est un peu longue, elle ne pose pas de problème de programmation particulier, sinon qu'il faut être véritablement exhaustif et tenir compte de toutes les conditions pré-requises à une substitution qui assurent la cohérence d'un traitement futur. Nous donnons un exemple d'exécution sachant que nous avons besoin de soumettre trois *chaînes* de caractères, enregistrées dans le *corpus d'entrée :*

cs_1, l'hôte de substitution, en *forme analysée* (première ligne du *corpus*) :
xQ_0⌊xTVQ{p}xTVQ{q}⁄xQ_0⌋xSQ_0⌈xE_3⦃xBOP⦄≡⦃xPG⦅(⦅xE_1⦃xBOP⦄≡⦃xPG⦅(⦅xTV_[0]_1⦃L/p⁄xTV_1⦄xTV_[0]_2⦃L/q⁄xTV_2⦄xPD⦅)⦆⁄xE_1⦄xE_2⦃xBOP⦄≡⦃xPG⦅(⦅xTV_[0]_3⦃L/q⁄xTV_3⦄xTV_[0]_4⦃L/p⁄xTV_4⦄xPD⦅)⦆⁄xE_2⦄xPD⦅)⦆⁄xE_3⦄⁄xSQ_0⌉
généré à partir de la forme brute : ⌊pq⌋⌈≡(≡(pq)≡(qp))⌉.

cs_2, le *terme* à substituer (deuxième ligne du corpus) : p

cs_3, l'*expression* utilisée comme substituant, en *forme analysée* (troisième ligne du *corpus*) :
xE_2⦃xMOP_[1]⦄h⦃xPG⦅(⦅xTV_[0]_2⦃L/r⁄xTV_2⦄xPD⦅)⦆⁄xE_2⦄

Nous obtenons :

TESTSUBSTITU, 1, 24, 1:
➢➲⌊pq⌋⌈≡(≡(pq)≡(qp))⌉➲≥p≤▷h(r)◁ ⁘⌊qrh⌋⌈≡(≡(h(r)q)≡(qh(r)))⌉⁘ ▶◀≺

L'*expression* ➲⌊pq⌋⌈≡(≡(pq)≡(qp))⌉➲ est la version dépouillée (en format brut) de l'hôte, l'*encapsulation* : ≥p≤ est le *terme* à substituer, la forme ▷h(r)◁ est le substituant et le résultat se trouve dans l'*encapsulation* : ⁘⌊qrh⌋⌈≡(≡(h(r)q)≡(qh(r)))⌉⁘ .

6.11 La directive d'extensionalité et traitement des filtres syntaxiques et sémantiques

Pour traiter les *directives d'extensionalité*, *4.4.6.4 La directive d'extensionalité, p. 51* et *4.5.4.4 Les deux directives d'extensionalité de l'ontologie, p. 69,* il s'agit avant tout de s'assurer que les *catégories syntaxico-sémantiques* des entités que l'*inscription* à analyser contient, sont inscrites dans le *système* (dans la *bibliothèque*). Cette condition est de l'ordre du domaine sémantique. Cette *directive* ne conduit pas à une *transformation* comme c'est le cas avec les autres *règles d'inférences*, mais au respect ou non d'un principe, celui d'extensionalité. Par conséquent, le traitement de cette règle passe par onze filtres, pour certains avec une couleur principalement syntaxique, pour d'autres avec une teinte sémantique. Dans l'exemple de GO_i, EXTENS, donné dans sa version complète dans l'annexe *Traitement de la directive d'extensionalité, p. 273*, nous traitons ces filtres de la façon suivante :

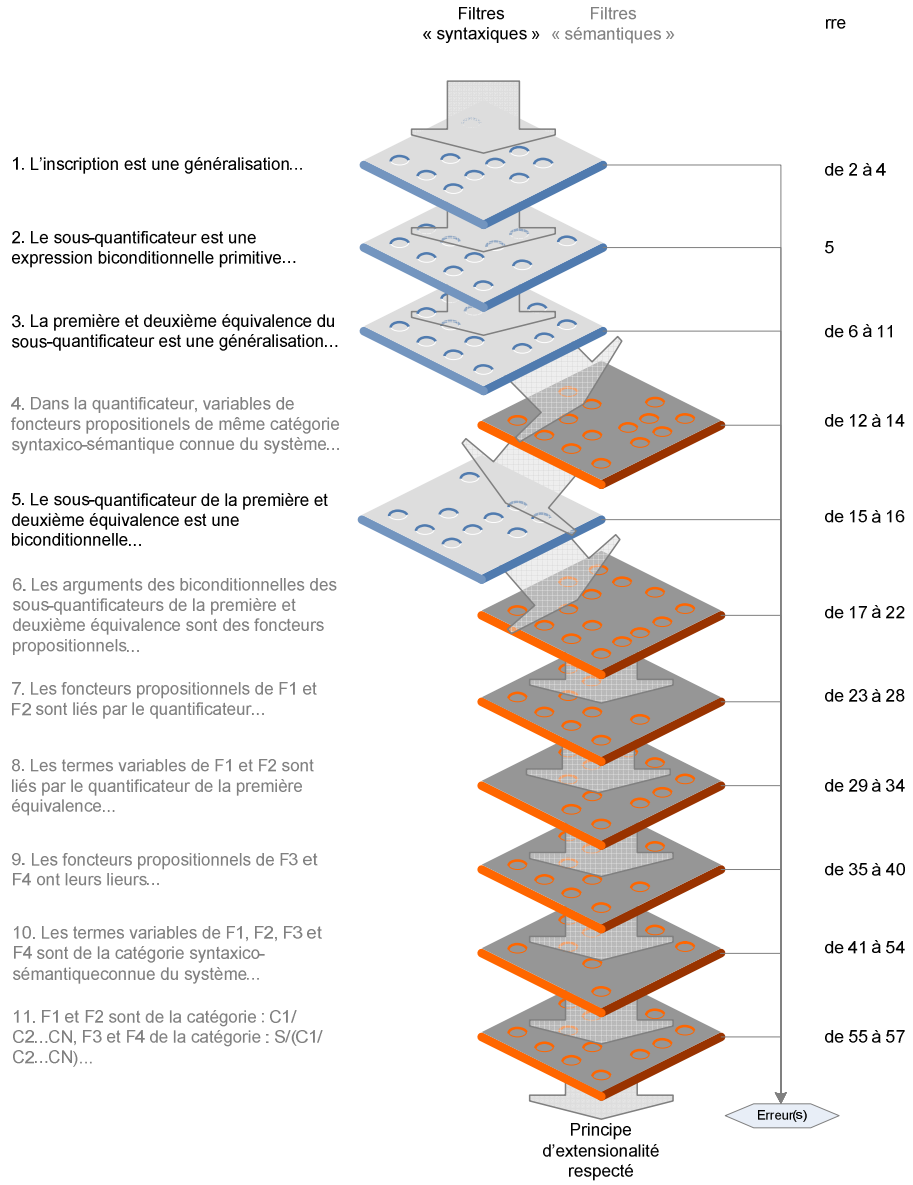


Figure 56. Filtres syntaxiques et sémantiques pour le respect du principe d'extensionalité

Nous n'avons pas jugé nécessaire de donner ici une GO_i dédiée à la recherche dans la *bibliothèque* ou la *librairie* des *catégories syntaxico-sémantiques*. Ce qui serait similaire au processus utilisé dans le traitement de la *règle d'inférences de détachement* (voir le traitement de recherche d'informations dans la *librairie* dans la GO_i de la *règle d'inférences de détachement*, à la section *6.8. Traitement de la règle d'inférences de détachement, p. 164*). Nous avons simplement simulé aux *rre* 53 et 55, une réponse positive.

6.12 La génération d'une bibliothèque

Il nous est apparu intéressant de donner un exemple de création d'une *bibliothèque* d'*inscriptions* analysées (en format *langage pivot*, LO_p) converties à partir d'*inscriptions* en format « brut ». Notre GO_i, GENLIB, que l'on trouve dans l'annexe *Génération d'une bibliothèque, p. 277,* parcourt un *corpus d'entrée* et génère dans le *corpus tampon* la *bibliothèque* d'*inscriptions* analysées.

Evidemment, il n'est pas question de traiter les *règles d'inférences* et d'envisager de faire de la *déduction* sans disposer d'une *bibliothèque* de tout ce que le *système* connaît au préalable.

6.13 Enchaînement de GO_i

L'enchaînement de GO_i, TOUT, que nous donnons dans l'annexe *Enchaînement des GOi d'analyses, de détermination de catégories, de calculs propositionnels, de conversion et de dépouillement, p. 279* est très pratique pour obtenir un maximum d'informations sur une *inscription* donnée ; par exemple :

cs : ⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉

TOUT, 1, 12, 1:
➢⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉__/__⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉►(∀pq)((p⊃q)≡~((p∧~(q))))◄/CAT=⦃⟦1:cat⦄S/SS⦃cst⦄⊃⦃ctx⦄(--)⦃⟧⦄/TABLE=⦃1,0,1,1⦄≺

ou encore

cs : ⌊qpr⌋⌈≡(ω[p]\q/(r)≡(~(≡(pq))r))⌉

TOUT, 1, 12, 1:
➢⌊qpr⌋⌈≡(ω[p]\q/(r)≡(~(≡(pq))r))⌉__/__⌊pqr⌋⌈≡(ω[p]\q/(r)≡(~(≡(pq))r))⌉►(∀pqr)(ω[p]\q/(r)≡(~((p≡q))≡r))◄/CAT=⦃⟦1:cat⦄((S/S)/S)/S⦃cst⦄ω⦃ctx⦄[-]⦃⟧⦄⟦2:cat⦄(S/S)/S⦃cst⦄ω[-]⦃ctx⦄\-/⦃⟧⦄⟦3:cat⦄S/S⦃cst⦄ω[-]\-/⦃ctx⦄(-)⦃⟧⦄/TABLE=⦃0,1,1,0,1,0,0,1⦄≺

En fin de *chaîne*, nous trouvons la détermination de la *catégorie syntaxico-sémantique* (/CAT=⦃…⦄) et la table de vérité du *terme constant* (/TABLE=⦃…⦄).

6.14 Remplacement des termes variables et des termes constants par des entités linguistiques en langue naturelle

Il nous est apparu intéressant de donner un exemple de GO_i rudimentaire qui remplace les éléments morphosyntaxiques et symboliques d'une *thèse-définition* par des assertions en langue naturelle. L'exemple ne traite pratiquement qu'un seul cas. On trouve cette petite GO_i, CONDISUBEX, dans l'annexe *Remplacement des éléments morphosyntaxiques et symboliques par des formules en langue naturelle, p. 280*.

Voyons le résultat obtenu par cette GO_i à la *thèse-définition* qui lui est destinée :

cs : ⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉

CONDISUBEX, 1, 43, 1: ➢⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉►Quel que soit "il fait beau temps", "on sort le chien", on définit : "si il fait beau alors on sort le chien", par la définition : il est faux de dire que il fait beau et on ne sort pas le chien◄≺

Nous soulignons, que si cet exemple ne présente pas d'intérêt en ce qui concerne la pure mécanique des *SL*, ce type de traitement est néanmoins le dernier niveau d'interfaçage important vers des solutions applicatives mises en œuvre et *formalisées* par les *SL*. Il répond en particulier à notre cahier des charges section *5.2.1 Outils de transposition et réversibilité, p. 83*.

6.15 Conclusion

Par les différents exemples de GO_i que nous avons présentés dans ce chapitre, nous montrons que nous avons atteint notre objectif de faisabilité. Nous avons aussi prouvé que le *LRE* pouvait servir de métalangage adapté aux *SL* et en particulier aux *règles d'inférences*. De plus, le *LRE* permet d'automatiser des processus de génération et de *transformation* d'*expressions* complexes. Si nos travaux s'adaptent au *SL*, ils peuvent parfaitement convenir à des logiques plus classiques comme celles du 1er ordre[112].

[112] Qui peut le plus, peut le moins !

7. Perspectives et développements futurs

Dans ce chapitre, nous explorons, tout d'abord, qu'elles sont les extensions possibles de notre approche à d'autres domaines de connaissances. Nous survolons quelques aspects applicatifs auxquels nous avons mis au défi le RE.

Nous proposons, ensuite, des améliorations du LRE et du logiciel prototype RE.

Dans une troisième partie nous ouvrons un horizon vers une nouvelle solution informatique, un interpréteur de LO_{SL}, qui pourrait se construire à partir de nos GO_i.

Nous nous arrêtons aussi sur le projet OWL qui envisage le traitement de la plus grande encyclopédie du monde : le Web.

En dernière partie, nous investiguons quels développements pourraient conduire le RE à devenir un moteur de stratégie, soit un outil d'aide à la déduction.

7.1 Extension du LRE vers d'autres domaines de connaissances

En se référant à nos exemples du tableau de bord, nous rencontrons des problèmes et questions similaires dans des *domaines de connaissances* comme la taxonomie des êtres vivants.

La taxonomie est un des domaines qui illustre bien les efforts réalisés dans la « description » de connaissances. Prenons le taxon du chat comme exemple similaire à celui du chocolat (*Figure 11, p. 25*) et qui peut enrichir les questions que nous nous posions. Il s'agit de décrire l'être vivant, nommé Tim, le chat unique de la voisine, sachant qu'il est un vertébré et un mammifère :

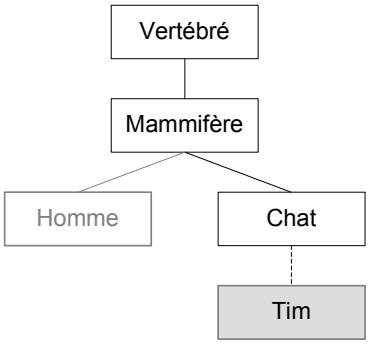


Figure 57. Taxon très simplifié du chat

Essayons d'enrichir ce modèle pour répondre à la question : « Si un chat perd sa tête, est-il toujours un mammifère ? ». Une nouvelle propriété doit être introduite dans le taxon. Insérons Animal à une tête[113] entre les propriétés Vertébré et Mammifère et étendons le taxon en vérifiant, en passant, qu'il s'applique pour les reptiles :

[113] Ne nous tenons pas compte des aberrations de la nature qui peut générer des monstres comme des moutons à deux têtes !

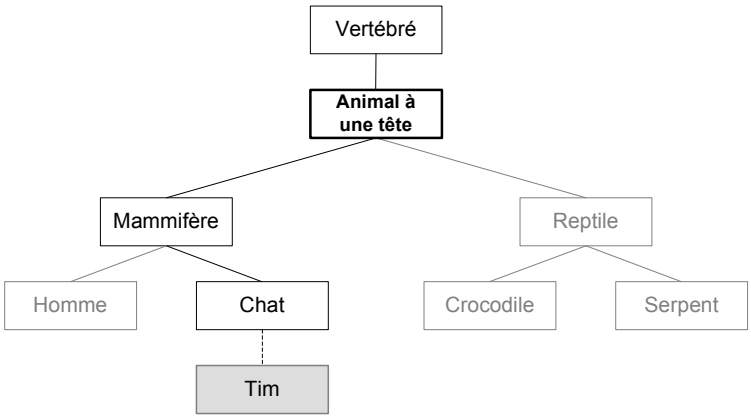


Figure 58. Introduction de la propriété Animal à une tête dans le taxon du chat

Le modèle couvre nos attentes ; tous les vertébrés ont une tête. Procédons de la même manière pour introduire la propriété Quadrupède, afin de répondre à la question : « Quels animaux ont quatre pattes ? » :

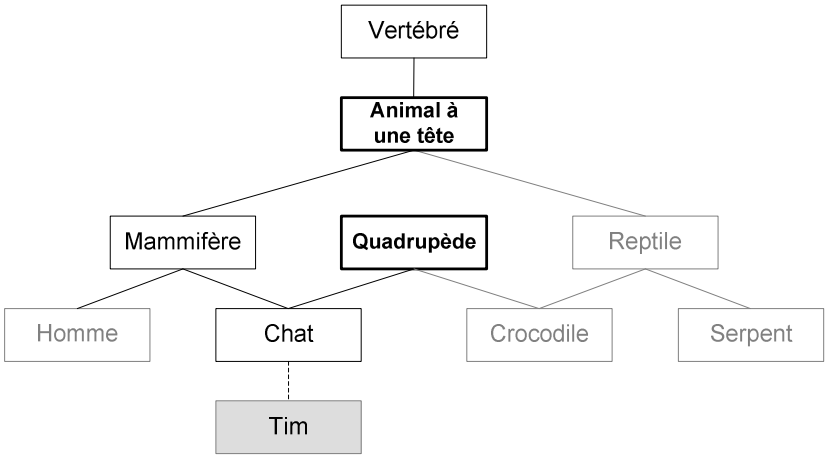


Figure 59. Introduction de la propriété Quadrupède dans le taxon du chat

La représentation montre aisément que tous les hommes ne sont pas des quadrupèdes. Comme dans nos exemples de tableau de bord, il a fallu introduire une propriété « parallèle », Quadrupède, qui ne s'insère pas dans l'arborescence initiale ; une propriété qui se combine à d'autres.

D'un point de vue purement biologique, quelle devraient être les contraintes qui nous permettent de substituer Chien à la place de ce Chat ? Avec les matériaux dont nous disposons, si toutes les propriétés qu'hérite Chat sont celles qui s'appliquent à Chien, la substitution est autorisée. Les connaissances biologiques que nous avons utilisées ici procèdent de l'axe sémantique du langage.

Donnons-nous une forme similaire du taxon du Chat à celle du Chocolat (voir *Figure 10, p. 24*) :

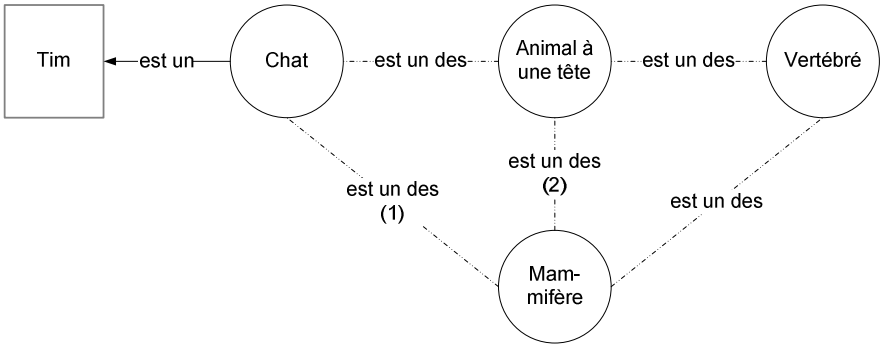


Figure 60. Extrait du taxon du chat

Répondons maintenant à la question : « Un chat qui n'a plus de tête est-il toujours un mammifère ? ». Supprimons la propriété Animal à une tête :

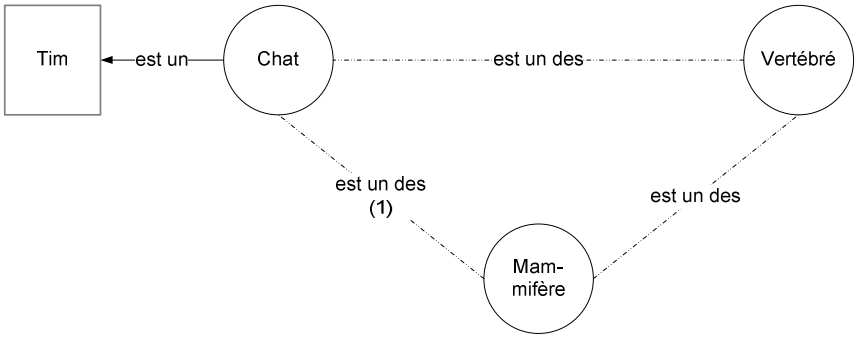


Figure 61. Suppression de la propriété Animal à une tête dans l'extrait du taxon du chat

Nous obtenons des questions et des réponses similaires à notre exploration du taxon relatif au chocolat dans le tableau de bord. Dès lors nous pouvons appliquer notre développement concernant le *maintien d'une propriété.*

Notre exploration montre que pour deux *domaines de connaissances* nous aboutissons à des questions similaires. Nos questions relatives à la suppression d'une propriété conduisent à l'argument de De Morgan[114], à savoir si :

Tout chat est un animal, donc toute tête d'un chat est une tête d'un animal.

S'intéressé à ce niveau de questions, nécessite de pénétrer des théories qui étudient la relation de partie à tout. C'est ce dont traite la *méréologie* de Leśniewski. Dès lors, de nouvelles GO_i peuvent être développées pour traiter ce *SL* que nous n'avons pas abordé dans notre développement[115].

7.2 Autres domaines applicatifs du LRE

Même si le *LRE* a été élaboré dans la perspective de faciliter la manipulation d'*inscriptions* leśniewskiennes, nous avons exploré plusieurs domaines auxquels notre langage pourrait s'appliquer.

C'est, par exemple, le cas de notre grammaire GO_c (*5.7 Programmation et analyse détaillée de la GOc, p. 136*) qui traite des grammaires génératives et transformationnelles.

Nous donnons ici d'autres exemples. Evidemment, la liste qui suit n'est pas exhaustive.

---

[114] Voir le chapitre dédié à ce sujet dans l'ouvrage de Nadine Gessler (GESSLER, 2005, p. 6 et suivantes).

[115] Il faut noter que les GO_i développées pour la *protothétique* et l'*ontologie*, en ce qui concerne le traitement syntaxique, s'appliquent aussi à la *méréologie*. Il reste à vérifier le traitement sémantique comme les notions d' « ingrédience ».

Nous pouvons parfaitement envisager de développer un ensemble de GO_i qui traite de l'évolution phonologique d'une langue comme le latin vers le français en passant par le bas latin et le français médiéval. Par exemple, l'utilisation des théories en phonologie de Roman Jakobson et plus particulièrement de la représentation en « triangles » des consonnes et voyelles (JAKOBSON, 1963, p. 136 et suivantes) nous permettrait d'écrire des *rre* de modification d'évolution phonétique comme :

rre : ☞‹a›P‹b›⇒‹a›B‹b›☜
cs : A**P**ICULA
cr_1 : A**B**ICULA
…
cr_i : A**B**EILLE
…
cr_{i+1} : AVEILLE

Le *LRE* a été défini pour répondre aux besoins des *SL*. En dehors de ce domaine, notre langage convient à tous les traitements dédiés à des langages formels comme les langages informatiques de programmation. Le *RE* permet, par exemple de tester, d'analyser, de *formaliser* tous les langages qu'une *EBNF* peut définir.

La valeur des données informatiques des entreprises augmente d'année en année. Cette même valeur croit encore avec la qualité et la fiabilité de ses informations. Par exemple, si l'adresse d'un chantier sur lequel il faut qu'un technicien de maintenance se rende, n'est pas enregistrée dans un format rigoureux que peut comprendre un GPS[116] pour la transformer en coordonnées géographiques, il sera vain de demander à un navigateur de conduire un utilisateur à cette adresse. Nous avons utilisé des GO_i et le *RE* pour convertir des adresses mal formées dans une base de données vers un format acceptable par GPS. Dans le même ordre d'idées, le *LRE* et le *RE* peuvent être utilisés comme moyen de filtrage, vérification de cohérences sur des données informatiques. Ils s'adaptent aussi à la migration de données entre systèmes d'informations qui nécessitent des *transpositions* d'informations.

C'est probablement le lambda calcul qui devrait, à premier titre, venir enrichir notre approche. La notion de réécriture de termes et de fonctions dans ce domaine et notre métalangage, *LRE*, combiné avec une base de calcul sérieuse des fonctions donnerait une *puissance* particulière à nos outils qui permettraient de traiter non seulement des *expressions* logiques, mais aussi des formules mathématiques complexes avec un minimum de programmation de GO_i.

7.3 Le langage ontologique de La Toile

La taxonomie, la classification, l'indexation sont des capacités sous-jacentes et nécessaires aux traitements de connaissances réunies de façon encyclopédique. Depuis quelques années, les spécialistes de la présentation de contenus sur La Toile, ont très vite compris qu'il fallait se donner des outils sérieux d'accès aux connaissances de ce qui devenait la plus grande encyclopédie du monde. C'est un nouveau défi scientifique, le projet Ontologic Web Language (*OWL*[117]). Plusieurs axes sont définis dans ce projet visant à mettre de l'ordre dans l'anarchie des contenus à disposition sur le réseau mondial. Il s'agit de se donner des formats normalisés pour mémoriser les connaissances. Personne ne trouvera d'intérêt à l'existence de la plus grande bibliothèque sans disposer d'outils sophistiqués de recherche. Il faut donc s'engager dans une *formalisation* sérieuse des règles d'accès aux structures des contenus mémorisés. Si des scientifiques de toutes les disciplines sont appelés à participer à cette mission, ce sont principalement, et pour des raisons évidentes, les informaticiens qui sont au centre de ces travaux. Cette discipline sait maîtriser le traitement de structures comme les listes, les arborescences et les graphes d'objets divers. Elle a montré que les opérations de tris,

[116] Global Positioning System (Système de positionnement).

[117] On trouve la référence et la source principale des travaux OWL sur le site Internet WC3 / OWL Ontology Web Language (McGUINESS & VAN HARMELEN, 2004).

d'insertions, de recherches n'ont plus de secret. Le développement des théories de la compilation a permis d'exceller dans le traitement des *LF*. Et, à partir des années 80, l'avènement de l'approche objet a orienté tous les développements informatiques à maîtriser la notion de classes[118] et d'objets[119]. En se documentant sur le projet *OWL*, le lecteur pourra noter à quel point le projet est coloré par l'héritage de la programmation orientée objet. L'approche objet a permis d'apporter de la rigueur dans la programmation des ordinateurs ; mais elle ne peut pas être considérée comme une théorie des classes. En effet, certains aspects de l'implantation de la programmation objet peuvent permettre aux programmeurs l'introduction de malfaçons du même genre que celles décrites dans nos exemples des tableaux de bord[120]. En dehors des aspects de types d'objets et de propriétés apportés par la notion de classe, la programmation orientée objet se borne à une dimension de traitement purement syntaxique. Le projet *OWL*, même s'il hérite d'une partie de cette approche, tient compte d'une certaine dimension sémantique. Cette dimension se base sur des processus de traitement par inférences. Dès lors, il est principalement un axe de recherche qui doit aboutir à la définition d'un métalangage ou à plusieurs couches de métalangages. *OWL* doit tenir compte de la multitude de formats de mémorisation déjà référencés sur Internet. Il n'est pas imaginable que l'on puisse restructurer cette immense quantité d'informations du jour au lendemain. Mais il nécessitera des *restructurations* de l'information existante dans des nouveaux formats. Ces *transformations* nécessiteront une attention particulière pour éviter les types de malfaçons que nous avons recensées.

Comme nous l'avons vu, les questions issues de la taxonomie ou autres volontés de classement peuvent participer à l'élaboration d'encyclopédies. Par conséquent, comme nous avons proposé un formalisme qui évite certaines ambiguïtés dans ces domaines, quel que soit le niveau de notre contribution, il ne peut qu'enrichir le traitement des connaissances.

7.4 Amélioration du RE

Nous recensons un certain nombre de grands axes de développements et d'améliorations qui peuvent se baser sur nos travaux pour des évolutions futures.

Il s'agirait de revisiter *LRE*, en particulier pour lui intégrer un calcul propositionnel des *fonctions propositionnelles*, de l'*epsilon* et autres *termes constants* de l'*ontologie (*voir en particulier la section *4.6 Les interprétations de foncteurs et matrices de vérités, p. 70*).

Comme nous l'avons vu dans nos GO_i qui traitent des *SL* (chapitre 6*. GOi destinées aux logiques de Leśniewski, p. 151*), plus particulièrement, pour le traitement de la *règle de distribution des quantificateurs* (section *6.9.3. Création d'une table de combinaison, p. 167*),

---

[118] Nous prenons ici la notion de classes utilisée en informatique et pas celle de la logique des classes traitée par les logiciens.

[119] La littérature sur la programmation objet est très riche. Le père de la programmation objet est Allan C. Kay avec le système Smalltalk. Tous les ouvrages qui traitent d'un langage de programmation traitent de l'approche objets. Le lecteur pourra se renseigner à ce sujet en consultant, par exemple, les ouvrages de Brad J. Cox (COX B. J., 1987), de S. Alagic (ALAGIC, 1989) et de M. Ayache, A. Flory (AYACHE & FLORY, 1996).

[120] Afin d'illustrer notre propos : à partir d'une classe générique définie dans le système, appelée la racine, on peut créer autant de classes que désirées qui hériteront elles-mêmes des propriétés de cette racine. Dans le même sens, tous les objets définis ont un ancêtre commun, un objet générique. Chaque objet est typé par une classe ou appartient à une classe. On considère qu'un objet ne peut pas être lui-même une classe. Donc, chaque objet est doté d'un certain nombre de propriétés et d'un certain nombre de méthodes ; sachant que les méthodes sont des petits programmes que l'on peut appeler permettant, par exemple, de créer de nouveaux objets, ou d'exécuter un traitement sur l'objet lui-même, etc. Les environnements de développement informatique offrent des mécanismes d'accès aux propriétés des objets. Mais, si de façon sous-jacente, les propriétés des objets déterminent des classes, elles ne sont pas explicitement inscrites dans le modèle. Par exemple, si un objet comme rectangle se définit avec trois propriétés : largeur, longueur et surface, toutes trois typées comme des nombres réels, même si la classe des réels existe dans le système, elle n'est pas « attachée » ou héritée explicitement de la racine. Dès lors, la programmation orientée objet ne suit pas la cohérence de la théorie des classes ; elle en applique les notions principales. Ceci n'est pas un handicap, l'implantation du modèle sur ordinateur ayant prouvé son efficacité depuis maintenant plus de vingt ans. En revanche, cela permet différentes implantations avec des interprétations qui peuvent devenir confuses.

nous pourrions introduire dans le *LRE* un certain nombre de nouvelles *fonctions* dédiées aux traitements combinatoires.

Nous envisageons aussi une amélioration du logiciel, *RE*, pour le faire passer de statut de prototype à celui de logiciel grand public.

Même si de façon interne et au moment de la compilation, le *RE* numérote les *rre*, il nécessite que le programmeur les numérote manuellement et parallèlement en particulier pour faciliter les branchements. Pour rendre la programmation plus conviviale, il faudrait apporter un moyen de définir des étiquettes relatives auxquelles nous pourrions nous référer pour les branchements.

Comme les GO_i traitent tous les aspects d'imbrication d'*expressions* et de profondeur du parenthèsage, nous pourrions envisager de convertir l'éditeur de *rre* en interpréteur, qui au fur et à mesure de l'édition, colore les *encapsulations* avec des couleurs différentes pour simplifier leur lecture[121]. Dans le même ordre d'idées que la colorisation des *expressions*, nous pourrions automatiser l'*encapsulation* que nous opérons à chaque phase d'analyse et la visualiser en forme arborescente pour simplifier des manipulations d'insertion, de suppression, de modification.

Le logiciel prototype a été dessiné avec une approche de processus parallèles ; tout d'abord pour pouvoir le stopper durant un long traitement qui pourrait s'itérer indéfiniment et ensuite pour prendre connaissance de résultats intermédiaires. Cette approche complexifie un peu le logiciel. Evidemment, la structure de données et l'algorithme les plus délicats sont ceux qui traitent les *variables isoformes*. Pour des questions de temps réponses nous avons dû aussi les rendre un peu plus complexes que prévus initialement. Il pourrait rester des anomalies qui ne sont détectables qu'avec une batterie importante de tests. Comme notre objectif se limite à une question de faisabilité, ce travail n'a pas été approfondi et devrait faire partie de travaux futurs.

Il y a des défauts de « cosmétique » en ce qui concerne notre police de caractères. Par exemple, l'espacement entre les *délimiteurs* de *quantificateurs* et *sous-quantificateurs* n'ont pas été assez travaillés et rendent la lecture difficile. Il est donc nécessaire de réaménager cette police de caractères.

Convertir l'ensemble de GO_i destinées aux *SL* en un logiciel (un interpréteur) qui traite directement les *inscriptions* et leurs bonnes formations ferait faire un grand pas à nos travaux.

7.5 Un logiciel interpréteur spécifique aux systèmes leśniewskiens

Le *LRE* et le *RE* traitent un champ d'applications plus large que les *SL*. La vérification de la faisabilité, que notre travail démontre, permet d'envisager le développement d'un logiciel spécialisé et qui intègre tout ce que nos GO_i dédiées aux *SL* couvrent, soit GO ($\subset GO^*$). Il s'agirait d'élaborer un grand interpréteur et exécuteur qui réunissent toutes les fonctions que nos exemples de GO_i présentent dans le chapitre 6. *GOi destinées aux logiques de Leśniewski, p. 151*.

En reprenant l'ensemble de nos GO_i et en les traduisant, dans un premier temps en diagrammes syntaxiques, base à une programmation traditionnelle de compilation (ou d'interprétation) et, dans un deuxième temps, en convertissant cette représentation dans un langage informatique traditionnel (WIRTH, 1976, p. 288). Cet interpréteur n'aurait plus la *puissance* du *LRE*. En revanche, il serait spécialisé aux *SL*, donc, à un sous ensemble LO_{SL} de *LO*. Avec cette approche, l'interpréteur remplacerait l'ensemble des GO_i. Cela permettrait de diminuer d'une couche le traitement par lesquelles le *RE* passe actuellement. De plus, les améliorations que nous proposons ci-dessus, facilitant l'interaction entre l'utilisateur et la nouvelle *solution* informatique, seraient plus faciles à intégrer.

[121] Comme le font les versions récentes d'Excel lors de la saisie de formules.

Evidemment, il s'agit là d'un effort industriel. Nous estimons ce type de logiciel à plus de cent mille lignes de programme. Alors que le *RE* n'en fait que dix mille.

7.6 Un moteur d'exécution de stratégie pour la déduction ou la preuve

Indiscutablement, le *RE* présenterait une valeur supplémentaire si nous pouvions étendre ses capacités à un outil d'aide à la *déduction*, qui nécessite à son niveau actuel de *puissance* encore beaucoup d'interventions manuelles. Nous proposons comme développement futur une nouvelle approche qui intègre un moteur de stratégie de *déduction*. Revenons sur les grandes questions à résoudre que nous avons rencontrées lors de notre étude :

1) mettre en œuvre un système d'aide qui traite les matrices de vérités (*4.6 Les interprétations de foncteurs et matrices de vérités, p. 70*), pour les *foncteurs propositionnels* ;

2) prévoir un moteur de stratégie exécutant des scripts ou des scénarios de l'enchaînement des *règles d'inférences* et des métarègles des *SL*.

Sachant qu'une *déduction* a toujours un objectif, quelque chose que l'on veut prouver ou montrer, nous avons commencé à explorer cette nouvelle direction dont voici quelques orientations à titre d'exemples.

Mettre en œuvre un moteur de stratégie, c'est essayer d'appliquer à un moment donné toutes les *règles d'inférences* et toutes les métarègles, cela sur la base de l'état d'un *SL* préalablement construit avec ses *thèses*. Le plus délicat est de définir quelle règle doit être exécutée avant une autre, sachant que chaque règle génère une nouvelle *thèse*. Pour se donner un début de stratégie nous avons exhibé une « compétence » des règles qui donne des voies de construction d'un moteur de stratégie. Une règle qui opère sur une seule *inscription* est dite autonome, dans les autres cas nous considérons qu'elle n'est pas autonome parce qu'elle nécessite l'utilisation d'éléments que le *système* connaît déjà. Prenons les *règles d'inférences* et les métarègles que nous avons abordées dans notre ouvrage :

(1) les *règles d'inférences de définition* ne sont pas autonomes. Elles nécessitent l'accès aux *inscriptions* et aux *catégories syntaxico-sémantiques* que connaît le *SL* à un moment donné. La GO_i que nous avons donnée concernant ces règles se présente comme un filtre qui vérifie la bonne formation des *thèses*. Notre cahier des charges ne prétend pas à traiter la génération de nouvelles *thèses-définition*, tâche que nous n'avons pas envisagée d'informatiser et qui est laissée à un niveau qui nécessite des compétences intellectuelles humaines. Par conséquent notre moteur de stratégie ferait référence à ce que ces règles ont généré préalablement comme *thèses* ;

(2) la *règle d'inférences de détachement* (ci-après : DET) n'est pas autonome. Elle fait usage d'au moins deux *inscriptions*. Par conséquent, la *solution* doit pouvoir parcourir toutes les *inscriptions* préalablement introduites dans le *SL* pour les comparer à celle qu'il s'agit de traiter ;

(3) la *règle d'inférences de distribution des quantificateurs* (ci-après : DIS) est autonome. Elle ne nécessite qu'une *généralisation* sur laquelle elle peut venir s'appliquer ;

(4) les *règles d'inférences de substitution* (ci-après : SUB) ne sont pas autonomes puisqu'elles nécessitent, en particulier, un substituant ;

(5) les *directives d'extensionalité* (ci-après : EXT) ne sont pas autonomes puisqu'elles doivent accéder aux *catégories syntaxico-sémantiques* que le *SL* contient à un moment donné ;

(6) la métarègle de la commutation des arguments d'une *fonction logique* (COM) est autonome. Elle ne nécessite de disposer que d'une *inscription* ;

(7) la métarègle de l'introduction de la *biconditionnelle* (INT) n'est pas autonome, elle nécessite deux *inscriptions* pour être appliquée ;

(8) et, la métarègle d'associativité (ASS) est autonome, puisqu'elle n'opère que sur une *inscription.*

Nous allons considérer les sept règles comme bloc d'exécution (par exemple : nos GO_i destinées aux *SL*) que peut utiliser le moteur de stratégie, les règles autonomes : DIS, COM et ASS et les règles qui ne sont pas autonomes : DET, SUB, EXT et INT.

A titre d'analogie, un moteur de stratégie détermine la performance d'un joueur. Les règles qui indiquent comment une pièce d'échec se déplace sur un échiquier sont comparables aux règles d'un *SL*. L'ensemble des déplacements qui a pour objectif la victoire est de l'ordre de la stratégie. Cette stratégie se développe en étapes qui chacune d'elles a une « cible » à atteindre, comme la capture d'une pièce. Pour mieux comprendre la notion de cible, prenons le schéma d'une *fonction logique* à laquelle nous voulons appliquer la *règle d'inférences de détachement* :

≡(**≡(<w><x>)**≡(<y><z>))

Si l'objectif est d'obtenir la deuxième équivalence, ≡(<y><z>), la cible d'une étape de la stratégie et de trouver ou de construire, à partir de ce que le *SL* connaît préalablement, la première équivalence, **≡(<w><x>)**. Donnons-nous une suite d'actions possibles, un script d'actions, qui pourrait répondre à notre cible :

(1) en parcourant l'ensemble des *inscriptions* du *SL*, nous trouvons non seulement le schéma, mais aussi la forme exacte de la première équivalence. Dans ce cas, on applique la DET et la cible est atteinte. Sinon,

(2) nous ne trouvons qu'un ou plusieurs schémas et dans ce cas nous essayons d'appliquer DIS en parcourant le *SL*, pour transformer les *généralisations* en *fonctions logiques* et essayons de l'appliquer comme première équivalence à la DET. Sinon,

(3) nous ne trouvons qu'un ou plusieurs schémas et dans ce cas nous essayons d'appliquer SUB en essayant d'utiliser les *termes* de la cible comme substituant. Comme pour ce qui précède, on essaie d'appliquer ensuite la DET. Sinon,

(4) nous essayons la combinaison de DIS et SUB. Sinon,

(5) nous pouvons procéder de la même manière avec les métarègles ;

(6) etc.

Ces étapes constituent le script d'un module de stratégie. Il est basé sur une séquence d'applications des règles. Donnons-nous un élément d'une stratégie : à une *inscription* traitée à un moment donné, nous dégageons une cible d'une règle qui n'est pas autonome et que nous voulons appliquer. Nous exécutons ensuite, dans un premier temps, toutes les règles autonomes et, dans un deuxième temps, toutes les règles non autonomes, en parcourant l'ensemble des *inscriptions* du *SL*, jusqu'à ce que la cible soit atteinte ou jusqu'à ce qu'il soit nécessaire de déclarer forfait. Nous représentons graphiquement l'algorithme de ce module de stratégie de la façon suivante :

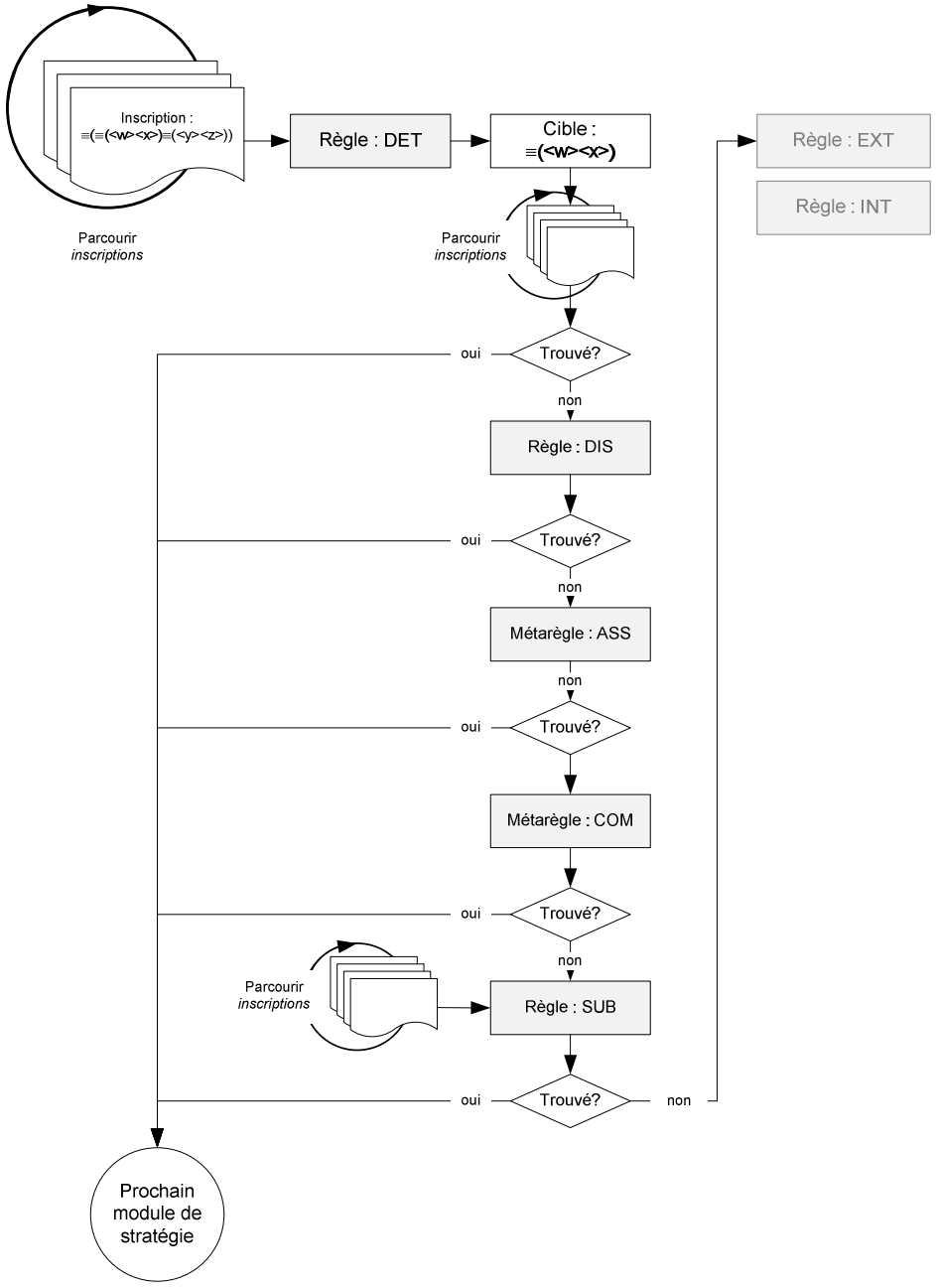


Figure 62. Exemple d'un module de stratégie

Nous insistons sur le fait que nous cherchons à proposer des outils d'aide informatisés et non pas une solution qui fait de la *déduction*. Nous aurons, donc, besoin en addition aux règles (DIS, COM et ASS, DET, SUB, EXT et INT) de blocs d'exécution comme :

- IUQ : soumettre une question à l'interface utilisateur, attendre une sélection ou une option de l'utilisateur et continuer en fonction de la réponse ;
- CIB : déterminer la cible d'une règle qui n'est pas autonome ;
- PIN : chercher la prochaine *inscription* pour une opération ;
- CSB : chercher un autre substituant dans un ensemble (acceptable) de *signes* ;
- SCR : passer à un autre script ;
- etc.

Définissons la notion de script comme une table à quatre colonnes :

(1) le numéro du bloc à exécuter ;

(2) le bloc à exécuter ;

(3) l'action à faire si le bloc à exécuter a réussi ;

(4) l'action à faire si le bloc à exécuter ne réussit pas.

Pour chaque cellule de la troisième et quatrième colonne, si plusieurs choix peuvent être suivis, nous ajoutons, à l'action à prendre, un niveau de préférence (un poids) facilitant la sélection du bloc à exécuter suivant. Nous pouvons maintenant élaborer le script de la *Figure 62, p. 183 :*

	Bloc à exécuter	**Si réussi, exécuter**	**Si pas réussi, exécuter**
1.	DET, CIB, PIN	(1) SCR (2) IUQ	(1) aller à 2.
2.	DIS	(1) SCR (2) IUQ	(1) aller à 3.
3.	ASS	(1) SCR (2) IUQ	(1) aller à 4.
4.	COM	(1) SCR (2) IUQ	(1) aller à 5.
5.	SUB, CBS, PIN	(1) SCR (2) IUQ	(1) IUQ (2) aller à 2. (3) aller à 6.
6.	INT	(1) SCR (2) IUQ	(1) aller à 2. (2) aller à 7. (3) IUQ
7.	SRC	(1) DET	(1) SUB_CHG (2) IUQ
…	…	…	…

En enchaînant les scripts de module de stratégie et en interrogeant l'utilisateur de temps à autres, nous pouvons imaginer aboutir à l'objectif que nous pouvons nous donné d'une *déduction*.

7.7 Conclusion

Tout au long de notre développement, à part l'exemple de grammaire générative et transformationnelle traitant du chat, notre GO_c (*5.7 Programmation et analyse détaillée de la GOc, p. 136*), nous avons élaboré le *LRE = L(GRE)* et le logiciel *RE* en fonction du cahier des charges destiné au *SL*. Dans ce chapitre, nous donnons des exemples d'extensions de notre *domaine de connaissances* (les tableaux de bord) et de domaines applicatifs (comme le projet *OWL*) à d'autres. Ces aspects montrent que le *LRE* peut s'étendre à un vaste univers de traitements.

En apportant quelques améliorations, nous pouvons envisager de disposer d'un *LRE* et d'une logiciel *RE* destiné à un large public.

A partir de nos GO_i, le développement d'un logiciel comme éditeur et interpréteur du *LO*, donc, spécialisé pour les langages des *SL*, auquel on apporterait la fonction de moteur de stratégie, nous apparaît comme une valeur ajoutée à l'état de nos travaux.

8. Bilan et conclusion de la recherche

Dans ce dernier chapitre, nous faisons le bilan de notre recherche en consolidant les résultats que nous avons obtenus.

Transposition, *système*, *déduction*, langage et grammaire sont les notions les plus utilisées dans notre ouvrage. Mais, dû au nombre de niveaux et de domaines traités, ce sont les concepts de langage et de grammaire qui présentent le plus haut risque de confusions.

Nous nous sommes donné comme objectif de *formaliser* un *domaine de connaissances*, celui des tableaux de bord. Pour atteindre la première partie de notre objectif, nous avons sélectionné la théorie des ensembles et un des *SL*, l'*ontologie*. Dans une première étape nous avons dû approcher le processus de *transposition* :

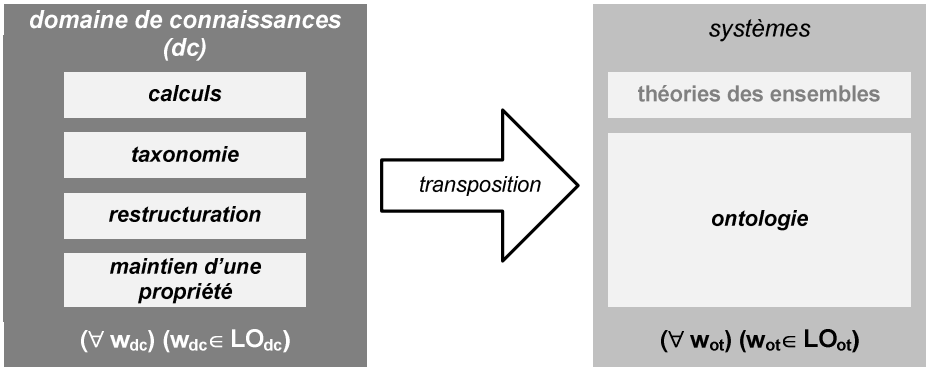


Figure 63. Transposition du *domaine de connaissances* à l'*ontologie*

Le *domaine de connaissances* est formé d'un langage, constitué par une infinité de *phrases*, w_{dc}, appartenant au langage, LO_{dc}. L'*ontologie* est constituée d'un ensemble infini de *phrases*, w_{ot}, qui appartient au langage, LO_{ot}. Les *axiomes* de la *protothétique* et de l'*ontologie* sont, par exemple, des phrases de LO_{ot}. Les *règles d'inférences* des *SL* indiquent comment une *thèse*, aussi une *phrase* de LO_{or} doit être correctement construite. Donc, ces règles fonctionnent comme une sorte de grammaire de LO, une GO_i. Dans la deuxième partie de notre objectif, nous avons voulu traiter les *définitions inductives*, les *catégories syntaxico-sémantiques* et le calcul propositionnel. Une dimension métalinguistique qui vient se greffer sur les langages LO_{dc} et LO_{ot} (voir la section : *5.10 Liste des sous-langages de LO et des sous-grammaires de GO*, p. 147* qui donne le classement des langages et des grammaires concernées) :

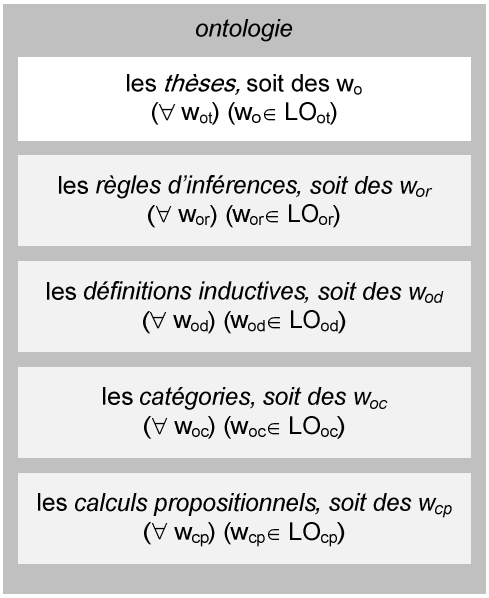


Figure 64. Constituants de l'*ontologie*

L'union, $LO_{dc} \cup LO_{ot} \cup LO_{or} \cup LO_{od} \cup LO_{oc} \cup LO_{cp} \cup \ldots$ constitue le langage « ontologique » que nous avons nommé : *LO*, sachant que $LO = L(GO)$ et sachant que $GO = GO_1 \cup GO_2 \cup \ldots \cup GO_i \cup GO_{i+1} \cup \ldots$ et que $GO \subset GO^*$.

Nous avons montré d'une part, que l'*ontologie* répondait à notre besoin de *transposition* et, d'autre part, qu'à l'aide de la *déduction*, nous pouvions traiter nos *restructurations*.

Afin de faciliter le travail des contrôleurs de gestion, nous nous sommes proposé d'automatiser une partie de la mise en œuvre des *transpositions* et des *restructurations* de notre *domaine de connaissances*. A partir d'un cahier des charges, nous avons développé un logiciel prototype, le *RE* (V4.4). Comme les connaissances informatiques actuelles ne permettent pas de « traduire » directement le *LO* en *solution* informatique, nous avons dû définir un nouveau langage intermédiaire : le *LRE*. Comme nous l'avons vu dans le cadre des *LF*, langage ne va pas sans grammaire. Il a fallu donc définir une grammaire, *GRE*, qui gouverne les *phrases* du *LRE*. C'est la *GRE* qui constitue la partie intrinsèque du logiciel *RE*, soit $LRE = L(GRE)$.

Nous avons défini le *LRE* comme le langage grammatical de toutes les grammaires, GO_i, qui gouvernent les *phrases* de *LO*, soit $LO = L(GO^*)$. Par conséquent, nous ajoutons une couche linguistique dont nous devons tenir compte :

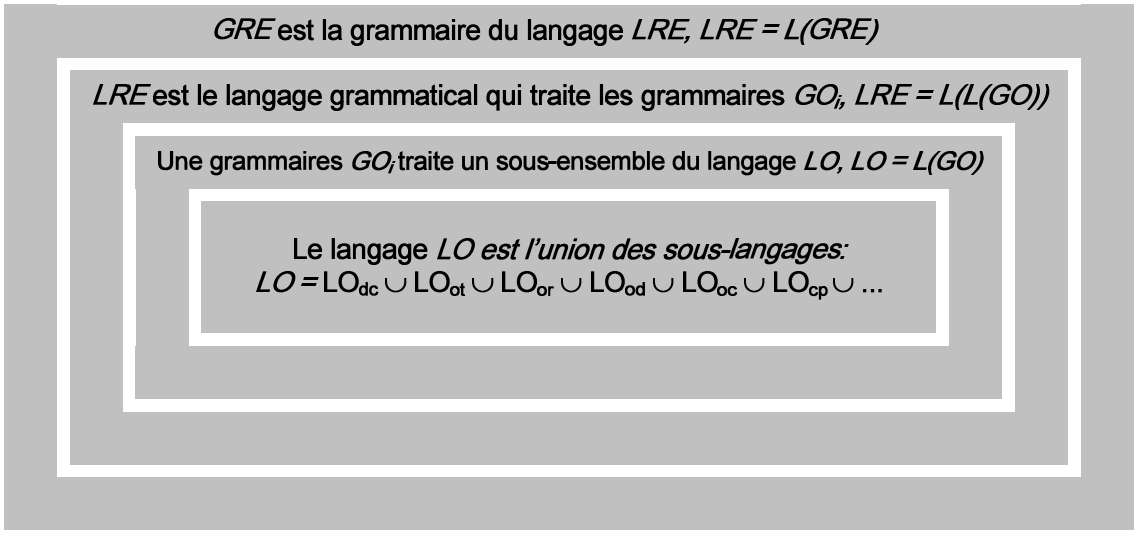


Figure 65. Les différents niveaux de langage de notre recherche

Notre logiciel *RE* est piloté par la *GRE* qui contrôle une quelconque GO_i qui peut lui être soumise. Une fois la compilation de la GO_i réalisée, le *RE* peut assimiler une ou plusieurs *phrases* de *LO* et exécuter le processus itératif de réécriture sur cette ou ces dernières *phrases*.

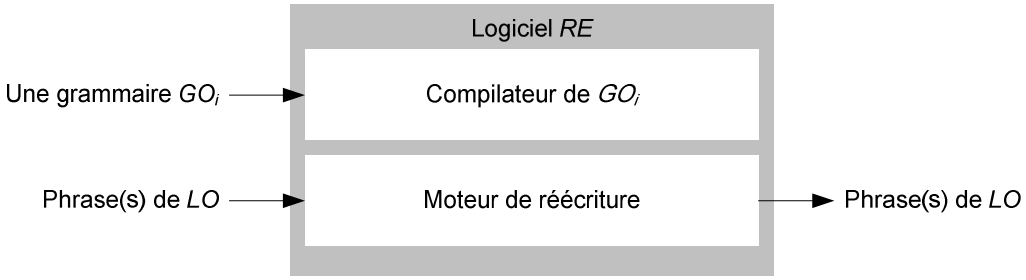


Figure 66. Processus de traitement du *RE*

En résumé et comme synthèse, tenant compte de notre perspective de développement d'un moteur de stratégie pour la déduction, nous avons :

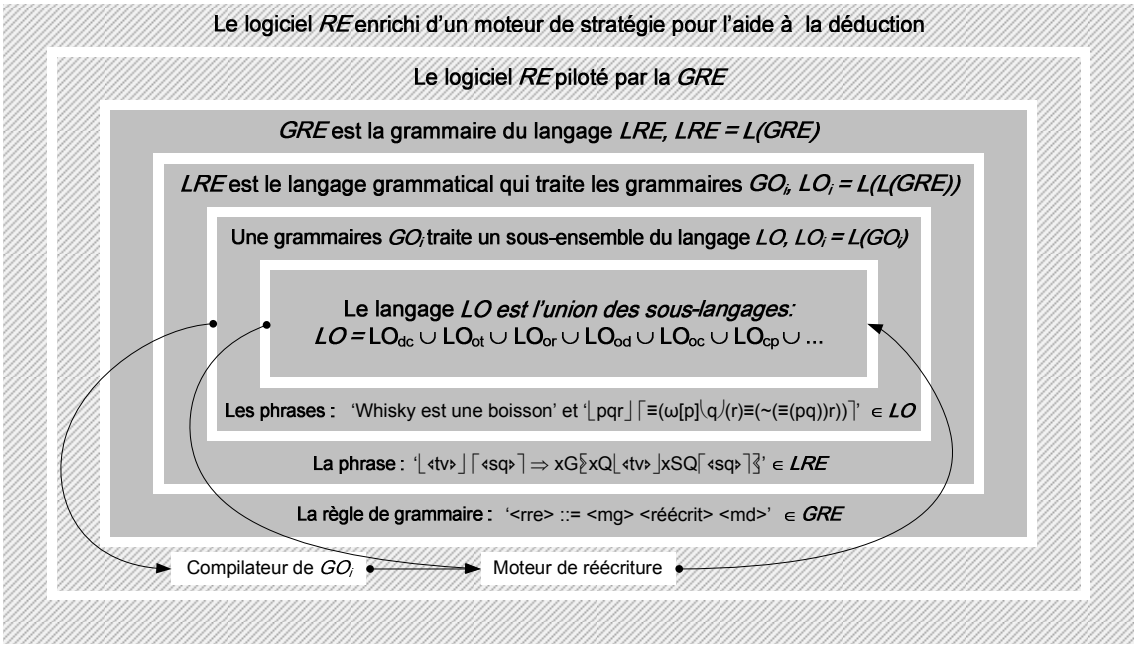


Figure 67. Synthèse

Nous avons pu montrer que la troisième partie de notre objectif, à savoir mettre au défi le *LRE = L(GRE)* et le *RE* avec les *SL*, a été atteinte.

Nos questions et hypothèses n'ont la prétention de couvrir que le premier pas d'une recherche beaucoup plus large. Nous avons fait allusion au potentiel qu'elle offre pour des nouveaux développements et des nouvelles perspectives. Etant données les réponses que nous avons obtenues à nos questions et à nos hypothèses, nous pouvons conclure que cette recherche a abouti. Elle a fait la preuve de la faisabilité escomptée à travers le *LRE* et peut servir de fondation pour une deuxième étape qui pourrait, par l'apport d'un moteur de stratégie, non seulement servir aux contrôleurs de gestion, mais aussi aux logiciens eux-mêmes.

* * *

Bibliographie

The Protégé Ontology Editor and Knowledge Acquisition System. (2011, April). (S. U. Medicine, Producer, & Standford University) Retrieved Juin 19, 2011, from Protégé: http://protege.standford.edu

ABITEBOUL, S., & GRUMBACH, S. (1987, Septembre). COL: A Logic-based Language for Complex Objects. *INRIA - Rapports de Recherche*(714), pp. 1-31.

ABRAHAM, M. (1995). *Analyse sémantico-cognitive des verbes de mouvement et d'activité; Contributions méthodologique à la constitution d'un dictionnaire informatique des verbes.* Paris, France: Thèse de doctorat EHESS.

ALAGIC, S. (1989). *Object-Oriented Database Programming* (Texts and Monographs in Computer Science ed.). New York, USA: Springer-Verlag New York Inc.

ASF+SDF Meta-Environment contributors. (2006). *The Meta-Environment - WebHome*. (A. M.-E. contributors, Editor, & A. M.-E. contributors, Producer) Retrieved juin 1, 2011, from The Meta-Environment: http://www.meta-environment.org/Meta-Environment/WebHome

AUILLANS, P. (2005). *Modélisation de réseaux sémantiques par des hypergraphes et applications* (Vol. Thèse de doctorat). Bordeaux, France: LaBRI, Université Bordeaux 1.

AUTEBART, J.-M. (1994). *Théorie des langages et des automates.* Paris, France: Masson.

AUTHIER, M., & LEVY, P. (1992). *Les arbres de connaissances.* Paris, France: Editions La Découverte.

AYACHE, M., & FLORY, A. (1996). *Approche Orientée Objet, Concepts et utilisation* (P.I.Q. Poche ed.). Paris, France: Ed. ECONOMICA.

BAADER, F., & NIPKOW, T. (1998). *Term Rewriting and All That.* Cambridge: Cambridge University Press.

BACHIMONT, B. (2000). *Engagement sémantique et engagement ontologique : conception et réalisation d'ontologies en Ingénieries des connaissances.* Paris, France: Eyrolles.

BACKHOUSE, R. C. (1979). *Syntax of Programming Languages, Theory and Practive.* (C. Hoare, Ed.) London, Angleterre: Prentice-Hall Internationa.

BARENDREGT, H. (1984). *The Lambda Calculus, Its Syntax and Semantics* (Vols. 103, in Studies in Logic and the Foundations of Mathématics). Amsterdam, North-Hollande: Elsevier Science Publishing Company.

BARENDREGT, H. (1991). *Lambda Calculi With Type* (Vol. Handbook of logic and computer science II). (S. AMBRAMSKI, D. GABBAY, & T. MAIBAUM, Eds.) Oxford Press University.

BAUDRILLARD, J. (1968). *Le système des objets.* Paris: Editions Gallimard.

BENZECRI, J.-P., & Collaborateurs. (1980). *Pratique de l'Analyse des Données - 1. Analyse des correspondances, exposé élémentaire* (Vol. 1). (Dunod, Ed.) Paris, France: Bordas.

BENZECRI, J.-P., & Collaborateurs. (1981). *Pratique de l'analyse des données - 3. Linguistique et lexicologie* (Vol. 3). (Dunod, Ed.) Paris, France: Bordas.

BENZECRI, J.-P., BASTIN, C., BOURGARIT, C., & CAZES, P. (1980). *Pratique de l'analyse de données - 2. Abrégé théoriques, études de cas modèle* (Vol. 2). (Dunod, Ed.) Paris, France: Bordas.

BERGE, C. (1983). *Graphes* (3e ed.). (Gauthier-Villars, Ed.) Paris, France: Bordas.

BIBEL, W., DAVIS, M., EDER, E., EISINGER, N., FITTING, M., HODGES, W., et al. (1993). *Handbook of Logic in Artificial Intelligence and Logic Programming - Logical Foundation* (Vol. 1). (D. GABBAY, C. HOGGER, & J. ROBINSON, Eds.) New York, U.S.A.: Oxford University Press Inc.

BIEDERMAN, I., BIZZI, E., DRETSKE, F., GALIANA, H., GOLDMAN, A., HOLLERBACH, J., et al. (1990). *An Invitation to Cognitive Science - Visual Cognition and Action* (Vol. 2). (D. OSHERSON, S. KOSSLYN, & J. HOLLERBACH, Eds.) Cambridge, Masschussets, U.S.A.: Masschussets Institute of Technology.

BISKRI, I. (1995). *La Grammaire Catégorielle Combinatoire Applicative dans le cadre de la Grammaire Applicative et Cognitive.* Paris, France: Thèse de doctorat EHESS.

BLACHE, P. (2001). *Les grammaires de propriétés (des contraintes pour le traitement automatique des langues naturelles).* Paris, France: HERMES Science Publications.

BLOCK, N., CAREY, S., CHURCHLAND, P., HOLYOAK, K., LEWOTIN, R., OSHERSON, D., et al. (1990). *An Inivitation to Cognitive Science - Thinking* (Vol. 3). (D. OSHERSON, & E. SMITH, Eds.) Cambridge, Massachusetts, U.S.A.: Massachussets Institute of Technology.

BOUVIER, A. (1982). *La théorie des ensembles* (3ème édition refondue ed.). Paris, France: Presse Universitaires de France / Que sais-je?

CafeObj Project Members. (2010, février 24). *CafeObj official homepage*. (Y. CHIBA, Editor) Retrieved juin 1, 2011, from CafeObj official homepage: http://www.ldl.jaist.ac.jp/cafeobj/index.html

CASANOVA, G. (1972). *L'algèbre de boole* (3e ed.). Paris, France: Presses Universitaires de France (Que sais-je ?).

CHANDRASEKARAN, B., JOSEPHSON, J., & BENJAMINS, V. (2011, June 11). *AITopics / Ontologies*. (SideBar, Producer, & AAAI 2000-2010) Retrieved Juin 19, 2011, from AITopics / Ontologies: http://www.aaai.org/AITopics/pmwiki/pmwiki.php./AITopics/ontologies

CHAUVIRE, C. (1995). *Pierce et la signification - Introduction à la logique du vague.* Paris, France: Presses Universitaires de France.

CHENIQUE, F. (1974). *Comprendre la logique moderne, classes, propositions et prédicats* (Série "logique et informatique" ed., Vol. 1). Paris, France: DUNOD / BORDAS.

CHENIQUE, F. (1974). *Comprendre la logique moderne, logiques non classiques, relations et structures* (série "logique et informatique" ed., Vol. 2). Paris, France: DUNOD / BORDAS.

CHENIQUE, F. (1975). *Eléments de logique classique, L'art de penser et de juger* (Série "logique et informatique" ed., Vol. 1). Paris, France: DUNOD / BORDAS.

CHENIQUE, F. (1975). *Eléments de logique classique, L'art de raisonner, Dunod, Paris 1975.* (série "logique et informatique" ed., Vol. 2). Paris, France: DUNOD / BORDAS.

CHOMSKY, N. (1959). On certain formal properties of grammars. *Information and Control, 2*(2).

CHOMSKY, N. (1967). *Le langage et la pensée.* Paris, France: Payot.

CHOMSKY, N. (1971). *Aspect de la théorie syntaxique* (Ordre Philosophique ed.). Paris, France: Seuil.

COLMERAUER, A. (1970, Septembre). Les systèmes-Q ou un formalisme pour analyser et synthétiser des phrases sur ordinateur. (C. N. Canada, Ed.) *Publication Interne*(43), pp. 1-46.

COLMERAURER, A. (1996). *Les bases de Prolog IV.* Université de Marseille. Marseilles: Laboratoire d'informatique de Marseille.

CORI, R., & LASCAR, D. (1993). *Logique mathématique - Cours et exercices - II. Fonctions récursives, théorème de Go¨del, théorie des ensembles, théorie de modèles* (Vol. 2). Paris, France: Masson.

CORI, R., & LASCAR, D. (1994). *Logique mathématique - Cours et exercices - 1. Calcul propositionnel, algèbres de Boole, calcul des prédicats* (Vol. 1). Paris, France: Masson.

CORMEN, T., LEISERSON, C., RIVEST, R., & STEIN, C. (2002). *Introduction à l'algorithmique* (2e ed.). Paris, France: Dunod.

COX, B. J. (1987). *Object Oriented Programming, An Evolutionary Approach* (2ème ed.). (A.-W. P. Company, Ed.) Reading, Massachusetts, USA: Productivity Products International, Inc.

COX, E. (1997). *La logique floue.* (M. CLERC, Trans.) Paris, France: International Thomson Publishing France.

D'ARCUS, B., & GIASSON, F. (2009, November 4). *Bibliographic Ontology Specification.* Retrieved Juin 19, 2011, from The Bibliographic Ontology: http://bibliontology.com/specification

DEHORNOY, P. (2000). *Mathématiques de l'informatique.* Paris, France: Dunod.

DELEDALLE, G. (1987). *Charles S. Peirce, phénoménologue et sémioticien.* Amsterdam/Philadelphia: John Benjamins Publishing Company.

DELEDALLE, G. (1990). *Lire Peirce Aujourd'hui.* Burxelles, Belgique: De Boeck-Wesmael, s.a.

DELSARTE, P., & THAYSE, A. (2001). *Logique pour le traitement de la langue naturelle (application à la langue française).* Paris, France: Hermes Science Publications (Europe Ltd).

DESCLES, J.-P. (1990). *Langages applicatifs, langues naturelle et cognition.* Paris, France: Hermès.

DESCLES, J.-P. (n.d.). Réseaux sémantiques : La nature logique et linguistiques des relateurs. *Langages*(87), pp. 55-87.

DOWEK, G. (1995). *La logique.* Evreux, France: Dominos / Flammarion.

DUBOIS, J. (1969). *Grammaire structurale du français : la phrase et les transformations* (langue et langage ed.). Paris, France: Librairie Larousse.

DUBOIS, J., & LAGANE, R. (1973). *La nouvelle grammaire du français.* Paris, France: Librairie Larousse.

EUZENAT, J., & SHVAIKO, P. (2007). *Ontology Matching.* New York, USA: Springer.

FISHBURN, P. (1973). *Les mathématiques de la décision.* (UNESCO, Ed., & E. COHEN, Trans.) Paris, La Haye: Gauthier-Villars, Mouton.

FORSTER, K., GARRETT, M., HALLE, M., HIGGINBOTHAM, J., LARSON, R., LASNIK, H., et al. (1990). *An Invitation to Cognitive Science - Language* (Vol. 1). (D. OSHERSON, & H. LASNIK, Eds.) Cambridge, Massachusetts, U.S.A.: Massachussets Institute of Technology.

FREGE, G. (1971). *Ecrits logiques et philosophiques.* Paris, France: Editions du Seuil.

FURST, F., & TRICHET, F. (2006, Avril 20). *Axiom-based ontology matching: a method and an experiment.* Retrieved juin 19, 2011, from archives-ouvertes: http://hal.archives-ouvertes.fr/docs/00/06/63/56/PDF/RR_05.02.pdf

GALLAIRE, H. (1976). *Techniques de compilation - Méthodes d'Analyse Syntaxiques.* Toulouse, France: CEPADUES-EDITIONS.

GENOUVRIER, E., & PEYTARD, J. (1970). *Linguistique et enseignement du français.* Paris, France: Libriarie Larousse.

GESSLER, N. (2005). *Introduction à l'oeuvre de S. Lesniewski, Fascicule III : La méréologie* (292 ed.). (CdRS, Ed.) Neuchâtel, Suisse: Université de Neuchâtel.

GESSLER, N. (2007). *Introduction à l'oeuvre de S. Lesniewsky, Fascicule V: Lesniewski lecteur de Frege.* Neuchâtel, Suisse: CdRS, Université de Neuchâtel.

GESSLER, N., JORAY, P., & DEGRANGE, C. (2005). *Le logicisme catégoriel.* Neuchâtel, Suisse: Centre de Recherches Sémiologiques.

GINISTI, J.-P. (1988). *Présentation de la logique comibatoire en vue de ses applications* (Vols. 103, in Mathématiques et Sciences Humaines). (NUMDAM, Ed.) Paris, France: Centre d'analyse et de mathématiques sociales de l'EHESS.

GINSBOURG, S. (1979). *The mathematical theory of context-free languages.* New York: MacGraw-Hill.

GRAY, P. (1984). *Logic, Algebra and Databases* (Vols. 1-294). Chichester, England: Ellis Horwood Limited.

GREIMAS, A. (1966). *Sémantique structurale.* Paris, France: Librairie Larousse.

GRISWOLD, R. E. (2011). *SNOBOL4.ORG : SNOBOL4 Ressources*. (P. BUDNE, Editor) Retrieved juin 2011, from SNOBOL4: http://www.snobol4.org

GRIZE, F. (1981). *Barbara - Analyse de données informelles à l'aide de réseaux systémiques.* (F. d. Cahiers de méthodes quantitatives, Ed.) Neuchâtel, Suisse: Université de Neuchâtel.

GRIZE, J.-B. (1967). Logique. In J. PIAGET, *Logique et connaissance scientifique* (pp. 135-399). Paris, France: Editions Gallimard.

GRIZE, J.-B. (1967). *Logique, dans Logique et connaissance scientifique.*

GRIZE, J.-B. (1971). *Logique moderne, Fascicule II : Logique des propositions et des prédicats, tables de vérité et axiomatisation.* Paris, France: Mouton / Gauthier-Villars.

GRIZE, J.-B. (1972). *Logique moderne, Fascicule I : Logique des propositions et des prédicats, déduction naturelle* (2ème ed.). Paris, France: Mouton / Gauthier-Villars.

GRIZE, J.-B. (1973). *Logique moderne, Fascicule III : Implications-modalités, logiques polyvalentes, logique combinatoire, ontologie et méréologie de Lesniewski.* Paris, France: Mouton / Gauthier-Villars.

GRIZE, J.-B. (1990). *Logique et langage.* Gap, France: Orphys.

HARRISON, M. A. (1978). *Introduction to Formal language theory.* Addison-Wesley.

HOPCROFT, J. E., & ULLMANN, J. D. (1979). *Introduction to automata theory,langauges and computation.* Addison-Wesley.

HOPCROFT, J., & ULLMAN, J. (1969). *Formal Languages and their Relation to Automata.* Reading, Mass.: Addison-Wesley.

HOTTOIS, G. (1989). *Penser la logique - Un introduction technique, théorique et philosophique à la logique formelle.* Bruxelles, Belgique: De Boeck-Wesmael S.A.

HULL, R., & KING, R. (1987, September). Semantic Database Modeling: Survey Application, and Research Issues. *ACM Computing Surveys, 19*(3), pp. 201-260.

INRIA. (2011). *Le langage Caml: Accueil.* (INRIA, Editor) Retrieved juin 1, 2011, from Le langage Caml: http://caml.inria.fr/index.fr.html

ISAAC, A. (2005). *Conception et utilisations d'ontologies pour l'indexation de documents audiovisiels* (Vol. Thèse de doctorat). (LaLIC, Ed.) Paris, France: Université Paris IV.

JAKOBSON, R. (1963). *Essais de linguisitique générale* (Editions de Minuit ed.). (N. RUWET, Trans.) Paris, France: Editions de Minuit.

JAKOBSON, R. (1973). *Essais de linguistique générale - Rapport internes et externes du langage* (Vol. 2). Paris, France: Les Editions de Minuit.

JAMBU, M. (1978). *Classification automatique pour l'analyse de données - 1. Méthodes et algorithmes* (Vol. I). (Dunod, Ed.) Paris, France: Bordas.

JAMBU, M., & LEBEAUX, M.-O. (1978). *Classification automatique pour l'analyse de données 2. Logiciels* (Vol. II). (Dunod, Ed.) Paris, France: Bordas.

JORAY, P. (2005). *La quantification dans la logique moderne.* Paris, France: L'Harmattan.

JURAFSKY, D., & MARTIN, J. (2000). *Speech and Language Processing - An Introduction to Natural Language Processing, Computational Linguistic, and Speech Recognition.* Upper Saddle River, New Jersey, U.S.A.: Prentice-Hall, Inc.

KRIVINE, J.-L. (2007). *Théorie des ensembles, Nouvelle bibliothèque mathématique* (2e ed., Vol. 5). Paris, France: Cassini.

LAFOURCADE, M., & PRINCE, V. (2004). Modélisation de l'Hyperonymie de Réseaux Sémantiques et de Vecteurs Conceptuels. *JADT'04; Journées Internationnales d'Analyse Statistiques de Données Textuelles, Louvain-la-Neuve* (pp. 1-8). Montpellier: LiRMM : Laboratoire d'Informatique, de Robotique et Microélectronique de Montpellier.

LASSAIGNE, R., & ROUGEMONT (DE), M. (1996). *Logique et complexité.* Paris, France: Hermès.

LE PRIOL, F. (2000). *Extraction et capitalisation automatiques de connaissances à partir de documents textuels. SEEK-JAVA : identification et interprétation de relations entre concepts.* (C. LaLIC, Ed.) Paris, France: Université Paris-Sorbonne.

LEGUERN, M. (1973). *Sémantique et la métaphore et de la métonymie.* Paris, France: Librairie Larousse.

LERESCHE, G. (1976). *Introduction mathématique à la logique.* (D. PERRET, Ed.) Lausanne, Suisse: Spes S.A. (DUNOD).

LEŚNIEWSKI, S. (1989). *Sur les fondements de la mathématiques. Fragments (Discussions préalables, méréologie, ontologie).* (G. Kalinowski, Trans.) Paris: Hermès.

LEŚNIEWSKI, S. (1992). *Collected Works I, II.* (S. Surma, J. Srzednicki, & V. Rickey, Eds.) Varsovie, Pologne: D.I. Barnett.

LEVY, G. (1994). *Algorithme Combinatoire, Méthodes Constructives.* (P. BURCH, Ed.) Paris, France: DUNOD.

LIRET, A. (2000). *Intégration de Mécanismes de Réécriture dans un Système de Satisfaction de Contraintes* (Vol. Thèse de doctorat). Paris, France: Université Paris VI.

LORIA-CNRS. (2011). *Elan Home Page*. (LORIA, Editor, & CNRS) Retrieved Juin 01, 2011, from ELAN: Rule based programming using strategic rewriting: http://elan.loria.fr/overview/elan-page-description.html

MAKOVELSKI, A. (1978). *Histoire de la logique.* (G. DUPOND, Trans.) Moscou, U.R.S.S.: Editions du Progès.

MARTINET, A., & COLLABORATEURS. (1968). *Le langage.* Paris, France: Editions Gallimard.

McGUINESS, D. L., & VAN HARMELEN, F. (Eds.). (2004, February 10). *OWL Web Ontology Language*. Retrieved Mars 1, 2011, from www.W3.org/owl-features/: http://www.W3.org/owl-features/

Members of Maude Team. (2011). *Maude*. (C. S. Laboratory, Producer, & University of Illinois at Urbana-Champaign) Retrieved juin 1, 2011, from The Maude System: http://maude.cs.uiuc.edu

MENDELSON, E. (1987). *Introduction to Mathematical Logic* (3e ed.). Belmont, California, U.S.A.: Wadworth, Inc.

MIEVILLE, D. (1984). *Un développement des systèmes logiques de S. Lesniwski. Protothétique-Ontologie-Méréologie.* Berne, Francfort/M., New York, Suisse: P. Lang.

MIEVILLE, D. (1991). *Introduction à la théorie des systèmes formel (Première partie)* (2 ed., Vol. 1). Neuchâtel, Suisse: CdRS, Université de Neuchâtel.

MIEVILLE, D. (1991). *Introduction à la théorie des systèmes formels (Deuxième partie)* (2 ed., Vol. 2). Neuchâtel, Suisse: CdRS, Université de Neuchâtel.

MIEVILLE, D. (1991). *La négation, une étude logique.* Neuchâtel, Suisse: CdRS, Université de Neuchâtel.

MIEVILLE, D. (2004). *Introduction à l'oeuvre de S. Lesniewski, Fascicule II: L'ontologie* (CdRS ed.). Neuchâtel, Suisse: Université de Neuchâtel.

MIEVILLE, D. (2007). *Introduction à l'oeuvre de S. Lesniewski, Fascicule I : La protothétique* (2ème ed.). (CdRS, Ed.) Neuchâtel, Suisse: Université de Neuchâtel.

MIEVILLE, D. (2009). *Introduction à l'oeuvre de S. Lesniewski, Fascicule VI: La métalangue d'une syntaxe inscriptionnelle.* Neuchâtel: CdRS, Université de Neuchâtel.

NOY, N., & MCGUINNESS, D. (2007, May 8). *Ontology Development 101: A Guide to Creating Your First Ontology*. Retrieved Juin 19, 2011, from http://protege.stanford.edu/publications/ontology-development: http://protege.stanford.edu/publications/ontology-development/ontology101-noy-mcguinness.html

OHLEBUSCH, E. (2002). *Advanced Topics in Term Rewriting.* Springer-Verlag.

PASCU, A. (2004). *La Logique de la Détermination d'Objets.* (E. D. Langages, Ed.) Paris, France: Université Paris IV.

PASSANT, A. (2009). *Technologie du WEB sémantique pour l'Entreprise 2.0* (Vol. Thèse de doctorat). Paris, France: Université Paris IV.

PEETERS, M. (2006). *Introduction à l'oeuvre de S. Liesniewski, FasciculeIV: L'oeuvre de jeunesse.* Neuchâtel, Suisse: CdRS, Université de Neuchâtel.

PICHON, J. (1989). *Théorie des ensembles - Logiques des entiers.* Paris, France: Ellipses, EDITION MARKETING.

PIERCE, C. S. (1995). *Le raisonnement et la logique des choses.* (C. CHAUVIRE, P. THIBAUD, & C. TIERCELIN, Trans.) Paris, France: Les éditions du cerf.

POLLOCK, J.-Y. (1998). *Langage et cognition : introduction au programme minimaliste de la grammaire générative.* Paris, France: Presses Universitaires de France.

QUINE, W. (1963). *Set Theory and its Logic* (revised edition 1967 ed.). Cambridge, Massachuset, U.S.A.: The Belknap Press of Harward University Press.

RENAUD, F. (1996). *Sémantique du temps et lambda-calcul.* Paris, France: Presse Univeristaires de France.

RICKEY, F. V. (1985). Interpretation of Lesniewsli'sOntoly. *Dialectica,* p. 39.3.

ROUILHAN, P. (1988). *Frege. les paradoxes de la représentation.* Paris, France: Editions de minuit.

ROUILHAN, P. (1996). *Russel et le cercle des paradoxes.* Paris, France: Presses Universitaires de France.

ROULET, E. (1975). *F. De Saussure, Cours de linguisitque générale.* Paris, France: HATIER.

RUWET, N. (1969). *Introduction à la grammaire générative.* Paris, France: Plon.

SABAH, G. (1990). *L'intelligence artificielle et le langage, représentation des connaissances* (2 ed.). Paris, France: Hermes.

SANCHEZ-MAZAS. (1982). Algebraic and Arithmetical Translations of Normative Systems and Applications in Legal Informatics. In A. MARTINO, & A. A. MARTINO (Ed.), *Deontic Logic, Computational Linguistics and Legal Information Systems* (Vol. 2, pp. 169-201). Amsterdam, Hollande: North-Holland Publishing Company.

SHETTY, R. T. (2008). *Enrichissement des réseaux sémantiques par la proximité de concepts* (Vol. Thèse de doctorat). Paris, France: Ecole Nationale Supérieure des Mines de Paris.

SITRI, F. (1998). *L'objet du débat, La construction des objets de discours dans des situations argumentatives orales.* Paris, France: Presses Sorbonne Nouvelle.

VAN DALEN, D., DEGEN, J., EBERT, P., ROSSBERG, M., GESSLER, N., GINISTI, J.-P., et al. (2007). *Contemporary Perspective on Logicism and the Foundation of Mathematics.* (P. Joray, Ed.) Neuchâtel: CdRS, Univesité de Neuchâtel.

VAX, L. (1982). *Lexique - logique.* Paris, France: Presses Universitaires de France.

WIELEMAKER, J. (2008). *Prolog, SWI-Prolog, Reference Manual* (5ème ed.). Amsterdam, Hollande: University of Amsterdam, Human-Computer Studies (HCS, formely SWI).

WIRTH, N. (1976). *Algorithms+Data Structures=Programs* (SERIES IN AUTOMATIC COMPUTATION ed.). Englewood Cliffs, New Jersey, U.S.A.: PRENTICE-HALL, INC.

ZAHND, J. (1998). *Logique élémentaire - Cours de base pour informaticiens.* Lausanne, Suisse: Presses polytechniques et universitaires romandes.

ZIMMERMANN, A. (2008). *Sémantiques des réseaux de connaissances : gestion de l'hétérogénéité fondée sur le principe de médiation* (Vol. Thèse de doctorat). Grenoble, France: Université Joseph Fourier - Grenoble 1.

Index

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

Z

Table des abréviations

C	Dans le *LRE* : *constante* de *chaîne* de caractères.
cr	Dans le *LRE* : *chaîne* de caractères résultante de l'exécution d'une *rre*.
cs	Dans le *LRE* : *chaîne* de caractères soumise au *filtrage de motif* d'un *mg* de *rre*.
DC	*Domaine de connaissances*.
ebf	Expression bien formée.
EBNF	Dans les *LF* : forme normale de Backus-Naur étendue.
GO	Mis pour grammaire « ontologique ». GO_i est mis pour n'importe quelle *GO*. *GO** est mis pour toutes les GO_i.
GRE	Grammaire de réécriture.
LF	Langage(s) formel(s).
LO	Langage « ontologique ».
LRE	Langage de réécriture.
md	Dans le *LRE* : membre de droite d'une *rre*.
mg	Dans le *LRE* : membre de gauche d'une *rre*.
RE	*Solution* ou logiciel de réécriture (ou en anglais : Rewrite Engine).
rre	Dans le *LRE* : règle de réécriture.
SF	Système(s) formel(s) du 1er ordre.
SL	Système(s) leśniewskien(s), logique(s) de Leśniewski.
V	Dans le *LRE* : une *variable* de *chaîne* de caractères.

ANNEXES

Exemples de grammaires GO_i

Le chat, version 1

↳GRAM↲
☞GRAMNAME⇒LECHAT☜
✎ version 1
✎ But de la re-grammaire : analyser une phrase rudimentaire, construire sa structure générative
✎ et lui faire subir une transformation en préfixant le complément circonstanciel de lieu, soit :
✎ en entrée : "le chat boit le lait dans la coupe"
✎ en sortie : "dans la coupe, le chat boit le lait"

↳PARAMS↲
☞MAXGRAMLOOP⇒10☜
☞MAXRULELOOP⇒10☜
↳ENDPARAMS↲

↳SETS↲
☞verbetrans3persdupresent⇒"boit", "mange"☜
☞articlemasculin⇒"le"☜
☞articlefeminin⇒"la"☜
☞substantif⇒"chat", "lait", "coupe"☜
☞preplieu⇒"dans"☜
↳ENDSETS↲

↳RULES↲

✎ REMPLACER LES ESPACES TYPOGRAPHIQUES PAR UNE ENCAPSULATION : {}

✎ Traiter le premier mot de la phrase
✎ Structure de la rre : $C1_{atom}$ - Va_{atom} - $C2_{atom}$ - Vb_{atom} - $C1_{atom}$,
✎ où C1 : " et C2 : espace
✎ Attention : l'espace se voit mal dans une rre suivant la typographie
☞"‹a› ‹b›"⇒{‹a›}"‹b›"☜ ✎ re-règle n°1

✎ Traiter tous les autres mots sauf le dernier
✎ Structure de la re-règle : $C1_{atom}$ - Va_{atom} - $C2_{atom}$ - Vb_{atom} - $C3_{atom}$ - Vc_{atom} - $C4_{atom}$
✎ où C1 : {, C2 : }", C3 : espace, C4 : "
☞{‹a›}"‹b› ‹c›"⇒{‹a›}{‹b›}"‹c›"☜ ✎ re-règle n° 2

✎ Traiter le dernier mot
✎ Structure de la re-règle : Va_{atom} - $C2_{atom}$ - Vb_{atom} - $C2_{atom}$
✎ où C2 : "
☞‹a›"‹b›"⇒‹a›{‹b›}☜ ✎ re-règle n° 3

✎ RECHERCHER ET ENCAPSULER LES ARTICLES MASCULINS
✎ si l'on trouve un article masculin, on lui apporte 2 niveau d'information : déterminant et genre ;
✎ on ne traite pas le nombre
✎ la première couche d'encapsulation préfixe le genre, la deuxième la catégorie grammaticale,
✎ soit : <categorie>[<genre>[…]]
☞‹x›{‹a↳IN(articlemasculin)↲›}‹r›⇒‹x›(art[m["‹a›"]])‹r›☜ ✎ re-règle n° 4

✎ RECHERCHER ET ENCAPSULER LES ARTICLES FEMININS
✎ on procède comme pour l'article masculin
☞‹x›{‹a↳IN(articlefeminin)↲›}‹r›⇒‹x›(art[f["‹a›"]])‹r›☜ ✎ re-règle n° 5

✎ RECHERCHER ET ENCAPSULER LES SUBSTANTIFS
✎ on anticipe les constructions syntaxiques en partant du principe qu'un article précède un nom
✎ en d'autres termes, un substantif ne peut être dépisté que si il est précédé d'un article
☞‹x›(art[‹m›[‹a›]]){‹s↳IN(substantif)↲›}‹r›⇒‹x›(art[‹m›[‹a›]])(subs["‹s›"])‹r›☜ ✎re-règle n° 6

✎ RECHERCHER LES PREPOSITIONS DE LIEU
☞‹x›{‹l↳IN(preplieu)↲›}‹r›⇒‹x›(plieu["‹l›"])‹r›☜ ✎re-règle n° 7

✎ RECHERCHER LES SYNTAGME NOMINAUX
☞‹x›(art[‹m›[‹a›]])(subs[‹s›])‹r›⇒‹x›sn[art[‹m›[‹a›]]subs[‹s›]]‹r›☜ ✎ re-règle n° 8

✎ RECHERCHER UN VERBE TRANSITIF
☞‹x›sn[‹a›]{‹v›}sn[‹b›]‹r›⇒‹x›sn[‹a›](verbt3pp["‹v›"])sn[‹b›]‹r›☜ ✎ re-règle n° 9

✎ CONSTRUCTION DU SYNTAGME VERBAL ET DU COMPLEMENT D'OBJET DIRECT
☞‹x›sn[‹a›](verbt3pp[‹v›])sn[‹b›](‹r›⇒sv[sn[‹a›]verbt3pp[‹v›]cd[sn[‹b›]]](‹r›☜ ✎ re-règle n° 10

✎ CONSTRUCTION D'UN COMPLEMENT CIRCONSTANCIEL DE LIEU
☞sv[‹v›](plieu[‹l›])sn[art[‹m›[‹a›]]subs[‹s›]]‹r›⇒
sv[‹v›]cc[plieu[‹l›]sn[art[‹m›[‹a›]]subs[‹s›]]]‹r›☜ ✎ re-règle n° 11

✎ EXEMPLES DE TRAITEMENT D'ERREUR
☞‹x›{‹y›⇒Erreur : traitement incomplet↳STOP↲☜ ✎ re-règle n° 12
☞‹x›}‹y›⇒Erreur : traitement incomplet↳STOP↲☜ ✎ re-règle n° 13
☞‹x›(‹y›⇒Erreur : traitement incomplet↳STOP↲☜ ✎ re-règle n° 14
☞‹x›)‹y›⇒Erreur : traitement incomplet↳STOP↲☜ ✎ re-règle n° 15

✎ ENCAPSULER LE TOUT COMME ETANT UNE PHRASE
☞sv[‹v›]cc[‹c›]⇒ph[sv[‹v›]cc[‹c›]]☜ ✎ re-règle n° 16

✎ PRE-FIXER LA PHRASE AVEC LE COMPLEMENT CIRCONSTANCIEL
✎ suivi d'une virgule
☞ph[sv[‹v›]cc[‹c›]]⇒ph[cc[‹c›], sv[‹v›]]☜ ✎ re-règle n° 17

✎ DEPOUILLER L'EXPRESSION DE SES ENCAPSULATIONS
☞ph[‹p›]⇒‹p›☜ ✎ re-règle n° 18
☞cc[plieu["‹p›"]sn[art[f["‹a›"]]subs["‹s›"]]]‹z›⇒‹p› ‹a› ‹s›‹z›☜ ✎ re-règle n° 19
☞‹a›, sv[‹b›cd[‹c›]]⇒‹a›, ‹b›‹c›☜ ✎ re-règle n° 20
☞‹a›, ‹x›sn[art[‹b›]subs[‹c›]]‹y›⇒‹a›, ‹x›art[‹b›]subs[‹c›]‹y›☜ ✎ re-règle n° 21
☞‹a›, ‹b›art[‹c›["‹d›"]]subs["‹e›"]‹f›⇒‹a›, ‹b› ‹d› ‹e› ‹f›☜ ✎ re-règle n° 22
☞‹a›, ‹b› verb‹c›["‹d›"] ‹f›⇒‹a›, ‹b› ‹d› ‹f›☜ ✎ re-règle n° 23

↳ENDRULES↲
↳ENDGRAM↲

Le chat, version 2

↳GRAM↲
☞GRAMNAME⇒LECHATDEUX☜
✎ version 2
✎ But de la re-grammaire : analyser une phrase rudimentaire,
✎ construire sa structure générative
✎ et lui faire subir une transformation en préfixant le complément circonstanciel de lieu,
✎ Tester que le genre est le même pour les unités linguisitiques d'un syntagme nominal
✎ soit :
✎ en entrée : "le chat boit le lait dans la coupe"
✎ en sortie : "dans la coupe, le chat boit le lait"

↳PARAMS↲
☞MAXGRAMLOOP⇒10☜
☞MAXRULELOOP⇒10☜
↳ENDPARAMS↲

↳SETS↲
☞verbetrans3persdupresent⇒"boit", "mange"☜
☞articlemasculin⇒"le"☜
☞articlefeminin⇒"la"☜
☞substantifmasculin⇒"chat","lait"☜ ✎ version 2
☞substantiffeminin⇒"coupe"☜ ✎ version 2
☞substantif⇒substantifmasculin,substantiffeminin☜
☞preplieu⇒"dans"☜
↳ENDSETS↲

↳RULES↲
✎ But de la re-grammaire : analyser une phrase rudimentaire,
✎ construire sa structure générative
✎ et lui faire subir une transformation en préfixant le complément circonstanciel de lieu, soit :
✎ en entrée : "le chat boit le lait dans la coupe"
✎ en sortie : "dans la coupe, le chat boit le lait"

✎ TRAITER LES ESPACES TYPOGRAPHIQUES
☞"‹a› ‹b›"⇒{‹a›}"‹b›"☜ ✎ 1
☞{‹a›}"‹b› ‹c›"⇒{‹a›}{‹b›}"‹c›"☜ ✎ 2
☞‹a›"‹b›"⇒‹a›{‹b›}☜ ✎ 3

✎ RECHERCHER ET ENCAPSULER LES ARTICLES MASCULINS
✎ avec leur genre et se donner un marqeur : ●
☞‹x›{‹a›↳IN(articlemasculin)↲›}‹r›⇒‹x›(●art[m["‹a›"]])‹r›☜ ✎ 4

✎ RECHERCHER ET ENCAPSULER LES ARTICLES FEMININS
✎ avec leur genre et se donner un marqeur : ●
☞‹x›{‹a›↳IN(articlefeminin)↲›}‹r›⇒‹x›(●art[f["‹a›"]])‹r›☜ ✎ 5

✎ RECHERCHER ET ENCAPSULER LES NOMS MASCULINS (version 2)
✎ avec leur genre et se donner un marqeur : ●
☞‹x›{‹a›↳IN(substantifmasculin)↲›}‹r›⇒‹x›(●subs[m["‹a›"]])‹r›☜ ✎ 6

✎ RECHERCHER ET ENCAPSULER LES NOMS FEMININS (version 2)
✎ avec leur genre et se donner un marqeur : ●
☞‹x›{‹a›↳IN(substantiffeminin)↲›}‹r›⇒‹x›(●subs[f["‹a›"]])‹r›☜ ✎ 7

✎ TESTER SI POUR TOUTES LES FORMES : ARTICLE / SUBSTANTIF
✎ LE GENRE EST IDENTIQUE (version 2)
✎ utilisation de la variable miroir : ‹genre›
✎ à chaque itération réussie les marqueurs : ● sont éffacés
☞‹a›(●art[‹genre›[‹art›]])(●subs[‹genre›[‹nom›]])‹z›⇒
‹a›(art[‹genre›[‹art›]])(subs[‹genre›[‹nom›]])‹z›☜ ✎ 8

✎ TESTER S'IL Y A UNE ERREUR (version 2)
✎ une erreur se repère s'il reste des marqueurs : ●
☞‹a›(●art[‹genre1›[‹art›]])(●subs[‹genre2›[‹nom›]])‹z›⇒l'article : ‹art› n'a pas le même genre que le nom : ‹nom› !↳STOP↲☜ ✎ 9

✎ RECHERCHER LES PREPOSITIONS DE LIEU
☞‹x›{‹l›↳IN(preplieu)↲›}‹r›⇒‹x›(plieu["‹l›"])‹r›☜ ✎ 10

✎ RECHERCHER LES SYNTAGME NOMINAUX
☞‹x›(art[‹m›[‹a›]])(subs[‹g›[‹s›]])‹r›⇒‹x›sn[art[‹m›[‹a›]]subs[‹g›[‹s›]]]‹r›☜ ✎ 11

✎ RECHERCHER LE VERBE TRANSITIF
☞‹x›sn[‹a›]{‹v›}sn[‹b›]‹r›⇒‹x›sn[‹a›](verbt3pp["‹v›"])sn[‹b›]‹r›☜ ✎ 12

✎ CONSTRUCTION DU SYNTAGME VERBAL ET DU COMPLEMENT DIRECT
☞‹x›sn[‹a›](verbt3pp[‹v›])sn[‹b›](‹r›⇒sv[sn[‹a›]verbt3pp[‹v›]cd[sn[‹b›]]](‹r›☜ ✎ 13

✎ CONSTRUCTION D'UN COMPLEMENT CIRCONSTANCIEL DE LIEU
☞sv[‹v›](plieu[‹l›])sn[art[‹m›[‹a›]]subs[‹s›]]‹r›⇒
sv[‹v›]cc[plieu[‹l›]sn[art[‹m›[‹a›]]subs[‹s›]]]‹r›☜ ✎ 14

✎ TRAITEMENT D'ERREURS
☞‹x›{‹y›⇒Erreur : traitement incomplet↳STOP↲☜ ✎ 15
☞‹x›}‹y›⇒Erreur : traitement incomplet↳STOP↲☜ ✎ 16
☞‹x›(‹y›⇒Erreur : traitement incomplet↳STOP↲☜ ✎ 17
☞‹x›)‹y›⇒Erreur : traitement incomplet↳STOP↲☜ ✎ 18

✎ ENCAPSULER LE TOUT COMME ETANT UNE PHRASE
☞sv[‹v›]cc[‹c›]⇒ph[sv[‹v›]cc[‹c›]]☜ ✎ 19

✎ PRE-FIXER LA PHRASE AVEC LE COMPLEMENT CIRCONSTANCIEL
☞ph[sv[‹v›]cc[‹c›]]⇒ph[cc[‹c›], sv[‹v›]]☜ ✎ 20

✎ DEPOUILLER L'EXPRESSION DE SES ENCAPSULATIONS
☞ph[‹p›]⇒‹p›☜ ✎ 21
☞cc[plieu["‹p›"]sn[art[‹g1›["‹a›"]]subs[‹g2›["‹s›"]]]]‹z›⇒‹p› ‹a› ‹s›‹z›☜ ✎ 22
☞‹a›, sv[‹b›cd[‹c›]]⇒‹a›, ‹b›‹c›☜ ✎ 23
☞‹a›, ‹x›sn[art[‹b›]subs[‹c›]]‹y›⇒‹a›, ‹x›art[‹b›]subs[‹c›]‹y›☜ ✎ 24
☞‹a›, ‹b›art[‹g1›["‹d›"]]subs[‹g2›["‹e›"]]‹f›⇒‹a›, ‹b› ‹d› ‹e› ‹f›☜ ✎ 25
☞‹a›, ‹b› verb‹c›["‹d›"] ‹f›⇒‹a›, ‹b› ‹d› ‹f›☜ ✎ 26

↳ENDRULES↲
↳ENDGRAM↲

Préanalyse, analyse et application de la procédure définitoire

Le bloc de base qui conduit à la version analysée des *inscriptions* leśniewskiennes et correspondant au format *langage pivot*, est constitué de trois GO_i. Une grammaire principale,

ANALYSE, qui en appelle successivement deux autres PREANALYSE et PROCDEF. Nous trouvons ci-après ces trois GO_i :

Analyse

↳GRAM↲
☞GRAMNAME⇒ANALYSE☜

↳INCLUDE↲
☞PREANALYSE⇒"D:\TheseL\GramXAnalyse\GX01PreAnalyse.txt"☜
☞PROCDEF⇒"D:\TheseL\GramXAnalyse\GX02ProcDef.txt"☜
↳ENDINCLUDE↲

↳PARAMS↲
☞MAXGRAMLOOP⇒2☜
☞MAXRULELOOP⇒5☜
↳ENDPARAMS↲

↳RULES↲
🖉 entrer la formule depuis INPUT_CORPUS
🖉 ⌊‹a›⌉ dépiste si l'on a soumis une généralisation comme Cs
🖉 ≡‹a› dépiste si l'on a faire à une Cs de type formule logique
🖉 ⁅‹id›⁆ est un identificateur d'inscription comme « Axiome 3 »
☞⌊‹a›⌉⁅‹id›⁆⇒➢⁅‹id›⁆⌊‹a›⌉▶⌊‹a›⌉◀≺☜🖉[1]
☞≡‹a›⇒➢≡‹a›▶≡‹a›◀≺☜🖉[2]

🖉 PREPARER ET EXECUTER L'ANALYSE DES TERMINAUX
🖉 ET CONSTRUCTION DES ENCAPSULATIONS DES TERMES ET EXPRESSIONS
☞↳EXECUTEGRAM(PREANALYSE)↲☜🖉[3]

🖉 VERIFIER PROCEDURE DEFINITOIRE ET
🖉 TRIER LES VARIABLES DANS LES DIFFERENTES EXPRESSIONS
☞↳EXECUTEGRAM(PROCDEF)↲☜🖉[4]

☞↳STOP↲☜🖉[]
↳ENDRULES↲
↳ENDGRAM↲

Préanalyse

↳GRAM↲
☞GRAMNAME⇒PREANALYSE☜

🖉 la formule à analyser se trouve encapsulée entre :
🖉 ▶‹formule›◀, elle doit être traitée comme : ➢‹avant›▶‹formule›◀‹après›≺

↳PARAMS↲
☞MAXGRAMLOOP⇒50☜
☞MAXRULELOOP⇒400☜
↳ENDPARAMS↲

↳SETS↲

🖉 Variables
☞vp⇒"p","q","r","s"☜🖉 variables propositionnelles
☞vns⇒"A","B","C","D"☜🖉 variables de nom singulier
☞vng⇒"a","b"☜🖉 variables de nom général
☞vn⇒vns,vng☜🖉 les symboles de variable de nom
☞vp_vn⇒vp,vns,vng☜🖉 tous les symboles de variable

🖉 Fonctions Propositionnelles
☞fp2⇒"f","g","𝓔"☜🖉 fonctions propositionnelles à deux places
☞fp1⇒"h","𝓐"☜🖉 fonctions propositionnelles à une place
☞fp⇒fp2,fp1☜🖉 toutes les fonctions propositionnelles

☞vp_vn_fp⇒vp,vns,vng,fp☜🖉 tous les symboles de termes qui peuvent être quantifiés

🖉 Constantes, termes constants
☞bic⇒"≡"☜🖉 biconditionnelle
☞cp0⇒"V","F","T"☜🖉 constantes propositionnelles sans contexte
☞tc1⇒"~","¬","α","!","↦","↓","𝓔","𝓐"☜🖉 constantes unaires
☞tc2⇒"μ","Ц","ε","∧","∨","⊃","𝓔"☜🖉 constantes avec contexte binaire

☞tcn⇒"Δ","δ"☜✎ constantes n_aires
☞tcmc⇒"τ","ω"☜✎ constantes multi-contextes
☞tcac⇒tc1,tc2,tcn,tcmc☜✎ les constantes avec contexte(s) sans BIC

☞mop⇒tc1,fp1☜✎ opérateurs unaires
☞bop⇒tc2,bic,fp2☜✎ opérateurs binaires
☞xop⇒tcn,tcmc☜✎ opérateurs complexes, multi-contextes

✎ Délimiteurs
☞qg⇒"⌊"☜ ✎ délimiteur gauche de quantificateur
☞qd⇒"⌋"☜ ✎ délimiteur droit de quantificateur
☞sqg⇒"⌈"☜ ✎ délimiteur gauche de sous-quantificateur
☞sqd⇒"⌉"☜ ✎ délimiteur droit de sous-quantificateur
☞qsqg⇒qg,sqg☜ ✎ délimiteurs q et sq gauches
☞qsqd⇒qd,sqd☜ ✎ délimiteurs q et sq droits
☞qsq⇒qsqg,qsqd☜ ✎ tous les délimiteurs q et sq

✎ Parenthèses gauches et délimiteurs gauches de contexte
☞pgA⇒"("☜
☞pgB⇒"["☜
☞pgC⇒"{"☜
☞pgD⇒"<"☜
☞pgE⇒"⟮"☜
☞pgF⇒"⟨"☜
✎ ...
☞pg⇒pgA,pgB,pgC,pgD,pgE,pgF☜✎ ...

✎ Parenthèses droites et délimiteurs droits de contexte
☞pdA⇒")"☜
☞pdB⇒"]"☜
☞pdC⇒"}"☜
☞pdD⇒">"☜
☞pdE⇒"⟯"☜
☞pdF⇒"⟩"☜
✎ ...
☞pd⇒pdA,pdB,pdC,pdD,pdE,pdF☜✎ ...

✎ Enveloppes constituées de délimiteurs
☞q_evp⇒"⌊⌋"☜ ✎ enveloppe Q délimiteurs
☞sq_evp⇒"⌈⌉"☜ ✎ enveloppe SQ délimiteurs
☞par_evp⇒"()","[]","{}","<>","⌒","⟨⟩"☜ ✎... enveloppes, délimiteurs de contexte jumeaux
☞evp⇒q_evp,sq_evp,par_evp☜ ✎ tous les délimiteurs ouvrants et fermants

✎ Métasymboles
☞mi_sy⇒"•"☜✎ marqueur d'invalidité
☞dg_sy⇒"╠"☜✎ marqueur gauche d'encapsulation de décomposition
☞dd_sy⇒"╣"☜✎ marqueur droite d'encapsulation de décomposition
☞mtg_sy⇒"⎧"☜✎ marqueur gauche d'encapsulation de terminaux
☞mtd_sy⇒"⎫"☜✎ marqueur droite d'encapsulation de terminaux
☞meg_sy⇒"⎩"☜✎ meta-marqueur gauche d'encapsulation
☞med_sy⇒"⎭"☜✎ meta-marqueur droite d'encapsulation
☞meg1_sy⇒"⎨"☜✎ meta-marqueur temporaire gauche d'encapsulation
☞med1_sy⇒"⎬"☜✎ meta-marqueur temporaire droite d'encapsulation
☞meg2_sy⇒"⎡"☜✎ meta-marqueur temporaire gauche d'encapsulation d'énumération
☞med2_sy⇒"⎤"☜✎ meta-marqueur temporaire droite d'encapsulation d'énumération
☞meg3_sy⇒"⎢"☜✎ meta-marqueur temporaire gauche d'encapsulation d'énumération
☞med3_sy⇒"⎥"☜✎ meta-marqueur temporaire droite d'encapsulation d'énumération
☞meg4_sy⇒"⎣"☜✎ meta-marqueur temporaire gauche d'encapsulation d'énumération
☞med4_sy⇒"⎦"☜✎ meta-marqueur temporaire droite d'encapsulation d'énumération
☞meg5_sy⇒"⎪"☜✎ meta-marqueur temporaire gauche d'encapsulation d'énumération
☞med5_sy⇒"⎪"☜✎ meta-marqueur temporaire droite d'encapsulation d'énumération
☞trinoirg_sy⇒"▶"☜✎ meta-marqueur triangle noir gauche d'encapsulation
☞trinoird_sy⇒"◀"☜✎ meta-marqueur triangle noir droite d'encapsulation
☞triblancg_sy⇒"▷"☜✎ meta-marqueur triangle blanc gauche d'encapsulation
☞triblancd_sy⇒"◁"☜✎ meta-marqueur triangle blanc droite d'encapsulation
☞trijolig_sy⇒"➢"☜✎ meta-marqueur triangle joli gauche d'encapsulation
☞trijolid_sy⇒"➢"☜✎ meta-marqueur triangle joli droite d'encapsulation
☞boulenoire_sy⇒"●"☜✎ meta-marqueur boule noire
☞bouleblanche_sy⇒"○"☜✎ meta-marqueur boule blanche
☞flecheg_sy⇒"→"☜✎ meta-marqueur flèche gauche

☞fleched_sy⇒"←"☜✐ meta-marqueur flèche droite
☞fe_sy⇒"⁄"☜✐ meta-marqueur de fin d'encapsulation
☞espace_sy⇒" "☜✐ espace parasite
☞metaMarqueurs_sy⇒mi_sy,dg_sy,dd_sy,mtg_sy,mtd_sy,meg_sy,med_sy,meg1_sy,med1_sy,meg2_sy,
med2_sy,meg3_sy,med3_sy,
meg4_sy,med4_sy,meg5_sy,med5_sy,fe_sy,espace_sy,
trinoirg_sy,trinoird_sy,triblancg_sy,triblancd_sy,trijolig_sy,trijolid_sy,boulenoire_sy,bouleblanche_sy,fleche
g_sy,fleched_sy☜✐ tous les meta-marqueurs

☞zero_sy⇒"0"☜
☞moinsun_sy⇒"-1"☜
↳ENDSETS↲

↳RULES↲

✐ VERIFICATION DU FORMAT DE Cs
✐ **
✐ la zone, où devrait se trouver Cs, est-elle vide ?
☞➢‹avant›►◄‹apres›≺⇒
ERREUR : ➢‹avant›►_erreur_fomule_vide◄‹apres›≺, trouvé formule à analyser vide !↳STOP↲☜✐[1]
✐ vérifier le format de l'encapsulation principale « pivot » : ➢...►‹formule›◄...≺
☞➢‹avant›►‹v›◄‹apres›≺⇒➢‹avant›►‹v›◄‹apres›≺/ok☜✐[2]

✐ TRAITER LES TERMINAUX
✐ ********************************

✐ DECOMPOSITION DE LA FORMULE
✐ Une règle formée avec la fonction EXPLODE n'est exécutée qu'une seule fois !
☞➢‹avant›►‹formule›◄‹apres›≺/ok⇒
➢‹avant›►↳EXPLODE(‹formule›,"╠","╣")↲◄‹apres›≺☜✐[3]

✐ NUMEROTER LES DELIMITEURS
✐ Une règle formée avec la fonction DELIMNUM n'est exécutée qu'une seule fois !
☞➢‹avant›►‹a›◄‹apres›≺⇒➢‹avant›►↳DELIMNUM(‹a›,evp)↲◄‹apres›≺☜✐[4]

✐ ADAPTER LES ENCAPSULATIONS DES NUMEROS DE DELIMITEUR
✐ ET SUPRIMMER LES SEPARATEURS DE DECOMPOSITION DE DELIMITEURS
☞➢‹avant›►‹a›╠‹d↳IN(pg)↲›[‹i↳INTEGER↲›]╣‹z›◄‹apres›≺⇒
➢‹avant›►‹a›•tPG_‹i›[‹d›]‹z›◄‹apres›≺☜✐[5]
☞➢‹avant›►‹a›╠‹d↳IN(pd)↲›[‹i↳INTEGER↲›]╣‹z›◄‹apres›≺⇒
➢‹avant›►‹a›•tPD_‹i›[‹d›]‹z›◄‹apres›≺☜✐[6]
☞➢‹avant›►‹a›╠⌊[‹i↳INTEGER↲›]╣‹z›◄‹apres›≺⇒➢‹avant›►‹a›•tQG_‹i›⌊[⌋]‹z›◄‹apres›≺☜✐[7]
☞➢‹avant›►‹a›╠⌋[‹i↳INTEGER↲›]╣‹z›◄‹apres›≺⇒➢‹avant›►‹a›•tQD_‹i›⌋[⌊]‹z›◄‹apres›≺☜✐[8]
☞➢‹avant›►‹a›╠⌈[‹i↳INTEGER↲›]╣‹z›◄‹apres›≺⇒➢‹avant›►‹a›•tSQG_‹i›⌈[⌉]‹z›◄‹apres›≺☜✐[9]
☞➢‹avant›►‹a›╠⌉[‹i↳INTEGER↲›]╣‹z›◄‹apres›≺⇒➢‹avant›►‹a›•tSQD_‹i›⌉[⌈]‹z›◄‹apres›≺☜✐[10]

✐ TRAITER LES GENERALISATIONS
✐ **
✐ ENCAPSULER LES QUANTIFICATEURS ET LES SOUS-QUANTIFICATEURS
✐ La combinaison quantificateur et sous-quantificateur prend le même indice (généralisation)
✐ Les délimiteurs de quantificateurs et de sous-quantificateurs sont dépouillés de leurs indices
✐ Se donner un compteur de généralisation(s)
☞➢‹avant›►‹a›◄‹apres›≺⇒➢‹avant›►‹a›◄‹apres›≺/gencnt=0☜✐[11]
✐ encapsuler, marquer avec : • mis pour : à valider
☞➢‹avant›►‹a›•tQG_‹i›⌊[⌋‹b›•tQD_‹i›⌊[⌋]•tSQG_‹j›⌈[⌉‹c›•tSQD_‹j›⌉[⌈]‹z›◄‹apres›≺
/gencnt=‹numgen›⇒
➢‹avant›►‹a›•xQ_‹numgen›⌊‹b›⁄xQ_‹numgen›⌋•xSQ_‹numgen›⌈‹c›⁄xSQ_‹numgen›⌉‹z›
◄‹apres›≺/gencnt=↳PLUS(‹numgen›,1)↲☜✐[12]
✐ effacer le compteur de sous-généralisation(s)
☞➢‹avant›►‹a›◄‹apres›≺/gencnt=‹numgen›⇒➢‹avant›►‹a›◄‹apres›≺☜✐[13]

✐ ENCAPSULER LES VP (VARIABLE PROPOSITIONELLE) OU VN (VARIABLE DE NOM)
✐ EN TV (TERME VARIABLE) OU LIEURS DANS LES QUANTIFICATEURS
☞➢‹avant›►‹a›xQ_‹i›⌊‹b›╠‹v↳IN(vp_vn)↲›╣‹c›⁄xQ_‹i›⌋‹z›◄‹apres›≺⇒
➢‹avant›►‹a›xQ_‹i›⌊‹b›•xTVQ{‹v›}‹c›⁄xQ_‹i›⌋‹z›◄‹apres›≺☜✐[14]

✐ ENCAPSULER LES FP (FONCTION PROPOSITIONNELLE)
✐ EN TV (TERME VARIABLE) DANS LES QUANTIFICATEURS
☞➢‹avant›►‹a›xQ_‹i›⌊‹b›╠‹v↳IN(fp)↲›╣‹c›⁄xQ_‹i›⌋‹z›◄‹apres›≺⇒
➢‹avant›►‹a›xQ_‹i›⌊‹b›•xTVQ{‹v›}‹c›⁄xQ_‹i›⌋‹z›◄‹apres›≺☜✐[15]

✐ démarquer la généralisation dominante, indice 0

☞➢‹avant›►•xQ_0⌊‹b›/xQ_0⌋•xSQ_0⌈‹c›/xSQ_0⌉◄‹apres›≺⇒
➢‹avant›►xQ_0⌊‹b›/xQ_0⌋xSQ_0⌈‹c›/xSQ_0⌉◄‹apres›≺☜🖉[16]

🖉 TRAITER UNE FL (FONCTION LOGIQUE) / DEFINIENDUM
🖉 **********
🖉 TRAITER LES TC (TERME CONSTANT) DES FL AVEC CONTEXTE(S)
🖉 Le contexte primitif: S/SS de BIC est traité pour lui-même
🖉 Encapsuler et marquer le tcac (terme constant avec contexte), provisoirement, du definiendum
🖉 du sous-quantificateur dominant d'une thèses, sachant que l'expression du
🖉 sous-quantificateur est une fonction logique biconditionnelle primitive.
☞➢‹avant›►‹a›xSQ_0⌈╠═╣•tPG_‹j›↳INTEGER⇙›❴(❵╠‹v›↳IN(tcac)⇙›╣‹b›/xSQ_0⌉‹z›◄‹apres›≺⇒
➢‹avant›►‹a›xSQ_0⌈╠═╣•tPG_‹j›❴(❵•xTCAC❴‹v›❵‹b›/xSQ_0⌉‹z›◄‹apres›≺☜🖉[17]
🖉 se donner un compteur de contextes pour le tcac
☞➢‹avant›►‹a›◄‹apres›≺⇒➢‹avant›►‹a›◄‹apres›≺/cntctx=1☜🖉[18]
🖉 ENCAPSULER LE PREMIER CONTEXTE DU TCAC
☞➢‹avant›►‹a›•xTCAC❴‹t›↳IN(tcac)⇙›❵•tPG_‹i›❴‹r›↳IN(pg)⇙›❵‹b›•tPD_‹i›❴‹s›↳IN(pd)⇙›❵‹z›◄
‹apres›≺/cntctx=‹u›⇒
➢‹avant›►‹a›•xTCAC❴‹t›❵•xCTX_‹u›⌊xPG❴‹r›❵‹b›xPD❴‹s›❵/xCTX_‹u›⌋→/←‹z›◄
‹apres›≺/cntctx=↳PLUS(‹u›,1)⇙☜🖉[19]
🖉 ENCAPSULER LES CONTEXTES SUIVANTS DU TCAC
☞➢‹avant›►‹a›→/←•tPG_‹i›❴‹r›↳IN(pg)⇙›❵‹b›•tPD_‹i›❴‹s›↳IN(pd)⇙›❵‹z›◄‹apres›≺/cntctx=‹u›⇒
➢‹avant›►‹a›•xCTX_‹u›⌊xPG❴‹r›❵‹b›xPD❴‹s›❵/xCTX_‹u›⌋→/←‹z›◄‹apres›≺
/cntctx=↳PLUS(‹u›,1)⇙☜🖉[20]
🖉 supprimer le compteur de contextes
☞➢‹avant›►‹a›◄‹apres›≺/cntctx=‹u›⇒➢‹avant›►‹a›◄‹apres›≺☜🖉[21]
🖉 ENCAPSULER LE TCAC ET LES CTX PAR DES MARQUEURS PROVISOIRES DE FL
🖉 (FONCTION LOGIQUE)
☞➢‹avant›►‹a›•xTCAC❴‹t›❵‹b›→/←‹z›◄‹apres›≺⇒
➢‹avant›►‹a›•provFLTN❵xTCAC❴‹t›❵‹b›/provFLTN❳‹z›◄‹apres›≺☜🖉[22]
🖉 ENCAPSULER LES TV DES CTX PAR UN MARQUEUR TERME
☞➢‹avant›►‹a›•provFLTN❵‹b›•xCTX_‹j›⌊‹c›╠‹v›╣‹d›/xCTX_‹j›⌋‹e›/provFLTN❳‹z›◄‹apres›≺⇒
➢‹avant›►‹a›•provFLTN❵‹b›•xCTX_‹j›⌊‹c›•xTV_0❴D/‹v›/xTV_0❵‹d›/xCTX_‹j›⌋‹e›
/provFLTN❳‹z›◄‹apres›≺☜🖉[23]
🖉 DEMARQUER LES CONTEXTES
☞➢‹avant›►‹a›•provFLTN❵‹b›•xCTX_‹j›⌊‹c›/xCTX_‹j›⌋‹e›/provFLTN❳‹z›◄‹apres›≺⇒
➢‹avant›►‹a›•provFLTN❵‹b›xCTX_‹j›⌊‹c›/xCTX_‹j›⌋‹e›/provFLTN❳‹z›◄‹apres›≺☜🖉[24]

🖉 TRAITER LES FP (FONCTION PROPOSITIONNELLE) ET LES TC (TERME CONSTANT)
🖉 **********
🖉 ENCAPSULER ET MARQUER TOUS LES TC
☞➢‹avant›►‹a›╠‹v›↳IN(bop)⇙›╣‹z›◄‹apres›≺⇒➢‹avant›►‹a›•xBOP❵‹v›❲‹z›◄‹apres›≺☜🖉[25]
☞➢‹avant›►‹a›╠‹v›↳IN(mop)⇙›╣‹z›◄‹apres›≺⇒➢‹avant›►‹a›•xMOP❵‹v›❲‹z›◄‹apres›≺☜🖉[26]
☞➢‹avant›►‹a›╠‹v›↳IN(xop)⇙›╣‹z›◄‹apres›≺⇒➢‹avant›►‹a›•xXOP❵‹v›❲‹z›◄‹apres›≺☜🖉[27]
🖉 ENCAPSULER ET MARQUER LES CONSTANTES PROPOSITIONNELLES
☞➢‹avant›►‹a›╠‹v›↳IN(cp0)⇙›╣‹z›◄‹apres›≺⇒➢‹avant›►‹a›•xCP❵‹v›❲‹z›◄‹apres›≺☜🖉[28]
☞INUTILE⇒NE RIEN FAIRE☜🖉[29]

🖉 TRAITER LES EXPRESSIONS
🖉 **********
🖉 Se donner un compteur de termes et d'expressions
☞➢‹avant›►‹a›◄‹apres›≺⇒➢‹avant›►‹a›◄‹apres›≺◊/cnttrm=1/cntexp=1◊☜🖉[30]

🖉 NIVEAU 0
🖉 **********

🖉 (E0) ENCAPSULER LA FORMULE LOGIQUE TERME-NOUVEAU (provFLTN) : xE ::= <fltn>
🖉 Ne pas surcharger FLTN, remplacer par une encapsulation d'expression !
☞➢‹avant›►‹a›•provFLTN❵‹v›/provFLTN❳‹z›◄‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊⇒
➢‹avant›►‹a›•xE_‹ce›❵‹v›/xE_‹ce›❳‹z›◄‹apres›≺◊/cnttrm=‹ct›/cntexp=↳PLUS(‹ce›,1)⇙◊☜🖉[31]

🖉 (T0) ENCAPSULER LES TERMES
🖉 Encapsuler les termes variables par une encapsulation terme (terme variable lié ou libre)
☞➢‹avant›►‹a›╠‹v›↳IN(vp_vn)⇙›╣‹z›◄‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊⇒
➢‹avant›►‹a›•pTV_‹ct›❵L/‹v›/pTV_‹ct›❳‹z›◄‹apres›≺
◊/cnttrm=↳PLUS(‹ct›,1)⇙/cntexp=‹ce›◊☜🖉[32]

🖉 (T0) ENCAPSULER LES CONSTANTES PROP. PAR UNE ENCAPSULATION DE TERME
☞➢‹avant›►‹a›•xCP❵‹v›❲‹z›◄‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊⇒
➢‹avant›►‹a›•pTV_‹ct›❵C/‹v›/pTV_‹ct›❳‹z›◄‹apres›≺
◊/cnttrm=↳PLUS(‹ct›,1)⇙/cntexp=‹ce›◊☜🖉[33]

NIVEAU 1
****
(E1) ENCAPSULER TOUTES LES EXPRESSIONS DE TYPE : E ::= ⌈<provT>⌉
et démarquer xE
☞➢‹avant›▶‹a›•xSQ_‹i›⌈•pTV_‹j›[‹v›/pTV_‹j›]/xSQ_‹i›⌉‹z›◀‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊⇒
➢‹avant›▶‹a›•xSQ_‹i›⌈xE_‹ce›[•xTV_‹j›[‹v›/xTV_‹j›]/xE_‹ce›]/xSQ_‹i›⌉‹z›◀‹apres›≺
◊/cnttrm=‹ct›/cntexp=↳PLUS(‹ce›,1)↵◊☜✎[34]

(E1) ENCAPSULATION DE TERMES SOUMIS A UN OPERATEUR UNAIRE,
DE TYPE E ::= <encap_xmonop><encap_xpg><encap_xt><encap_xpd>

>>> MONOP_NIVEAU_E1 <<<

(E1) cerner un mop
☞➢‹avant›▶‹a›•xMOP⟩‹o›↳IN(mop)↵›⟩•tPG_‹i›⌈‹pg›↳IN(pg)↵›⌋‹v›•tPD_‹i›⌈‹pd›↳IN(pd)↵›⌋‹z›◀
‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊⇒
➢‹avant›▶‹a›●→mark_xmop⟩‹o›⟩mark_tpg_‹i›⌈‹pg›⌋○→‹v›←○mark_tpd_‹i›⌈‹pd›⌋←●‹z›◀
‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark1☜✎[35]
(E1) tester si un seul terme
☞➢‹avant›▶‹a›○→•pTV_‹j›[‹v›/pTV_‹j›]←○‹z›◀‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark1⇒
➢‹avant›▶‹a›○→•xTV_‹j›[‹v›/xTV_‹j›]←○‹z›◀‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark2☜✎[36]
(E1) si ok, encapsuler l'expression
☞➢‹avant›▶‹a›●→mark_xmop⟩‹o›⟩mark_tpg_‹i›⌈‹pg›⌋○→‹v›←○mark_tpd_‹i›⌈‹pd›⌋←●‹z›◀
‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark2⇒
➢‹avant›▶‹a›•xE_‹ce›[xMOP⟩‹o›⟩xPG⌈‹pg›⌋‹v›xPD⌈‹pd›⌋/xE_‹ce›]‹z›◀
‹apres›≺◊/cnttrm=‹ct›/cntexp=↳PLUS(‹ce›,1)↵◊☜✎[37]
(E1) si non, effacer les curseurs et passer à monop suivant
☞➢‹avant›▶‹a›●→‹v›←●‹z›◀‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark1⇒
➢‹avant›▶‹a›‹v›‹z›◀‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark2☜✎[38]
☞➢‹avant›▶‹a›○→‹v›←○‹z›◀‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark2⇒
➢‹avant›▶‹a›‹v›‹z›◀‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊☜✎[39]

(E1) ALLER A >>> MONOP_NIVEAU_E1 >>>
☞↳IFMATCH_GOTORULE(35)↵☜✎ monop T suivant [40]
effacer les marqueurs
☞➢‹avant›▶‹a›mark_xmop⟩‹o›⟩mark_tpg_‹i›⌈‹pg›⌋‹v›mark_tpd_‹i›⌈‹pd›⌋‹z›◀
‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊⇒
➢‹avant›▶‹a›•xMOP⟩‹o›⟩•tPG_‹i›⌈‹pg›⌋‹v›•tPD_‹i›⌈‹pd›⌋‹z›◀
‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊☜✎[41]

NIVEAU 2
****
META-EXPRESSION DE PAIRES TT (NIVEAU 2), de type :
E ::= <encap_xbinop><encap_xpg><encap_xt><encap_xt><encap_xpd>

>>> TT_NIVEAU_2 <<<

(TT2) détecter les paires de termes : ●→<xT><xT>←● et les garder marqués pour l'analyse des lieurs
☞➢‹avant›▶‹a›•pTV_‹j›[‹p›/pTV_‹j›]•pTV_‹k›[‹q›/pTV_‹k›]‹z›◀‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊
⇒
➢‹avant›▶‹a›●→•xTV_‹j›[‹p›/xTV_‹j›]•xTV_‹k›[‹q›/xTV_‹k›]←●‹z›◀‹apres›≺
◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark1☜✎[42]
(TT2) étendre la détection à : ●→<pg>...<pd>←●
☞➢‹avant›▶‹a›•tPG_‹i›⌈‹pg›↳IN(pg)↵›⌋●→‹v›←●•tPD_‹i›⌈‹pd›↳IN(pd)↵›⌋‹z›◀‹apres›≺
◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark1⇒
➢‹avant›▶‹a›●→•tPG_‹i›⌈‹pg›⌋‹v›•tPD_‹i›⌈‹pd›⌋←●‹z›◀‹apres›≺
◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark2☜✎[42]
(TT2) encapsuler : E ::= <xBOP><pg><xT><xT><pd>
☞➢‹avant›▶‹a›•xBOP⟩‹o›↳IN(bop)↵›⟩●→‹v›←●‹z›◀‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark2⇒
➢‹avant›▶‹a›•xE_‹ce›[xBOP⟩‹o›⟩●→‹v›←●/xE_‹ce›]‹z›◀‹apres›≺
◊/cnttrm=‹ct›/cntexp=↳PLUS(‹ce›,1)↵◊/mark3☜✎[44]
(TT2) supprimer encapsulation des <pg>...<pd> et des cursuers : ●→←●
☞➢‹avant›▶‹a›●→•tPG_‹i›⌈‹pg›⌋‹v›•tPD_‹i›⌈‹pd›⌋←●‹z›◀‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark3⇒
➢‹avant›▶‹a›xPG⌈‹pg›⌋‹v›xPD⌈‹pd›⌋‹z›◀‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊☜✎[45]

(TT2) ALLER A >>> TT_NIVEAU_2 <<<
☞↳IFMATCH_GOTORULE(42)↵☜✎ paire TT suivante [46]

NIVEAU 3
****

✎ ENCAPSULER LES EXPRESSIONS (NIVEAU E3), de type :
✎ xE ::= <encap_xmonop><encap_xpg><encap_xe><encap_xpd>

✎ >>> NIVEAU_E3 <<<
✎ >>> MONOP_NIVEAU_E3 <<<

✎ (E3) cerner un mop
☞➢‹avant›▶‹a›•xMOP⦊‹o›↳IN(mop)↲›⦉•tPG_‹i›⦇‹pg›↳IN(pg)↲›⦈‹v›•tPD_‹i›⦇‹pd›↳IN(pd)↲›⦈‹z›◀
‹apres›⩤◊/cnttrm=‹ct›/cntexp=‹ce›◊⇒
➢‹avant›▶‹a›●→mark_xmop⦊‹o›⦉mark_tpg_‹i›⦇‹pg›⦈○→‹v›←○mark_tpd_‹i›⦇‹pd›⦈←●‹z›◀
‹apres›⩤◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark1☜✎[47]
✎ (E3) tester si expression interne unique et, si oui, démarquer
☞➢‹avant›▶‹a›○→•xE_‹i›⦋‹v›⁄xE_‹i›⦌←○‹z›◀‹apres›⩤◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark1⇒
➢‹avant›▶‹a›xE_‹i›⦋‹v›⁄xE_‹i›⦌‹z›◀‹apres›⩤◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark2☜✎[48]
✎ (E3) si expression encapsuler par une nouvelle encapsulation d'expression
☞➢‹avant›▶‹a›●→mark_xmop⦊‹o›⦉mark_tpg_‹i›⦇‹pg›⦈‹v›mark_tpd_‹i›⦇‹pd›⦈←●‹z›◀
‹apres›⩤◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark2⇒
➢‹avant›▶‹a›•xE_‹ce›⦋xMOP⦊‹o›⦉xPG⦇‹pg›⦈‹v›xPD⦇‹pd›⦈⁄xE_‹ce›⦌‹z›◀
‹apres›⩤◊/cnttrm=‹ct›/cntexp=↳PLUS(‹ce›,1)↲◊/mark_suivant☜✎[49]
✎ (E3) si non, effacer les curseurs et passer à monop suivant
☞➢‹avant›▶‹a›●→‹v›←●‹z›◀‹apres›⩤◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark1⇒
➢‹avant›▶‹a›‹v›‹z›◀‹apres›⩤◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark2☜✎[50]
☞➢‹avant›▶‹a›○→‹v›←○‹z›◀‹apres›⩤◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark2⇒
➢‹avant›▶‹a›‹v›‹z›◀‹apres›⩤◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark_suivant☜✎[51]

✎ (E3) ALLER A >>> MONOP_NIVEAU_E3 <<<
☞➢‹avant›▶‹a›◀‹apres›⩤◊‹x›◊/mark_suivant⇒
➢‹avant›▶‹a›◀‹apres›⩤◊‹x›◊↳GOTORULE(47)↲☜✎[52]
✎ (E3) effacer les marqueurs
☞➢‹avant›▶‹a›mark_xmop⦊‹o›⦉mark_tpg_‹i›⦇‹pg›⦈‹v›mark_tpd_‹i›⦇‹pd›⦈‹z›◀
‹apres›⩤◊/cnttrm=‹ct›/cntexp=‹ce›◊⇒
➢‹avant›▶‹a›•xMOP⦊‹o›⦉•tPG_‹i›⦇‹pg›⦈‹v›•tPD_‹i›⦇‹pd›⦈‹z›◀
‹apres›⩤◊/cnttrm=‹ct›/cntexp=‹ce›◊☜✎[53]

✎ ENCAPSULER : encap_xe ::= <encap_xbinop><encap_xpg><encap_xt><encap_xe><encap_xpd>|
✎ <encap_xbinop><encap_xpg><encap_xe><encap_xt><encap_xpd>|
✎ <encap_xbinop><encap_xpg><encap_xe><encap_xe><encap_xpd>
✎ (NIVEAU E3) POUR LES BOP
✎ (E3) détecter une paire : ●→<T><E>←●, garder les marqueurs pour l'analyse des lieurs
☞➢‹avant›▶‹a›•pTV_‹j›⦋‹p›⁄pTV_‹j›⦌•xE_‹k›⦋‹q›⁄xE_‹k›⦌‹z›◀‹apres›⩤◊/cnttrm=‹ct›/cntexp=‹ce›◊⇒
➢‹avant›▶‹a›●→•xTV_‹j›⦋‹p›⁄xTV_‹j›⦌xE_‹k›⦋‹q›⁄xE_‹k›⦌←●‹z›◀‹apres›⩤
◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark1☜✎[54]
✎ (E3) détecter une paire : ●→<E><T>←●, garder les marqueurs pour l'analyse des lieurs
☞➢‹avant›▶‹a›•xE_‹k›⦋‹p›⁄xE_‹k›⦌•pTV_‹j›⦋‹q›⁄pTV_‹j›⦌‹z›◀‹apres›⩤◊/cnttrm=‹ct›/cntexp=‹ce›◊⇒
➢‹avant›▶‹a›●→xE_‹k›⦋‹p›⁄xE_‹k›⦌•xTV_‹j›⦋‹q›⁄xTV_‹j›⦌←●‹z›◀‹apres›⩤
◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark1☜✎[55]
✎ (E3) détecter les paires : ●→<E><E>←●
☞➢‹avant›▶‹a›•xE_‹k›⦋‹p›⁄xE_‹k›⦌•xE_‹j›⦋‹q›⁄xE_‹j›⦌‹z›◀‹apres›⩤◊/cnttrm=‹ct›/cntexp=‹ce›◊⇒
➢‹avant›▶‹a›●→xE_‹k›⦋‹p›⁄xE_‹k›⦌xE_‹j›⦋‹q›⁄xE_‹j›⦌←●‹z›◀‹apres›⩤
◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark1☜✎[56]
✎ (E3) étendre la détection à : ●→<pg>...<pd>←●
☞➢‹avant›▶‹a›•tPG_‹i›⦇‹pg›↳IN(pg)↲›⦈●→‹v›←●•tPD_‹i›⦇‹pd›↳IN(pd)↲›⦈‹z›◀‹apres›⩤
◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark1⇒
➢‹avant›▶‹a›●→xPG⦇‹pg›⦈‹v›xPD⦇‹pd›⦈←●‹z›◀‹apres›⩤◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark2☜✎[57]
✎ (E3) encapsuler : E::= xBOP<pg>...<pd>
☞➢‹avant›▶‹a›•xBOP⦊‹o›↳IN(bop)↲›⦉●→‹v›←●‹z›◀‹apres›⩤◊/cnttrm=‹ct›/cntexp=‹ce›◊/mark2⇒
➢‹avant›▶‹a›•xE_‹ce›⦋xBOP⦊‹o›⦉‹v›⁄xE_‹ce›⦌‹z›◀‹apres›⩤
◊/cnttrm=‹ct›/cntexp=↳PLUS(‹ce›,1)↲◊/mark_suivant☜✎[58]

✎ (E3) ALLER A >>> NIVEAU_E3 <<<
☞➢‹avant›▶‹a›◀‹apres›⩤◊‹x›◊/mark_suivant⇒
➢‹avant›▶‹a›◀‹apres›⩤◊‹x›◊↳GOTORULE(47)↲☜✎[59]

✎ NIVEAU 4
✎ **********
✎ (E4) DEMARQUER LES EXPRESSIONS UNIQUES DANS UN SOUS-QUANTIFICATEUR
☞➢‹avant›▶‹a›•xSQ_‹i›⎡•xE_‹j›⦋‹b›⁄xE_‹j›⦌⁄xSQ_‹i›⎤‹z›◀‹apres›⩤◊/cnttrm=‹ct›/cntexp=‹ce›◊⇒
➢‹avant›▶‹a›•xSQ_‹i›⎡xE_‹j›⦋‹b›⁄xE_‹j›⦌⁄xSQ_‹i›⎤‹z›◀‹apres›⩤◊/cnttrm=‹ct›/cntexp=‹ce›◊☜✎[60]

NIVEAU 5
****
(E5) ENCAPSULATION DES EXPRESSIONS DE GENERALISATION, de type :
E ::= <encap_xQ><encap_xSQ>
➢‹avant›►‹a›•xQ_‹i›↳INTEGER↵›⌊‹b›/xQ_‹i›⌋•xSQ_‹i›⌈‹c›/xSQ_‹i›⌉‹z›◄‹apres›≺
◊/cnttrm=‹ct›/cntexp=‹ce›◊⇒
➢‹avant›►‹a›•xE_‹ce›xQ_‹i›⌊‹b›/xQ_‹i›⌋xSQ_‹i›⌈‹c›/xSQ_‹i›⌉/xE_‹ce›‹z›◄‹apres›≺
◊/cnttrm=‹ct›/cntexp=↳PLUS(‹ce›,1)↵◊[61]

(E5) ALLER A >>> NIVEAU 3 <<<
↳IFMATCH_GOTORULE(47)↵[62]

(E6) ENCAPSULER LES FORMULES LOGIQUES (BICONDITIONNELLES) / NIVEAU DOMINANT
➢‹avant›►•xBOP≡•xPG_‹i›(‹v›•xPD_‹i›)◄‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊⇒
➢‹avant›►•xE_‹ce›xBOP≡xPG(‹v›xPD)/xE_‹ce›◄‹apres›≺
◊/cnttrm=‹ct›/cntexp=↳PLUS(‹ce›,1)↵◊[63]

(E7) DEMARQUER L'ENCAPSULATION DE L'EXPRESSION PRINCIPALE
DES FORMULES LOGIQUES
➢‹avant›►•xE_‹i›‹v›◄‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊⇒
➢‹avant›►xE_‹i›‹v›◄‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊[64]
(E7) DEMARQUER LES ENCAPSULATIONS DES EXPRESSIONS DES SOUS-QUANTIFICATEURS
➢‹avant›►‹a›xSQ_‹i›⌈•xE_‹j›‹z›◄‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊⇒
➢‹avant›►‹a›xSQ_‹i›⌈xE_‹j›‹z›◄‹apres›≺◊/cnttrm=‹ct›/cntexp=‹ce›◊[65]

VERIFICATIONS
➢‹avant›►‹a›◄‹apres›≺◊‹x›◊/mark‹i›⇒
ERREUR : ➢‹avant›►‹a›◄‹apres›≺◊‹x›◊_erreur_/mark‹i›,
trouvé encore une marque parasite ! ↳STOP↵[66]
➢‹avant›►‹a›→‹z›◄‹apres›≺◊‹x›◊⇒
ERREUR : ➢‹avant›►‹a›_erreur_→‹z›◄‹apres›≺◊‹x›◊,
trouvé encore un curseur parasite : → ! ↳STOP↵[67]
➢‹avant›►‹a›←‹z›◄‹apres›≺◊‹x›◊⇒
ERREUR : ➢‹avant›►‹a›_erreur_←‹z›◄‹apres›≺◊‹x›◊,
trouvé encore un curseur parasite : ← ! ↳STOP↵[68]
➢‹avant›►‹a›╠‹z›◄‹apres›≺◊‹x›◊⇒
ERREUR : ➢‹avant›►‹a›_erreur_╠‹z›◄‹apres›≺◊‹x›◊,
trouvé encore un marqueur parasite : ╠ ! ↳STOP↵[69]
➢‹avant›►‹a›╣‹z›◄‹apres›≺◊‹x›◊⇒
ERREUR : ➢‹avant›►‹a›_erreur_╣‹z›◄‹apres›≺◊‹x›◊,
trouvé encore un marqueur parasite : ╣ ! ↳STOP↵[70]
➢‹avant›►‹a›•xQ‹z›◄‹apres›≺◊‹x›◊⇒
ERREUR : ➢‹avant›►‹a›_erreur_•xQ‹z›◄‹apres›≺◊‹x›◊,
trouvé encore un marqueur parasite : • ! ↳STOP↵[71]
➢‹avant›►‹a›•xS‹z›◄‹apres›≺◊‹x›◊⇒
ERREUR : ➢‹avant›►‹a›_erreur_•xS‹z›◄‹apres›≺◊‹x›◊,
trouvé encore un marqueur parasite : • ! ↳STOP↵[72]
➢‹avant›►‹a›•xB‹z›◄‹apres›≺◊‹x›◊⇒
ERREUR : ➢‹avant›►‹a›_erreur_•xB‹z›◄‹apres›≺◊‹x›◊,
trouvé encore un marqueur parasite : • ! ↳STOP↵[73]
➢‹avant›►‹a›•pT‹z›◄‹apres›≺◊‹x›◊⇒
ERREUR : ➢‹avant›►‹a›_erreur_•xTC‹z›◄‹apres›≺◊‹x›◊,
trouvé encore un marqueur parasite : • ! ↳STOP↵[74]
➢‹avant›►‹a›•xM‹z›◄‹apres›≺◊‹x›◊⇒
ERREUR : ➢‹avant›►‹a›_erreur_•xM‹z›◄‹apres›≺◊‹x›◊,
trouvé encore un marqueur parasite : • ! ↳STOP↵[75]
➢‹avant›►‹a›•xE‹z›◄‹apres›≺◊‹x›◊⇒
ERREUR : ➢‹avant›►‹a›_erreur_•xE‹z›◄‹apres›≺◊‹x›◊,
trouvé encore un marqueur parasite : • ! ↳STOP↵[76]
➢‹avant›►‹a›•prov‹z›◄‹apres›≺◊‹x›◊⇒
ERREUR : ➢‹avant›►‹a›_erreur_•prov‹z›◄‹apres›≺◊‹x›◊,
trouvé encore un marqueur parasite : • ! ↳STOP↵[77]

supprimer les compteurs
➢‹avant›►‹a›◄‹apres›≺◊‹x›◊⇒➢‹avant›►‹a›◄‹apres›≺[78]

↳STOP↵[79]
↳ENDRULES↵
↳ENDGRAM↵

Procédure définitoire

↳GRAM↲
☞GRAMNAME⇒PROCDEF☜

↳PARAMS↲
☞MAXGRAMLOOP⇒50☜
☞MAXRULELOOP⇒400☜
↳ENDPARAMS↲

↳SETS↲

🖉 Fonctions Propositionnelles
☞fp2⇒"f","g"☜🖉 fonctions propositionnelles à deux places
☞fp1⇒"h"☜🖉 fonctions propositionnelles à une place
☞fp⇒fp2,fp1☜🖉 toutes les fonctions propositionnelles

🖉 Constantes, termes constants
☞bic⇒"≡"☜🖉 biconditionnelle
☞cp0⇒"V","F","T"☜🖉 connstantes sans contexte
☞tc1⇒"~","¬","α","!","♭","↓"☜🖉 constantes unaires
☞tc2⇒"μ","Ц","ε","∧","∨","⊃"☜🖉 constantes avec contexte binaire
☞tcn⇒"Δ","δ"☜🖉 constantes n_aires
☞tcmc⇒"τ","ω"☜🖉 constantes multi-contextes
☞tcac⇒tc1,tc2,tcn,tcmc☜🖉 les constantes avec contexte(s) sans BIC

🖉 Parenthèses gauches et délimiteurs gauches de contexte
☞pgA⇒"("☜
☞pgB⇒"["☜
☞pgC⇒"{"☜
☞pgD⇒"<"☜
☞pgE⇒"⟮"☜
🖉 ...
☞pg⇒pgA,pgB,pgC,pgD,pgE☜🖉 ...

🖉 Parenthèses droites et délimiteurs droits de contexte
☞pdA⇒")"☜
☞pdB⇒"]"☜
☞pdC⇒"}"☜
☞pdD⇒">"☜
☞pdE⇒"⟯"☜
🖉 ...
☞pd⇒pdA,pdB,pdC,pdD,pdE☜🖉 ...
↳ENDSETS↲

↳RULES↲

🖉 VERIFIER QUE LA FORMULE N'EST PAS VIDE
☞➢‹avant›►◄‹apres›≺⇒
ERREUR : ➢‹avant›►_erreur_fomule_vide◄‹apres›≺, trouvé formule à analyser vide !↳STOP↲☜🖉[1]
🖉 VERIFIER QUE LA FORMULE A ANALYSER EST BIEN ENCAPSULEE PAR : ➢...►‹formule›◄...≺
☞➢‹avant›►‹v›◄‹apres›≺⇒➢‹avant›►‹v›◄‹apres›≺/ok☜🖉[2]
☞➢‹avant›►‹v›◄‹apres›≺⇒
ERREUR : ➢‹avant›►‹v›◄‹apres›≺, le format d'entré n'est pas correcte !↳STOP↲☜🖉[3]
☞➢‹avant›►‹v›◄‹apres›≺/ok⇒➢‹avant›►‹v›◄‹apres›≺☜🖉[4]

🖉 TRAITER LES QUANTIFICATEURS
🖉 **
🖉 VERIFIER QUE LES QUANTIFICATEURS NE SONT PAS VIDES
☞➢‹avant›►‹a›xQ_‹i›⌊/xQ_‹i›⌋‹b›◄‹apres›≺⇒
ERREUR : ➢‹avant›►‹a›xQ_‹i›⌊_erreur_/xQ_‹i›⌋‹b›◄‹apres›≺,
trouvé un quantificateur vide !↳STOP↲☜🖉[5]

🖉 >>> QUANTIFICATEUR <<<
🖉 VERIFIER QUE PAS DE REPETITION DE TERMES VARIABLES DANS LES QUANTIFICATEURS
🖉 cerner le contenu d'un quantificateur
☞➢‹avant›►‹a›xQ_‹i›⌊‹v›/xQ_‹i›⌋‹z›◄‹apres›≺⇒
➢‹avant›►‹a›mark_xQ_‹i›⌊●→‹v›←●/mark_xQ_‹i›⌋‹z›◄‹apres›≺/mark☜🖉[6]

🖉 >>> TESTER_REDONDANCE <<<
🖉 marquer le premier
☞➢‹avant›►‹a›●→•xTVQ⟦‹v›⟧‹b›←●‹z›◄‹apres›≺/mark⇒

➢‹avant›▶‹a›●→•xTVQ_mark[‹v›]‹b›←●‹z›◀‹apres›≺/mark1✍✎[7]
✎ tester le premier avec les suivants
✍➢‹avant›▶‹a›●→•xTVQ_mark[‹v›]‹b›•xTVQ[‹v›]‹c›←●‹z›◀‹apres›≺/mark1⇒
ERREUR : ➢‹avant›▶‹a›●→•xTVQ_mark[_erreur_‹v›]‹b›•xTVQ[_erreur_‹v›]‹c›←●‹z›◀‹apres›≺
/mark1, trouvé une répétition de termes variables dans un quantificateur !↳STOP⏎✍✎[8]
✎ déplacer curseur, marquer le nouveau premier et démarquer définitivement le précédent
✍➢‹avant›▶‹a›●→•xTVQ_mark[‹v›]‹b›←●‹z›◀‹apres›≺/mark1⇒
➢‹avant›▶‹a›xTVQ[‹v›]●→‹b›←●‹z›◀‹apres›≺/mark✍✎[9]

✎ ALLER A >>> TESTER_REDONDANCE <<<
✍↳IFMATCH_GOTORULE(7)⏎✍✎ exécuter pour un quantificateur suivant [10]
✎ supprimer les curseurs
✍➢‹avant›▶‹a›●→←●‹z›◀‹apres›≺/mark⇒➢‹avant›▶‹a›‹z›◀‹apres›≺✍✎[11]

✎ ALLER A >>> QUANTIFICATEUR <<<
✍↳IFMATCH_GOTORULE(6)⏎✍✎ traiter sous-quantificateur suivant [12]
✎ démarquer les quantificateurs
✍➢‹avant›▶‹a›mark_xQ_‹i›⌊‹v›/mark_xQ_‹i›⌋‹b›◀‹apres›≺⇒
➢‹avant›▶‹a›xQ_‹i›⌊‹v›/xQ_‹i›⌋‹b›◀‹apres›≺✍✎[13]

✎ TRIER LES ARGUMENTS DES QUANTIFICATEURS
✎ on les trie pour l'analyse comparative entre le Definiendum et Definiens

✎ >>> TRIE_QUANTIFICATEUR <<<
✎ cerner un quantificateur
✍➢‹avant›▶‹a›xQ_‹i›⌊‹v›/xQ_‹i›⌋‹b›◀‹apres›≺⇒
➢‹avant›▶‹a›mark_xQ_‹i›⌊●→‹v›←●/mark_xQ_‹i›⌋‹b›◀‹apres›≺/mark✍✎[14]
✎ marquer les termes variables
✍➢‹avant›▶‹a›●→‹b›xTVQ[‹v›]‹c›←●‹d›◀‹apres›≺/mark⇒
➢‹avant›▶‹a›●→‹b›xtvq_mark[‹v›]‹c›←●‹d›◀‹apres›≺/mark✍✎[15]
✎ trier les termes variables du quantificateur
✍➢‹avant›▶‹a›●→‹v›←●‹z›◀‹apres›≺/mark⇒
➢‹avant›▶‹a›●→↳SORT(‹v›, ╚xtvq_mark[‹x›]╝)⏎←●‹z›◀‹apres›≺✍✎[16]
✎ démarquer les termes variables
✍➢‹avant›▶‹a›●→‹b›xtvq_mark[‹v›]‹c›←●‹z›◀‹apres›≺⇒
➢‹avant›▶‹a›●→‹b›xTVQ[‹v›]‹c›←●‹z›◀‹apres›≺✍✎[17]
✎ supprimer les curseurs
✍➢‹avant›▶‹a›●→‹v›←●‹z›◀‹apres›≺⇒➢‹avant›▶‹a›‹v›‹z›◀‹apres›≺✍✎[18]

✎ ALLER A >>> TRIE_QUANTIFICATEUR <<<
✍↳IFMATCH_GOTORULE(14)⏎✍✎ exécuter pour un quantificateur suivant [19]
✎ supprimer les marques
✍➢‹avant›▶‹a›mark_xQ_‹i›⌊‹v›/mark_xQ_‹i›⌋‹z›◀‹apres›≺⇒
➢‹avant›▶‹a›xQ_‹i›⌊‹v›/xQ_‹i›⌋‹z›◀‹apres›≺✍✎[20]

✎ TRAITER LE(S) CONTEXTE(S) DU DEFINIENDUM
✎ **********
✎ Attention : filtre les axiomes et les formules logiques, n'accepte que les thèses, donc que les tcac !
✎ dépister une Thèse
✍➢‹avant›▶‹a›xE_‹i›[xTCAC[‹tc↳IN(tcac)⏎›]‹b›◀‹apres›≺⇒
➢‹avant›▶‹a›xE_‹i›[xTCAC[‹tc›]‹b›◀‹apres›≺/these/sans_contexte✍✎[21]
✎ cerner un contexte
✍➢‹avant›▶‹a›xCTX_‹i↳INTEGER⏎›[‹v›/xCTX_‹i›]‹z›◀‹apres›≺/these/sans_contexte⇒
➢‹avant›▶‹a›xCTX_‹i›[‹v›/xCTX_‹i›]‹z›◀‹apres›≺/these✍✎[22]
✎ vérification, y a_t_il un contexte ?
✍➢‹avant›▶‹a›◀‹apres›≺/these/sans_contexte⇒
ERREUR : ➢‹avant›▶‹a›◀‹apres›≺/these/sans_contexte, pas trouvé de contexte !↳STOP⏎✍✎[23]

✎ >>> TRIE_CONTEXTE <<<
✎ cerner et marquer un contexte
✍➢‹avant›▶‹a›xCTX_‹i↳INTEGER⏎›[xPG[‹pg↳IN(pg)⏎›]•xTV_0[‹p›/xTV_0]‹z›◀‹apres›≺/these⇒
➢‹avant›▶‹a›mark_xctx_‹i›[xPG[‹pg›]●→•xTV_0[‹p›/xTV_0]‹z›◀‹apres›≺
/these/[‹i›]/mark1✍✎[24]
✍➢‹avant›▶‹a›•xTV_0[‹p›/xTV_0]xPD[‹pd↳IN(pd)⏎›]/xCTX_‹i›]‹z›◀‹apres›≺/these/[‹i›]/mark1⇒
➢‹avant›▶‹a›•xTV_0[‹p›/xTV_0]←●xPD[‹pd›]/mark_xctx_‹i›]‹z›◀‹apres›≺/these/mark2✍✎[25]
✍➢‹avant›▶‹a›●→‹v↳NOTEMPTY⏎›←●‹z›◀‹apres›≺/these/mark2⇒
➢‹avant›▶‹a›●→‹v›←●‹z›◀‹apres›≺/these/markPasVide✍✎[26]
✎ fin du processus si erreur
✍➢‹avant›▶‹a›●→‹v›←●‹z›◀‹apres›≺/these/mark2⇒
ERREUR : ➢‹avant›▶‹a›●→_erreur_‹v›←●‹z›◀‹apres›≺/these/mark2,

trouvé un contexte vide !↳STOP[27]
✎ marquer le(s) terme(s) du contexte courant
☞➢‹avant›▶‹a›●→‹b›•xTV_0[‹v›/xTV_0]‹c›←●‹z›◀‹apres›≮/these/markPasVide⇒
➢‹avant›▶‹a›●→‹b›mark_xtv_0[‹v›/xTV_0]‹c›←●‹z›◀‹apres›≮/these/markPasVide[28]
✎ TRIER LES TERMES DANS LES CONTEXTES
☞➢‹avant›▶‹a›●→‹v›←●‹z›◀‹apres›≮/these/markPasVide⇒
➢‹avant›▶‹a›●→↳SORT(‹v›, mark_xtv_0[‹x›/xTV_0])←●‹z›◀‹apres›≮/these[29]
✎ démarquer les termes
☞➢‹avant›▶‹a›●→‹b›mark_xtv_0[‹v›/xTV_0]‹c›←●‹z›◀‹apres›≮/these⇒
➢‹avant›▶‹a›●→‹b›•xTV_0[‹v›/xTV_0]‹c›←●‹z›◀‹apres›≮/these[30]
✎ supprimer les curseurs
☞➢‹avant›▶‹a›●→‹v›←●‹z›◀‹apres›≮/these⇒
➢‹avant›▶‹a›‹v›‹z›◀‹apres›≮/these[31]

✎ ALLER A >>> TRIE_CONTEXTE <<<
☞↳IFMATCH_GOTORULE(24) exécuter pour les contextes suivants [32]
✎ démarquer le(s) contexte(s) - garder la marque d'identification d'une Thèse
☞➢‹avant›▶‹a›mark_xctx_‹i›[‹v›/mark_xctx_‹i›]‹z›◀‹apres›≮/these⇒
➢‹avant›▶‹a›xCTX_‹i›[‹v›/xCTX_‹i›]‹z›◀‹apres›≮/these[33]

✎ REGLE D'INFERENCE DE DEFINITON :
✎ LES ARGUMENTS DU CONTEXTE DU DEFINIENDUM SONT DES TERMES VARIABLES
✎ ne traite que les THESES et pour cette version que dans la généralisation dominante
✎ extraire la suite des termes du quantificateur (courant)
☞➢‹avant›▶‹a›xQ_‹i↳INTEGER⌋[‹v›/xQ_‹i›]‹z›◀‹apres›≮/these⇒
➢‹avant›▶‹a›xQ_‹i›[○→‹v›←○/xQ_‹i›]‹z›◀‹apres›≮/these/mark1[34]

✎ >>> CONTEXTE_TV <<<
✎ cerner le contenu du sous-quantificateur dominant
☞➢‹avant›▶xQ_‹i↳INTEGER⌋[‹a›/xQ_‹i›]xSQ_‹i›⌈‹b›/xSQ_‹i›⌉◀‹apres›≮/these/mark1⇒
➢‹avant›▶xQ_‹i›[‹a›/xQ_‹i›]xSQ_‹i›⌈●→‹b›←●/xSQ_‹i›⌉◀‹apres›≮/these/mark2[35]
✎ cerner le contenu de l'encapsulation de l'expression biconditionnelle du sous-quantificateur dominant
☞➢‹avant›▶‹a›●→xE_‹i›[‹v›/xE_‹i›]←●‹z›◀‹apres›≮/these/mark2⇒
➢‹avant›▶‹a›xE_‹i›[●→‹v›←●/xE_‹i›]‹z›◀‹apres›≮/these/mark3[36]
✎ cerner le contenu de l'expression biconditionnelle du sous-quantificateur dominant
☞➢‹avant›▶‹a›●→xBOP≡⌋xPG(⌋‹v›xPD(⌋)←●‹z›◀‹apres›≮/these/mark3⇒
➢‹avant›▶‹a›xBOP≡⌋xPG(⌋●→‹v›←●xPD(⌋)‹z›◀‹apres›≮/these/mark4[37]
✎ cerner le contenu de l'expression tcac du sous-quantificateur dominant et sélectionner les contextes
☞➢‹avant›▶‹a›●→xE_‹i›[xTCAC{‹tn›}‹v›/xE_‹i›]‹b›←●‹z›◀‹apres›≮/these/mark4⇒
➢‹avant›▶‹a›xE_‹i›[xTCAC{‹tn›}●→‹v›←●/xE_‹i›]‹b›‹z›◀‹apres›≮/these/mark5[38]
✎ cerner le contexte suivant en évitant les contextes marqués
☞➢‹avant›▶‹a›●→xctx_mark_‹i↳INTEGER⌋[‹v›/xctx_mark_‹i›]‹c›←●‹z›◀‹apres›≮/these/mark5⇒
➢‹avant›▶‹a›xctx_mark_‹i›[‹v›/xctx_mark_‹i›]●→‹c›←●‹z›◀‹apres›≮/these/mark5[39]
✎ marquer le contexte suivant
☞➢‹avant›▶‹a›●→xCTX_‹i↳INTEGER⌋[‹v›/xCTX_‹i›]‹c›←●‹z›◀‹apres›≮/these/mark5⇒
➢‹avant›▶‹a›●→xctx_mark_‹i›[‹v›/xctx_mark_‹i›]←●‹c›‹z›◀‹apres›≮/these/mark5[40]
✎ cerner le contenu du contexte
☞➢‹avant›▶‹a›●→xctx_mark_‹i›[xPG(‹pg↳IN(pg)⌋)•xTV_0[‹p›/xTV_0]‹z›◀‹apres›≮/these/mark5⇒
➢‹avant›▶‹a›xctx_mark_‹i›[xPG(‹pg›)●→•xTV_0[‹p›/xTV_0]‹z›◀‹apres›≮/these/[‹i›]/mark6[41]
☞➢‹avant›▶‹a›●→•xTV_0[‹p›/xTV_0]xPD(‹pd↳IN(pd)⌋)/xctx_mark_‹i›]←●‹z›◀
‹apres›≮/these/[‹i›]/mark6⇒
➢‹avant›▶‹a›●→•xTV_0[‹p›/xTV_0]←●xPD(‹pd›)/xctx_mark_‹i›]‹z›◀‹apres›≮/these/mark7[42]

✎ tester l'équivalence entre les tv du quantificateur et du contexte,
✎ si trouvé alors terme variable, sinon terme constant !
☞➢‹avant›▶‹a›○→‹b›xTVQ{‹p›}‹c›←○‹x›●→•xTV_0[‹type›/‹p›/xTV_0]‹e›←●‹z›◀
‹apres›≮/these/mark7⇒
➢‹avant›▶‹a›○→‹b›xTVQ{‹p›}‹c›←○‹x›•xTV_0[‹type›/‹p›/xTV_0]●→‹e›←●‹z›◀
‹apres›≮/these/mark7[43]
✎ renommer les formes •xTV restantes en xTC
☞➢‹avant›▶‹a›●→‹d›•xTV_0[‹p›/xTV_0]‹e›←●‹z›◀‹apres›≮/these/mark7⇒
➢‹avant›▶‹a›●→‹d›xTC_0[‹p›/xTC_0]‹e›←●‹z›◀‹apres›≮/these/mark7[44]
✎ un definiendum n'accepte pas de variable libre !
☞➢‹avant›▶‹a›●→‹d›xTC_0[‹p›/xTC_0]‹e›←●‹z›◀‹apres›≮/these/mark7⇒
ERREUR : ➢‹avant›▶‹a›●→‹d›_erreur_xTC_0[‹p›/xTC_0]‹e›←●‹z›◀
‹apres›≮/these/mark7, trouvé un terme constant dans le Definiendum !↳STOP[45]
✎ supprimer les curseurs
☞➢‹avant›▶‹a›●→‹v›←●‹z›◀‹apres›≮/these/mark7⇒

➢‹avant›►‹a›‹v›‹z›◄‹apres›≺/these/mark1☜🖉[46]

🖉 ALLER A >>> CONTEXTE_TV <<<
☞↳IFMATCH_GOTORULE(35)🖒☜🖉 exécuter terme varaible pour contexte suivant [47]
🖉 effacer les marqeurs (on garde la marque Thèse pour la suite)
☞➢‹avant›►‹a›●→←●‹z›◄‹apres›≺/these/mark5⇒➢‹avant›►‹a›‹z›◄‹apres›≺/these☜🖉[48]
☞➢‹avant›►‹a›○→‹v›←○‹z›◄‹apres›≺/these⇒➢‹avant›►‹a›‹v›‹z›◄‹apres›≺/these☜🖉[49]
☞➢‹avant›►‹a›xctx_mark_‹i›↳INTEGER🖒›[‹v›/xctx_mark_‹i›]‹z›◄‹apres›≺/these⇒
➢‹avant›►‹a›xCTX_‹i›[‹v›/xCTX_‹i›]‹z›◄‹apres›≺/these☜🖉[50]

🖉 REGLE D'INFERENCE DE DEFINITION :
🖉 AUCUN SIGNE DU DEFINIENDUM N'EST REPETE
🖉 extraire le contenu du definiendum (courant)
🖉 cerner le contenu du sous-quantificateur dominant
☞➢‹avant›►xQ_‹i›↳INTEGER🖒›⌊‹a›/xQ_‹i›⌋xSQ_‹i›⌈‹b›/xSQ_‹i›⌉◄‹apres›≺/these⇒
➢‹avant›►xQ_‹i›⌊‹a›/xQ_‹i›⌋xSQ_‹i›⌈●→‹b›←●/xSQ_‹i›⌉◄‹apres›≺/these/mark1☜🖉[51]
🖉 cerner le contenu de l'encapsulation de l'expression biconditionnelle du sous-quantificateur dominant
☞➢‹avant›►‹a›●→xE_‹i›[‹v›/xE_‹i›]←●‹z›◄‹apres›≺/these/mark1⇒
➢‹avant›►‹a›xE_‹i›[●→‹v›←●/xE_‹i›]‹z›◄‹apres›≺/these/mark2☜🖉[52]
🖉 cerner le contenu de l'expression biconditionnelle du sous-quantificateur dominant
☞➢‹avant›►‹a›●→xBOP⟩≡⟨xPG(›‹v›xPD)←●‹z›◄‹apres›≺/these/mark2⇒
➢‹avant›►‹a›xBOP⟩≡⟨xPG(›●→‹v›←●xPD)‹z›◄‹apres›≺/these/mark3☜🖉[53]
🖉 cerner le contenu de l'expression tcac du sous-quantificateur dominant et sélectionner les contextes
☞➢‹avant›►‹a›●→xE_‹i›xTCAC‹tn›‹v›/xE_‹i›‹b›←●‹z›◄‹apres›≺/these/mark3⇒
➢‹avant›►‹a›xE_‹i›xTCAC‹tn›●→‹v›←●/xE_‹i›‹b›‹z›◄‹apres›≺/these/mark4☜🖉[54]
🖉 rechercher une équivalence entre les tv du Definiendum
☞➢‹avant›►‹a›●→‹b›•xTV_0[‹p›/xTV_0]‹c›•xTV_0[‹p›/xTV_0]‹d›←●‹z›◄‹apres›≺/these/mark4⇒
ERREUR : ➢‹avant›►‹a›●→‹b›•xTV_0[_erreur_‹p›/xTV_0]‹c›•xTV_0[_erreur_‹p›/xTV_0]‹d›←●‹z›◄
‹apres›≺/these/mark4, trouvé des termes (‹p›) répétés dans le Definiendum !↳STOP🖒☜🖉[55]
🖉 supprimer les marqueurs des xTV_0
🖉 supprimer les curseurs, le marqueur et l'indicateur de Thèse
☞➢‹avant›►‹a›●→‹v›←●‹z›◄‹apres›≺/these/mark4⇒
➢‹avant›►‹a›‹v›‹z›◄‹apres›≺☜🖉[56]

🖉 TRAITER LES TERMES VARIABLES LIES ET LES TERMES CONSTANTS (VARIABLES LIBRES)
🖉 **********
🖉 Il s'agit de rechercher tous les TV de l'inscription (l'un après l'autre) et de remonter
🖉 dans l'arborescence des généralisations pour trouver si l'on en fait mention
🖉 et cela jusqu'à la généralisation de niveau 0.
🖉 Ce traitement est valable pour les axiomes et les thèses / il n'est pas valable pour le formules logiques !

🖉 On commence par éliminer le cas des termes constitués d'une constante propositionnelle :
🖉 si le xT est une constante propositionnelle, la démarquer
☞➢‹avant›►‹a›•xTV_‹i›[C/‹p›↳IN(cp0)🖒›/xTV_‹i›]‹z›◄‹apres›≺⇒
➢‹avant›►‹a›xTV_[-1]_‹i›[C/‹p›/xTV_‹i›]‹z›◄‹apres›≺☜🖉[57]

🖉 Préparer un paramètre de numéro de quantificateur qui servira à traiter tous les niveaux de
🖉 parenthèsage sachant qu'un tv peut être lié par un quantificateur imbriqué dans une expression.
☞➢‹avant›►‹a›◄‹apres›≺⇒➢‹avant›►‹a›◄‹apres›≺/q_cour[0]/q_arbo[0]/mark☜🖉[58]
🖉 Chercher le numéro de sous-quantificateur le plus élevé, soit l'expression la plus profonde.
🖉 Dû au traitement qui a suivi DELIMNUM dans la re-grammaire : PREANALYSE, nous savons
🖉 que : (1) le quantificateur dominant est toujours numéroté avec 0, (2) les niveaux suivants et possibles
🖉 sont numéroté dans l'ordre. La re-règle qui suit, construite sur une variable miroir : noucour, s'itère
🖉 tant que que l'on trouve un quantificateur numéroté avec la valeur du compteur courant
🖉 de la variable miroir
☞➢‹avant›►‹a›xSQ_‹nocour›⌈‹z›◄‹apres›≺/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark⇒
➢‹avant›►‹a›xSQ_‹nocour›⌈‹z›◄‹apres›≺
/q_cour[↳PLUS(‹nocour›,1)🖒]/q_arbo[↳PLUS(‹noarbo›,1)🖒]/mark☜🖉[59]
🖉 La re-règle précédente a compter un coup de trop, donc ajuster le compteur (numéro moins 1)
☞➢‹avant›►‹a›◄‹apres›≺/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark⇒
➢‹avant›►‹a›◄‹apres›≺/q_cour[↳MINUS(‹nocour›,1)🖒]/q_arbo[↳MINUS(‹noarbo›,1)🖒]☜🖉[60]

🖉 Nous repèrons le début d'une structure d'itération par une étiquette (virtuelle). Cette section de
🖉 traitement marque le début du traitement des sous-quantificateurs. On va les parcourir un à un.

🖉 >>> PROCHAIN_SQ <<<
🖉 Sélectionner le sous-quantificateur courant et positionner les curseurs de xTV. Pour que les deux
🖉 re-règles qui suivent ne s'exécute qu'une seule fois, on fait évoluer des marqueurs de fin de chaîne.
☞➢‹avant›►‹a›xSQ_‹nocour›⌈‹z›◄‹apres›≺/q_cour[‹nocour›]/q_arbo[‹noarbo›]⇒
➢‹avant›►‹a›xSQ_‹nocour›⌈➲⇨‹z›◄‹apres›≺/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark1☜🖉[61]

☞➢‹avant›►‹a›/xSQ_‹nocour›⎤‹z›◄‹apres›≺/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark1⇒
➢‹avant›►‹a›⇦⊂/xSQ_‹nocour›⎤‹z›◄‹apres›≺/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv☜✎[62]

✎ On se donne un deuxième branchement, donc une boucle imbriquée qui va parcourir les tv
✎ du sous-quantificateur que l'on a cerné précédement.

✎ >>> PROCHAIN_TV <<<
✎ Sélectionner le prochain xTV et remettre le compteur noarbo à nocour.
☞➢‹avant›►‹a›⇨‹b›•xTV_‹i›[‹p›/xTV_‹i›]‹c›⇦‹z›◄‹apres›≺
/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv⇒
➢‹avant›►‹a›‹b›⇨•xTV_‹i›[‹p›/xTV_‹i›]‹c›⇦‹z›◄‹apres›≺
/q_cour[‹nocour›]/q_arbo[‹nocour›]/mark1☜✎[63]

✎ Nous nous donnons un troisième branchement qui doit permettre pour un tv donné de retrouver
✎ son quantificateur associé

✎ >>> PROCHAIN_NIVEAU_ARBO <<<
✎ Sélectionner le quantificateur avec le numéro d'arborescence courant.
☞➢‹avant›►‹a›xQ_‹noarbo›⌊‹z›◄‹apres›≺/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark1⇒
➢‹avant›►‹a›xQ_‹noarbo›⌊▷‹z›◄‹apres›≺/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark2☜✎[64]
☞➢‹avant›►‹a›/xQ_‹noarbo›⌋‹z›◄‹apres›≺/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark2⇒
➢‹avant›►‹a›◁/xQ_‹noarbo›⌋‹z›◄‹apres›≺
/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mention_pas_trouve☜✎[65]

✎ Cerner le terme variable dans l'encapsulation du terme. On mémorise dans les paramètres de fin
✎ de chaîne le tv trouvé : ⟦‹p›⟧
☞➢‹avant›►‹a›⇨•xTV_‹i›↳INTEGER↲[‹type›/‹p›/xTV_‹i›]‹z›◄‹apres›≺
/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mention_pas_trouve⇒
➢‹avant›►‹a›⇨•xTV_‹i›[‹type›/‹p›/xTV_‹i›]‹z›◄‹apres›≺/⟦‹p›⟧
/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mention1_pas_trouve☜✎[66]
✎ Tester si le xTV cerné est mentionné dans le quantificateur sélectionné ou non.
☞➢‹avant›►‹a›▷‹b›xTVQ⟦‹p›⟧‹c›◁‹z›◄‹apres›≺
/⟦‹p›⟧/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mention1_pas_trouve⇒
➢‹avant›►‹a›▷‹b›xTVQ⟦‹p›⟧‹c›◁‹z›◄‹apres›≺
/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mention2_pas_trouve☜✎[67]
✎ Tester si le xTV est mentionné dans le quantificateur numéroté noarbo
☞➢‹avant›►‹a›▷‹b›◁‹d›⇨•xTV_‹i›↳INTEGER↲[‹p›/xTV_‹i›]‹z›◄‹apres›≺
/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mention2_pas_trouve⇒
➢‹avant›►‹a›‹b›‹d›xTV_[‹noarbo›]_‹i›[‹p›/xTV_‹i›]⇨‹z›◄‹apres›≺
/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark_prochain_tv☜✎[68]
✎ Si trouvé traiter le prochain xTV, supprimer les curseurs de quantificateur,

✎ aller à >>> PROCHAIN_TV <<<
☞➢‹avant›►‹a›◄‹apres›≺/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark_prochain_tv⇒
➢‹avant›►‹a›◄‹apres›≺/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv↳GOTORULE(63)↲☜✎[69]
✎ Si pas trouvé, monter dans l'arborescence des généralisations (soit la re-règle : 71).
✎ Mais, d'abord vérifier le niveau de la généralisation. Si on ne trouve pas de quantificateur pour cette
✎ variable c'est qu'elle est libre !
☞➢‹avant›►‹a›◄‹apres›≺/q_cour[‹nocour›]/q_arbo[0]/mention‹mark›_pas_trouve⇒
ERREUR : ➢‹avant›►‹a›◄‹apres›≺
/q_cour[‹nocour›]/q_arbo[0]/mention‹mark›_pas_trouve, trouvé variable libre !↳STOP↲☜✎[70]
✎ Supprimer les curseurs du quantificateur…
☞➢‹avant›►‹a›▷‹b›◁‹c›◄‹apres›≺‹X›/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mention‹mark›_pas_trouve
⇒➢‹avant›►‹a›‹b›‹c›◄‹apres›≺/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mention‹mark›_pas_trouve☜✎[71]
✎ …et monter dans l'arborescence des généralisations.
☞➢‹avant›►‹a›◄‹apres›≺‹X›/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mention‹mark›_pas_trouve⇒➢‹avant›
►‹a›◄‹apres›≺/q_cour[‹nocour›]/q_arbo[↳MINUS(‹noarbo›,1)↲]/prochain_niveau_arbo☜✎[72]

✎ Le nouveau niveau d'arborescense est donné. Il s'agit maintenant d'aller cerner le quantificateur
✎ de ce prochain niveau.

✎ aller à >>> PROCHAIN_NIVEAU_ARBO <<<
☞➢‹avant›►‹a›◄‹apres›≺/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_niveau_arbo⇒
➢‹avant›►‹a›◄‹apres›≺/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark1↳GOTORULE(64)↲☜✎[73]

✎ Si le marqueur : prochain_tv est inscrit en fin de chaîne, alors monter dans la hiérarchies
✎ des sous-quantificateurs.
✎ Mais il faut tout d'abord, tester si on est arrivé à la fin des niveaux.
☞➢‹avant›►‹a›◄‹apres›≺/q_cour[0]/q_arbo[‹noarbo›]/prochain_tv⇒
➢‹avant›►‹a›◄‹apres›≺/q_cour[0]/q_arbo[‹noarbo›]/fin1☜✎[74]

✎ Sinon on initialise les paramètres pour l'itération suivante.
✎ Supprimer tous les curseurs
☞➢‹avant›▶‹a›▷‹b›◁‹c›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv⇒
➢‹avant›▶‹a›‹b›‹c›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv☜✎[75]
☞➢‹avant›▶‹a›➲‹b›➲‹c›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv⇒
➢‹avant›▶‹a›‹b›‹c›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv☜✎[76]
☞➢‹avant›▶‹a›⇨‹b›⇦‹c›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv⇒
➢‹avant›▶‹a›‹b›‹c›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv☜✎[77]
✎ Après cette initialisation, on peut remonter dans la hiérarchie des sous-quantificateurs

✎Aller à >>> PROCHAIN_SQ <<<
☞➢‹avant›▶‹a›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv⇒
➢‹avant›▶‹a›◀‹apres›≮/q_cour[↳MINUS(‹nocour›,1)↲]/q_arbo[‹noarbo›]↳GOTORULE(61)↲☜✎[78]

✎ Supprimer tous les curseurs
☞➢‹avant›▶‹a›▷‹b›◁‹c›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/fin1⇒
➢‹avant›▶‹a›‹b›‹c›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/fin1☜✎[79]
☞➢‹avant›▶‹a›➲‹b›➲‹c›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/fin1⇒
➢‹avant›▶‹a›‹b›‹c›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/fin1☜✎[80]
☞➢‹avant›▶‹a›⇨‹b›⇦‹c›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/fin1⇒
➢‹avant›▶‹a›‹b›‹c›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/fin1☜✎[81]
✎ Supprimer tous les marqueurs et tous les compteurs.
☞➢‹avant›▶‹a›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/fin1⇒➢‹avant›▶‹a›◀‹apres›≮☜✎[82]
☞➢‹avant›▶‹a›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]⇒➢‹avant›▶‹a›◀‹apres›≮☜✎[83]

✎ TRAITER LES FONCTIONS PROPOSITIONNELLES LIEES
✎ **********
✎ préparer un paramètre de numéro de quantificateur
☞➢‹avant›▶‹a›◀‹apres›≮⇒➢‹avant›▶‹a›◀‹apres›≮/q_cour[0]/q_arbo[0]/mark☜✎[84]
✎ chercher le numéro de sous-quantificateur le plus élevé (le plus profond)
☞➢‹avant›▶‹a›xSQ_‹nocour›⎡‹z›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark⇒
➢‹avant›▶‹a›xSQ_‹nocour›⎡‹z›◀‹apres›≮
/q_cour[↳PLUS(‹nocour›,1)↲]/q_arbo[↳PLUS(‹noarbo›,1)↲]/mark☜✎[85]
✎ ajuster les compteurs (numéro moins 1)
☞➢‹avant›▶‹a›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark⇒
➢‹avant›▶‹a›◀‹apres›≮/q_cour[↳MINUS(‹nocour›,1)↲]/q_arbo[↳MINUS(‹noarbo›,1)↲]☜✎[86]

✎ >>> PROCHAIN_SQ <<<
✎ sélectionner le sous-quantificateur courant et positionner les curseurs de xTV
☞➢‹avant›▶‹a›xSQ_‹nocour›⎡‹z›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]⇒
➢‹avant›▶‹a›xSQ_‹nocour›⎡➲⇨‹z›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark1☜✎[87]
☞➢‹avant›▶‹a›⁄xSQ_‹nocour›⎤‹z›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark1⇒
➢‹avant›▶‹a›⇦➲⁄xSQ_‹nocour›⎤‹z›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv☜✎[88]

✎ >>> PROCHAIN_FP <<<
✎ sélectionner le prochain xMOP de type fp1 et remettre le compteur noarbo à nocour
☞➢‹avant›▶‹a›⇨‹b›xMOP⦃‹fp1↳IN(fp1)↲›⦄‹c›⇦‹z›◀‹apres›≮
/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv⇒
➢‹avant›▶‹a›‹b›⇨xMOP⦃‹fp1›⦄‹c›⇦‹z›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹nocour›]/mark1☜✎[89]
✎ sélectionner le prochain xMOP de type fp1 et remettre le compteur noarbo à nocour
☞➢‹avant›▶‹a›⇨‹b›xBOP⦃‹fp2↳IN(fp2)↲›⦄‹c›⇦‹z›◀‹apres›≮
/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv⇒
➢‹avant›▶‹a›‹b›⇨xBOP⦃‹fp2›⦄‹c›⇦‹z›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹nocour›]/mark1☜✎[90]

✎ >>> PROCHAIN_NIVEAU_ARBO <<<
✎ sélectionner le quantificateur avec le numéro d'arborescence courant
☞➢‹avant›▶‹a›xQ_‹noarbo›⎣‹z›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark1⇒
➢‹avant›▶‹a›xQ_‹noarbo›⎣▷‹z›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark2☜✎[91]
☞➢‹avant›▶‹a›⁄xQ_‹noarbo›⎦‹z›◀‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark2⇒
➢‹avant›▶‹a›◁⁄xQ_‹noarbo›⎦‹z›◀‹apres›≮
/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mention_pas_trouve☜✎[92]
✎ isoler le fp dans l'encapsulation du terme
☞➢‹avant›▶‹a›⇨x‹type›OP⦃‹fp›⦄‹z›◀‹apres›≮
/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mention_pas_trouve⇒
➢‹avant›▶‹a›⇨x‹type›OP⦃‹fp›⦄‹z›◀‹apres›≮
/⦃‹fp›⦄/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mention1_pas_trouve☜✎[93]
✎ tester si le fp est mentionné dans le quantificateur
☞➢‹avant›▶‹a›▷‹b›xTVQ⦃‹fp›⦄‹c›◁‹z›◀‹apres›≮
/⦃‹fp›⦄/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mention1_pas_trouve⇒

➢‹avant›►‹a›▷‹b›xTVQ[‹fp›]‹c›◁‹z›◄‹apres›≮
/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mention2_pas_trouve☜🖉[94]
🖉 le fp est mentionné dans le quantificateur numéroté noarbo
☞➢‹avant›►‹a›▷‹b›◁‹d›⇨x‹type›OP[‹fp›]‹z›◄‹apres›≮
/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mention2_pas_trouve⇒
➢‹avant›►‹a›‹b›‹d›x‹type›OP_[‹noarbo›]‹fp›]⇨‹z›◄‹apres›≮
/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark_prochain_tv☜🖉[95]
🖉 si trouvé traiter le prochain fp, supprimer les curseurs de quantificateur

🖉 Aller à >>> PROCHAIN_FP <<<
☞➢‹avant›►‹a›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark_prochain_tv⇒
➢‹avant›►‹a›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv↳GOTORULE(89)🖉☜🖉[96]
🖉 si pas trouvé, remonter dans l'arborescence des généralisations
🖉 d'abord vérifier le niveau de la généralisation
☞➢‹avant›►‹a›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[0]/mention‹mark›_pas_trouve⇒
ERREUR : ➢‹avant›►‹a›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[0]/mention‹mark›_pas_trouve,
trouvé fonction propositionnelle libre !↳STOP🖉☜🖉[97]
🖉 supprimer les curseurs de quantificateur
☞➢‹avant›►‹a›▷‹b›◁‹c›◄‹apres›≮‹X›/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mention‹mark›_pas_trouve
⇒➢‹avant›►‹a›‹b›‹c›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mention‹mark›_pas_trouve☜🖉[98]
🖉 remonter dans l'arborescence des généralisations
☞➢‹avant›►‹a›◄‹apres›≮‹X›/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mention‹mark›_pas_trouve⇒➢‹avant›
►‹a›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[↳MINUS(‹noarbo›,1)🖉]/prochain_niveau_arbo☜🖉[99]

🖉 Aller à >>> PROCHAIN_NIVEAU_ARBO <<<
☞➢‹avant›►‹a›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_niveau_arbo⇒
➢‹avant›►‹a›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/mark1↳GOTORULE(91)🖉☜🖉[100]
🖉 si prochain_tv, alors remonter dans la hiérarchies des sous-quantificateurs
🖉 tout d'abord, tester si fin
☞➢‹avant›►‹a›◄‹apres›≮/q_cour[0]/q_arbo[‹noarbo›]/prochain_tv⇒
➢‹avant›►‹a›◄‹apres›≮/q_cour[0]/q_arbo[‹noarbo›]/fin1☜🖉[101]

🖉 supprimer tous les curseurs
☞➢‹avant›►‹a›▷‹b›◁‹c›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv⇒
➢‹avant›►‹a›‹b›‹c›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv☜🖉[102]
☞➢‹avant›►‹a›➲‹b›➲‹c›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv⇒
➢‹avant›►‹a›‹b›‹c›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv☜🖉[103]
☞➢‹avant›►‹a›⇨‹b›⇦‹c›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv⇒
➢‹avant›►‹a›‹b›‹c›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv☜🖉[104]
🖉 remonter dans la hiérarchies des sous-quantificateurs

🖉 Aller à >>> PROCHAIN_SQ <<<
☞➢‹avant›►‹a›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/prochain_tv⇒
➢‹avant›►‹a›◄‹apres›≮/q_cour[↳MINUS(‹nocour›,1)🖉]/q_arbo[‹noarbo›]↳GOTORULE(87)🖉☜🖉[105]

🖉 supprimer tous les curseurs
☞➢‹avant›►‹a›▷‹b›◁‹c›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/fin1⇒
➢‹avant›►‹a›‹b›‹c›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/fin1☜🖉[106]
☞➢‹avant›►‹a›➲‹b›➲‹c›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/fin1⇒
➢‹avant›►‹a›‹b›‹c›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/fin1☜🖉[107]
☞➢‹avant›►‹a›⇨‹b›⇦‹c›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/fin1⇒
➢‹avant›►‹a›‹b›‹c›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/fin1☜🖉[108]
🖉 supprimer tous les marqueurs et compteurs
☞➢‹avant›►‹a›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]/fin1⇒➢‹avant›►‹a›◄‹apres›≮☜🖉[109]
☞➢‹avant›►‹a›◄‹apres›≮/q_cour[‹nocour›]/q_arbo[‹noarbo›]⇒➢‹avant›►‹a›◄‹apres›≮☜🖉[110]

🖉 FILTRER LES ERREURS
☞➢‹avant›►‹a›◄‹apres›≮‹u›/‹x›⇒
ERREUR : ➢‹avant›►‹a›◄‹apres›≮‹u›/‹x›, trouvé un paramètre parasite : ‹u›/‹x› !↳STOP🖉☜🖉[111]
☞➢‹avant›►‹a›●‹z›◄‹apres›≮⇒
ERREUR : ➢‹avant›►‹a›_erreur_●‹z›◄‹apres›≮, trouvé un curseur parasite : ● !↳STOP🖉☜🖉[112]
☞➢‹avant›►‹a›○‹z›◄‹apres›≮⇒
ERREUR : ➢‹avant›►‹a›_erreur_○‹z›◄‹apres›≮, trouvé un curseur parasite : ○ !↳STOP🖉☜🖉[113]
☞➢‹avant›►‹a›→‹z›◄‹apres›≮⇒
ERREUR : ➢‹avant›►‹a›_erreur_→‹z›◄‹apres›≮, trouvé un curseur parasite : → !↳STOP🖉☜🖉[114]
☞➢‹avant›►‹a›←‹z›◄‹apres›≮⇒
ERREUR : ➢‹avant›►‹a›_erreur_←‹z›◄‹apres›≮, trouvé un curseur parasite : ← !↳STOP🖉☜🖉[115]
☞➢‹avant›►‹a›▷‹z›◄‹apres›≮⇒
ERREUR : ➢‹avant›►‹a›_erreur_▷‹z›◄‹apres›≮, trouvé un curseur parasite : ▷ !↳STOP🖉☜🖉[116]

☞➢‹avant›▶‹a›◁‹z›◀‹apres›≮⇒
ERREUR : ➢‹avant›▶‹a›_erreur_◁‹z›◀‹apres›≮, trouvé un curseur parasite : ◁ !↳STOP↲☜✐[117]
☞➢‹avant›▶‹a›➲‹z›◀‹apres›≮⇒
ERREUR : ➢‹avant›▶‹a›_erreur_➲‹z›◀‹apres›≮, trouvé un curseur parasite : ➲ !↳STOP↲☜✐[118]
☞➢‹avant›▶‹a›⊂‹z›◀‹apres›≮⇒
ERREUR : ➢‹avant›▶‹a›_erreur_⊂‹z›◀‹apres›≮, trouvé un curseur parasite : ⊂ !↳STOP↲☜✐[119]
☞➢‹avant›▶‹a›⇨‹z›◀‹apres›≮⇒
ERREUR : ➢‹avant›▶‹a›_erreur_⇨‹z›◀‹apres›≮, trouvé un curseur parasite : ⇨ !↳STOP↲☜✐[120]
☞➢‹avant›▶‹a›⇦‹z›◀‹apres›≮⇒
ERREUR : ➢‹avant›▶‹a›_erreur_⇦‹z›◀‹apres›≮, trouvé un curseur parasite : ⇦ !↳STOP↲☜✐[121]
☞➢‹avant›▶‹a›•‹z›◀‹apres›≮⇒
ERREUR : ➢‹avant›▶‹a›_erreur_•‹z›◀‹apres›≮, élément pas traité !↳STOP↲☜✐[122]

☞↳STOP↲☜✐[123]
↳ENDRULES↲
↳ENDGRAM↲

Prenons un exemple, l'*axiome* 3 de la *protothétique* qui nécessite pour l'ensemble des deux GO_i six cent vingt-cinq itérations :

C_s : ⌊pg⌋⌈≡(⌊f⌋⌈≡(g(pp)≡(⌊r⌋⌈≡(f(rr)g(pp))⌉⌊r⌋⌈≡(f(rr)g(≡(p⌊q⌋⌈q⌉)p))⌉))⌉⌊q⌋⌈g(pq)⌉)⌉⦃PA3⦄

PREANALYSE, 1, 78,1:
➢⦃PA3⦄⌊pg⌋⌈≡(⌊f⌋⌈≡(g(pp)≡(⌊r⌋⌈≡(f(rr)g(pp))⌉⌊r⌋⌈≡(f(rr)g(≡(p⌊q⌋⌈q⌉)p))⌉))⌉⌊q⌋⌈g(pq)⌉)⌉▶xQ_0⌊•xTVQ⦃p⦄•xTVQ⦃g⦄/xQ_0⌋xSQ_0⌈xE_15⦃xBOP⦄≡⦃xPG❪❫xE_8⦃xQ_1⌊•xTVQ⦃f⦄/xQ_1⌋xSQ_1⌈xE_17⦃xBOP⦄≡⦃xPG❪❫xE_2⦃xBOP⦄g⦃xPG❪❫•xTV_1⦃L/p/xTV_1⦄•xTV_2⦃L/p/xTV_2⦄xPD❪❫⌋/xE_2⦄xE_16⦃xBOP⦄≡⦃xPG❪❫xE_9⦃xQ_2⌊•xTVQ⦃r⦄/xQ_2⌋xSQ_2⌈xE_7⦃xBOP⦄≡⦃xPG❪❫xE_3⦃xBOP⦄f⦃xPG❪❫•xTV_3⦃L/r/xTV_3⦄•xTV_4⦃L/r/xTV_4⦄xPD❪❫⌋/xE_3⦄xE_4⦃xBOP⦄g⦃xPG❪❫•xTV_5⦃L/p/xTV_5⦄•xTV_6⦃L/p/xTV_6⦄xPD❪❫⌋/xE_4⦄xPD❪❫⌋/xE_7⦄/xSQ_2⌉/xE_9⦄xE_10⦃xQ_3⌊•xTVQ⦃r⦄/xQ_3⌋xSQ_3⌈xE_18⦃xBOP⦄≡⦃xPG❪❫xE_5⦃xBOP⦄f⦃xPG❪❫•xTV_7⦃L/r/xTV_7⦄•xTV_8⦃L/r/xTV_8⦄xPD❪❫⌋/xE_5⦄xE_14⦃xBOP⦄g⦃xPG❪❫xE_13⦃xBOP⦄≡⦃xPG❪❫•xTV_9⦃L/p/xTV_9⦄xE_11⦃xQ_4⌊•xTVQ⦃q⦄/xQ_4⌋xSQ_4⌈xE_1⦃•xTV_10⦃L/q/xTV_10⦄/xE_1⦄/xSQ_4⌉/xE_11⦄xPD❪❫⌋/xE_13⦄•xTV_11⦃L/p/xTV_11⦄xPD❪❫⌋/xE_14⦄xPD❪❫⌋/xE_18⦄/xSQ_3⌉/xE_10⦄xPD❪❫⌋/xE_16⦄xPD❪❫⌋/xE_17⦄/xSQ_1⌉/xE_8⦄xE_12⦃xQ_5⌊•xTVQ⦃q⦄/xQ_5⌋xSQ_5⌈xE_6⦃xBOP⦄g⦃xPG❪❫•xTV_12⦃L/p/xTV_12⦄•xTV_13⦃L/q/xTV_13⦄xPD❪❫⌋/xE_6⦄/xSQ_5⌉/xE_12⦄xPD❪❫⌋/xE_15⦄/xSQ_0⌉◀≮

PROCDEF, 1, 109,1:
➢⦃PA3⦄⌊pg⌋⌈≡(⌊f⌋⌈≡(g(pp)≡(⌊r⌋⌈≡(f(rr)g(pp))⌉⌊r⌋⌈≡(f(rr)g(≡(p⌊q⌋⌈q⌉)p))⌉))⌉⌊q⌋⌈g(pq)⌉)⌉▶xQ_0⌊xTVQ⦃g⦄xTVQ⦃p⦄/xQ_0⌋xSQ_0⌈xE_15⦃xBOP⦄≡⦃xPG❪❫xE_8⦃xQ_1⌊xTVQ⦃f⦄/xQ_1⌋xSQ_1⌈xE_17⦃xBOP⦄≡⦃xPG❪❫xE_2⦃xBOP_[0]⦄g⦃xPG❪❫xTV_[0]_1⦃L/p/xTV_1⦄xTV_[0]_2⦃L/p/xTV_2⦄xPD❪❫⌋/xE_2⦄xE_16⦃xBOP⦄≡⦃xPG❪❫xE_9⦃xQ_2⌊xTVQ⦃r⦄/xQ_2⌋xSQ_2⌈xE_7⦃xBOP⦄≡⦃xPG❪❫xE_3⦃xBOP_[1]⦄f⦃xPG❪❫xTV_[2]_3⦃L/r/xTV_3⦄xTV_[2]_4⦃L/r/xTV_4⦄xPD❪❫⌋/xE_3⦄xE_4⦃xBOP_[0]⦄g⦃xPG❪❫xTV_[0]_5⦃L/p/xTV_5⦄xTV_[0]_6⦃L/p/xTV_6⦄xPD❪❫⌋/xE_4⦄xPD❪❫⌋/xE_7⦄/xSQ_2⌉/xE_9⦄xE_10⦃xQ_3⌊xTVQ⦃r⦄/xQ_3⌋xSQ_3⌈xE_18⦃xBOP⦄≡⦃xPG❪❫xE_5⦃xBOP_[1]⦄f⦃xPG❪❫xTV_[3]_7⦃L/r/xTV_7⦄xTV_[3]_8⦃L/r/xTV_8⦄xPD❪❫⌋/xE_5⦄xE_14⦃xBOP_[0]⦄g⦃xPG❪❫xE_13⦃xBOP⦄≡⦃xPG❪❫xTV_[0]_9⦃L/p/xTV_9⦄xE_11⦃xQ_4⌊xTVQ⦃q⦄/xQ_4⌋xSQ_4⌈xE_1⦃xTV_[4]_10⦃L/q/xTV_10⦄/xE_1⦄/xSQ_4⌉/xE_11⦄xPD❪❫⌋/xE_13⦄xTV_[0]_11⦃L/p/xTV_11⦄xPD❪❫⌋/xE_14⦄xPD❪❫⌋/xE_18⦄/xSQ_3⌉/xE_10⦄xPD❪❫⌋/xE_16⦄xPD❪❫⌋/xE_17⦄/xSQ_1⌉/xE_8⦄xE_12⦃xQ_5⌊xTVQ⦃q⦄/xQ_5⌋xSQ_5⌈xE_6⦃xBOP_[0]⦄g⦃xPG❪❫xTV_[0]_12⦃L/p/xTV_12⦄xTV_[5]_13⦃L/q/xTV_13⦄xPD❪❫⌋/xE_6⦄/xSQ_5⌉/xE_12⦄xPD❪❫⌋/xE_15⦄/xSQ_0⌉◀≮

Dépouillement des expressions en version contextuelle

Pour tester la GO_i, DEPX, nous donnons pour commencer une petite grammaire qui l'enchaîne à un traitement qui précède et qui conduit au format *langage pivot*.

Enchaînement de GO_i de tests pour DEPX

↳GRAM↲
☞GRAMNAME⇒TESTDEPX☜

↳INCLUDE↲
☞PREANALYSE⇒"D:\TheseL\GramXAnalyse\GX01PreAnalyse.txt"☜
☞PROCDEF⇒"D:\TheseL\GramXAnalyse\GX02ProcDef.txt"☜
☞DEPX⇒"D:\TheseL\GramXDepouille\GX05Depouille.txt"☜
↳ENDINCLUDE↲

↳PARAMS↲
☞MAXGRAMLOOP⇒2☜
☞MAXRULELOOP⇒5☜
↳ENDPARAMS↲

↳RULES↲
🖉 entrer la formule depuis INPUT_CORPUS
☞⌊‹a›⇒➢⌊‹a›▶⌊‹a›◀🡐☜🖉[1]
☞≡‹a›⇒➢≡‹a›▶≡‹a›◀🡐☜🖉[2]

🖉 PREPARER ET EXECUTER L'ANALYSE DES TERMINAUX
🖉 ET CONSTRUCTION DES TERMES ET EXPRESSIONS
☞↳EXECUTEGRAM(PREANALYSE)↲☜🖉[3]

🖉 VERIFIER PROCEDURE DEFINITOIRE, TRIER LES VARIABLES DANS LES
🖉 DIFFERENTES EXPRESSIONS, TESTER LES LIAISONS ENTRE LIEURS ET VARIABLES LIES
🖉 ET MEMORISERS CES LIAISONS
☞↳EXECUTEGRAM(PROCDEF)↲☜🖉[4]

🖉 DEPOUILLER
☞↳EXECUTEGRAM(DEPX)↲☜🖉[4]

☞↳STOP↲☜🖉[]
↳ENDRULES↲
↳ENDGRAM↲

DEPX

↳GRAM↲
☞GRAMNAME⇒DEPX☜
🖉 Dépouiller une formule de format : langage pivot de ses encapsulations

↳PARAMS↲
☞MAXGRAMLOOP⇒20☜
☞MAXRULELOOP⇒60☜
↳ENDPARAMS↲

↳SETS↲

🖉 Fonctions Propositionnelles
☞fp2⇒"f","g"☜🖉 fonctions propositionnelles à deux places
☞fp1⇒"h"☜🖉 fonctions propositionnelles à une place
☞fp⇒fp2,fp1☜🖉 toutes les fonctions propositionnelles

🖉 Constantes, termes constants
☞bic⇒"≡"☜🖉 biconditionnelle
☞cp0⇒"V","F","T"☜🖉 constantes propositionnelles sans contexte
☞tc1⇒"~","¬","α","!","♭","↓"☜🖉 constantes unaires
☞tc2⇒"μ","Ц","ε","∧","∨","⊃"☜🖉 constantes avec contexte binaire
☞tcn⇒"Δ","δ"☜🖉 constantes n_aires
☞tcmc⇒"τ","ω"☜🖉 constantes multi-contextes
☞tcac⇒tc1,tc2,tcn,tcmc☜🖉 les constantes avec contexte(s) sans BIC
☞mop⇒tc1,fp1☜🖉 opérateurs unaires
☞bop⇒tc2,bic,fp2☜🖉 opérateurs binaires
☞xop⇒tcn,tcmc☜🖉 opérateurs complexes, multi-contextes

🖉 Parenthèses gauches et délimiteurs gauches de contexte
☞pgA⇒"("☜
☞pgB⇒"["☜
☞pgC⇒"{"☜
☞pgD⇒"<"☜
☞pgE⇒"⟮"☜
🖉 ...
☞pg⇒pgA,pgB,pgC,pgD,pgE☜🖉 ...

🖉 Parenthèses droites et délimiteurs droits de contexte
☞pdA⇒")"☜
☞pdB⇒"]"☜
☞pdC⇒"}"☜
☞pdD⇒">"☜
☞pdE⇒"⟯"☜
🖉 ...

☞pd⇒pdA,pdB,pdC,pdD,pdE☜🖉 ...
↳ENDSETS↲

↳RULES↲
🖉 VERIFIER QUE LA FORMULE N'EST PAS VIDE
☞➢‹avant›▶◀‹apres›≺⇒
ERREUR : ➢‹avant›▶_erreur_fomule_vide◀‹apres›≺, trouvé formule à analyser vide !↳STOP↲☜🖉[1]
🖉 VERIFIER QUE LA FORMULE A ANALYSER EST BIEN ENCAPSULEE PAR : ➢...▶‹formule›◀...≺
☞➢‹avant›▶‹v›◀‹apres›≺⇒➢‹avant›▶‹v›◀‹apres›≺/ok☜🖉[2]
🖉 préparer la zone cible et de traitement
☞➢‹avant›▶‹v›◀‹apres›≺/ok⇒➢‹avant›▶‹v›◀‹apres›≺▷‹v›◁/ok☜🖉[3]

🖉 on va de l'extérieur vers l'intérieur des encapsulations

🖉 supprimer les encapsulations des généralisations
🖉 supprimer les encapsulations des quantificateurs
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xQ_‹i↳INTEGER↲›⌊‹v›/xQ_‹i›⌋‹z›◁/ok⇒
➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›⌊‹v›⌋‹z›◁/ok☜🖉[4]
🖉 supprimer les encapsulations des sous-quantificateurs
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xSQ_‹i↳INTEGER↲›⌈‹v›/xSQ_‹i›⌉‹z›◁/ok⇒
➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›⌈‹v›⌉‹z›◁/ok☜🖉[5]
🖉 supprimer les encapsulations des expressions
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xE_‹i↳INTEGER↲›{‹v›/xE_‹i›}‹z›◁/ok⇒
➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[6]
🖉 supprimer les encapsulations des délimiteurs gauches
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xPG‹v↳IN(pg)↲›‹z›◁/ok⇒
➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[7]
🖉 supprimer les encapsulations des délimiteurs droits
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xPD‹v↳IN(pd)↲›‹z›◁/ok⇒
➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[8]
🖉 supprimer les encapsulations des métatermes
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xTV_[‹lien›]_‹i↳INTEGER↲›{‹type›/‹v›/xTV_‹i›}‹z›◁/ok⇒
➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[9]
🖉 supprimer les encapsulations des contextes
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xCTX_‹i↳INTEGER↲›[‹v›/xCTX_‹i›]‹z›◁/ok⇒
➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[10]
🖉 supprimer les encapsulations des bop
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xBOP‹possiblelien›‹v↳IN(bop)↲›‹z›◁/ok⇒
➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[11]
🖉 supprimer les encapsulations des mop
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xMOP‹possiblelien›‹v↳IN(mop)↲›‹z›◁/ok⇒
➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[12]
🖉 supprimer les encapsulations des tcac
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xTCAC{‹v↳IN(tcac)↲›}‹z›◁/ok⇒
➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[13]
🖉 dépouiller les délimiteurs des tv (termes variables) des quantificateur
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xTVQ{‹v›}‹z›◁/ok⇒
➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[14]

🖉 vérification
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xT‹z›◁/ok⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_xT‹z›◁/ok,
trouvé encore encapsulation : xT !↳STOP↲☜🖉[15]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xE‹z›◁/ok⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_xE‹z›◁/ok,
trouvé encore encapsulation : xE !↳STOP↲☜🖉[16]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xP‹z›◁/ok⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_xP‹z›◁/ok, trouvé encore encapsulation : xP
!↳STOP↲☜🖉[17]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xQ‹z›◁/ok⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_xQ‹z›◁/ok,
trouvé encore encapsulation : xQ !↳STOP↲☜🖉[18]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xS‹z›◁/ok⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_xS‹z›◁/ok,
trouvé encore encapsulation : xQ !↳STOP↲☜🖉[19]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xC‹z›◁/ok⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_xC‹z›◁/ok,
trouvé encore encapsulation : xC !↳STOP↲☜🖉[20]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xM‹z›◁/ok⇒

ERREUR : ➢‹avant›►‹milieu›◄‹apres›≺▷‹a›_erreur_xM‹z›◁/ok,
trouvé encore encapsulation : xM !↳STOP↲☜✎[21]
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›xB‹z›◁/ok⇒
ERREUR : ➢‹avant›►‹milieu›◄‹apres›≺▷‹a›_erreur_xB‹z›◁/ok,
trouvé encore encapsulation : xB !↳STOP↲☜✎[22]
✎ supprimer la marque
☞➢‹x›≺▷‹y›◁/ok⇒➢‹x›≺▷‹y›◁☜✎[23]

☞↳STOP↲☜✎[24]
↳ENDRULES↲
↳ENDGRAM↲

Détermination des catégories syntaxico-sémantiques

Pour tester la GO_i, CATEGORIE, nous nous donnons une GO_i qui chaîne les grammaires nécessaires à mettre dans la *forme analysée* la *chaîne* de caractères qu'elle peut traiter.

GO_i de tests pour la détermination des catégories syntaxico-sémantiques

↳GRAM↲
☞GRAMNAME⇒TESTCAT☜

↳INCLUDE↲
☞PREANALYSE⇒"D:\TheseL\GramXAnalyse\GX01PreAnalyse.txt"☜
☞PROCDEF⇒"D:\TheseL\GramXAnalyse\GX02ProcDef.txt"☜
☞CATEGORIE⇒"D:\TheseL\GramXCategorie\GX03Categorie.txt"☜
↳ENDINCLUDE↲

↳PARAMS↲
☞MAXGRAMLOOP⇒2☜
☞MAXRULELOOP⇒5☜
↳ENDPARAMS↲

↳RULES↲
✎ entrer la formule depuis INPUT_CORPUS
☞⌊‹a›⇒➢⌊‹a›►⌊‹a›◄≺☜✎[1]
☞≡‹a›⇒➢≡‹a›►≡‹a›◄≺☜✎[2]

✎ PREPARER ET EXECUTER ANALYSE DES TERMINAUX ET
✎ ENCAPSULATION DES TERMES ET EXPRESSIONS
☞↳EXECUTEGRAM(PREANALYSE)↲☜✎[3]

✎ VERIFIER LA PROCEDURE DEFINITOIRE, TRIER LES VARIABLES,
✎ ETALBIR LES LIAISONS POUR LES TERMES VARIABLES LIES
☞↳EXECUTEGRAM(PROCDEF)↲☜✎[4]

✎ ANALYSER ET INSCRIRE LES CATEGORIES SYNTAXICO-SEMANTIQUES
☞↳EXECUTEGRAM(CATEGORIE)↲☜✎[5]

☞↳STOP↲☜✎[6]
↳ENDRULES↲
↳ENDGRAM↲

CATEGORIE

↳GRAM↲
☞GRAMNAME⇒CATEGORIE☜
✎ CONSTRUCTION DES CATEGORIES SYNTAXICO-SEMANTIQUES
✎ dans cette version :
✎ ne fonctionne que pour un definiendum de niveau 0
✎ ne traite pas les définitions comme : Dp13 de la protothétique de type S/(S/SS)/SS
✎ la formule à analyser se trouve encapsulée entre : ►‹formule›◄, elle doit être traitée comme :
➢‹avant›►‹formule›◄‹apres›≺
✎ on ajoute un zone de traitement : ➢‹avant›►‹formule›◄‹apres›≺▷‹zonedetraitement›◁
✎ les erreurs sont signalées mais ne stoppent pas l'analyse !

↳PARAMS↲
☞MAXGRAMLOOP⇒2☜

☞MAXRULELOOP⇒20☜
↳ENDPARAMS↲

↳SETS↲
🖉 Variables
☞vp⇒"p","q","r","s"☜🖉 variables propositionnelles
☞vns⇒"A","B","C","D"☜🖉 variables de nom singulier
☞vng⇒"a","b"☜🖉 variables de nom générique
☞vn⇒vns,vng☜🖉 les symboles de variable de nom

🖉 Fonctions Propositionnelles
☞fp2⇒"f","g"☜🖉 fonctions propositionnelles à deux places
☞fp1⇒"h"☜🖉 fonctions propositionnelles à une place
☞fp⇒fp2,fp1☜🖉 toutes les fonctions propositionnelles

🖉 Constantes, termes constants
☞bic⇒"≡"☜🖉 biconditionnelle
☞tc0⇒"V","F","T"☜🖉 connstantes sans contexte
☞tc1⇒"~","¬","α","!","♭","↓"☜🖉 constantes unaires
☞tc2⇒"μ","Ц","ε","∧","∨","⊃"☜🖉 constantes avec contexte binaire
☞tcn⇒"Δ","δ"☜🖉 constantes n_aires
☞tcmc⇒"τ","ω"☜🖉 constantes multi-contextes
☞tcac⇒tc1,tc2,tcn,tcmc☜🖉 les constantes avec contexte(s) sans BIC

🖉 Parenthèses gauches et délimiteurs gauches de contexte
☞pgA⇒"("☜
☞pgB⇒"["☜
☞pgC⇒"{"☜
☞pgD⇒"<"☜
☞pgE⇒"⟮"☜
🖉 ...
☞pg⇒pgA,pgB,pgC,pgD,pgE☜🖉 ...

🖉 Parenthèses droites et délimiteurs droits de contexte
☞pdA⇒")"☜
☞pdB⇒"]"☜
☞pdC⇒"}"☜
☞pdD⇒">"☜
☞pdE⇒"⟯"☜
🖉 ...
☞pd⇒pdA,pdB,pdC,pdD,pdE☜🖉 ...
↳ENDSETS↲

↳RULES↲
🖉 copier la formule dans la zone de traitement
☞➢‹avant›▶xQ_‹i›⌊‹a›/xQ_‹i›⌋xSQ_‹i›⌈‹b›/xSQ_‹i›⌉◀‹apres›◁⇒
➢‹avant›▶xQ_‹i›⌊‹a›/xQ_‹i›⌋xSQ_‹i›⌈‹b›/xSQ_‹i›⌉◀‹apres›◁
▷xQ_‹i›⌊‹a›/xQ_‹i›⌋xSQ_‹i›⌈‹b›/xSQ_‹i›⌉◁☜🖉[1]

🖉 CONSTRUCTION DES CATEGORIES SYNTAXICO-SEMANTIQUES
🖉 **********
🖉 à part pour bic qui est prédéfinie comme terme primitif, seules les thèses sont concernées

🖉 par défaut : axiome : /CAT=⦃⟦1:cat⦄S/SS⦃cst⦄≡⦃ctx⦄(--)⦃⟧⦄
☞➢‹avant›▶‹formule›◀‹apres›◁▷‹a›◁⇒➢‹avant›▶‹formule›◀‹apres›◁▷‹a›◁/axiome☜🖉[2]
🖉 identifier si l'on a affaire à une Thèse, dépister une Thèse
☞➢‹avant›▶‹formule›◀‹apres›◁▷‹a›xE_‹i›⦃xTCAC⦃‹tn›↳IN(tcac)↲›⦄‹b›◁/axiome⇒
➢‹avant›▶‹formule›◀‹apres›◁▷‹a›xE_‹i›⦃xTCAC⦃‹tn›⦄‹b›◁/these☜🖉[3]
🖉 inscrire des paramètres temporaires
☞➢‹avant›▶‹formule›◀‹apres›◁▷‹a›◁/these⇒
➢‹avant›▶‹formule›◀‹apres›◁▷‹a›◁
/these/TMP=⦃CAT=/tcacTmp=/varPerm0=?/varTmp=?/catTmp=S/⦄☜🖉[4]

🖉 >>> CONTEXTE <<<
🖉 extraire le contenu du sous-quantificateur (courant)
☞➢‹avant›▶‹formule›◀‹apres›◁▷‹a›xQ_‹i›↳INTEGER↲⌊‹b›/xQ_‹i›⌋xSQ_‹i›⌈‹c›/xSQ_‹i›⌉‹z›◁
/these/TMP=⦃‹u›⦄⇒
➢‹avant›▶‹formule›◀‹apres›◁▷‹a›xQ_‹i›⌊‹b›/xQ_‹i›⌋xSQ_‹i›⌈•→‹c›←•/xSQ_‹i›⌉‹z›◁
/these/TMP=⦃‹u›⦄/mark1☜🖉[5]
🖉 extraire le contenu de la métaexpression tcac (Definiendum)
☞➢‹avant›▶‹formule›◀‹apres›◁▷‹a›•→‹b›xE_‹i›⦃xTCAC⦃‹tn›↳IN(tcac)↲›⦄‹v›/xE_‹i›⦄‹c›←•‹z›◁

/these/TMP=⟦‹u›⟧/mark1⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›‹b›xE_‹i›⟦●→xTCAC⟦‹tn›⟧‹v›←●/xE_‹i›⟧‹c›‹z›◁
/these/TMP=⟦‹u›⟧/mark2☜🖉[6]
🖉 éjecter et mémoriser le tcac
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→xTCAC⟦‹t›⟧‹v›←●‹z›◁
/these/TMP=⟦CAT=‹x›/tcacTmp=‹m›/‹n›⟧/mark2⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xTCAC⟦‹t›⟧●→‹v›←●‹z›◁
/these/TMP=⟦CAT=‹x›/tcacTmp=‹t›/‹n›⟧/mark3☜🖉[7]
🖉 il ne reste plus que la liste des contextes
🖉 partir du dernier pas traité, serrer de la droite vers la gauche en éliminant le dernier traité
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→‹b›xctx_mark_‹i›⟦‹v›/xctx_mark_‹i›⟧←●‹z›◁
/these/TMP=‹u›/mark3⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→‹b›←●xctx_mark_‹i›⟦‹v›/xctx_mark_‹i›⟧‹z›◁
/these/TMP=‹u›/mark3☜🖉[8]
🖉 cerner le nouveau contexte pas traité et créer une nouvelle structure de mémorisation de catégorie
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→‹b›xCTX_‹i›⟦‹v›/xCTX_‹i›⟧←●‹z›◁
/these/TMP=⟦CAT=‹u›/‹w›⟧/mark3⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›‹b›xctx_mark_‹i›⟦●→‹v›←●/xctx_mark_‹i›⟧‹z›◁
/these/TMP=⟦CAT=⟦num⟧‹i›⟦cat⟧⟦cst⟧⟦ctx⟧⟦⟧‹u›/‹w›⟧/mark3☜🖉[9]
🖉 extraire les délimiteurs du contexte courant, mémoriser les délimiteurs du contexte
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→xPG⟦‹pg↳IN(pg)⇙›⟧xTV_⟦‹noarbo›⟧_0⟦D/‹p›/xTV_0⟧
xPD⟦‹pd↳IN(pd)⇙›⟧←●‹z›◁/these/TMP=⟦CAT=⟦num⟧‹o›⟦cat⟧⟦cst⟧⟦ctx⟧⟦⟧‹u›/‹x›⟧/mark3⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xPG⟦‹pg›⟧●→xTV_⟦‹noarbo›⟧_0⟦D/‹p›/xTV_0⟧←●xPD⟦‹pd›⟧‹z›◁
/these/TMP=⟦CAT=⟦num⟧‹o›⟦cat⟧⟦cst⟧⟦ctx⟧pg⟦‹pg›⟧pd⟦‹pd›⟧⟦⟧‹u›/‹x›⟧/mark3☜🖉[10]

🖉 >>> TERME_VARIABLE <<<
🖉 détermine la catégorie des termes variables, détecte la catégorie des noms
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→‹b›xTV_⟦‹noarbo›⟧_0⟦D/‹v↳IN(vn)⇙›/xTV_0⟧‹c›←●‹z›◁
/these/TMP=⟦‹x›/varPerm‹n›=‹m›/varTmp=‹o›/‹k›⟧/mark3⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→‹b›xtv_mark_⟦‹noarbo›⟧_0⟦D/‹v›/xTV_0⟧‹c›←●‹z›◁
/these/TMP=⟦‹x›/varPerm‹n›=‹m›/varTmp=N/‹k›⟧/mark4☜🖉[11]
🖉 détecte la catégorie des propositions
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→‹b›xTV_⟦‹noarbo›⟧_0⟦D/‹v↳IN(vp)⇙›/xTV_0⟧‹c›←●‹z›◁
/these/TMP=⟦‹x›/varPerm‹n›=‹m›/varTmp=‹o›/‹k›⟧/mark3⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→‹b›xtv_mark_⟦‹noarbo›⟧_0⟦D/‹v›/xTV_0⟧‹c›←●‹z›◁
/these/TMP=⟦‹x›/varPerm‹n›=‹m›/varTmp=S/‹k›⟧/mark4☜🖉[12]
🖉 si premier coup, enregistrer la nouvelle catégorie
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/these/TMP=⟦‹x›/varPerm0=‹m›/varTmp=‹o›/‹k›⟧/mark4⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/these/TMP=⟦‹x›/varPerm1=‹o›/varTmp=‹o›/‹k›⟧/mark4☜🖉[13]
🖉 si pas premier coup comparer
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/these/TMP=⟦‹x›/varPerm1=‹m›/varTmp=‹m›/‹k›⟧/mark4⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁
/these/TMP=⟦‹x›/varPerm1=‹m›/varTmp=‹m›/‹k›⟧/mark4_ok☜🖉[14]
🖉 mémoriser erreur
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/these/TMP=⟦‹x›/varPerm1=‹m›/varTmp=‹o›/‹k›⟧/mark4⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁
/these/TMP=⟦‹x›/varPerm1=‹m›/varTmp=‹o›/‹k›⟧/mark4/
[erreur : termes variables de catégories différentes dans un même contexte]☜🖉[15]

🖉 augmente d'une position le degré du contexte,
🖉 met à jour la catégorie temporaire et réinitialise la marque
🖉 attention : le « moins » dans ctx⟧pg⟦‹pg›⟧-pd⟦‹pd›⟧⟦ n'est pas un caractère parasite
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁
/these/TMP=⟦CAT=⟦num⟧‹o›⟦cat⟧⟦cst⟧⟦ctx⟧pg⟦‹pg›⟧‹d›pd⟦‹pd›⟧⟦⟧‹f›
/varPerm1=‹g›/‹h›/catTmp=‹m›⟧/mark4_ok⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁
/these/TMP=⟦CAT=⟦num⟧‹o›⟦cat⟧⟦cst⟧⟦ctx⟧pg⟦‹pg›⟧‹d›-pd⟦‹pd›⟧⟦⟧‹f›
/varPerm1=‹g›/‹h›/catTmp=‹m›‹g›⟧/mark3☜🖉[16]

🖉 ALLER A >>> TERME_VARIABLE <<<
☞↳IFMATCH_GOTORULE(11)⇙☜🖉 prochain Terme Variable [17]
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁
/these/TMP=⟦CAT=⟦num⟧‹o›⟦cat⟧⟦cst⟧⟦ctx⟧pg⟦‹pg›⟧‹d›pd⟦‹pd›⟧⟦⟧‹f›
/varPerm1=‹g›/‹h›/catTmp=‹m›⟧/mark3⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁
/these/TMP=⟦CAT=⟦num⟧‹o›⟦cat⟧‹m›⟦cst⟧⟦ctx⟧‹pg›‹d›‹pd›⟦⟧‹f›
/varPerm1=‹g›/‹h›/catTmp=(‹m›)/⟧☜🖉[18]
🖉 supprimer les curseurs

☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→‹v›←●‹z›◁/these/TMP=⟦‹x›⟧⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›‹v›‹z›◁/these/TMP=⟦‹x›⟧☜🖉[19]

🖉 ALLER A >>> CONTEXTE <<<
☞↳IFMATCH_GOTORULE(5)⮰☜🖉 prochain contexte [20]

🖉 supprimer les curseurs
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→←●‹z›◁/these/TMP=⟦‹x›⟧/mark3⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›‹z›◁/these/TMP=⟦‹x›⟧☜🖉[21]
🖉 supprimer les marques de contexte
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xctx_mark‹z›◁/these/TMP=⟦‹x›⟧⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xCTX‹z›◁/these/TMP=⟦‹x›⟧☜🖉[22]
🖉 supprimer les marques de termes
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xtv_mark‹z›◁/these/TMP=⟦‹x›⟧⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xTV‹z›◁/these/TMP=⟦‹x›⟧☜🖉[23]

🖉 mettre à jour les constantes dans les paramètres catégories
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁
/these/TMP=⟦CAT=‹e›⟦num⟧‹i›⟦cat⟧‹c›⟦cst⟧⟦ctx⟧‹d›⟦⟧‹f›/tcacTmp=‹t›/‹x›⟧⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁
/these/TMP=⟦CAT=‹e›⟦num⟧‹i›⟦cat⟧‹c›⟦cst⟧⟦ctx⟧‹d›⟦⟧‹f›/tcacTmp=‹t›/‹x›/cstTmp=‹t›⟧/mark☜🖉[24]
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁
/these/TMP=⟦CAT=‹e›⟦num⟧‹i›⟦cat⟧‹c›⟦cst⟧⟦ctx⟧‹d›⟦⟧‹f›/tcacTmp=‹t›/‹x›/cstTmp=‹g›⟧/mark⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁
/these/TMP=⟦CAT=‹e›⟦mark_num⟧‹i›⟦cat⟧‹c›⟦cst⟧‹g›⟦ctx⟧‹d›⟦⟧‹f›
/tcacTmp=‹t›‹d›/‹x›/cstTmp=‹g›‹d›⟧/mark☜🖉[25]
🖉 supprimer les marqueurs et autres paramètres inutiles
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/these/TMP=⟦CAT=‹c›⟦mark_num⟧‹d›⟦‹e›⟧‹f›/‹g›⟧/mark⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/these/TMP=⟦CAT=‹c›⟦‹d›:‹e›⟧‹f›/‹g›⟧/mark☜🖉[26]
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/these/TMP=⟦CAT=‹c›/‹d›⟧/mark⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/these/TMP=⟦CAT=‹c›⟧☜🖉[27]

🖉 FILTRER LES ERREURS
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/these/‹u›/mark‹x›⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/these/‹u›/mark‹x›/[erreur : marqueur parasite : mark‹x›]☜🖉[28]
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●‹z›◁‹y›⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›_erreur_●‹z›◁‹y›/[erreur : curseur parasite : ●]☜🖉[29]
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›○‹z›◁‹y›⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›_erreur_○‹z›◁‹y›/[erreur : curseur parasite : ○]☜🖉[30]
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›→‹z›◁‹y›⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›_erreur_→‹z›◁‹y›/[erreur : curseur parasite : →]☜🖉[31]
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›←‹z›◁‹y›⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›_erreur_←‹z›◁‹y›/[erreur : curseur parasite : ←]☜🖉[32]

🖉 copier la catégorie dans les métaparamèters
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/axiome⇒
➢‹avant›▶‹formule›◀‹apres›/CAT=⟦⟦1:cat⟧S/SS⟧‹cst⟧≡⟧‹ctx⟧(--)⟧⟧≺☜🖉[33]
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁‹b›/CAT=⟦‹c›⟧‹d›≺/these/TMP=⟦CAT=‹e›⟧⇒
➢‹avant›▶‹formule›◀‹apres›/CAT=⟦‹e›⟧‹d›≺☜🖉[34]
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/these/TMP=⟦CAT=‹e›⟧⇒
➢‹avant›▶‹formule›◀‹apres›/CAT=⟦‹e›⟧≺☜🖉[35]

🖉 copier une possible erreur dans les métaparamètres
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/‹param›/[erreur : ‹err›]⇒
➢‹avant›▶‹formule›◀‹apres›/CAT=⟦erreur : ‹err›⟧≺☜🖉[36]

☞↳STOP⮰☜🖉[37]
↳ENDRULES⮰
↳ENDGRAM⮰

Voici une premier exemple :

cs : ⌊qpr⌋⌈≡(ω[p]\q/(r)≡(~(≡(pq))r))⌉

TESTCAT, 1, 1, 1: ➢⌊qpr⌋⌈≡(ω[p]\q/(r)≡(~(≡(pq))r))⌉▶⌊qpr⌋⌈≡(ω[p]\q/(r)≡(~(≡(pq))r))⌉◀≺

PREANALYSE, 1, 78, 1:
➢⌊qpr⌋⌈≡(ω[p]\q/(r)≡(~(≡(pq))r))⌉▶xQ_0⌊•xTVQ⟦q⟧•xTVQ⟦p⟧•xTVQ⟦r⟧/xQ_0⌋xSQ_0⌈xE_5⟦xBOP⟧≡⟧x
PG⟦⟦⟧xE_1⟦xTCAC⟦ω⟧xCTX_1⟦xPG⟦⟦⟧•xTV_0⟦D/p/xTV_0⟧xPD⟦⟧⟧/xCTX_1⟧xCTX_2⟦xPG⟦⟧•xTV_0⟦D
/q/xTV_0⟧xPD⟦⟧⟧/xCTX_2⟧xCTX_3⟦xPG⟦⟧•xTV_0⟦D/r/xTV_0⟧xPD⟦⟧⟧/xCTX_3⟧/xE_1⟧xE_4⟦xBOP⟧≡

⟨xPG[(]xE_3[xMOP]~⟨xPG[(]xE_2[xBOP]≡⟨xPG[(]•xTV_1[L/p/xTV_1]•xTV_2[L/q/xTV_2]xPD[)]/
xE_2]xPD[)]/xE_3]•xTV_3[L/r/xTV_3]xPD[)]/xE_4]xPD[)]/xE_5]/xSQ_0⟩◄≺

PROCDEF, 1, 109, 1:
➢⌊qpr⌋⌈≡(ω[p]\q/(r)≡(~(≡(pq))r))⌉►xQ_0⌊xTVQ{p}xTVQ{q}xTVQ{r}/xQ_0⌋xSQ_0⌈xE_5[xBOP]≡⟨xP
G[(]xE_1[xTCAC{ω}xCTX_1[xPG[]xTV_[0]_0[D/p/xTV_0]xPD[]]/xCTX_1]xCTX_2[xPG[(]xTV_[0
]_0[D/q/xTV_0]xPD[)]/xCTX_2]xCTX_3[xPG[(]xTV_[0]_0[D/r/xTV_0]xPD[)]/xCTX_3]/xE_1]xE_
4[xBOP]≡⟨xPG[(]xE_3[xMOP]~⟨xPG[(]xE_2[xBOP]≡⟨xPG[(]xTV_[0]_1[L/p/xTV_1]xTV_[0]_2[L
/q/xTV_2]xPD[)]/xE_2]xPD[)]/xE_3]xTV_[0]_3[L/r/xTV_3]xPD[)]
/xE_4]xPD[)]/xE_5]/xSQ_0⟩◄≺

CATEGORIE, 1, 35, 1:
➢⌊qpr⌋⌈≡(ω[p]\q/(r)≡(~(≡(pq))r))⌉►xQ_0⌊xTVQ{p}xTVQ{q}xTVQ{r}/xQ_0⌋xSQ_0⌈xE_5[xBOP]≡⟨xP
G[(]xE_1[xTCAC{ω}xCTX_1[xPG[]xTV_[0]_0[D/p/xTV_0]xPD[]]/xCTX_1]xCTX_2[xPG[(]xTV_[0
]_0[D/q/xTV_0]xPD[)]/xCTX_2]xCTX_3[xPG[(]xTV_[0]_0[D/r/xTV_0]xPD[)]/xCTX_3]/xE_1]xE_
4[xBOP]≡⟨xPG[(]xE_3[xMOP]~⟨xPG[(]xE_2[xBOP]≡⟨xPG[(]xTV_[0]_1[L/p/xTV_1]xTV_[0]_2[L
/q/xTV_2]xPD[)]/xE_2]xPD[)]/xE_3]xTV_[0]_3[L/r/xTV_3]xPD[)]/xE_4]xPD[)]/xE_5]/xSQ_0⟩◄
/CAT=⟩[**1:cat**⟩**((S/S)/S)/S**⟨**cst**⟩**ω**⟨**ctx**⟩**[-]**⟨]⟩[**2:cat**⟩**(S/S)/S**⟨**cst**⟩**ω[-]**⟨**ctx**⟩**\-/**⟨]⟩
[**3:cat**⟩**S/S**⟨**cst**⟩**ω[-]\-/**⟨**ctx**⟩**(-)**⟨]⟩≺

Voici un deuxième exemple basé sur la *catégorie* des *noms* :

cs : ⌊a⌋⌈≡(!{a}~(⌊A⌋⌈~(ε{Aa})⌉))⌉

TESTCAT, 1, 1, 1: ➢⌊a⌋⌈≡(!{a}~(⌊A⌋⌈~(ε{Aa})⌉))⌉►⌊a⌋⌈≡(!{a}~(⌊A⌋⌈~(ε{Aa})⌉))⌉◄≺

PREANALYSE, 1, 78, 1:
➢⌊a⌋⌈≡(!{a}~(⌊A⌋⌈~(ε{Aa})⌉))⌉►xQ_0⌊•xTVQ{a}/xQ_0⌋xSQ_0⌈xE_6[xBOP]≡⟨xPG[(]xE_1[xTCAC{!}x
CTX_1[xPG[{]•xTV_0[D/a/xTV_0]xPD[}]/xCTX_1]/xE_1]xE_5[xMOP]~⟨xPG[(]xE_4[xQ_1⌊•xTVQ{
A}/xQ_1⌋xSQ_1⌈xE_3[xMOP]~⟨xPG[(]xE_2[xBOP]ε⟨xPG[{]•xTV_1[L/A/xTV_1]•xTV_2[L/a/xTV_2
]xPD[}]/xE_2]xPD[)]/xE_3]/xSQ_1⌉/xE_4]xPD[)]/xE_5]xPD[)]/xE_6]/xSQ_0⟩◄≺

PROCDEF, 1, 109, 1:
➢⌊a⌋⌈≡(!{a}~(⌊A⌋⌈~(ε{Aa})⌉))⌉►xQ_0⌊xTVQ{a}/xQ_0⌋xSQ_0⌈xE_6[xBOP]≡⟨xPG[(]xE_1[xTCAC{!}x
CTX_1[xPG[{]xTV_[0]_0[D/a/xTV_0]xPD[}]/xCTX_1]/xE_1]xE_5[xMOP]~⟨xPG[(]xE_4[xQ_1⌊xT
VQ{A}/xQ_1⌋xSQ_1⌈xE_3[xMOP]~⟨xPG[(]xE_2[xBOP]ε⟨xPG[{]xTV_[1]_1[L/A/xTV_1]xTV_[0]_
2[L/a/xTV_2]xPD[}]/xE_2]xPD[)]/xE_3]/xSQ_1⌉/xE_4]xPD[)]/xE_5]xPD[)]/xE_6]/xSQ_0⟩◄≺

CATEGORIE, 1, 35, 1:
➢⌊a⌋⌈≡(!{a}~(⌊A⌋⌈~(ε{Aa})⌉))⌉►xQ_0⌊xTVQ{a}/xQ_0⌋xSQ_0⌈xE_6[xBOP]≡⟨xPG[(]xE_1[xTCAC{!}x
CTX_1[xPG[{]xTV_[0]_0[D/a/xTV_0]xPD[}]/xCTX_1]/xE_1]xE_5[xMOP]~⟨xPG[(]xE_4[xQ_1⌊xT
VQ{A}/xQ_1⌋xSQ_1⌈xE_3[xMOP]~⟨xPG[(]xE_2[xBOP]ε⟨xPG[{]xTV_[1]_1[L/A/xTV_1]xTV_[0]_
2[L/a/xTV_2]xPD[}]/xE_2]xPD[)]/xE_3]/xSQ_1⌉/xE_4]xPD[)]/xE_5]xPD[)]/xE_6]/xSQ_0⟩◄
/CAT=⟩[**1:cat**⟩**S/N**⟨**cst**⟩**!**⟨**ctx**⟩**{-}**⟨]⟩≺

Calculs propositionnels

Comme pour les autres exemples, nous fournissons ici la GO_i concernée, CALCULUS, et la GO_i qui permet de la tester depuis une *inscription* « brute ».

GO_i de tests pour CALCULUS

↳GRAM↲
☞GRAMNAME⇒TESTCALC☜

↳INCLUDE↲
☞PREANALYSE⇒"D:\TheseL\GramXAnalyse\GX01PreAnalyse.txt"☜
☞PROCDEF⇒"D:\TheseL\GramXAnalyse\GX02ProcDef.txt"☜
☞CATEGORIE⇒"D:\TheseL\GramXCategorie\GX03Categorie.txt"☜
☞CALCULUS⇒"D:\TheseL\GramXCalculus\GX04Calculus.txt"☜
↳ENDINCLUDE↲

↳PARAMS↲
☞MAXGRAMLOOP⇒2☜
☞MAXRULELOOP⇒5☜
↳ENDPARAMS↲

↳RULES↲
✎ entrer la formule depuis INPUT_CORPUS

☞⌊‹a›⇒➢⌊‹a›▶⌊‹a›◀≮☜🖉[1]
☞≡‹a›⇒➢≡‹a›▶≡‹a›◀≮☜🖉[2]

🖉 PREPARER ET EXECUTER L'ANALYSE DES TERMINAUX ET CONSTRUCTION DES
🖉 ENCAPSULATIONS DE TERMES D'EXPRESSIONS
☞↳EXECUTEGRAM(PREANALYSE)⏎☜🖉[3]

🖉 VERIFIER PROCEDURE DEFINITOIRE, TRIER LES VARIABLES DANS LES DIFFERENTES
🖉 EXPRESSIONS, ETABLIR LES RELATIONS LIEURS A VARIABLES LIEES
☞↳EXECUTEGRAM(PROCDEF)⏎☜🖉[4]

🖉 ANALYSER ET INSCRIRE LES CATEGORIES SYNTAXICO-SEMANTIQUES
☞↳EXECUTEGRAM(CATEGORIE)⏎☜🖉[5]

🖉 ANALYSER, CALCULER ET INSCRIRE LES TABLES DE VERITE
☞↳EXECUTEGRAM(CALCULUS)⏎☜🖉[6]

☞↳STOP⏎☜🖉[]
↳ENDRULES⏎
↳ENDGRAM⏎

GO_i CALCULUS

↳GRAM⏎
☞GRAMNAME⇒CALCULUS☜
🖉 Version 1 : valable pour les thèses, les axiomes et les formules logiques, ne calcule pas ε !
🖉 la formule à analyser se trouve encapsulée entre :
🖉 ▶‹formule›◀, elle doit être traitée comme : ➢‹avant›▶‹formule›◀‹apres›≮
🖉 on ajoute une zone de traitement : ➢‹avant›▶‹formule›◀‹apres›≮▷‹zonedetraitement›◁
🖉 les erreurs sont signalées, mais ne stoppent pas l'analyse !

↳PARAMS⏎
☞MAXGRAMLOOP⇒2☜
☞MAXRULELOOP⇒20☜
↳ENDPARAMS⏎

↳SETS⏎

🖉 Variables
☞vp⇒"p","q","r","s"☜🖉 variables propositionnelles
☞vns⇒"A","B","C","D"☜🖉 variables de nom singulier
☞vng⇒"a","b"☜🖉 variables de nom générique
☞vn⇒vns,vng☜🖉 les symboles de variable de nom
☞vp_vn⇒vp,vns,vng☜🖉 tous les symboles de variable

🖉 Fonctions Propositionnelles
☞fp2⇒"f","g"☜🖉 fonctions propositionnelles à deux places
☞fp1⇒"h"☜🖉 fonctions propositionnelles à une place
☞fp⇒fp2,fp1☜🖉 toutes les fonctions propositionnelles
☞vp_vn_fp⇒vp,vns,vng,fp☜🖉 tous les symboles de termes qui peuvent être quantifiés

🖉 Constantes, termes constants
☞bic⇒"≡"☜🖉 biconditionnelle
☞tc0⇒"V","F","T"☜🖉 connstantes sans contexte
☞tc1⇒"~","¬","α","!","♭","↓"☜🖉 constantes unaires
☞tc2⇒"μ","Ц","ε","∧","∨","⊃"☜🖉 constantes avec contexte binaire
☞tcn⇒"Δ","δ"☜🖉 constantes n_aires
☞tcmc⇒"τ","ω"☜🖉 constantes multi-contextes
☞tcac⇒tc1,tc2,tcn,tcmc☜🖉 les constantes avec contexte(s) sans BIC
☞mop⇒tc1,fp1☜🖉 opérateurs unaires
☞bop⇒tc2,bic,fp2☜🖉 opérateurs binaires
☞xop⇒tcn,tcmc☜🖉 opérateurs complexes, multi-contextes

🖉 Parenthèses gauches et délimiteurs gauches de contexte
☞pgA⇒"("☜
☞pgB⇒"["☜
☞pgC⇒"{"☜
☞pgD⇒"<"☜
☞pgE⇒"⟨"☜
🖉 ...
☞pg⇒pgA,pgB,pgC,pgD,pgE☜🖉 ...

✎ Parenthèses droites et délimiteurs droits de contexte
☞pdA⇒")"☜
☞pdB⇒"]"☜
☞pdC⇒"}"☜
☞pdD⇒">"☜
☞pdE⇒"⁾"☜
✎ ...
☞pd⇒pdA,pdB,pdC,pdD,pdE☜✎ ...

☞zero_sy⇒"0"☜✎ symbole 0
☞monopindefini⇒fp1☜✎ opérateurs binaires non définis
☞binopindefini⇒"ε",fp2☜✎ opérateurs binaires non définis
☞these_sy⇒"t"☜✎ symbole de marque de formule de type thèse
↳ENDSETS↲

↳RULES↲
✎ VERIFIER LA FORMULE EN ENTREE
✎ vérifier que la formule n'est pas vide
☞➢‹avant›►◄‹apres›≺⇒
ERREUR : ➢‹avant›►_erreur_fomule_vide◄‹apres›≺, trouvé formule à analyser vide !↳STOP↲☜✎[1]
✎ vérifier que la formule à analyser est bien encapsulée par : ➢...►‹formule›◄...≺
☞➢‹avant›►‹v›◄‹apres›≺⇒➢‹avant›►‹v›◄‹apres›≺/ok☜✎[2]
☞➢‹avant›►‹v›◄‹apres›≺⇒
ERREUR : ➢‹avant›►‹v›◄‹apres›≺, le format d'entré n'est pas correcte !↳STOP↲☜✎[3]
☞➢‹avant›►‹v›◄‹apres›≺/ok⇒➢‹avant›►‹v›◄‹apres›≺☜✎[4]
✎ copier la formule dans la zone de traitement
☞➢‹avant›►‹v›◄‹apres›≺⇒➢‹avant›►‹v›◄‹apres›≺▷‹v›◁☜✎[5]

✎ par définition, définition de F, du FAUX, aller à >>> INSCRIRE_RESULTAT <<<
☞➢‹avant›►‹formule›◄‹apres›≺
▷xE_‹I›❴xBOP❵≡❰xPG❴❵xTV_❴-1❵_‹J›❴C/F⁄xTV_‹j›❵xE_‹K›❴‹E›⁄xE_‹k›❵xPD❴❵❳⁄xE_‹i›❵◁⇒
➢‹avant›►‹formule›◄‹apres›≺
▷xE_‹I›❴0,0❵❴xBOP❵≡❰xPG❴❵xTV_❴-1❵_‹J›❴C/F⁄xTV_‹j›❵xE_‹K›❴‹E›⁄xE_‹k›❵xPD❴❵❳⁄xE_‹i›❵◁
↳GOTORULE(109)↲☜✎[6]

✎ par définition, définition de V, du VRAI, aller à >>> INSCRIRE_RESULTAT <<<
☞➢‹avant›►‹formule›◄‹apres›≺
▷xE_‹I›❴xBOP❵≡❰xPG❴❵xTV_❴-1❵_‹J›❴C/V⁄xTV_‹j›❵xE_‹K›❴‹E›⁄xE_‹k›❵xPD❴❵❳⁄xE_‹i›❵◁⇒
➢‹avant›►‹formule›◄‹apres›≺
▷xE_‹I›❴1,1❵❴xBOP❵≡❰xPG❴❵xTV_❴-1❵_‹J›❴C/V⁄xTV_‹j›❵xE_‹K›❴‹E›⁄xE_‹k›❵xPD❴❵❳⁄xE_‹i›❵◁
↳GOTORULE(109)↲☜✎[7]

✎ par définition, définition de T, du TAUTOLOGIE, aller à >>> INSCRIRE_RESULTAT <<<
☞➢‹avant›►‹formule›◄‹apres›≺
▷xE_‹I›❴xBOP❵≡❰xPG❴❵xTV_❴-1❵_‹J›❴C/T⁄xTV_‹j›❵xE_‹K›❴‹E›⁄xE_‹k›❵xPD❴❵❳⁄xE_‹i›❵◁⇒
➢‹avant›►‹formule›◄‹apres›≺
▷xE_‹I›❴1,1❵❴xBOP❵≡❰xPG❴❵xTV_❴-1❵_‹J›❴C/T⁄xTV_‹j›❵xE_‹K›❴‹E›⁄xE_‹k›❵sPD❴❵❳⁄xE_‹i›❵◁
↳GOTORULE(109)↲☜✎[8]

✎ ASSIGNER A CHAQUE VARIABLE PROPOSITIONNELLE UNE TABLE DE VERITE
✎ **********
✎ y compris pour l'ontologie : noms singuliers et noms généraux,
✎ même si on ne les calcule pas (sans connaître l'univers de référence)
✎ calculer la puissance des tables de vérité en fonction du nombre de tv_‹lien›_lie distincts dans
✎ - le Definiens dominant d'une Thèse
✎ - l'expression dominante d'une Formule Logique
✎ - le sous-quantificateur dominant d'un axiome

✎ inscrire temporairement une constante : la base 2
☞➢‹avant›►‹formule›◄‹apres›≺▷‹v›◁⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹v›◁/TMP❲tvListe=❴❵/base=❴2❵❳☜✎[9]

✎ SELECTION DU TYPE D'INSCRIPTION
✎ Sélectionner toute la formule (Axiomes, Thèses et Formules Logiques)
✎ et la marquer par défaut comme n'importe quel type (af)
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›◁/TMP❲‹u›❳⇒
➢‹avant›►‹formule›◄‹apres›≺▷●→‹a›←●◁/af/TMP❲‹u›❳☜✎[10]
✎ cerner le contenu de la généralisation dominante (Axiomes et Thèses)⌈⌊⌉⌋
☞➢‹avant›►‹formule›◄‹apres›≺▷●→xQ_‹i›⌊‹v›⁄xQ_‹i›⌋xSQ_‹i›⌈‹w›⁄xSQ_‹i›⌉←●◁/af/TMP❲‹u›❳⇒
➢‹avant›►‹formule›◄‹apres›≺▷●→xQ_‹i›⌊‹v›⁄xQ_‹i›⌋xSQ_‹i›⌈‹w›⁄xSQ_‹i›⌉←●◁
/a/TMP❲‹u›❳/mark1☜✎[11]

éjecter le quantificateur dominant (Axiomes et Thèses)
☞➢‹avant›▶‹formule›◀‹apres›≮▷‹a›●→xQ_‹i›↳INTEGER↵›⌊‹b›/xQ_‹i›⌋‹v›←●‹z›◁
/a/TMP⦃‹u›⦄/mark1⇒
➢‹avant›▶‹formule›◀‹apres›≮▷‹a›xQ_‹i›⌊‹b›/xQ_‹i›⌋●→‹v›←●‹z›◁/a/TMP⦃‹u›⦄/mark2☜✎[12]
sélectionner le contenu du sous-quantificateur dominant (Axiomes et Thèses)
☞➢‹avant›▶‹formule›◀‹apres›≮▷‹a›●→xSQ_‹i›↳INTEGER↵›⌈‹v›/xSQ_‹i›⌉←●‹z›◁
/a/TMP⦃‹u›⦄/mark2⇒
➢‹avant›▶‹formule›◀‹apres›≮▷‹a›xSQ_‹i›⌈●→‹v›←●/xSQ_‹i›⌉‹z›◁/a/TMP⦃‹u›⦄☜✎[13]
identifier Formule Logique ou Axiome/Thèse
☞➢‹avant›▶‹formule›◀‹apres›≮▷‹a›◁/af/TMP⦃‹u›⦄⇒
➢‹avant›▶‹formule›◀‹apres›≮▷‹a›◁/f/TMP⦃‹u›⦄☜✎[14]

CALCULER LES VALEURS DES TABLES POUR LES TV DU SOUS-QUANTIFICATEUR DOMINANT
sélectionner le contenu de la métaexpression du sous-quantificateur dominant ou
de la Formule Logique (Axiomes, Thèses et Formules Logiques)
☞➢‹avant›▶‹formule›◀‹apres›≮▷‹a›●→xE_‹i›↳INTEGER↵›⦃‹v›/xE_‹i›⦄←●‹z›◁/‹x›/TMP⦃‹u›⦄⇒
➢‹avant›▶‹formule›◀‹apres›≮▷‹a›xE_‹i›⦃●→‹v›←●/xE_‹i›⦄‹z›◁/‹x›/TMP⦃‹u›⦄☜✎[15]
identifier Thèse et si trouvé cerner le Definiens
☞➢‹avant›▶‹formule›◀‹apres›≮▷‹a›●→‹b›xE_‹i›⦃xTCAC⦃‹tc›↳IN(tcac)↵›⦄‹c›/xE_‹i›⦄‹v›←●‹z›◁
/a/TMP⦃‹u›⦄⇒➢‹avant›▶‹formule›◀‹apres›≮▷‹a›‹b›xE_‹i›⦃xTCAC⦃‹tc›⦄‹c›/xE_‹i›⦄●→‹v›←●‹z›◁
/t/TMP⦃‹u›⦄☜✎[16]

détecter les variables et les copier dans les paramètres temporaires
☞➢‹avant›▶‹formule›◀‹apres›≮▷‹a›●→‹b›xTV_⟦‹lien›↳INTEGER↵›⟧_‹i›↳INTEGER↵›⦃L/‹p›/xTV_‹i›⦄‹c›←●‹z›◁/‹te›/TMP⦃‹s›tvListe=⦃‹t›⦄/‹u›⦄⇒
➢‹avant›▶‹formule›◀‹apres›≮▷‹a›‹b›xTV_⟦‹lien›⟧_‹i›⦃L/‹p›/xTV_‹i›⦄●→‹c›←●‹z›◁
/‹te›/TMP⦃‹s›tvListe=⦃‹t›⦄‹p›⦃⦄/‹u›⦄☜✎[17]

supprimer les curseurs
☞➢‹avant›▶‹formule›◀‹apres›≮▷‹a›●→‹v›←●‹z›◁/‹te›/TMP⦃‹s›tvListe=⦃‹t›⦄/‹u›⦄⇒
➢‹avant›▶‹formule›◀‹apres›≮▷‹a›‹v›‹z›◁/‹te›/TMP⦃‹s›tvListe=⦃‹t›⦄/‹u›⦄/mark1☜✎[18]
supprimer les termes variables copiés et redondants dans les paramètres temporaires
☞➢‹avant›▶‹formule›◀‹apres›≮▷‹a›◁/‹te›/TMP⦃‹c›tvListe=⦃‹d›⦄‹p›⦃‹e›⦄‹p›⦃‹f›⦄/‹u›⦄/mark1⇒
➢‹avant›▶‹formule›◀‹apres›≮▷‹a›◁/‹te›/TMP⦃‹c›tvListe=⦃‹d›⦄‹p›⦃‹e›‹f›⦄/‹u›⦄/mark1☜✎[19]

compter le nombre de termes variables
☞➢‹avant›▶‹formule›◀‹apres›≮▷‹a›◁/‹te›/TMP⦃‹c›tvListe=⦃‹v›⦄/‹u›⦄/mark1⇒
➢‹avant›▶‹formule›◀‹apres›≮▷‹a›◁
/‹te›/TMP⦃‹c›tvListe=⦃‹v›⦄/nbVar=⦃↳COUNT(‹v›,╚⦃‹x›⦃╝)↵⦄/‹u›⦄/mark2☜✎[20]
calculer la puissance des tables de vérité en fonction du nombre de termes variables distincts
☞➢‹avant›▶‹formule›◀‹apres›≮▷‹a›◁/‹te›/TMP⦃‹w›/nbVar=⦃‹nbvar›⦄‹u›/base=⦃‹base›⦄⦄/mark2⇒
➢‹avant›▶‹formule›◀‹apres›≮▷‹a›◁
/‹te›/TMP⦃‹w›/nbVar=⦃‹nbvar›⦄/Puis=⦃↳POWER(‹base›,‹nbvar›)↵⦄‹u›/base=⦃‹base›⦄⦄/mark3☜✎[21]
initialiser le marquage de la combinatoire des termes variables,
soit la forme inversée des tables de vérité en fonction de la puissance de la combinatoire
et mémoriser les compteurs
☞➢‹avant›▶‹formule›◀‹apres›≮▷‹a›◁
/‹te›/TMP⦃‹c›/nbVar=⦃‹nbvar›⦄/Puis=⦃‹puis›⦄/base=⦃‹base›⦄⦄/mark3⇒
➢‹avant›▶‹formule›◀‹apres›≮▷‹a›◁
/‹te›/TMP⦃‹c›/TailleCouple=⦃↳DIV(‹puis›,2)↵⦄/NbOcc=⦃1⦄/TVnum=⦃‹nbvar›⦄/TV‹nbvar›=⦃?⦄
/nbVar=⦃‹nbvar›⦄/Puis=⦃‹puis›⦄/base=⦃‹base›⦄⦄/mark4☜✎[22]

>>> SELECT_TV <<<
sélectionner le terme variable courant dans les paramètres temporaires
☞➢‹avant›▶‹formule›◀‹apres›≮▷‹a›◁
/‹te›/TMP⦃‹c›tvListe=⦃‹d›⦃‹mavar›⦄‹e›⦄/TailleCouple=⦃‹couple›⦄/NbOcc=⦃‹occ›⦄/TVnum=⦃‹tvnum›⦄
/TV‹i›=⦃‹p›⦄/nbVar=⦃‹f›⦄/Puis=⦃‹puis›⦄/base=⦃‹g›⦄⦄/mark4⇒
➢‹avant›▶‹formule›◀‹apres›≮▷‹a›◁
/‹te›/TMP⦃‹c›tvListe=⦃‹d›⦃‹mavar›⦄‹e›⦄/TailleCouple=⦃‹couple›⦄/NbOcc=⦃‹occ›⦄/TVnum=⦃‹tvnum›⦄
/TV‹i›=⦃‹mavar›⦄/nbVar=⦃‹f›⦄/Puis=⦃‹puis›⦄/base=⦃‹g›⦄⦄/mark5☜✎[23]

>>> PROCHAIN_TV <<<
A partir du tv sélectionné dans les paramètres, chercher toutes les occurrences
du même tv dans la formule et lui affecter les paramètres.
cerner le xTV
☞➢‹avant›▶‹formule›◀‹apres›≮▷‹a›xTV_⟦‹lien›⟧_‹i›⦃L/‹v›/xTV_‹i›⦄‹z›◁/‹te›/‹param›/mark5⇒
➢‹avant›▶‹formule›◀‹apres›≮▷‹a›xTV_⟦‹lien›⟧_‹i›_fait⦃L/⇨‹v›⇦/xTV_‹i›⦄‹z›◁
/‹te›/‹param›/mark6☜✎[24]

variables équivalentes ?

☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›⇨‹mavar›⇦‹z›◁
/‹te›/TMP[‹d›tvListe={‹e›}/TailleCouple={‹couple›}/NbOcc={‹occ›}/TVnum={‹tvnum›}
/TV‹i›={‹mavar›}/nbVar={‹g›}/Puis={‹puis›}/base={‹h›}]/mark6⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›⇨‹mavar›⇦‹z›◁
/‹te›/TMP[‹d›tvListe={‹e›}/TailleCouple={‹couple›}/NbOcc={‹occ›}/TVnum={‹tvnum›}
/TV‹i›={‹mavar›}/nbVar={‹g›}/Puis={‹puis›}/base={‹h›}]/mark7☜✐[25]

✐ variable trouvée
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›xTV_[‹lien›]_‹i›_fait[L/⇨‹v›⇦/xTV_‹i›]‹b›◁
/‹te›/TMP[‹d›tvListe={‹e›}/TailleCouple={‹couple›}/NbOcc={‹occ›}/TVnum={‹tvnum›}
/TV‹icur›={‹mavar›}/nbVar={‹g›}/Puis={‹puis›}/base={‹h›}]/mark7⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›xTV_[‹lien›]_‹i›_[‹tvnum›_‹couple›_‹occ›][L/‹v›/xTV_‹i›]‹b›◁
/‹te›/TMP[‹d›tvListe={‹e›}/TailleCouple={‹couple›}/NbOcc={‹occ›}/TVnum={‹tvnum›}
/TV‹icur›={‹mavar›}/nbVar={‹g›}/Puis={‹puis›}/base={‹h›}]/mark5☜✐[26]

✐ variable pas trouvée
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›⇨‹v›⇦‹b›◁
/‹te›/TMP[‹d›tvListe={‹e›}/TailleCouple={‹couple›}/NbOcc={‹occ›}/TVnum={‹tvnum›}
/TV‹icur›={‹mavar›}/nbVar={‹g›}/Puis={‹puis›}/base={‹h›}]/mark6⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›‹v›‹b›◁
/‹te›/TMP[‹d›tvListe={‹e›}/TailleCouple={‹couple›}/NbOcc={‹occ›}/TVnum={‹tvnum›}
/TV‹icur›={‹mavar›}/nbVar={‹g›}/Puis={‹puis›}/base={‹h›}]/mark5☜✐[27]

✐ ALLER A >>> SELECT_TV <<<
☞↳IFMATCH_GOTORULE(23)⏎☜✐ tv suivant [28]

✐ supprimer le marqueur _fait
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›_fait[‹b›◁/‹te›/‹param›/mark5⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›[‹b›◁/‹te›/‹param›/mark5☜✐[29]

✐ préparer le tv suivant; diminuer, augmenter et re-initialiser les variables de mémorisation
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›◁
/‹te›/TMP[‹d›tvListe={‹e›}/TailleCouple={‹couple›}/NbOcc={‹occ›}
/TVnum={‹tvnum›}/TV‹i›={‹mavar›}/nbVar={‹g›}/Puis={‹puis›}/base={‹h›}]/mark5⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›◁
/‹te›/TMP[‹d›tvListe={‹e›}/TailleCouple={↳DIV(‹couple›,2)⏎}/NbOcc={↳MULT(‹occ›,2)⏎}
/TVnum={↳MINUS(‹tvnum›,1)⏎}/TV↳MINUS(‹tvnum›,1)⏎={‹mavar›}/nbVar={‹g›}
/Puis={‹puis›}/base={‹h›}]/mark4☜✐[30]

✐ ALLER A >>> SELECT_TV <<<
☞↳IFMATCH_GOTORULE(23)⏎☜✐ tv suivant [31]

✐ INSCRIRE TOUTES LES TABLES DE VERITE ASSOCIEES AUX TV
☞➢‹avant›►‹formule›◄‹apres›≺
▷‹a›xTV_[‹lien›]_‹i›_[‹tvnum›_‹couple›_‹occ›][L/‹mavar›/xTV_‹i›]‹b›◁/‹te›/‹y›/mark4⇒
➢‹avant›►‹formule›◄‹apres›≺
▷‹a›xTV_[‹lien›]_‹i›[↳TRUTHTABLE(‹couple›,‹occ›)⏎][L/‹mavar›/xTV_‹i›]‹b›◁
/‹te›/‹y›/mark4☜✐[32]

✐ supprimer les paramètres temporaires et la marque ; sauf la puissance nécessaire pour après
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›◁/‹te›/‹y›/Puis={‹p›}‹z›]/mark4⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›◁/‹te›/Puis={‹p›}/mark0☜✐[33]

✐ TRAITER LES TERMES CONSTITUES D'UNE CONSTANTE PROPOSITIONNELLE : F, V ET T
✐ **********

✐ >>> CONSTANTE_PROP <<<
✐ cerner une constante propositionnelle
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›xTV_[-1]_‹i›[C/F/xTV_‹i›]‹z›◁/‹te›/Puis={‹puis›}/mark0⇒
➢‹avant›►‹formule›◄‹apres›≺
▷‹a›●→xTV_[-1]_‹i›_fait[0][C/F/xTV_‹i›]←●‹z›◁/‹te›/Puis={‹puis›}{↳MINUS(‹puis›,1)⏎}
/marque1/[0]☜✐ [34]
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›xTV_[-1]_‹i›[C/V/xTV_‹i›]‹z›◁/‹te›/Puis={‹puis›}/mark0⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›●→xTV_[-1]_‹i›_fait[1][C/V/xTV_‹i›]←●‹z›◁
/‹te›/Puis={‹puis›}{↳MINUS(‹puis›,1)⏎}/marque1/[1]☜✐ [35]
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›xTV_[-1]_‹i›[C/T/xTV_‹i›]‹z›◁/‹te›/Puis={‹puis›}/mark0⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›●→xTV_[-1]_‹i›_fait[1][C/T/xTV_‹i›]←●‹z›◁
/‹te›/Puis={‹puis›}{↳MINUS(‹puis›,1)⏎}/marque1/[1]☜✐ [36]

✐ assigner la table
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›●→xTV_[-1]_‹i›_fait[‹v›][‹ct›/xTV_‹i›]‹b›←●‹z›◁

/‹te›/Puis=[‹puis›][‹p↳OUTOF(zero_sy)↵›]/marque1/[‹verit›]⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→xTV_[-1]_‹i›_fait[‹v›,‹verit›][‹ct›/xTV_‹i›]‹b›←●‹z›◁
/‹te›/Puis=[‹puis›][↳MINUS(‹p›,1)↵]/marque1/[‹verit›]☜✐ [37]

✐ supprimer les curseurs et re-initilialiser les paramètres
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→‹v›←●‹z›◁/‹te›/Puis=[‹puis›][‹pu›]/marque1/[‹n›]⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›‹v›‹z›◁/‹te›/Puis=[‹puis›]/mark0☜✐ [38]

✐ ALLER A >>> CONSTANTE_PROP <<<
☞↳IFMATCH_GOTORULE(34)↵☜✐ cp suivant [39]

✐ supprimer les marques
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›_fait‹z›◁/‹te›/Puis=[‹puis›]/mark0⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›‹z›◁/‹te›/Puis=[‹puis›]/mark0☜✐ [40]

✐ TRAITER LES GENERALISATIONS SOUMISES A LA DEFINITION :
✐ Dp8, le FAUX, soit : ≡(F⌊p⌋⌊p⌋) ou F≡(∀p)(p)

✐ >>> GENE_FAUX <<<
✐ cerner une généralisation et vérifier qu'il n'y ait qu'un tv dans le quantificateur
☞➢‹avant›▶‹formule›◀‹apres›≺
▷‹a›xQ_‹i›⌊xTVQ[‹p↳IN(vp_vn_fp)↵›]/xQ_‹i›⌋xSQ_‹i›⌈‹v›/xSQ_‹i›⌉‹z›◁/‹te›/Puis=[‹puis›]/mark0⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›➲xQ_mark_‹i›⌊xTVQ[‹p›]/xQ_‹i›⌋xSQ_‹i›⌈⇨‹v›⇦/xSQ_‹i›⌉⊂‹z›◁
/‹te›/Puis=[‹puis›]/[‹p›]/markA☜✐ [41]
✐ si quantificateur avec un seul terme trouvé, cerner l'expression du sous-quantificateur
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›⇨xE_‹i›[‹v›/xE_‹i›]⇦‹z›◁/‹te›/Puis=[‹puis›]/[‹Qp›]/markA⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›⇨xE_‹i›[◊→‹v›←◊/xE_‹i›]⇦‹z›◁
/‹te›/Puis=[‹puis›]/[‹Qp›]/markB☜✐ [42]
✐ cerner un terme unique possible xTV du sous-quantificateur
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◊→xTV_[‹lien↳INTEGER↵›]_‹i›[‹verit›][‹X›/‹p›/xTV_‹i›]←◊
‹z›◁/‹te›/Puis=[‹puis›]/[‹Qp›]/markB⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◊→xTV_[‹lien›]_‹i›[‹verit›][‹X›/‹p›/xTV_‹i›]←◊
‹z›◁/‹te›/Puis=[‹puis›]/[‹Qp›]/[‹p›]/markC☜✐ [43]
✐ si pas trouvé terme unique, re-initialiser en supprimant les curseurs,
✐ mais garder la marque sur l'encapsulation du quantificateur
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›➲‹b›⇨‹c›◊→‹d›←◊‹e›⇦‹f›⊂‹z›◁
/‹te›/Puis=[‹puis›]/[‹Qp›]/markB⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›‹b›‹c›‹d›‹e›‹f›‹z›◁/‹te›/Puis=[‹puis›]/mark0☜✐ [44]
✐ si trouvé terme unique, comparer les deux variables
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/‹te›/Puis=[‹puis›]/[‹p›]/[‹p›]/markC⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/‹te›/Puis=[‹puis›]/markD☜✐ [45]
✐ si trouvé terme unique xTV, supprimer la table de vérité du terme unique
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◊→xTV_[‹lien›]_‹i›[‹verit›][‹v›/xTV_‹i›]←◊‹z›◁
/‹te›/Puis=[‹puis›]/markD⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xTV_[‹lien›]_‹i›[‹v›/xTV_‹i›]‹z›◁
/‹te›/Puis=[‹puis›]/markD☜✐ [46]
✐ et si trouvé, marquer l'expression du sous-quantificateur comme calculée
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›⇨xE_‹i›[‹v›/xE_‹i›]⇦‹z›◁/‹te›/Puis=[‹puis›]/markD⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_calc_‹i›[‹v›/xE_‹i›]‹z›◁/‹te›/Puis=[‹puis›]/markD☜✐ [47]
✐ si pas équiforme, re-initialiser en supprimant les curseurs,
✐ mais garder la marque sur l'encapsulation du quantificateur
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›➲‹b›⇨‹c›◊→‹d›←◊‹e›⇦‹f›⊂‹z›◁
/‹te›/Puis=[‹puis›]/[‹Qp›]//[‹Ep›]/markC⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›‹b›‹c›‹d›‹e›‹f›‹z›◁/‹te›/Puis=[‹puis›]☜✐ [48]
✐ étendre à l'expression du quantificateur
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_‹i›[➲‹v›⊂/xE_‹i›]‹z›◁/‹te›/Puis=[‹puis›]/markD⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_calc_‹i›➲[0]⊂[‹v›/xE_‹i›]‹z›◁
/‹te›/Puis=[‹puis›][↳MINUS(‹puis›,1)↵]/markE☜✐ [49]

✐ construire la table du FAUX
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›➲[‹v›]⊂‹z›◁
/‹te›/Puis=[‹puis›][‹p↳OUTOF(zero_sy)↵›]/markE⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›➲[‹v›,0]⊂‹z›◁
/‹te›/Puis=[‹puis›][↳MINUS(‹p›,1)↵]/markE☜✐ [50]

✐ supprimer les curseurs
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›➲‹v›⊂‹z›◁/‹te›/Puis=[‹puis›][‹p›]/markE⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›‹v›‹z›◁/‹te›/Puis=[‹puis›]/markF☜✐ [51]
✐ modifier les marques de la généralisation
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›mark_xQ_‹z›◁/‹te›/Puis=[‹puis›]/markF⇒

➢‹avant›►‹formule›◄‹apres›≺▷‹a›calc_xQ_‹z›◁/‹te›/Puis=[‹puis›]☜🖉 [52]

🖉 ALLER A >>> GENE_FAUX <<<
☞↳IFMATCH_GOTORULE(41)⇙☜🖉 prochaine généralisation "FAUX" [53]

🖉 supprimer les marques de quantificateur
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›xQ_mark_‹z›◁/‹te›/Puis=[‹puis›]/mark‹m›⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›xQ_‹z›◁/‹te›/Puis=[‹puis›]/mark‹m›☜🖉 [54]
🖉 supprimer le paramètre de la puissance
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›◁/‹te›/Puis=[‹puis›]‹mark›⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›◁/‹te›/mark0☜🖉 [55]

🖉 TRAITER UN DEFINIENDUM
🖉 marquer l'expression
☞➢‹avant›►‹formule›◄‹apres›≺
▷‹a›xSQ_0⌈xE_‹i›↳INTEGER⇙›[xBOP]≡⦃xPG[(]xE_‹j›↳INTEGER⇙›[‹z›◁/t/mark0⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›xSQ_0⌈xE_‹i›[xBOP]≡⦃xPG[(]xE_DUM_‹j›[‹z›◁/t/mark0☜🖉 [56]

🖉 CALCULER LES META-EXPRESSIONS TT
🖉 **********

🖉 >>> TRAITER_TT <<<

🖉 cerner une expression de type : xTV - xTV
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›xE_‹i›[xBOP]‹op›⦃xPG[‹pg›↳IN(pg)⇙›]
xTV_[‹lienP›]_‹tp1›[‹tableP›][L/‹p›]xTV_[‹lienQ›]_‹tq1›[‹tableQ›][L/‹q›]
xPD[‹pd›↳IN(pd)⇙›]/xE_‹i›[‹z›◁/‹te›/mark0⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›●→xE_‹i›[xBOP]‹op›⦃xPG[‹pg›]
xTV_[‹lienP›]_‹tp1›[L/‹p›]xTV_[‹lienQ›]_‹tq1›[L/‹q›]
xPD[‹pd›]/xE_‹i›[←●‹z›◁/‹te›/[‹tableP›][‹tableQ›]/mark1/[‹op›]☜🖉[57]
🖉 tester si xBOP est défini
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›◁/‹te›/mark1/[‹bop›↳IN(binopindefini)⇙›]⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›◁/‹te›/mark0/[erreur : opérateur ‹bop› pas défini]☜🖉[58]
🖉 CALCULER LA META-EXPRESSION TT AVEC xBOP
☞➢‹avant›►‹formule›◄‹apres›≺
▷‹a›●→xE_‹i›[‹v›/xE_‹i›[←●‹z›◁/‹te›/[‹tableP›][‹tableQ›]/mark1/[‹op›]⇒
➢‹avant›►‹formule›◄‹apres›≺
▷‹a›xE_calc_‹i›[↳TRUTHCALC(‹op›,‹tableP›,‹tableQ›)⇙›][‹v›/xE_‹i›[‹z›◁/‹te›/mark0☜🖉[59]

🖉 ALLER A >>> TRAITER_TT <<<
☞↳IFMATCH_GOTORULE(57)⇙☜🖉 paire TT suivante [60]

🖉 CALCULER LA EXPRESSION CONSTITUEE D'UN OPERATEUR UNAIRE APPLIQUE A UN TERME
🖉 **********

🖉 >>> TRAITER_MONO_E(T) <<<

🖉 cerner une expression MOP et isoler xMOP
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›xE_‹i›[xMOP]‹mop›↳IN(mop)⇙›⦃‹v›/xE_‹i›[‹z›◁/‹te›/mark0⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›●→xE_mark_‹i›[xMOP_mark]‹mop›⦃◊→‹v›←◊/xE_‹i›[←●‹z›◁
/‹te›/mark1/[‹mop›]☜🖉[61]
🖉 vérifier si on a bien un terme (xTV) contenu entre les curseurs
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›◊→xPG[‹pg›↳IN(pg)⇙›]xTV_[‹lien›]_‹ti›[‹table›][‹p›/xTV_‹ti›]
xPD[‹pd›↳IN(pd)⇙›]←◊‹z›◁/‹te›/mark1/[‹mop›]⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›xPG[‹pg›]xTV_[‹lien›]_‹ti›[‹p›/xTV_‹ti›]xPD[‹pd›]‹z›◁
/‹te›/mark2/[‹mop›]/[‹table›]☜🖉[62]
🖉 tester si xMOP est défini
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›◁/‹te›/mark2/[‹mop›↳IN(monopindefini)⇙›]/[‹val›]⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›◁/‹te›/mark0/[erreur : opérateur ‹mop› pas défini]☜🖉[63]
🖉 calculer l'expression
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›●→xE_mark_‹i›↳INTEGER⇙›[‹v›/xE_‹i›]←●‹z›◁
/‹te›/mark2/[‹mop›]/[‹table›]⇒
➢‹avant›►‹formule›◄‹apres›≺
▷‹a›●→xE_mark_‹i›[↳TRUTHCALC(‹mop›,‹table›,"0,0,0,0")⇙›][‹v›/xE_‹i›]←●‹z›◁/‹te›/mark3☜🖉[64]
🖉 si trouvé, supprimer curseurs et marquer le monop
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›●→xE_mark_‹i›[‹table›][xMOP_mark]‹mop›⦃‹v›/xE_‹i›]←●
‹z›◁/‹te›/mark3⇒➢‹avant›►‹formule›◄‹apres›≺▷‹a›xE_calc_‹i›[‹table›][xMOP_calc]‹mop›⦃‹v›
/xE_‹i›[‹z›◁/‹te›/mark0☜🖉[65]
🖉 supprimer les curseurs si pas monop de terme
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›●→‹b›◊→‹c›←◊‹d›←●‹z›◁/‹te›/mark1/[‹mop›]⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›‹b›‹c›‹d›‹z›◁/‹te›/mark0☜🖉[66]

✎ ALLER A >>> TRAITER_MONO_E(T) <<<
☞↳IFMATCH_GOTORULE(61)⏎☞✎ mono E(T) suivant [67]
✎ supprimer tous les marqueurs de xMOP
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xMOP_mark‹z›◁/‹te›/mark0⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xMOP‹z›◁/‹te›/mark0☞✎[68]
✎ supprimer tous les marqueurs de xE
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_mark‹z›◁/‹te›/mark0⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE‹z›◁/‹te›/mark0☞✎[69]

✎ CALCULER LES EXPRESSIONS E
✎ **

✎ >>> TRAITER_EXPRESSION <<<

✎ CALCULER L'EXPRESSION CONSTITUEE D'UN OPERATEUR UNAIRE APPLIQUE A E(E)
✎ **
✎ cerner expression suivi de xMOP, suivi des délimiteurs, isoler xMOP
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_‹i›[xMOP]‹mop›↳IN(mop)⏎›⦃‹v›/xE_‹i›⦄‹z›◁/‹te›/mark0⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→xE_mark_‹i›[xMOP_mark]‹mop›⦃◊→‹v›←◊/xE_‹i›⦄←●‹z›◁
/‹te›/mark1/[‹mop›]☞✎[70]
✎ vérifier si l'on a bien une expression contenue entre les curseurs
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◊→xPG[‹pg›↳IN(pg)⏎›]xE_calc_‹k›[‹table›][‹b›/xE_‹k›]
xPD[‹pd›↳IN(pd)⏎›]←◊‹z›◁/‹te›/mark1/[‹mop›]⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xPG[‹pg›]◊→xE_calc_‹k›[‹table›][‹b›/xE_‹k›]←◊xPD[‹pd›]‹z›◁
/‹te›/mark2/[‹mop›]☞✎[71]
✎ si trouvé expression interne calculée
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◊→xE_calc_‹k›[‹table›][‹b›/xE_‹k›]←◊‹z›◁/‹te›/mark2/[‹mop›]
⇒➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_calc_‹k›[‹b›/xE_‹k›]‹z›◁/‹te›/mark3/[‹mop›]/[‹table›]☞✎[72]
✎ si pas trouvé expression interne calculée, supprimer curseurs
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→‹b›◊→‹c›←◊‹d›←●‹z›◁/‹te›/mark1/[‹mop›]⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›‹b›‹c›‹d›‹z›◁/‹te›/mark0☞✎[73]
✎ si trouvé expression interne calculée, tester si xMOP est défini
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/‹te›/mark3/[‹mop›↳IN(monopindefini)⏎›]/[‹table›]⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/‹te›/mark0/[erreur : opérateur ‹mop› pas défini]☞✎[74]
✎ calculer l'expression
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→xE_mark_‹i›↳INTEGER⏎›⦃‹v›/xE_‹i›⦄←●
‹z›◁/‹te›/mark3/[‹mop›]/[‹val›]⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_calc_‹i›[↳TRUTHCALC(‹mop›,‹val›,"0,0,0,0")⏎›][‹v›/xE_‹i›]
‹z›◁/‹te›/mark0☞✎[75]

✎ ALLER A >>> TRAITER_EXPRESSION <<<
☞↳IFMATCH_GOTORULE(70)⏎☞✎ meta-expression suivante [76]
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xMOP_mark‹z›◁/‹te›/mark0⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xMOP‹z›◁/‹te›/mark0☞✎[77]
✎ supprimer tous les marqueurs de xE
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_mark‹z›◁/‹te›/mark0⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE‹z›◁/‹te›/mark0☞✎[78]

✎ CALCULER L'EXPRESSION CONSTITUEE D'UN OPERATEUR BINAIRE
✎ **

✎ >>> TRAITER_PAIRE <<<

✎ cerner une métaexpression non calculée↳INTEGER⏎
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_‹i›[xBOP]‹op›⦃‹v›/xE_‹i›⦄‹z›◁/‹te›/mark0⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◊→xE_mark_‹i›[xBOP]‹op›⦃●→‹v›←●/xE_‹i›⦄←◊‹z›◁
/‹te›/[‹op›]/mark1_pp☞✎[79]
✎ serrer les curseurs entre les délimiteurs (terminé par une encapsulation d'expression)
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→xPG[‹pg›↳IN(pg)⏎›]x‹v›xPD[‹pd›↳IN(pd)⏎›]←●‹z›◁
/‹te›/[‹op›]/mark1_pp⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xPG[‹pg›]●→x‹v›←●xPD[‹pd›]‹z›◁/‹te›/[‹op›]/mark2_pp☞✎[80]
✎ serrer les curseurs entre les délimiteurs (terminé par une encapsulation de terme)
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→xPG[‹pg›↳IN(pg)⏎›]x‹v›xPD[‹pd›↳IN(pd)⏎›]←●‹z›◁
/‹te›/[‹op›]/mark1_pp⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xPG[‹pg›]●→x‹v›←●xPD[‹pd›]‹z›◁/‹te›/[‹op›]/mark2_pp☞✎[81]
✎ cerner <E><xTV>
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→xE_calc_‹i›[‹tableE›][‹E›/xE_‹i›]
xTV_[‹lien›]_‹ti›[‹tableT›][‹T›/xTV_‹ti›]←●‹z›◁/‹te›/[‹bop›]/mark2_pp⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_calc_‹i›[‹E›/xE_‹i›]
xTV_[‹lien›]_‹ti›[‹T›/xTV_‹ti›]‹z›◁/‹te›/[‹bop›]/[[‹tableE›][‹tableT›]]/mark3_pp☞✎[82]

✎ cerner <xTV><E>
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→xTV_[‹lien›]_‹ti›[‹tableT›][‹T›/xTV_‹ti›]
xE_calc_‹j›[‹tableE›][‹E›/xE_‹j›]←●‹z›◁/‹te›/[‹bop›]/mark2_pp⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xTV_[‹lien›]_‹ti›[‹T›/xTV_‹ti›]
xE_calc_‹j›[‹E›/xE_‹j›]‹z›◁/‹te›/[‹bop›]/[[‹tableT›][‹tableE›]]/mark3_pp☜✎[83]
✎ cerner <E><E>
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→xE_calc_‹i›[‹tableE1›][‹b›/xE_‹i›]
xE_calc_‹j›[‹tableE2›][‹c›/xE_‹j›]←●‹z›◁/‹te›/[‹bop›]/mark2_pp⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_calc_‹i›[‹b›/xE_‹i›]
xE_calc_‹j›[‹c›/xE_‹j›]‹z›◁/‹te›/[‹bop›]/[[‹tableE1›][‹tableE2›]]/mark3_pp☜✎[84]
✎ tester si xBOP est défini
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁
/‹te›/[‹bop↳IN(binopindefini)↲›]/[[‹table1›][‹table2›]]/mark3_pp⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/‹te›/mark0/[erreur : opérateur ‹bop› pas défini]☜✎[85]
✎ calculer la métaexpression avec BOP
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◊→xE_mark_‹i›[‹v›/xE_‹i›]←◊‹z›◁
/‹te›/[‹bop›]/[[‹table1›][‹table2›]]/mark3_pp⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_calc_‹i›[↳TRUTHCALC(‹bop›,‹table1›,‹table2›)↲][‹v›/xE_‹i›]
‹z›◁/‹te›/mark0☜✎[86]
✎ si pas trouvé une paire acceptable : supprimer les curseurs,
✎ mais garder la marque sur l'expression principale
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◊→‹b›●→‹c›←●‹d›←◊‹z›◁/‹te›/‹x›mark‹mark›_pp⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›‹b›‹c›‹d›‹z›◁/‹te›/mark0☜✎[87]

✎ TRAITER L'EXPRESSION PRINCIPALE D'UNE THESE
✎ si thèse reporter le résultat dans l'expression du sous-quantificateur dominant
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xSQ_0⌈
xE_mark_‹i›[xBOP]≡⌋xPG[(]xE_DUM_‹j›[‹v›/xE_‹j›]xE_calc_‹k›[‹table›][‹z›◁/t/mark0⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xSQ_0⌈
xE_calc_‹i›[‹table›][xBOP]≡⌋xPG[(]xE_calc_‹j›[‹v›/xE_‹j›]xE_calc_‹k›[‹z›◁/t/mark0☜✎[88]

✎ TRAITER LES EXPRESSIONS DE TYPE GENERALISATION
✎ cerner une sous-généralisation
✎ les généralisations soumises à la définition : Dp8, le FAUX,
✎ soit : ≡(F⌊p⌋⌊p⌋) ou F≡(∀p)(p) sont traitées avant
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_‹k›[xQ_‹i›⌊
‹v›/xQ_‹i›⌋xSQ_‹i›⌈‹w›/xSQ_‹i›⌉/xE_‹k›]‹z›◁/‹te›/mark0⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→xE_mark_‹k›[xQ_‹i›⌊
‹v›/xQ_‹i›⌋xSQ_‹i›⌈•→‹w›←•/xSQ_‹i›⌉/xE_‹k›]←●‹z›◁/‹te›/markA☜✎ [89]
✎ cerner l'expression du sous/sous-quantificateur de la sous-généralisation, extraire sa table
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›•→xE_calc_‹i›[‹table›][‹v›/xE_‹i›]←•‹z›◁/‹te›/markA⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_calc_‹i›[‹v›/xE_‹i›]‹z›◁/‹te›/markB/[‹table›]☜✎ [90]
✎ remonter la table au niveau de l'expression de la sous-généralisation et supprimer les marqueurs
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→xE_mark_‹k›[xQ_‹i›⌊‹v›/xQ_‹i›⌋xSQ_‹i›⌈
‹w›/xSQ_‹i›⌉/xE_‹k›]←●‹z›◁/‹te›/markB/[‹table›]⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_calc_‹k›[‹table›][xQ_‹i›⌊‹v›/xQ_‹i›⌋xSQ_‹i›⌈
‹w›/xSQ_‹i›⌉/xE_‹k›]‹z›◁/‹te›/mark0☜✎ [91]
✎ supprimer les curseurs si pas trouvé
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›●→‹b›•→‹c›←•‹d›←●‹z›◁/‹te›/markA⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›‹b›‹c›‹d›‹z›◁/‹te›/mark0☜✎ [92]

✎ ALLER A >>> TRAITER_PAIRE <<<
☞↳IFMATCH_GOTORULE(79)↲☜✎ expression suivante [93]

✎ initialiser les paramètres
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/‹te›/‹x›mark‹mark›_pp⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›◁/‹te›/mark0☜✎[94]
✎ supprimer les marques d'expression
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_mark_‹z›◁/‹te›/‹x›mark0⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_‹z›◁/‹te›/mark0☜✎[95]

✎ ALLER A >>> TRAITER_EXPRESSION <<<
☞↳IFMATCH_GOTORULE(70)↲☜✎ expression suivante [96]

✎ supprimer tous les marqueurs xE_calc_
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_calc_‹z›◁/‹te›/mark0⇒
➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xE_‹z›◁/‹te›/mark0☜✎[97]
✎ supprimer tous les marqueurs de monop
☞➢‹avant›▶‹formule›◀‹apres›≺▷‹a›xmop_calc‹z›◁/‹te›/mark0⇒

➢‹avant›►‹formule›◄‹apres›≺▷‹a›xMOP‹z›◁/‹te›/mark0☜🖉[98]
🖉 supprimer les marqueurs
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›◁/‹te›/mark0⇒➢‹avant›►‹formule›◄‹apres›≺▷‹a›◁☜🖉[99]

🖉 VERIFICATIONS
🖉 signale les erreurs sans stopper le processus
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›◁/‹te›/mark‹mark›⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›◁/‹te›/_erreur_mark‹mark›/
[erreur : marqueur parasite : mark‹mark›]☜🖉[100]
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›xmop_mark⦗‹mop›⦘‹z›◁⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›_erreur_xmop_mark⦗‹mop›⦘‹z›◁/
[erreur : marqueur parasite : xmop_mark]☜🖉[101]
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›mark_xG_‹i›⦗‹v›⁄mark_xG_‹i›⦘‹z›◁⇒
➢‹avant›►‹formule›◄‹apres›≺▷‹a›_erreur_mark_xG_‹i›⦗‹v›⁄xG_‹i›⦘‹z›◁/
[erreur : marqueur parasite : mark_xG]☜🖉[102]
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›◊‹z›◁⇒➢‹avant›►‹formule›◄‹apres›≺▷‹a›_erreur_◊‹z›◁/
[erreur : curseur parasite : ◊]☜🖉[103]
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›○‹z›◁≺⇒➢‹avant›►‹formule›◄‹apres›≺▷‹a›_erreur_○‹z›◁/
[erreur : curseur parasite : ○]☜🖉[104]
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›●‹z›◁⇒➢‹avant›►‹formule›◄‹apres›≺▷‹a›_erreur_●‹z›◁/
[erreur : curseur parasite : ●]☜🖉[105]
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›•‹z›◁⇒➢‹avant›►‹formule›◄‹apres›≺▷‹a›_erreur_•‹z›◁/
[erreur : curseur parasite : •]☜🖉[106]
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›→‹z›◁⇒➢‹avant›►‹formule›◄‹apres›≺▷‹a›_erreur_→‹z›◁/
[erreur : curseur parasite : →]☜🖉[107]
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›←‹z›◁⇒➢‹avant›►‹formule›◄‹apres›≺▷‹a›_erreur_←‹z›◁/
[erreur : curseur parasite : ←]☜🖉[108]

🖉 >>> INSCRIRE_RESULTAT <<<

🖉 inscrire le résultat
☞➢‹avant›►‹formule›◄‹apres›/TABLE=⦗‹x›⦘≺▷‹a›xE_‹i›↳INTEGER↲›❪‹t›❫‹b›◁⇒
➢‹avant›►‹formule›◄‹apres›/TABLE=⦗‹t›⦘≺☜🖉[109]
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›xE_‹i›↳INTEGER↲›❪‹t›❫‹b›◁⇒
➢‹avant›►‹formule›◄‹apres›/TABLE=⦗‹t›⦘≺☜🖉[110]

🖉 inscrire une erreur
☞➢‹avant›►‹formule›◄‹apres›≺▷‹a›◁‹x›/[erreur : ‹err›]⇒
➢‹avant›►‹formule›◄‹apres›/TABLE=⦗erreur : ‹err›⦘≺☜🖉[111]

☞↳STOP↲☜🖉[112]
↳ENDRULES↲
↳ENDGRAM↲

Voici un exemple d'exécution de la GO_i, CALCULUS, pour calculer la table du *definiens*, la *thèse-définition* de la conditionnelle :

cs : ⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉

Nous donnons ici quelques pas illustratifs des résultats de tout l'enchaînement du traitement de la re-grammaire qui teste CALCULUS (annexe *GOi de tests pour CALCULUS, p. 231*) :

TESTCALC, 1, 1,1: ➢⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉►⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉◄≺

PREANALYSE, 1, 78,1:
➢⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉►xQ_0⌊•xTVQ⦗p⦘•xTVQ⦗q⦘⁄xQ_0⌋xSQ_0⌈xE_5⦗xBOP⦘≡⦗xPG❪(❫xE_1⦗xTCAC⦗⊃⦘xCTX_1⦗xPG❪(❫•xTV_0⦗D/p⁄xTV_0⦘•xTV_0⦗D/q⁄xTV_0⦘xPD❪)❫⁄xCTX_1⦘⁄xE_1⦘xE_4⦗xMOP⦘~⦗xPG❪(❫xE_3⦗xBOP⦘∧⦗xPG❪(❫•xTV_1⦗L/p⁄xTV_1⦘xE_2⦗xMOP⦘~⦗xPG❪(❫•xTV_2⦗L/q⁄xTV_2⦘xPD❪)❫⁄xE_2⦘xPD❪)❫⁄xE_3⦘xPD❪)❫⁄xE_4⦘xPD❪)❫⁄xE_5⦘⁄xSQ_0⌉◄≺

A la fin de l'exécution de PROCDEF, voici le format en *langage pivot* de l'*inscription* :

PROCDEF, 1, 109,1:
➢⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉►xQ_0⌊xTVQ⦗p⦘xTVQ⦗q⦘⁄xQ_0⌋xSQ_0⌈xE_5⦗xBOP⦘≡⦗xPG❪(❫xE_1⦗xTCAC⦗⊃⦘xCTX_1⦗xPG❪(❫xTV_⦗0⦘_0⦗D/p⁄xTV_0⦘xTV_⦗0⦘_0⦗D/q⁄xTV_0⦘xPD❪)❫⁄xCTX_1⦘⁄xE_1⦘xE_4⦗xMOP⦘~⦗xPG❪(❫xE_3⦗xBOP⦘∧⦗xPG❪(❫xTV_⦗0⦘_1⦗L/p⁄xTV_1⦘xE_2⦗xMOP⦘~⦗xPG❪(❫xTV_⦗0⦘_2⦗L/q⁄xTV_2⦘xPD❪)❫⁄xE_2⦘xPD❪)❫⁄xE_3⦘xPD❪)❫⁄xE_4⦘xPD❪)❫⁄xE_5⦘⁄xSQ_0⌉◄≺

Après la détermination de la *catégorie syntaxico-sémantique* produite par la re-grammaire CATEGORIE, nous obtenons :

CATEGORIE, 1, 35,1:
➢⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉▶xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈xE_5[xBOP}≡⦑xPG[(]xE_1[xTCAC{⊃}xCTX_1[xPG[(]xTV_[0]_0[D/p/xTV_0]xTV_[0]_0[D/q/xTV_0]xPD[)]/xCTX_1]/xE_1]xE_4[xMOP}~⦑xPG[(]xE_3[xBOP}∧⦑xPG[(]xTV_[0]_1[L/p/xTV_1]xE_2[xMOP}~⦑xPG[(]xTV_[0]_2[L/q/xTV_2]xPD[)]/xE_2]xPD[)]/xE_3]xPD[)]/xE_4]xPD[)]/xE_5]/xSQ_0⌉◀/CAT=}[1:cat}S/SS⦑cst}⊃⦑ctx}(--)⦑]]}⦒

Arrêtons-nous aux étapes intéressantes de CALCULUS en prenant d'abord le résultat de l'affectation des tables de vérité aux *termes variables* :

CALCULUS, 1, 33,1:
➢⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉▶xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈xE_5[xBOP}≡⦑xPG[(]xE_1[xTCAC{⊃}xCTX_1[xPG[(]xTV_[0]_0[D/p/xTV_0]xTV_[0]_0[D/q/xTV_0]xPD[)]/xCTX_1]/xE_1]xE_4[xMOP}~⦑xPG[(]xE_3[xBOP}∧⦑xPG[(]xTV_[0]_1[L/p/xTV_1]xE_2[xMOP}~⦑xPG[(]xTV_[0]_2[L/q/xTV_2]xPD[)]/xE_2]xPD[)]/xE_3]xPD[)]/xE_4]xPD[)]/xE_5]/xSQ_0⌉◀/CAT=}[1:cat}S/SS⦑cst}⊃⦑ctx}(--)⦑]]}⦒
▷xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈xE_5[xBOP}≡⦑xPG[(]xE_1[xTCAC{⊃}xCTX_1[xPG[(]xTV_[0]_0[D/p/xTV_0]xTV_[0]_0[D/q/xTV_0]xPD[)]/xCTX_1]/xE_1]xE_4[xMOP}~⦑xPG[(]xE_3[xBOP}∧⦑xPG[(]**xTV_[0]_1[1,1,0,0]**[L/p/xTV_1]xE_2[xMOP}~⦑xPG[(]**xTV_[0]_2[1,0,1,0]**[L/q/xTV_2]xPD[)]/xE_2]xPD[)]/xE_3]xPD[)]/xE_4]xPD[)]/xE_5]/xSQ_0⌉◁/t/Puis=}[4]/mark0

Prenons ensuite un extrait du calcul de l'expression ~(q) :

CALCULUS, 1, 69,1:
➢⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉▶xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈xE_5[xBOP}≡⦑xPG[(]xE_1[xTCAC{⊃}xCTX_1[xPG[(]xTV_[0]_0[D/p/xTV_0]xTV_[0]_0[D/q/xTV_0]xPD[)]/xCTX_1]/xE_1]xE_4[xMOP}~⦑xPG[(]xE_3[xBOP}∧⦑xPG[(]xTV_[0]_1[L/p/xTV_1]xE_2[xMOP}~⦑xPG[(]xTV_[0]_2[L/q/xTV_2]xPD[)]/xE_2]xPD[)]/xE_3]xPD[)]/xE_4]xPD[)]/xE_5]/xSQ_0⌉◀/CAT=}[1:cat}S/SS⦑cst}⊃⦑ctx}(--)⦑]]}⦒
▷xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈xE_5[xBOP}≡⦑xPG[(]xE_DUM_1[xTCAC{⊃}xCTX_1[xPG[(]xTV_[0]_0[D/p/xTV_0]xTV_[0]_0[D/q/xTV_0]xPD[)]/xCTX_1]/xE_1]xE_4[xMOP}~⦑xPG[(]xE_3[xBOP}∧⦑xPG[(]xTV_[0]_1[1,1,0,0][L/p/xTV_1]**xE_calc_2[0,1,0,1]**[xMOP_calc}~⦑xPG[(]xTV_[0]_2[L/q/xTV_2]xPD[)]/xE_2]xPD[)]/xE_3]xPD[)]/xE_4]xPD[)]/xE_5]/xSQ_0⌉◁/t/mark0

Voici, maintenant, l'étape après le calcul de l'expression du *definiens* :

CALCULUS, 1, 99,1:
➢⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉▶xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈xE_5[xBOP}≡⦑xPG[(]xE_1[xTCAC{⊃}xCTX_1[xPG[(]xTV_[0]_0[D/p/xTV_0]xTV_[0]_0[D/q/xTV_0]xPD[)]/xCTX_1]/xE_1]xE_4[xMOP}~⦑xPG[(]xE_3[xBOP}∧⦑xPG[(]xTV_[0]_1[L/p/xTV_1]xE_2[xMOP}~⦑xPG[(]xTV_[0]_2[L/q/xTV_2]xPD[)]/xE_2]xPD[)]/xE_3]xPD[)]/xE_4]xPD[)]/xE_5]/xSQ_0⌉◀/CAT=}[1:cat}S/SS⦑cst}⊃⦑ctx}(--)⦑]]}⦒
▷xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈**xE_5[1,0,1,1]**[xBOP}≡⦑xPG[(]xE_1[xTCAC{⊃}xCTX_1[xPG[(]xTV_[0]_0[D/p/xTV_0]xTV_[0]_0[D/q/xTV_0]xPD[)]/xCTX_1]/xE_1]xE_4[xMOP}~⦑xPG[(]xE_3[xBOP}∧⦑xPG[(]xTV_[0]_1[L/p/xTV_1]xE_2[xMOP_calc}~⦑xPG[(]xTV_[0]_2[L/q/xTV_2]xPD[)]/xE_2]xPD[)]/xE_3]xPD[)]/xE_4]xPD[)]/xE_5]/xSQ_0⌉◁

Et, nous présentons, ici, la dernière étape où l'on trouve le résultat inscrit (en gras) :

CALCULUS, 1, 110,1:
➢⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉▶xQ_0⌊xTVQ{p}xTVQ{q}/xQ_0⌋xSQ_0⌈xE_5[xBOP}≡⦑xPG[(]xE_1[xTCAC{⊃}xCTX_1[xPG[(]xTV_[0]_0[D/p/xTV_0]xTV_[0]_0[D/q/xTV_0]xPD[)]/xCTX_1]/xE_1]xE_4[xMOP}~⦑xPG[(]xE_3[xBOP}∧⦑xPG[(]xTV_[0]_1[L/p/xTV_1]xE_2[xMOP}~⦑xPG[(]xTV_[0]_2[L/q/xTV_2]xPD[)]/xE_2]xPD[)]/xE_3]xPD[)]/xE_4]xPD[)]/xE_5]/xSQ_0⌉◀/CAT=}[1:cat}S/SS⦑cst}⊃⦑ctx}(--)⦑]]}⦒/**TABLE=}1,0,1,1}**⦒

Conversions d'inscriptions de la version contextuelle à la version catégorielle

Nous fournissons ici la version complète de la GO_i qui permet la conversion d'une *inscription* notée en version contextuelle et en format *langage pivot,* et d'être convertie en version catégorielle et de *forme analysée,* soit la GO_i, CONVERTXC. A celle-ci est associée

une grammaire spécifique au dépouillement de la version catégorielle analysée, DEPC. Afin de tester ces nouveaux modules, nous donnons une grammaire qui les enchaîne.

Enchaînement de GO_i pour tester la conversion de notations

```
↳GRAM↲
☞GRAMNAME⇒TESTCONV☜

↳INCLUDE↲
☞PREANALYSE⇒"D:\TheseL\GramXAnalyse\GX01PreAnalyse.txt"☜
☞PROCDEF⇒"D:\TheseL\GramXAnalyse\GX02ProcDef.txt"☜
☞CONVERTXC⇒"D:\TheseL\GramXConvert\GX06Convert.txt"☜
☞DEPC⇒"D:\TheseL\GramCDepouille\GC06Depouille.txt"☜
↳ENDINCLUDE↲

↳PARAMS↲
☞MAXGRAMLOOP⇒20☜
☞MAXRULELOOP⇒50☜
↳ENDPARAMS↲

↳RULES↲
🖉 entrer la formule depuis INPUT_CORPUS
☞⎣‹a›⇒➢⎣‹a›▶⎣‹a›◀≺☜🖉[1]
☞≡‹a›⇒➢≡‹a›▶≡‹a›◀≺☜🖉[2]

🖉 ANALYSE DES TERMINAUX ET CONSTRUCTION DES TERMES ET EXPRESSIONS
☞↳EXECUTEGRAM(PREANALYSE)↲☜🖉[3]

🖉 PROCEDURE DEFINITOIRE, TRIER LES TV, ETABLIR LES LIAISONS LIEURS VARAIABLES LIEES
☞↳EXECUTEGRAM(PROCDEF)↲☜🖉[4]

🖉 CONVERTIR VERS LA FORME CATEGORIELLE
☞↳EXECUTEGRAM(CONVERTXC)↲☜🖉[5]

🖉 DEPOUILLER LA FORME CATEGORIELLE
☞➢‹avant›▶‹v›◀‹apres›≺▷‹w›◁⇒➢‹avant›▶‹w›◀‹apres›≺☜🖉[6]
☞↳EXECUTEGRAM(DEPC)↲☜🖉[7]

☞↳STOP↲☜🖉[]
↳ENDRULES↲
↳ENDGRAM↲
```

GO_i de conversion de la notation contextuelle vers la version catégorielle

```
↳GRAM↲
☞GRAMNAME⇒CONVERTXC☜
🖉 convertir de la notation contextuelle vers la version catégorielle
🖉 la formule à analyser se trouve encapsulée entre : ▶‹formule›◀,
🖉 elle doit être traitée comme : ➢‹avant›▶‹formule›◀‹après›≺

↳PARAMS↲
☞MAXGRAMLOOP⇒10☜
☞MAXRULELOOP⇒50☜
↳ENDPARAMS↲

↳SETS↲
🖉 Fonctions Propositionnelles
☞fp2⇒"f","g"☜🖉 fonctions propositionnelles à deux places
☞fp1⇒"h"☜🖉 fonctions propositionnelles à une place
☞fp⇒fp2,fp1☜🖉 toutes les fonctions propositionnelles

🖉 Constantes, termes constants
☞bic⇒"≡"☜🖉 biconditionnelle
☞cp0⇒"V","F","T"☜🖉 constantes propositionnelles sans contexte
☞tc1⇒"~","¬","α","!","‣","↓"☜🖉 constantes unaires
☞tc2⇒"μ","Ц","ε","∧","∨","⊃"☜🖉 constantes avec contexte binaire
☞tcn⇒"Δ","δ"☜🖉 constantes n_aires
☞tcmc⇒"τ","ω"☜🖉 constantes multi-contextes
☞tcac⇒tc1,tc2,tcn,tcmc☜🖉 les constantes avec contexte(s) sans BIC
☞tcac_convert⇒"≡","ε","∧","∨","⊃"☜🖉 les constantes binaires converties dans les definienda
☞mop⇒tc1,fp1☜🖉 opérateurs unaires
```

☞bop⇒tc2,bic,fp2☜🖉 opérateurs binaires
☞xop⇒tcn,tcmc☜🖉 opérateurs complexes, multi-contextes

🖉 Parenthèses gauches et délimiteurs gauches de contexte
☞pgA⇒"("☜
☞pgB⇒"["☜
☞pgC⇒"{"☜
☞pgD⇒"<"☜
☞pgE⇒"\("☜
🖉 ...
☞pg⇒pgA,pgB,pgC,pgD,pgE☜🖉 ...

🖉 Parenthèses droites et délimiteurs droits de contexte
☞pdA⇒")"☜
☞pdB⇒"]"☜
☞pdC⇒"}"☜
☞pdD⇒">"☜
☞pdE⇒"/)"☜
🖉 ...
☞pd⇒pdA,pdB,pdC,pdD,pdE☜🖉 ...

☞ptype⇒"x","c"☜🖉 type de conversion
☞termetype⇒"L","C"☜🖉 type de terme sans les TV
↳ENDSETS🡻

↳RULES🡻
🖉 VERIFIER QUE LA FORMULE N'EST PAS VIDE
☞➢‹avant›►◄‹apres›≺⇒
ERREUR : ➢‹avant›►_erreur_fomule_vide◄‹apres›≺, trouvé formule à analyser vide !↳STOP🡻☜🖉[1]
🖉 VERIFIER QUE LA FORMULE A ANALYSER EST BIEN ENCAPSULEE PAR : ➢...►‹formule›◄...≺
☞➢‹avant›►‹v›◄‹apres›≺⇒➢‹avant›►‹v›◄‹apres›≺/ok☜🖉[2]
🖉 préparer la zone cible et de traitement
☞➢‹avant›►‹v›◄‹apres›≺/ok⇒➢‹avant›►‹v›◄‹apres›≺▷‹v›◁/ok☜🖉[3]

🖉 CONVERTIR TOUS LES xTV/L et xTV/C
🖉 **********
🖉 se donner une marque et un curseur
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›◁/ok⇒➢‹avant›►‹milieu›◄‹apres›≺▷➲‹a›◁/mark1_t☜🖉[4]
☞➢‹avant›►‹milieu›◄‹apres›≺▷➲‹b›xE_‹i›[xTCAC[‹tcac›]‹v›/xE_‹i›]‹z›◁/mark1_t⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹b›xE_‹i›[xTCAC[‹tcac›]‹v›/xE_‹i›]➲‹z›◁/mark1_t☜🖉[5]

🖉 >>> TRAITER_T <<<
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›➲‹b›xTV_[‹lien›]_‹i›↳INTEGER🡻›[‹type›↳IN(termetype)🡻›/‹z›◁
/mark1_t⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›‹b›➲•cTV_[‹lien›]_‹i›[‹type›/‹z›◁/[‹i›]/mark2_t☜🖉[6]
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›➲‹b›/xTV_‹i›]‹z›◁/[‹i›]/mark2_t⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›‹b›/cTV_‹i›]➲‹z›◁/mark1_t☜🖉[7]

🖉 ALLER A >>> TRAITER_T <<<
☞↳IFMATCH_GOTORULE(6)🡻☜🖉 T suivant [8]
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›➲‹z›◁‹x›/mark‹m›_t⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›‹z›◁/ok☜🖉[9]

🖉 TRAITER LES FORMES PROFONDES
🖉 **********
🖉 définir un marqueur
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›◁/ok⇒➢‹avant›►‹milieu›◄‹apres›≺▷‹a›◁/mark1_tt☜🖉[10]

🖉 >>> TRAITER_TT <<<

🖉 dépister une expression constituée d'un xBOP
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›xE_‹i›[xBOP‹possiblelien›[‹op›]‹v›/xE_‹i›]‹z›◁/mark1_tt⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›➲xE_mark_‹i›[xBOP[‹op›]◊→‹v›←◊/xE_‹i›]➲‹z›◁
/[xBOP[‹op›]/mark2_tt☜🖉[11]

🖉 restreindre la sélection de l'expression du contenu de l'expression au contenu des délimiteurs
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›◊→xPG[‹pg›↳IN(pg)🡻›]•c‹v›]xPD[‹pd›↳IN(pd)🡻›]←◊‹z›◁
/[xBOP[‹op›]/mark2_tt⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›xPG[‹pg›]◊→•c‹v›]←◊xPD[‹pd›]‹z›◁/[xBOP[‹op›]/mark3_tt☜🖉[1
2]
🖉 si pas trouvé, supprimer les curseurs et la mémorisation de l'opérateur binaire

☞➢‹avant›►‹milieu›◄‹apres›≮▷‹a›➲‹b›◊→‹c›←◊‹d›⊂‹z›◁/[xBOP‡‹op›⇃]/mark2_tt⇒
➢‹avant›►‹milieu›◄‹apres›≮▷‹a›‹b›‹c›‹d›‹z›◁/mark1_tt☜🖉[13]
🖉 cerner si l'expression est formée d'une paire TT et, si oui, la convertir de op(TT) en (T op T),
🖉 supprimer les curseurs
☞➢‹avant›►‹milieu›◄‹apres›≮▷‹a›xPG[‹pg↳IN(pg)↵›]
◊→•cTV_[‹lienA›]_‹i↳INTEGER↵›[‹p›/cTV_‹i›]•cTV_[‹lienB›]_‹j↳INTEGER↵›[‹q›/cTV_‹j›]←◊
xPD[‹pd↳IN(pd)↵›]‹z›◁/[xBOP‡‹op↳OUTOF(fp2)↵›⇃]/mark3_tt⇒
➢‹avant›►‹milieu›◄‹apres›≮▷‹a›cPG[‹pg›]
cTV_[‹lienA›]_‹i›[‹p›/cTV_‹i›]cBOP‡‹op›⇃cTV_[‹lienB›]_‹j›[‹q›/cTV_‹j›]cPD[‹pd›]‹z›◁
/mark4_tt☜🖉[14]
☞➢‹avant›►‹milieu›◄‹apres›≮▷‹a›xPG[‹pg↳IN(pg)↵›]
◊→•cTV_[‹lienA›]_‹i↳INTEGER↵›[‹p›/cTV_‹i›]•cTV_[‹lienB›]_‹j↳INTEGER↵›[‹q›/cTV_‹j›]←◊
xPD[‹pd↳IN(pd)↵›]‹z›◁/[xBOP‡‹op↳IN(fp2)↵›⇃]/mark3_tt⇒
➢‹avant›►‹milieu›◄‹apres›≮▷‹a›cBOP‡‹op›⇃cPG[‹pg›]
cTV_[‹lienA›]_‹i›[‹p›/cTV_‹i›]cTV_[‹lienB›]_‹j›[‹q›/cTV_‹j›]cPD[‹pd›]‹z›◁/mark4_tt☜🖉[15]

🖉 si pas trouvé, supprimer les curseurs
☞➢‹avant›►‹milieu›◄‹apres›≮▷‹a›➲‹b›◊→‹c›←◊‹d›⊂‹z›◁/[xBOP‡‹op›⇃]/mark3_tt⇒
➢‹avant›►‹milieu›◄‹apres›≮▷‹a›➲‹b›‹c›‹d›‹z›◁/mark1_tt☜🖉[16]
🖉 si paire TT trouvée, convertir l'expression
☞➢‹avant›►‹milieu›◄‹apres›≮▷‹a›➲xE_mark_‹i›[xBOP‡‹op›⇃‹v›/xE_‹i›]⊂‹z›◁/mark4_tt⇒
➢‹avant›►‹milieu›◄‹apres›≮▷‹a›•cE_‹i›[‹v›/cE_‹i›]‹z›◁/mark1_tt☜🖉[17]

🖉 ALLER A >>> TRAITER_TT <<<
☞↳IFMATCH_GOTORULE(11)↵☜🖉 paire TT suivante [18]
🖉 supprimer les marques de xE_mark
☞➢‹avant›►‹milieu›◄‹apres›≮▷‹a›xE_mark_‹z›◁/mark1_tt⇒
➢‹avant›►‹milieu›◄‹apres›≮▷‹a›xE_‹z›◁/mark1_tt☜🖉[19]
🖉 initialiser la marque
☞➢‹avant›►‹milieu›◄‹apres›≮▷‹a›◁/mark1_tt⇒➢‹avant›►‹milieu›◄‹apres›≮▷‹a›◁/ok☜🖉[20]

🖉 TRAITER L'EXPRESSION : DEFINIENDUM / TNAC / CONTEXTES

🖉 >>> CONTEXTE <<<

🖉 cerner le contexte suivant
☞➢‹avant›►‹milieu›◄‹apres›≮▷‹a›xCTX_‹i›[xPG[‹pg↳IN(pg)↵›]xTV_‹x›]
xPD[‹pd↳IN(pd)↵›]/xCTX_‹i›]‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≮▷‹a›•cCTX_‹i›[cPG[‹pg›]
○→xTV_‹x›]←○cPD[‹pd›]/cCTX_‹i›]‹z›◁/mark1_d☜🖉[21]
🖉 convertir les termes variables du contexte
☞➢‹avant›►‹milieu›◄‹apres›≮▷‹a›○→xTV_[‹lien›]_0[D/‹p›/xTV_0]‹c›←○‹z›◁/mark1_d⇒
➢‹avant›►‹milieu›◄‹apres›≮▷‹a›cTV_[‹lien›]_0[D/‹p›/cTV_0]○→‹c›←○‹z›◁/mark1_d☜🖉[22]
🖉 supprimer les curseurs et initialiser le marqueur
☞➢‹avant›►‹milieu›◄‹apres›≮▷‹a›○→‹v›←○‹z›◁/mark1_d⇒
➢‹avant›►‹milieu›◄‹apres›≮▷‹a›‹v›‹z›◁/ok☜🖉[23]

🖉 ALLER A >>> CONTEXTE <<<
☞↳IFMATCH_GOTORULE(21)↵☜🖉 contexte suivant [24]

🖉 CONVERTIR LE TCAC
☞➢‹avant›►‹milieu›◄‹apres›≮▷‹a›xTCAC{‹tn↳IN(tcac)↵›}‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≮▷‹a›cTCAC{‹tn›}‹z›◁/mark1☜🖉[25]
🖉 TRAITER UNE xE CONTENANT UN TCAC
☞➢‹avant›►‹milieu›◄‹apres›≮▷‹a›xE_‹i›[cTCAC{‹op↳IN(tcac)↵›}‹v›/xE_‹i›]‹z›◁/mark1⇒
➢‹avant›►‹milieu›◄‹apres›≮▷‹a›•cE_‹i›[cTCAC{‹op›}‹v›/cE_‹i›]‹z›◁/mark2☜🖉[26]
🖉 supprimer les marques de contexte
☞➢‹avant›►‹milieu›◄‹apres›≮▷‹a›•cCTX_‹z›◁/mark2⇒
➢‹avant›►‹milieu›◄‹apres›≮▷‹a›cCTX_‹z›◁/mark2☜🖉[27]
☞➢‹avant›►‹milieu›◄‹apres›≮▷‹a›◁/mark2⇒➢‹avant›►‹milieu›◄‹apres›≮▷‹a›◁/ok☜🖉[28]
🖉 CONVERTIR LES TCAC DE TYPE BINOP
🖉 sachant que la biconditionnelle, le terme primitif est traité par les axiomes 1 et 2
🖉 dépister un cTCAC binaire, cerner le contexte à deux arguments
☞➢‹avant›►‹milieu›◄‹apres›≮▷‹a›•cE_‹i›[cTCAC{‹op↳IN(tcac_convert)↵›}cCTX_1[cPG[‹pg›]cTV_
[‹lienP›]_0[D/‹p›/cTV_0]cTV_[‹lienQ›]_0[D/‹q›/cTV_0]cPD[‹pd›]/cCTX_1]‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≮▷‹a›•cE_‹i›[cPG[‹pg›]cTV_[‹lienP›]_0[D/‹p›/cTV_0]cBOP‡‹op›⇃cTV_[
‹lienQ›]_0[D/‹q›/cTV_0]cPD[‹pd›]⊂‹z›◁/[‹i›]/mark☜🖉[29]
☞➢‹avant›►‹milieu›◄‹apres›≮▷‹a›⊂/cE_‹i›]‹z›◁/[‹i›]/mark⇒
➢‹avant›►‹milieu›◄‹apres›≮▷‹a›/cE_‹i›]‹z›◁/ok☜🖉[30]

✎ >>> TRAITER_MONOP_T <<<
✎ cerner une xE potentiellement constituée d'un <monop><pg><xT><pd>
✎ différente d'une fonction propositionnelle
☞➢‹avant›►‹milieu›◄‹apres›◁▷‹a›xE_‹i›[xMOP]‹mop↳IN(mop)↵›⦃‹v›/xE_‹i›⦄‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›◁▷‹a›●→tmp_xE_‹i›[xMOP]‹mop›⦃○→‹v›←○/tmp_xE_‹i›⦄←●‹z›◁
/mark1_m☜✎[31]
✎ cerner une xE potentiellement constituée d'un <monop><pg><xT><pd>
✎ appliquée à une fonction propositionnelle
☞➢‹avant›►‹milieu›◄‹apres›◁▷‹a›xE_‹i›[xMOP_[‹lien›]]‹mop↳IN(mop)↵›⦃‹v›/xE_‹i›⦄‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›◁▷‹a›●→tmp_xE_‹i›[xMOP]‹mop›⦃○→‹v›←○/tmp_xE_‹i›⦄←●‹z›◁
/mark1_m☜✎[32]
✎ dépister les délimiteurs
☞➢‹avant›►‹milieu›◄‹apres›◁▷‹a›○→xPG[‹pg↳IN(pg)↵›]•cTV_‹v›⦄xPD[‹pd↳IN(pd)↵›]←○‹z›◁
/mark1_m⇒
➢‹avant›►‹milieu›◄‹apres›◁▷‹a›xPG[‹pg›]○→•cTV_‹v›⦄←○xPD[‹pd›]‹z›◁/mark2_m☜✎[33]
✎ la forme xTV est-elle présente ? si oui supprimer la marque de •cTV
☞➢‹avant›►‹milieu›◄‹apres›◁▷‹a›○→•cTV_[‹lien↳INTEGER↵›]_‹i↳INTEGER↵›[L/‹p›/cTV_‹i›⦄←○
‹z›◁/mark2_m⇒
➢‹avant›►‹milieu›◄‹apres›◁▷‹a›cTV_[‹lien›]_‹i›[L/‹p›/cTV_‹i›⦄‹z›◁/mark3_m☜✎[34]
☞➢‹avant›►‹milieu›◄‹apres›◁▷‹a›○→•cTV_[1]_‹i↳INTEGER↵›[C/‹p›/cTV_‹i›⦄←○‹z›◁/mark2_m⇒
➢‹avant›►‹milieu›◄‹apres›◁▷‹a›cTV_[-1]_‹i›[C/‹p›/cTV_‹i›⦄‹z›◁/mark3_m☜✎[35]
✎ ... et si oui, convertir tmp_xE en cE
☞➢‹avant›►‹milieu›◄‹apres›◁▷‹a›●→tmp_xE_‹i›[xMOP]‹mop↳IN(mop)↵›⦃
xPG[‹pg›]‹v›xPD[‹pd›]/tmp_xE_‹i›⦄←●‹z›◁/mark3_m⇒
➢‹avant›►‹milieu›◄‹apres›◁▷‹a›•cE_‹i›[cMOP]‹mop›⦃
cPG[‹pg›]‹v›cPD[‹pd›]/cE_‹i›⦄‹z›◁/ok☜✎[36]
✎ ... et si non, supprimer les curseurs ○→‹v›←○
☞➢‹avant›►‹milieu›◄‹apres›◁▷‹a›○→‹v›←○‹z›◁/mark‹zmark›_m⇒
➢‹avant›►‹milieu›◄‹apres›◁▷‹a›‹v›‹z›◁/mark‹zmark›_m☜✎[37]
✎ ... et si non, supprimer les curseurs ●→‹v›←● mais garder la marque tmp_
☞➢‹avant›►‹milieu›◄‹apres›◁▷‹a›●→‹v›←●‹z›◁/mark‹zmark›_m⇒
➢‹avant›►‹milieu›◄‹apres›◁▷‹a›‹v›‹z›◁/ok☜✎[38]

✎ ALLER A >>> TRAITER_MONOP_T <<<
☞↳IFMATCH_GOTORULE(31)↵☜✎ monop suivant [39]
✎ effacer les marqueurs des tentatives tmp_
☞➢‹avant›►‹milieu›◄‹apres›◁▷‹a›tmp_‹z›◁/ok⇒➢‹avant›►‹milieu›◄‹apres›◁▷‹a›‹z›◁/ok☜✎[40]

✎ TRAITER TOUS LES TERMES UNIQUES ENCAPSULES PAR UNE EXPRESSION
✎ cela permet de traiter en particulier les expressions principales d'un sous-quantificateur
✎ qui ne sont pas composées d'un calcul fait pas un binop
☞➢‹avant›►‹milieu›◄‹apres›◁▷‹a›xE_‹i›⦃•cTV_[‹lien›]_‹j›[L/‹p›/cTV_‹i›]/xE_‹i›⦄‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›◁▷‹a›•cE_‹i›[cPG[(]cTV_[‹lien›]_‹j›[L/‹p›/cTV_‹i›]cPD[)]]/cE_‹i›⦄‹z›◁
/ok☜✎[41]

✎ >>> TRAITER_EXPRESSION <<<
✎ >>> EXPRESSION_SUIVANTE <<<

✎ on cerne le début de la première métaexpression trouvée de type xE
☞➢‹avant›►‹milieu›◄‹apres›◁▷‹a›xE_‹i↳INTEGER↵›⦃‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›◁▷‹a›●→xE_afaire_[•‹i›•]⦃‹z›◁/mark☜✎[42]
✎ on cerne la fin de la première expression trouvée de type xE
☞➢‹avant›►‹milieu›◄‹apres›◁▷‹a›●→xE_afaire_[•‹i›•]⦃‹v›/xE_‹i›⦄‹z›◁/mark⇒
➢‹avant›►‹milieu›◄‹apres›◁▷‹a›●→xE_afaire_‹i›⦃○→‹v›←○/xE_afaire_‹i›⦄←●‹z›◁
/afaire1_exp☜✎[43]

✎ >>> DESCENDRE <<<

✎ Y a-t-il encore une forme de type xE dans xE_afaire ? si oui on descend dans la formule sélectionnée
☞➢‹avant›►‹milieu›◄‹apres›◁▷‹a›○→‹b›xE_‹i↳INTEGER↵›⦃‹v›/xE_‹i›⦄‹c›←○‹z›◁/afaire1_exp⇒
➢‹avant›►‹milieu›◄‹apres›◁▷‹a›‹b›○→xE_afaire_‹i›⦃◊→‹v›←◊/xE_afaire_‹i›⦄←○‹c›‹z›◁
/afaire2_exp☜✎[44]
✎ modifier en dessus xE_afaire en xE_bloque et descendre
☞➢‹avant›►‹milieu›◄‹apres›◁▷‹a›●→xE_afaire_‹i›⦃‹v›/xE_afaire_‹i›⦄←●‹z›◁/afaire2_exp⇒
➢‹avant›►‹milieu›◄‹apres›◁▷‹a›[xE_bloque_‹i›]⦃‹v›/[xE_bloque_‹i›]⦄‹z›◁/afaire3_exp☜✎[45]
✎ déplacer les curseurs
☞➢‹avant›►‹milieu›◄‹apres›◁▷‹a›○→‹v›←○‹z›◁/afaire3_exp⇒
➢‹avant›►‹milieu›◄‹apres›◁▷‹a›●→‹v›←●‹z›◁/afaire4_exp☜✎[46]

☞➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›●→‹b›◊→‹v›←◊‹c›←●‹z›◁/afaire4_exp⇒
➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›●→‹b›○→‹v›←○‹c›←●‹z›◁/afaire5_exp☜✐[47]

✐ continuer à descendre, aller à >>> DESCENDRE <<<
☞➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›◁/afaire5_exp⇒
➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›◁/afaire1_exp↳GOTORULE(44)☞✐[48]

✐ supprimer curseurs
☞➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›●→‹v›←●‹z›◁/afaire1_exp⇒
➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›‹v›‹z›◁/afaire1_exp☜✐[49]
☞➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›○→‹v›←○‹z›◁/afaire1_exp⇒
➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›‹v›‹z›◁/afaire1_exp☜✐[50]
☞➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›◊→‹v›←◊‹z›◁/afaire1_exp⇒
➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›‹v›‹z›◁/afaire1_exp☜✐[51]

✐ xE suivante, aller à >>> EXPRESSION_SUIVANTE <<<
☞➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›◁/afaire1_exp⇒
➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›◁/ok↳GOTORULE(42)☞✐[52]

✐ éjecter un xE_bloque seul sans xE_afaire !
☞➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›xE_‹z›◁/ok⇒
➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›xE_‹z›◁/test☜✐[53]
☞➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›xE_afaire_‹z›◁/test⇒
➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›xE_afaire_‹z›◁/ok☜✐[54]
✐ il ne reste plus qu'un xE
☞➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›[xE_bloque_‹i›]{‹v›/[xE_bloque_‹i›]}‹z›◁/test⇒
➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›xE_afaire_‹i›{‹v›/xE_afaire_‹i›}‹z›◁/ok☜✐[55]

✐ TRAITER LES COMBINAISONS DE META-EXPRESSIONS A FAIRE
✐ **********
✐ donc à faire : cTxE, xEcT, cTcE, cEcT, xExE, xEcE, cExE, cEcE, mopE, xE(xG)

✐ >>> A_FAIRE<<<

✐ cerner la (première) métaexpression à faire
☞➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›xE_afaire_‹i›↳INTEGER☞›{‹v›/xE_afaire_‹i›}‹z›◁/ok⇒
➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›•cE_‹i›{◊→‹v›←◊/cE_‹i›}‹z›◁/mark1☜✐[56]

✐ isoler un opérateur binaire
☞➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›◊→xBOP{‹opsym›↳IN(bop)☞›⦃‹v›←◊‹z›◁/mark1⇒
➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›◊→‹v›←◊‹z›◁/mark2/[BOP{‹opsym›⦄]☜✐[57]
☞➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›◊→xBOP_[‹lien›]{‹opsym›↳IN(fp2)☞›⦃‹v›←◊‹z›◁/mark1⇒
➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›◊→‹v›←◊‹z›◁/mark2/[BOP{‹opsym›⦄]☜✐[58]
✐ isoler un opérateur unaire différent d'une fonction propositionnelle
☞➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›◊→xMOP{‹opsym›↳IN(mop)☞›⦃‹v›←◊‹z›◁/mark1⇒
➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›cMOP{‹opsym›⦄◊→‹v›←◊‹z›◁/mark2/[MOP{‹opsym›⦄]☜✐[59]
✐ isoler un opérateur unaire opérant sur une fonction propositionnelle
☞➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›◊→xMOP_[‹lien›]{‹opsym›↳IN(fp1)☞›⦃‹v›←◊‹z›◁/mark1⇒
➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›cMOP{‹opsym›⦄◊→‹v›←◊‹z›◁/mark2/[MOP{‹opsym›⦄]☜✐[60]

✐ xE_afaire encapsulant une généralisation et supprimer les curseurs
☞➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›◊→xQ_‹i›↳INTEGER☞›⌊‹v›/xSQ_‹i›⌉←◊‹z›◁/mark1⇒
➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›xQ_‹i›⌊‹v›/xSQ_‹i›⌉‹z›◁/ok☜✐[61]

✐ cerner <pg>...<pd>
☞➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›◊→‹type›↳IN(ptype)☞›PG[‹pg›↳IN(pg)☞›]‹v›
xPD[‹pd›↳IN(pd)☞›]←◊‹z›◁/mark2/[‹op›]⇒
➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›cPG[‹pg›]◊→‹v›←◊cPD[‹pd›]‹z›◁/mark3/[‹op›]☜✐[62]
☞➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›◊→‹type›↳IN(ptype)☞›PG[‹pg›↳IN(pg)☞›]‹v›
cPD[‹pd›↳IN(pd)☞›]←◊‹z›◁/mark2/[‹op›]⇒
➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›cPG[‹pg›]◊→‹v›←◊cPD[‹pd›]‹z›◁/mark3/[‹op›]☜✐[63]

✐ cerner cTxE
☞➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›◊→•cTV_[‹lien›]_‹i›↳INTEGER☞›{‹p›/cTV_‹i›}
xE_‹j›↳INTEGER☞›{‹q›/xE_‹j›}←◊‹z›◁/mark3/[BOP{‹op›⦄]⇒
➢‹avant›▶‹milieu›◀‹apres›≮▷‹a›●→cTV_[‹lien›]_‹i›{‹p›/cTV_‹i›}←●
○→cE_‹j›{‹q›/cE_‹j›}←○‹z›◁/mark4/[BOP{‹op›⦄]☜✐[64]

✎ cerner cTcE
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›◊→•cTV_[‹lien›]_‹i›↳INTEGER↶›[‹p›/cTV_‹i›]
•cE_‹j›↳INTEGER↶›[‹q›/cE_‹j›]←◊‹z›◁/mark3/[BOP|‹op›]]⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›●→cTV_[‹lien›]_‹i›[‹p›/cTV_‹i›]←●
○→cE_‹j›[‹q›/cE_‹j›]←○‹z›◁/mark4/[BOP|‹op›]]☜✎[65]
✎ cerner xEcT
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›◊→xE_‹i›↳INTEGER↶›[‹p›/xE_‹i›]
•cTV_[‹lien›]_‹j›↳INTEGER↶›[‹q›/cTV_‹j›]←◊‹z›◁/mark3/[BOP|‹op›]]⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›●→cE_‹i›[‹p›/cE_‹i›]←●
○→cTV_[‹lien›]_‹j›[‹q›/cTV_‹j›]←○‹z›◁/mark4/[BOP|‹op›]]☜✎[66]
✎ cerner cEcT
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›◊→•cE_‹i›↳INTEGER↶›[‹p›/cE_‹i›]
•cTV_[‹lien›]_‹j›↳INTEGER↶›[‹q›/cTV_‹j›]←◊‹z›◁/mark3/[BOP|‹op›]]⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›●→cE_‹i›[‹p›/cE_‹i›]←●
○→cTV_[‹lien›]_‹j›[‹q›/cTV_‹j›]←○‹z›◁/mark4/[BOP|‹op›]]☜✎[67]
✎ cerner xExE
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›◊→xE_‹i›↳INTEGER↶›[‹p›/xE_‹i›]
xE_‹j›↳INTEGER↶›[‹q›/xE_‹j›]←◊‹z›◁/mark3/[BOP|‹op›]]⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›●→cE_‹i›[‹p›/cE_‹i›]←●
○→cE_‹j›[‹q›/cE_‹j›]←○‹z›◁/mark4/[BOP|‹op›]]☜✎[68]
✎ cerner cExE
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›◊→•cE_‹i›↳INTEGER↶›[‹p›/cE_‹i›]
xE_‹j›↳INTEGER↶›[‹q›/xE_‹j›]←◊‹z›◁/mark3/[BOP|‹op›]]⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›●→cE_‹i›[‹p›/cE_‹i›]←●
○→cE_‹j›[‹q›/cE_‹j›]←○‹z›◁/mark4/[BOP|‹op›]]☜✎[69]
✎ cerner xEcE
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›◊→xE_‹i›↳INTEGER↶›[‹p›/xE_‹i›]
•cE_‹j›↳INTEGER↶›[‹q›/cE_‹j›]←◊‹z›◁/mark3/[BOP|‹op›]]⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›●→cE_‹i›[‹p›/cE_‹i›]←●
○→cE_‹j›[‹q›/cE_‹j›]←○‹z›◁/mark4/[BOP|‹op›]]☜✎[70]
✎ cerner cEcE
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›◊→•cE_‹i›↳INTEGER↶›[‹p›/cE_‹i›]
•cE_‹j›↳INTEGER↶›[‹q›/cE_‹j›]←◊‹z›◁/mark3/[BOP|‹op›]]⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›●→cE_‹i›[‹p›/cE_‹i›]←●
○→cE_‹j›[‹q›/cE_‹j›]←○‹z›◁/mark4/[BOP|‹op›]]☜✎[71]
✎ cerner MOP(cE)
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›◊→•cE_‹i›↳INTEGER↶›[‹p›/cE_‹i›]←◊‹z›◁/mark3/[MOP|‹op›]]⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›◊→cE_‹i›[‹p›/cE_‹i›]←◊‹z›◁/mark4/[MOP|‹op›]]☜✎[72]
✎ cerner MOP(xE)
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›◊→xE_‹i›↳INTEGER↶›[‹p›/xE_‹i›]←◊‹z›◁/mark3/[MOP|‹op›]]⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›◊→cE_‹i›[‹p›/cE_‹i›]←◊‹z›◁/mark4/[MOP|‹op›]]☜✎[73]

✎ convertir l'expression binaire
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›●→‹v›←●○→‹w›←○‹z›◁/mark4/[BOP|‹op›↳OUTOF(fp2)↶›]]⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›‹v›cBOP|‹op›]‹w›‹z›◁/ok☜✎[74]
✎ convertir l'expression binaire de type fonction propositionnelle
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›●→‹v›←●○→‹w›←○‹z›◁/mark4/[BOP|‹op›↳IN(fp2)↶›]]⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›cBOP|‹op›]‹v›‹w›‹z›◁/ok☜✎[75]

✎ convertir l'expression unaire
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›◊→‹v›←◊‹z›◁/mark4/[MOP|‹op›]]⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›‹v›‹z›◁/ok☜✎[76]

✎ ALLER A >>> A_FAIRE <<<
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›xE_afaire_‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›xE_afaire_‹z›◁/ok↳GOTORULE(56)↶☜✎[77]

✎ débloquer xE_bloque
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›[xE_bloque_‹i›↳INTEGER↶›][‹v›/[xE_bloque_‹i›]][‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›xE_‹i›[‹v›/xE_‹i›][‹z›◁/ok☜✎[78]

✎ ALLER A >>> TRAITER_EXPRESSION <<<
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›xE_‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›xE_‹z›◁/ok↳GOTORULE(42)↶☜✎[79]

✎ convertir toutes les expressions de tous les sous-quantifcateurs
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›xSQ_‹i›⌈•cE_‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›xSQ_‹i›⌈cE_‹z›◁/ok☜✎[80]

✎ CONVERTIR LES QUANTIFICATEURS
✎ dépister et convertir tous les quantificateurs
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xQ_‹i›⌊‹v›⁄xQ_‹i›⌋‹z›◁/ok⇒
➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›cQ_‹i›⦃(∀‹v›)⁄cQ_‹i›⦄‹z›◁/ok☜✎[81]
✎ marquer et convertir tous les TVQ dans les tous quantificateurs
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›cQ_‹i›⦃(∀‹b›xTVQ⦃‹tv›⦄‹c›)⁄cQ_‹i›⦄‹z›◁/ok⇒
➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›cQ_‹i›⦃(∀‹b›cTVQ_mark⦃‹tv›⦄‹c›)⁄cQ_‹i›⦄‹z›◁/ok☜✎[82]
✎ démarquer tous les TVQ
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›cQ_‹i›⦃(∀‹b›cTVQ_mark⦃‹tv›⦄‹c›)⁄cQ_‹i›⦄‹z›◁/ok⇒
➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›cQ_‹i›⦃(∀‹b›cTVQ⦃‹tv›⦄‹c›)⁄cQ_‹i›⦄‹z›◁/ok☜✎[83]

✎ CONVERTIR LES SOUS-QUANTIFICATEURS
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xSQ_‹i›⌈‹v›⁄xSQ_‹i›⌉‹z›◁/ok⇒
➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›cSQ_‹i›⦃‹v›⁄cSQ_‹i›⦄‹z›◁/ok☜✎[84]

✎ convertir / valider l'expression d'une formule logique, supprimer les délimiteurs de l'expression
☞➢‹avant›▶‹milieu›◀‹apres›≺▷•cE_‹i›⁅cPG❨❩‹v›cPD❨❩⁆⁄cE_‹i›⁅◁/ok⇒
➢‹avant›▶‹milieu›◀‹apres›≺▷cE_‹i›⁅‹v›⁄cE_‹i›⁅◁/ok☜✎[85]

✎ VERIFICATION
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›◁/mark‹x›⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›◁/mark‹x›_erreur_/mark‹x›,
marque pas traitée !↳STOP↲☜✎[86]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›●‹z›◁‹x›⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_●‹z›◁‹x›,
trouvé un curseur parasite : ● !↳STOP↲☜✎[87]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›○‹z›◁‹x›⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_○‹z›◁‹x›,
trouvé un curseur parasite : ○ !↳STOP↲☜✎[88]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›→‹z›◁‹x›⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_→‹z›◁‹x›,
trouvé un curseur parasite : → !↳STOP↲☜✎[89]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›←‹z›◁‹x›⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_←‹z›◁‹x›,
trouvé un curseur parasite : ← !↳STOP↲☜✎[90]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_mark‹z›◁‹x›⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_mark‹z›◁‹x›,
trouvé un marqueur parasite !↳STOP↲☜✎[91]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›mark‹z›◁‹x›⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_mark‹z›◁‹x›,
marqueur pas traité !↳STOP↲☜✎[92]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›•‹z›◁‹x›⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_•‹z›◁‹x›,
trouvé un marqueur parasite : •↳STOP↲☜✎[93]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xT_‹z›◁‹x›⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_xE_‹z›◁‹x›,
xT pas traité !↳STOP↲☜✎[94]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xE_‹z›◁‹x›⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_xE_‹z›◁‹x›,
xE pas traité !↳STOP↲☜✎[95]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xQ_‹z›◁‹x›⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_xQ_‹z›◁‹x›,
xQ pas traité !↳STOP↲☜✎[96]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xSQ_‹z›◁‹x›⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_xSQ_‹z›◁‹x›,
xSQ pas traité !↳STOP↲☜✎[97]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›xC‹z›◁‹x›⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_xC‹z›◁‹x›,
xCTX pas traité !↳STOP↲☜✎[98]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›⌈‹z›◁‹x›⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_⌈‹z›◁‹x›,
symbole pas converti : ⌈↳STOP↲☜✎[99]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›⌉‹z›◁‹x›⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_⌉‹z›◁‹x›,
symbole pas converti : ⌉↳STOP↲☜✎[100]

☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›⌊‹z›◁‹x›⇒
ERREUR : ➢‹avant›►‹milieu›◄‹apres›≺▷‹a›_erreur_⌊‹z›◁‹x›,
symbole pas converti : ⌊↳STOP↲☜✐[101]
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›⌋‹z›◁‹x›⇒
ERREUR : ➢‹avant›►‹milieu›◄‹apres›≺▷‹a›_erreur_⌋‹z›◁‹x›,
symbole pas converti : ⌋↳STOP↲☜✐[102]

☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›◁/ok⇒➢‹avant›►‹milieu›◄‹apres›≺▷‹a›◁☜✐[103]

☞↳STOP↲☜✐[104]
↳ENDRULES↲
↳ENDGRAM↲

Dépouillement des inscriptions notées en version catégorielle

↳GRAM↲
☞GRAMNAME⇒DEPC☜
✐ la formule à analyser se trouve encapsulée entre : ►‹formule›◄,
✐ elle doit être traitée comme : ➢‹avant›►‹formule›◄‹après›≺
✐ Dépouiller une inscription analysée et encapsulée de ses encapsulations en version catégorielle

↳PARAMS↲
☞MAXGRAMLOOP⇒20☜
☞MAXRULELOOP⇒60☜
↳ENDPARAMS↲

↳SETS↲
✐ Variables
☞vp⇒"p","q","r","s"☜✐ variables propositionnelles
☞vns⇒"A","B","C","D"☜✐ variables de nom singulier
☞vng⇒"a","b"☜✐ variables de nom générique
☞vn⇒vns,vng☜✐ les symboles de variable de nom
☞vp_vn⇒vp,vns,vng☜✐ tous les symboles de variable

✐ Fonctions Propositionnelles
☞fp2⇒"f","g"☜✐ fonctions propositionnelles à deux places
☞fp1⇒"h"☜✐ fonctions propositionnelles à une place
☞fp⇒fp2,fp1☜✐ toutes les fonctions propositionnelles

☞vp_vn_fp⇒vp,vns,vng,fp☜✐ tous les symboles de termes qui peuvent être quantifiés

✐ Constantes, termes constants
☞bic⇒"≡"☜✐ biconditionnelle
☞cp0⇒"V","F","T"☜✐ constantes propositionnelles sans contexte
☞tc1⇒"~","¬","α","!","♭","↓"☜✐ constantes unaires
☞tc2⇒"μ","Ц","ε","∧","∨","⊃"☜✐ constantes avec contexte binaire
☞tcn⇒"Δ","δ"☜✐ constantes n_aires
☞tcmc⇒"τ","ω"☜✐ constantes multi-contextes
☞tcac⇒tc1,tc2,tcn,tcmc☜✐ les constantes avec contexte(s) sans BIC
☞mop⇒tc1,fp1☜✐ opérateurs unaires
☞bop⇒tc2,bic,fp2☜✐ opérateurs binaires
☞xop⇒tcn,tcmc☜✐ opérateurs complexes, multi-contextes

✐ Parenthèses gauches et délimiteurs gauches de contexte
☞pgA⇒"("☜
☞pgB⇒"["☜
☞pgC⇒"{"☜
☞pgD⇒"<"☜
☞pgE⇒"⎛"☜
✐ ...
☞pg⇒pgA,pgB,pgC,pgD,pgE☜✐ ...

✐ Parenthèses droites et délimiteurs droits de contexte
☞pdA⇒")"☜
☞pdB⇒"]"☜
☞pdC⇒"}"☜
☞pdD⇒">"☜
☞pdE⇒"⎞"☜
✐ ...
☞pd⇒pdA,pdB,pdC,pdD,pdE☜✐ ...
↳ENDSETS↲

↳RULES↲
🖉 VERIFIER QUE LA FORMULE N'EST PAS VIDE
☞➢‹avant›►◄‹apres›≺⇒
ERREUR : ➢‹avant›►_erreur_fomule_vide◄‹apres›≺, trouvé formule à analyser vide !↳STOP↲☜🖉[1]
🖉 VERIFIER QUE LA FORMULE A ANALYSER EST BIEN ENCAPSULEE PAR : ➢...►‹formule›◄...≺
☞➢‹avant›►‹v›◄‹apres›≺⇒➢‹avant›►‹v›◄‹apres›≺/ok☜🖉[2]
🖉 préparer la zone cible et de traitement
☞➢‹avant›►‹v›◄‹apres›≺/ok⇒➢‹avant›►‹v›◄‹apres›≺▷‹v›◁/ok☜🖉[3]

🖉 on va de l'extérieur vers l'intérieur des encapsulations

🖉 supprimer les encapsulations des généralisations
☞INUTILE⇒NERIENFAIRE☜🖉[4]
🖉 supprimer les encapsulations des quantificateurs
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›cQ_‹i›↳INTEGER↲›[‹v›/cQ_‹i›]‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[5]
🖉 supprimer les encapsulations des sous-quantificateurs
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›cSQ_‹i›↳INTEGER↲›[‹v›/cSQ_‹i›]‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[6]
🖉 supprimer les encapsulations des métaexpressions
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›cE_‹i›↳INTEGER↲›[‹v›/cE_‹i›]‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[7]
🖉 supprimer les encapsulations des délimiteurs gauches
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›cPG[‹v›↳IN(pg)↲›]‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[8]
🖉 supprimer les encapsulations des délimiteurs droits
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›cPD[‹v›↳IN(pd)↲›]‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[9]

🖉 >>> CTVL <<<

🖉 supprimer les encapsulations des métatermes cTV / L➲
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›cTV_[‹lien›↳INTEGER↲›]_‹i›↳INTEGER↲›[L/‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›➲‹z›◁/[‹i›]/mark☜🖉[10]
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›➲‹v›/cTV_‹i›]‹z›◁/[‹i›]/mark⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[11]

🖉 ALLER A >>> CTVL <<<
☞↳IFMATCH_GOTORULE(10)↲☜🖉 cTV/L suivant [12]

🖉 supprimer les encapsulations des métatermes cTV / C
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›cTV_[-1]_‹i›↳INTEGER↲›[C/‹v›/cTV_‹i›]‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[13]
🖉 supprimer les encapsulations des métatermes cTV / D
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›cTV_[‹lien›↳INTEGER↲›]_0[D/‹v›/cTV_0]‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[14]
🖉 supprimer les encapsulations des contextes
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›cCTX_‹i›↳INTEGER↲›[‹v›/cCTX_‹i›]‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[15]
🖉 supprimer les encapsulations des bop
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›cBOP}‹v›↳IN(bop)↲›}‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[16]
🖉 supprimer les encapsulations des mop
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›cMOP}‹v›↳IN(mop)↲›}‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[17]
🖉 supprimer les encapsulations des tcac
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›cTCAC{‹v›↳IN(tcac)↲›}‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[18]
🖉 dépouiller les délimiteurs des tv (termes variables) dans la quantificateurs
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›cTVQ{‹v›↳IN(vp_vn_fp)↲›}‹z›◁/ok⇒
➢‹avant›►‹milieu›◄‹apres›≺▷‹a›‹v›‹z›◁/ok☜🖉[19]

🖉 vérification
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›cT‹z›◁/ok⇒
ERREUR : ➢‹avant›►‹milieu›◄‹apres›≺▷‹a›_erreur_cT‹z›◁/ok,
trouvé encore encapsulation : cT !↳STOP↲☜🖉[20]
☞➢‹avant›►‹milieu›◄‹apres›≺▷‹a›cE‹z›◁/ok⇒
ERREUR : ➢‹avant›►‹milieu›◄‹apres›≺▷‹a›_erreur_cE‹z›◁/ok,
trouvé encore encapsulation : cE !↳STOP↲☜🖉[21]

☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›cP‹z›◁/ok⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_cP‹z›◁/ok,
trouvé encore encapsulation : cP !↳STOP↲☜✐[22]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›cQ‹z›◁/ok⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_cQ‹z›◁/ok,
trouvé encore encapsulation : cQ !↳STOP↲☜✐[23]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›cS‹z›◁/ok⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_cS‹z›◁/ok,
trouvé encore encapsulation : cQ !↳STOP↲☜✐[24]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›cC‹z›◁/ok⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_cC‹z›◁/ok,
trouvé encore encapsulation : cC !↳STOP↲☜✐[25]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›cM‹z›◁/ok⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_cM‹z›◁/ok,
trouvé encore encapsulation : cM !↳STOP↲☜✐[26]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›cB‹z›◁/ok⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_cB‹z›◁/ok,
trouvé encore encapsulation : cB !↳STOP↲☜✐[27]
☞➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›cB‹z›◁/ok⇒
ERREUR : ➢‹avant›▶‹milieu›◀‹apres›≺▷‹a›_erreur_cB‹z›◁/ok,
trouvé encore encapsulation : cB !↳STOP↲☜✐[27]

✐ supprimer la marque
☞➢‹x›≺▷‹y›◁/ok⇒➢‹x›≺▷‹y›◁☜✐[28]

☞↳STOP↲☜✐[29]
↳ENDRULES↲
↳ENDGRAM↲

Traitement de la règle d'inférences de détachement

Le traitement de la *règle d'inférences de détachement* se fait à l'aide des deux GO_i : DETACH et COMPARE :

GO_i de la directive de détachement

↳GRAM↲
☞GRAMNAME⇒DETACH☜
✐ Essayer d'appliquer un « détachement »

↳INCLUDE↲
☞COMPARE⇒"D:\TheseL\GramXDetach\GXDABCompare.txt"☜
↳ENDINCLUDE↲

↳PARAMS↲
☞MAXGRAMLOOP⇒5☜
☞MAXRULELOOP⇒30☜
↳ENDPARAMS↲

↳LIBRARY↲

☞┖➢§Ap1⧙⌊qpr⌋⌈≡(≡(≡(pr)≡(qp))≡(rq))⌉▶xQ_0⌊xTVQ{p}xTVQ{q}xTVQ{r}⁄xQ_0⌋xSQ_0⌈xE_5⁅xBOP⁆≡⧘xPG⁅(⁆xE_4⁅xBOP⁆≡⧘xPG⁅(⁆xE_1⁅xBOP⁆≡⧘xPG⁅(⁆xTV_⌊0⌋_1⁅L/p⁄xTV_1⁆xTV_⌊0⌋_2⁅L/r⁄xTV_2⁆xPD⁅)⁆⁄xE_1⁆xE_2⁅xBOP⁆≡⧘xPG⁅(⁆xTV_⌊0⌋_3⁅L/q⁄xTV_3⁆xTV_⌊0⌋_4⁅L/p⁄xTV_4⁆xPD⁅)⁆⁄xE_2⁆xPD⁅)⁆⁄xE_4⁆xE_3⁅xBOP⁆≡⧘xPG⁅(⁆xTV_⌊0⌋_5⁅L/r⁄xTV_5⁆xTV_⌊0⌋_6⁅L/q⁄xTV_6⁆xPD⁅)⁆⁄xE_3⁆xPD⁅)⁆⁄xE_5⁆⁄xSQ_0⌉◀≺⇊⇇x⇊☜

☞┖➢§Ap2⧙⌊qpr⌋⌈≡(≡(p≡(qr))≡(≡(pq)r))⌉▶xQ_0⌊xTVQ{p}xTVQ{q}xTVQ{r}⁄xQ_0⌋xSQ_0⌈xE_5⁅xBOP⁆≡⧘xPG⁅(⁆xE_3⁅xBOP⁆≡⧘xPG⁅(⁆xTV_⌊0⌋_1⁅L/p⁄xTV_1⁆xE_1⁅xBOP⁆≡⧘xPG⁅(⁆xTV_⌊0⌋_2⁅L/q⁄xTV_2⁆xTV_⌊0⌋_3⁅L/r⁄xTV_3⁆xPD⁅)⁆⁄xE_1⁆xPD⁅)⁆⁄xE_3⁆xE_4⁅xBOP⁆≡⧘xPG⁅(⁆xE_2⁅xBOP⁆≡⧘xPG⁅(⁆xTV_⌊0⌋_4⁅L/p⁄xTV_4⁆xTV_⌊0⌋_5⁅L/q⁄xTV_5⁆xPD⁅)⁆⁄xE_2⁆xTV_⌊0⌋_6⁅L/r⁄xTV_6⁆xPD⁅)⁆⁄xE_4⁆xPD⁅)⁆⁄xE_5⁆⁄xSQ_0⌉◀≺⇊⇇x⇊☜

✐☞┖➢§Ap3⧙⌊pg⌋⌈≡(⌊f⌋⌈≡(g(pp)≡(⌊r⌋⌈≡(f(rr)g(pp))⌉⌊r⌋⌈≡(f(rr)g(≡(p⌊q⌋⌈q⌉)p))⌉))⌉⌊q⌋⌈g(pq)⌉)⌉▶xQ_0⌊xTVQ{g}xTVQ{p}⁄xQ_0⌋xSQ_0⌈xE_15⁅xBOP⁆≡⧘xPG⁅(⁆xE_8⁅xQ_1⌊xTVQ{f}⁄xQ_1⌋xSQ_1⌈xE_17⁅xBOP⁆≡⧘xPG⁅(⁆xE_2⁅xBOP_⌊0⌋⁆g⧘xPG⁅(⁆xTV_⌊0⌋_1⁅L/p⁄xTV_1⁆xTV_⌊0⌋_2⁅L/p⁄xTV_2⁆xPD⁅)⁆⁄xE_2⁆xE_16⁅xBOP⁆≡⧘xPG⁅(⁆xE_9⁅xQ_2⌊xTVQ{r}⁄xQ_2⌋xSQ_2⌈xE_7⁅xBOP⁆≡⧘xPG⁅(⁆xE_3⁅xBOP_⌊1⌋⁆f⧘xPG⁅(⁆xTV_⌊2⌋_3⁅L/r⁄xTV_3⁆xTV_⌊2⌋_4⁅L/r⁄xTV_4⁆xPD⁅)⁆⁄xE_3⁆xE_4⁅xBOP_⌊0⌋⁆g⧘xPG⁅(⁆xTV_⌊0⌋_5⁅L/p⁄xTV_5⁆xTV_⌊0⌋_6⁅L/p⁄xTV_6⁆xPD⁅)⁆⁄xE_4⁆xPD⁅)⁆⁄xE_7⁆⁄xSQ_2⌉⁄xE_9⁆xE_10⁅xQ_3⌊xTVQ{r}⁄xQ_3⌋xSQ_3⌈xE_18⁅xBOP⁆≡⧘xPG⁅(⁆xE_5⁅xBOP_⌊1⌋⁆f⧘xPG⁅(⁆xTV_⌊3⌋_7⁅L/r⁄xTV_7⁆xTV_⌊3⌋_8⁅L/r⁄xTV_8⁆xPD⁅)⁆⁄xE_5⁆xE_14⁅xBOP_⌊0⌋⁆g⧘xPG⁅(⁆xE_13⁅xBOP⁆≡⧘xPG⁅(⁆xTV_⌊0⌋_9⁅L/p⁄xTV_9⁆xE_11⁅xQ_4⌊xTVQ{q}⁄xQ_4⌋xSQ_4⌈xE_1⁆xTV_⌊4⌋_10⁅L/q⁄xTV_10⁆⁄xE_1⁆⁄xSQ_4⌉⁄xE_11⁆xPD⁅)⁆⁄xE_13⁆xTV_⌊0⌋_11⁅L/p⁄xTV_11⁆xPD⁅)⁆⁄xE_14⁆xPD⁅)⁆⁄xE_18⁆⁄xSQ_3⌉⁄xE_10⁆xPD⁅)⁆⁄xE_16⁆xPD⁅)⁆⁄xE_

17⟧⁄xSQ_1⎤⁄xE_8⟧xE_12⟦xQ_5⌊xTVQ⎨q⎬⁄xQ_5⌋xSQ_5⌈xE_6⟦xBOP_⎡0⎦⦊g⦉xPG⁅(⁆xTV_⎡0⎦_12⟦L/p⁄xTV_12⟧xTV_⎡5⎦_13⟦L/q⁄xTV_13⟧xPD⁅)⁆⁄xE_6⟧⁄xSQ_5⎤⁄xE_12⟧xPD⁅)⁆⁄xE_15⟧⁄xSQ_0⎤◀≺╜ ╙x╜ ☜

✐☞╙➢⟅Dp1⟆⌊pq⌋⌈≡(μ(pq)≡(p≡(pq)))⎤▶xQ_0⌊xTVQ⎨p⎬xTVQ⎨q⎬⁄xQ_0⌋xSQ_0⌈xE_4⟦xBOP⦊≡⦉xPG⁅(⁆xE_1⟦xTCAC⎨μ⎬xCTX_1⎡xPG⁅(⁆xTV_⎡0⎦_0⟦D/p⁄xTV_0⟧xTV_⎡0⎦_0⟦D/q⁄xTV_0⟧xPD⁅)⁆⁄xCTX_1⎦⁄xE_1⟧xE_3⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡0⎦_1⟦L/p⁄xTV_1⟧xE_2⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡0⎦_2⟦L/p⁄xTV_2⟧xTV_⎡0⎦_3⟦L/q⁄xTV_3⟧xPD⁅)⁆⁄xE_2⟧xPD⁅)⁆⁄xE_3⟧xPD⁅)⁆⁄xE_4⟧⁄xSQ_0⎤◀≺╜ ╙x╜ ☜

✐☞╙➢⟅FGp1⟆⌊pq⌋⌈≡(Ц(pq)≡(q≡(pq)))⎤▶xQ_0⌊xTVQ⎨p⎬xTVQ⎨q⎬⁄xQ_0⌋xSQ_0⌈xE_4⟦xBOP⦊≡⦉xPG⁅(⁆xE_1⟦xTCAC⎨Ц⎬xCTX_1⎡xPG⁅(⁆xTV_⎡0⎦_0⟦D/p⁄xTV_0⟧xTV_⎡0⎦_0⟦D/q⁄xTV_0⟧xPD⁅)⁆⁄xCTX_1⎦⁄xE_1⟧xE_3⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡0⎦_1⟦L/q⁄xTV_1⟧xE_2⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡0⎦_2⟦L/p⁄xTV_2⟧xTV_⎡0⎦_3⟦L/q⁄xTV_3⟧xPD⁅)⁆⁄xE_2⟧xPD⁅)⁆⁄xE_3⟧xPD⁅)⁆⁄xE_4⟧⁄xSQ_0⎤◀≺╜ ╙x╜ ☜

✐☞╙➢⟅Dp2⟆⌊pq⌋⌈≡(τ(pq)≡(≡(pq)≡(pq)))⎤▶xQ_0⌊xTVQ⎨p⎬xTVQ⎨q⎬⁄xQ_0⌋xSQ_0⌈xE_5⟦xBOP⦊≡⦉xPG⁅(⁆xE_1⟦xTCAC⎨τ⎬xCTX_1⎡xPG⁅(⁆xTV_⎡0⎦_0⟦D/p⁄xTV_0⟧xTV_⎡0⎦_0⟦D/q⁄xTV_0⟧xPD⁅)⁆⁄xCTX_1⎦⁄xE_1⟧xE_4⟦xBOP⦊≡⦉xPG⁅(⁆xE_2⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡0⎦_1⟦L/p⁄xTV_1⟧xTV_⎡0⎦_2⟦L/q⁄xTV_2⟧xPD⁅)⁆⁄xE_2⟧xE_3⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡0⎦_3⟦L/p⁄xTV_3⟧xTV_⎡0⎦_4⟦L/q⁄xTV_4⟧xPD⁅)⁆⁄xE_3⟧xPD⁅)⁆⁄xE_4⟧xPD⁅)⁆⁄xE_5⟧⁄xSQ_0⎤◀≺╜ ╙x╜ ☜

✐☞╙➢⟅Dp3⟆⌊p⌋⌈≡(α(p)≡(p⌊q⌋⌈≡(qq)⎤))⎤▶xQ_0⌊xTVQ⎨p⎬⁄xQ_0⌋xSQ_0⌈xE_5⟦xBOP⦊≡⦉xPG⁅(⁆xE_1⟦xTCAC⎨α⎬xCTX_1⎡xPG⁅(⁆xTV_⎡0⎦_0⟦D/p⁄xTV_0⟧xPD⁅)⁆⁄xCTX_1⎦⁄xE_1⟧xE_4⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡0⎦_1⟦L/p⁄xTV_1⟧xE_3⟦xQ_1⌊xTVQ⎨q⎬⁄xQ_1⌋xSQ_1⌈xE_2⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡1⎦_2⟦L/q⁄xTV_2⟧xTV_⎡1⎦_3⟦L/q⁄xTV_3⟧xPD⁅)⁆⁄xE_2⟧⁄xSQ_1⎤⁄xE_3⟧xPD⁅)⁆⁄xE_4⟧xPD⁅)⁆⁄xE_5⟧⁄xSQ_0⎤◀≺╜ ╙x╜ ☜

✐☞╙➢⟅Dp4⟆⌊p⌋⌈≡(~(p)≡(p⌊q⌋⌈q⎤))⎤▶xQ_0⌊xTVQ⎨p⎬⁄xQ_0⌋xSQ_0⌈xE_5⟦xBOP⦊≡⦉xPG⁅(⁆xE_1⟦xTCAC⎨~⎬xCTX_1⎡xPG⁅(⁆xTV_⎡0⎦_0⟦D/p⁄xTV_0⟧xPD⁅)⁆⁄xCTX_1⎦⁄xE_1⟧xE_4⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡0⎦_1⟦L/p⁄xTV_1⟧xE_3⟦xQ_1⌊xTVQ⎨q⎬⁄xQ_1⌋xSQ_1⌈xE_2⟦xTV_⎡1⎦_2⟦L/q⁄xTV_2⟧⁄xE_2⟧⁄xSQ_1⎤⁄xE_3⟧xPD⁅)⁆⁄xE_4⟧xPD⁅)⁆⁄xE_5⟧⁄xSQ_0⎤◀≺╜ ╙x╜ ☜

✐☞╙➢⟅Dp5⟆⌊pq⌋⌈≡(ω(pq)¬(≡(pq)))⎤▶xQ_0⌊xTVQ⎨p⎬xTVQ⎨q⎬⁄xQ_0⌋xSQ_0⌈xE_4⟦xBOP⦊≡⦉xPG⁅(⁆xE_1⟦xTCAC⎨ω⎬xCTX_1⎡xPG⁅(⁆xTV_⎡0⎦_0⟦D/p⁄xTV_0⟧xTV_⎡0⎦_0⟦D/q⁄xTV_0⟧xPD⁅)⁆⁄xCTX_1⎦⁄xE_1⟧xE_3⟦xMOP⦊¬⦉xPG⁅(⁆xE_2⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡0⎦_1⟦L/p⁄xTV_1⟧xTV_⎡0⎦_2⟦L/q⁄xTV_2⟧xPD⁅)⁆⁄xE_2⟧xPD⁅)⁆⁄xE_3⟧xPD⁅)⁆⁄xE_4⟧⁄xSQ_0⎤◀≺╜ ╙x╜ ☜

✐☞╙➢⟅Dp6⟆⌊p⌋⌈≡(τ(p)≡(pp))⎤▶xQ_0⌊xTVQ⎨p⎬⁄xQ_0⌋xSQ_0⌈xE_3⟦xBOP⦊≡⦉xPG⁅(⁆xE_1⟦xTCAC⎨τ⎬xCTX_1⎡xPG⁅(⁆xTV_⎡0⎦_0⟦D/p⁄xTV_0⟧xPD⁅)⁆⁄xCTX_1⎦⁄xE_1⟧xE_2⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡0⎦_1⟦L/p⁄xTV_1⟧xTV_⎡0⎦_2⟦L/p⁄xTV_2⟧xPD⁅)⁆⁄xE_2⟧xPD⁅)⁆⁄xE_3⟧⁄xSQ_0⎤◀≺╜ ╙x╜ ☜

✐☞╙➢⟅Dp7⟆⌊pqr⌋⌈≡(δ(pqr)≡(r¬(≡(pq))))⎤▶xQ_0⌊xTVQ⎨p⎬xTVQ⎨q⎬xTVQ⎨r⎬⁄xQ_0⌋xSQ_0⌈xE_5⟦xBOP⦊≡⦉xPG⁅(⁆xE_1⟦xTCAC⎨δ⎬xCTX_1⎡xPG⁅(⁆xTV_⎡0⎦_0⟦D/p⁄xTV_0⟧xTV_⎡0⎦_0⟦D/q⁄xTV_0⟧xTV_⎡0⎦_0⟦D/r⁄xTV_0⟧xPD⁅)⁆⁄xCTX_1⎦⁄xE_1⟧xE_4⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡0⎦_1⟦L/r⁄xTV_1⟧xE_3⟦xMOP⦊¬⦉xPG⁅(⁆xE_2⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡0⎦_2⟦L/p⁄xTV_2⟧xTV_⎡0⎦_3⟦L/q⁄xTV_3⟧xPD⁅)⁆⁄xE_2⟧xPD⁅)⁆⁄xE_3⟧xPD⁅)⁆⁄xE_4⟧xPD⁅)⁆⁄xE_5⟧⁄xSQ_0⎤◀≺╜ ╙x╜ ☜

✐☞╙➢⟅Dp8⟆≡(F⌊p⌋⌈p⎤)▶xE_3⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡-1⎦_2⟦C/F⁄xTV_2⟧xE_2⟦xQ_0⌊xTVQ⎨p⎬⁄xQ_0⌋xSQ_0⌈xE_1⟦xTV_⎡0⎦_1⟦L/p⁄xTV_1⟧⁄xE_1⟧⁄xSQ_0⎤⁄xE_2⟧xPD⁅)⁆⁄xE_3⟧◀≺╜ ╙x╜ ☜

✐☞╙➢⟅Dp9⟆≡(V~(F))▶xE_2⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡-1⎦_1⟦C/V⁄xTV_1⟧xE_1⟦xMOP⦊~⦉xPG⁅(⁆xTV_⎡-1⎦_2⟦C/F⁄xTV_2⟧xPD⁅)⁆⁄xE_1⟧xPD⁅)⁆⁄xE_2⟧◀≺╜ ╙x╜ ☜

✐☞╙➢⟅Dp10⟆≡(T⌊p⌋⌈≡(pp)⎤)▶xE_3⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡-1⎦_3⟦C/T⁄xTV_3⟧xE_2⟦xQ_0⌊xTVQ⎨p⎬⁄xQ_0⌋xSQ_0⌈xE_1⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡0⎦_1⟦L/p⁄xTV_1⟧xTV_⎡0⎦_2⟦L/p⁄xTV_2⟧xPD⁅)⁆⁄xE_1⟧⁄xSQ_0⎤⁄xE_2⟧xPD⁅)⁆⁄xE_3⟧◀≺╜ ╙x╜ ☜

✐☞╙➢⟅Dp11⟆⌊qpr⌋⌈≡(ω⎝pq⎠(r)≡(~(≡(pq))r))⎤▶xQ_0⌊xTVQ⎨p⎬xTVQ⎨q⎬xTVQ⎨r⎬⁄xQ_0⌋xSQ_0⌈xE_5⟦xBOP⦊≡⦉xPG⁅(⁆xE_1⟦xTCAC⎨ω⎬xCTX_1⎡xPG⁅⎝⁆xTV_⎡0⎦_0⟦D/p⁄xTV_0⟧xTV_⎡0⎦_0⟦D/q⁄xTV_0⟧xPD⁅⎠⁆⁄xCTX_1⎦xCTX_2⎡xPG⁅(⁆xTV_⎡0⎦_0⟦D/r⁄xTV_0⟧xPD⁅)⁆⁄xCTX_2⎦⁄xE_1⟧xE_4⟦xBOP⦊≡⦉xPG⁅(⁆xE_3⟦xMOP⦊~⦉xPG⁅(⁆xE_2⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡0⎦_1⟦L/p⁄xTV_1⟧xTV_⎡0⎦_2⟦L/q⁄xTV_2⟧xPD⁅)⁆⁄xE_2⟧xPD⁅)⁆⁄xE_3⟧xTV_⎡0⎦_3⟦L/r⁄xTV_3⟧xPD⁅)⁆⁄xE_4⟧xPD⁅)⁆⁄xE_5⟧⁄xSQ_0⎤◀≺╜ ╙x╜ ☜

✐☞╙➢⟅Dp12⟆⌊qpr⌋⌈≡(ω[p]⎝q⎠(r)≡(~(≡(pq))r))⎤▶xQ_0⌊xTVQ⎨p⎬xTVQ⎨q⎬xTVQ⎨r⎬⁄xQ_0⌋xSQ_0⌈xE_5⟦xBOP⦊≡⦉xPG⁅(⁆xE_1⟦xTCAC⎨ω⎬xCTX_1⎡xPG⁅[⁆xTV_⎡0⎦_0⟦D/p⁄xTV_0⟧xPD⁅]⁆⁄xCTX_1⎦xCTX_2⎡xPG⁅⎝⁆xTV_⎡0⎦_0⟦D/q⁄xTV_0⟧xPD⁅⎠⁆⁄xCTX_2⎦xCTX_3⎡xPG⁅(⁆xTV_⎡0⎦_0⟦D/r⁄xTV_0⟧xPD⁅)⁆⁄xCTX_3⎦⁄xE_1⟧xE_4⟦xBOP⦊≡⦉xPG⁅(⁆xE_3⟦xMOP⦊~⦉xPG⁅(⁆xE_2⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡0⎦_1⟦L/p⁄xTV_1⟧xTV_⎡0⎦_2⟦L/q⁄xTV_2⟧xPD⁅)⁆⁄xE_2⟧xPD⁅)⁆⁄xE_3⟧xTV_⎡0⎦_3⟦L/r⁄xTV_3⟧xPD⁅)⁆⁄xE_4⟧xPD⁅)⁆⁄xE_5⟧⁄xSQ_0⎤◀≺╜ ╙x╜ ☜

✐☞╙➢⟅Dp13⟆⌊pqg⌋⌈≡(Δ[gpq]≡(p~(g(pq)))))⎤▶xQ_0⌊xTVQ⎨g⎬xTVQ⎨p⎬xTVQ⎨q⎬⁄xQ_0⌋xSQ_0⌈xE_5⟦xBOP⦊≡⦉xPG⁅(⁆xE_1⟦xTCAC⎨Δ⎬xCTX_1⎡xPG⁅[⁆xTV_⎡0⎦_0⟦D/g⁄xTV_0⟧xTV_⎡0⎦_0⟦D/p⁄xTV_0⟧xTV_⎡0⎦_0⟦D/q⁄xTV_0⟧xPD⁅]⁆⁄xCTX_1⎦⁄xE_1⟧xE_4⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡0⎦_1⟦L/p⁄xTV_1⟧xE_3⟦xMOP⦊~⦉xPG⁅(⁆xE_2⟦xBOP_⎡0⎦⦊g⦉xPG⁅(⁆xTV_⎡0⎦_2⟦L/p⁄xTV_2⟧xTV_⎡0⎦_3⟦L/q⁄xTV_3⟧xPD⁅)⁆⁄xE_2⟧xPD⁅)⁆⁄xE_3⟧xPD⁅)⁆⁄xE_4⟧xPD⁅)⁆⁄xE_5⟧⁄xSQ_0⎤◀≺╜ ╙x╜ ☜

✐☞╙➢⟅Dp14⟆⌊pq⌋⌈≡(∧(pq)⌊h⌋⌈≡(p≡(h(p)h(q)))⎤)⎤▶xQ_0⌊xTVQ⎨p⎬xTVQ⎨q⎬⁄xQ_0⌋xSQ_0⌈xE_7⟦xBOP⦊≡⦉xPG⁅(⁆xE_1⟦xTCAC⎨∧⎬xCTX_1⎡xPG⁅(⁆xTV_⎡0⎦_0⟦D/p⁄xTV_0⟧xTV_⎡0⎦_0⟦D/q⁄xTV_0⟧xPD⁅)⁆⁄xCTX_1⎦⁄xE_1⟧xE_6⟦xQ_1⌊xTVQ⎨h⎬⁄xQ_1⌋xSQ_1⌈xE_5⟦xBOP⦊≡⦉xPG⁅(⁆xTV_⎡0⎦_1⟦L/p⁄xTV_1⟧xE_4⟦xBOP⦊≡⦉xPG⁅(⁆xE_2⟦xMOP_⎡1⎦⦊h⦉xPG⁅(⁆xTV_⎡0⎦_2⟦L/p⁄xTV_2⟧xPD⁅)⁆⁄xE_2⟧xE_3⟦xMOP_⎡1⎦⦊h⦉xPG⁅(⁆xTV_⎡0⎦_3⟦L/q⁄xTV_3⟧xPD⁅)⁆⁄xE_3⟧xPD⁅)⁆⁄xE_4⟧xPD⁅)⁆⁄xE_5⟧⁄xSQ_1⎤⁄xE_6⟧xPD⁅)⁆⁄xE_7⟧⁄xSQ_0⎤◀≺╜ ╙x╜ ☜

✐☞╙➢⟅Dp15⟆⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⎤▶xQ_0⌊xTVQ⎨p⎬xTVQ⎨q⎬⁄xQ_0⌋xSQ_0⌈xE_5⟦xBOP⦊≡⦉xPG⁅(⁆xE_1⟦xTCAC⎨⊃⎬xCTX_1⎡xPG⁅(⁆xTV_⎡0⎦_0⟦D/p⁄xTV_0⟧xTV_⎡0⎦_0⟦D/q⁄xTV_0⟧xPD⁅)⁆⁄xCTX_1⎦⁄xE_1⟧xE_4⟦xMOP⦊~⦉xP

G[(]xE_3[xBOP]∧⟨xPG[(]xTV_[0]_1[L/p/xTV_1[xE_2[xMOP]~⟨xPG[(]xTV_[0]_2[L/q/xTV_2[xPD[)]/xE_2[xPD[)]/xE_3[xPD[)]/xE_4[xPD[)]/xE_5⟩/xSQ_0]◀≮╝╚x╝☜

✎☞╚➢[Dp16⟩⌊pq⌋⌈≡(∨(pq)~(∧(~(p)~(q))))⌉▶xQ_0⌊xTVQ[p]xTVQ[q]⟩/xQ_0⌋xSQ_0⌈xE_6[xBOP]≡⟨xPG[(]xE_1[xTCAC[∨]xCTX_1[xPG[(]xTV_[0]_0[D/p/xTV_0[xTV_[0]_0[D/q/xTV_0[xPD[)]/xCTX_1]/xE_1[xE_5[xMOP]~⟨xPG[(]xE_4[xBOP]∧⟨xPG[(]xE_2[xMOP]~⟨xPG[(]xTV_[0]_1[L/p/xTV_1[xPD[)]/xE_2[xE_3[xMOP]~⟨xPG[(]xTV_[0]_2[L/q/xTV_2[xPD[)]/xE_3[xPD[)]/xE_4[xPD[)]/xE_5[xPD[)]/xE_6⟩/xSQ_0]◀≮╝╚x╝☜

✎☞╚➢[Dp17⟩⌊𝒜⌋⌈≡(ℰ<𝒜>⌊q⌋⌈𝒜(q)⌉)⌉▶xQ_0⌊xTVQ[𝒜]⟩/xQ_0⌋xSQ_0⌈xE_4[xBOP]≡⟨xPG[(]xE_1[xTCAC[ℰ]xCTX_1[xPG[<]xTV_[0]_0[D/𝒜/xTV_0[xPD[>]/xCTX_1]/xE_1[xE_3[xQ_1⌊xTVQ[q]⟩/xQ_1⌋xSQ_1⌈xE_2[xMOP]⟨xPG[(]xTV_[1]_1[L/q/xTV_1[xPD[)]/xE_2⟩/xSQ_1]/xE_3[xPD[)]/xE_4⟩/xSQ_0]◀≮╝╚x╝☜

✎☞╚➢[Dp18⟩⌊ℬ⌋⌈≡(ℰ\ℬ⌊pq⌋⌈ℬ(pq)⌉)⌉▶xQ_0⌊xTVQ[ℬ]⟩/xQ_0⌋xSQ_0⌈xE_4[xBOP]≡⟨xPG[(]xE_1[xTCAC[ℰ]xCTX_1[xPG[\]xTV_[0]_0[D/ℬ/xTV_0[xPD[/]/xCTX_1]/xE_1[xE_3[xQ_1⌊xTVQ[p]xTVQ[q]⟩/xQ_1⌋xSQ_1⌈xE_2[xBOP]ℬ⟨xPG[(]xTV_[1]_1[L/p/xTV_1[xTV_[1]_2[L/q/xTV_2[xPD[)]/xE_2⟩/xSQ_1]/xE_3[xPD[)]/xE_4⟩/xSQ_0]◀≮╝╚x╝☜

✎☞╚➢[Tp1:Ap1.SUB,p/q,q/r⟩⌊qr⌋⌈≡(≡(≡(qr)≡(rq))≡(rr))⌉▶xQ_0⌊xTVQ[q]xTVQ[r]⟩/xQ_0⌋xSQ_0⌈xE_5[xBOP]≡⟨xPG[(]xE_4[xBOP]≡⟨xPG[(]xE_1[xBOP]≡⟨xPG[(]xTV_[0]_1[L/q/xTV_1[xTV_[0]_2[L/r/xTV_2[xPD[)]/xE_1[xE_2[xBOP]≡⟨xPG[(]xTV_[0]_3[L/r/xTV_3[xTV_[0]_4[L/q/xTV_4[xPD[)]/xE_2[xPD[)]/xE_4[xE_3[xBOP]≡⟨xPG[(]xTV_[0]_5[L/r/xTV_5[xTV_[0]_6[L/r/xTV_6[xPD[)]/xE_3[xPD[)]/xE_5⟩/xSQ_0]◀≮╝╚x╝☜

☞╚➢[Tp2:Ap1.SUB,p/r,r/q=r,q/r=q⟩⌊qr⌋⌈≡(≡(≡(r=(qr))≡(≡(rq)r))≡(≡(qr)≡(rq)))⌉▶xQ_0⌊xTVQ[q]xTVQ[r]⟩/xQ_0⌋xSQ_0⌈xE_9[xBOP]≡⟨xPG[(]xE_7[xBOP]≡⟨xPG[(]xE_5[xBOP]≡⟨xPG[(]xTV_[0]_1[L/r/xTV_1[xE_1[xBOP]≡⟨xPG[(]xTV_[0]_2[L/q/xTV_2[xTV_[0]_3[L/r/xTV_3[xPD[)]/xE_1[xPD[)]/xE_5[xE_6[xBOP]≡⟨xPG[(]xE_2[xBOP]≡⟨xPG[(]xTV_[0]_4[L/r/xTV_4[xTV_[0]_5[L/q/xTV_5[xPD[)]/xE_2[xTV_[0]_6[L/r/xTV_6[xPD[)]/xE_6[xPD[)]/xE_7[xE_8[xBOP]≡⟨xPG[(]xE_3[xBOP]≡⟨xPG[(]xTV_[0]_7[L/q/xTV_7[xTV_[0]_8[L/r/xTV_8[xPD[)]/xE_3[xE_4[xBOP]≡⟨xPG[(]xTV_[0]_9[L/r/xTV_9[xTV_[0]_10[L/q/xTV_10[xPD[)]/xE_4[xPD[)]/xE_8[xPD[)]/xE_9⟩/xSQ_0]◀≮╝╚x╝☜

☞╚➢[Tp3:Ap2.SUB,p/r⟩⌊qr⌋⌈≡(≡(r=(qr))≡(≡(rq)r))⌉▶xQ_0⌊xTVQ[q]xTVQ[r]⟩/xQ_0⌋xSQ_0⌈xE_5[xBOP]≡⟨xPG[(]xE_3[xBOP]≡⟨xPG[(]xTV_[0]_1[L/r/xTV_1[xE_1[xBOP]≡⟨xPG[(]xTV_[0]_2[L/q/xTV_2[xTV_[0]_3[L/r/xTV_3[xPD[)]/xE_1[xPD[)]/xE_3[xE_4[xBOP]≡⟨xPG[(]xE_2[xBOP]≡⟨xPG[(]xTV_[0]_4[L/r/xTV_4[xTV_[0]_5[L/q/xTV_5[xPD[)]/xE_2[xTV_[0]_6[L/r/xTV_6[xPD[)]/xE_4[xPD[)]/xE_5⟩/xSQ_0]◀≮╝╚x╝☜

✎☞╚➢[Tp4:Tp2.DIS,q+r⟩≡(⌊qr⌋⌈≡(≡(r=(qr))≡(≡(rq)r))⌉⌊qr⌋⌈≡(≡(qr)≡(rq))⌉)▶xE_11[xBOP]≡⟨xPG[(]xE_9[xQ_0⌊xTVQ[q]xTVQ[r]⟩/xQ_0⌋xSQ_0⌈xE_7[xBOP]≡⟨xPG[(]xE_5[xBOP]≡⟨xPG[(]xTV_[0]_1[L/r/xTV_1[xE_1[xBOP]≡⟨xPG[(]xTV_[0]_2[L/q/xTV_2[xTV_[0]_3[L/r/xTV_3[xPD[)]/xE_1[xPD[)]/xE_5[xE_6[xBOP]≡⟨xPG[(]xE_2[xBOP]≡⟨xPG[(]xTV_[0]_4[L/r/xTV_4[xTV_[0]_5[L/q/xTV_5[xPD[)]/xE_2[xTV_[0]_6[L/r/xTV_6[xPD[)]/xE_6[xPD[)]/xE_7⟩/xSQ_0]/xE_9[xE_10[xQ_1⌊xTVQ[q]xTVQ[r]⟩/xQ_1⌋xSQ_1⌈xE_8[xBOP]≡⟨xPG[(]xE_3[xBOP]≡⟨xPG[(]xTV_[1]_7[L/q/xTV_7[xTV_[1]_8[L/r/xTV_8[xPD[)]/xE_3[xE_4[xBOP]≡⟨xPG[(]xTV_[1]_9[L/r/xTV_9[xTV_[1]_10[L/q/xTV_10[xPD[)]/xE_4[xPD[)]/xE_8⟩/xSQ_1]/xE_10[xPD[)]/xE_11⟩◀≮╝╚x╝☜

✎☞╚➢[Tp5:Tp3,Tp4,DET⟩⌊qr⌋⌈≡(≡(qr)≡(rq))⌉▶xQ_0⌊xTVQ[q]xTVQ[r]⟩/xQ_0⌋xSQ_0⌈xE_3[xBOP]≡⟨xPG[(]xE_1[xBOP]≡⟨xPG[(]xTV_[0]_1[L/q/xTV_1[xTV_[0]_2[L/r/xTV_2[xPD[)]/xE_1[xE_2[xBOP]≡⟨xPG[(]xTV_[0]_3[L/r/xTV_3[xTV_[0]_4[L/q/xTV_4[xPD[)]/xE_2[xPD[)]/xE_3⟩/xSQ_0]◀≮╝╚x╝☜

✎☞╚➢[Ao⟩⌊Ab⌋⌈≡(ε{Ab}∧(~(⌊B⌋⌈~(ε{BA})⌉)∧(⌊DC⌋⌈⊃(∧(ε{DA}ε{CA})ε{DC})⌉⌊D⌋⌈⊃(ε{DA}ε{Db})⌉))))⌉▶xQ_0⌊xTVQ[A]xTVQ[b]⟩/xQ_0⌋xSQ_0⌈xE_18[xBOP]≡⟨xPG[(]xE_1[xTCAC[ε]xCTX_1[xPG[{]xTV_[0]_0[D/A/xTV_0[xTV_[0]_0[D/b/xTV_0[xPD[}]]/xCTX_1]/xE_1[xE_17[xBOP]∧⟨xPG[(]xE_15[xMOP]~⟨xPG[(]xE_12[xQ_1⌊xTVQ[B]⟩/xQ_1⌋xSQ_1⌈xE_8[xMOP]~⟨xPG[(]xE_2[xBOP]ε⟨xPG[{]xTV_[1]_1[L/B/xTV_1[xTV_[0]_2[L/A/xTV_2[xPD[}]]/xE_2[xPD[)]/xE_8⟩/xSQ_1]/xE_12[xPD[)]/xE_15[xE_16[xBOP]∧⟨xPG[(]xE_13[xQ_2⌊xTVQ[C]xTVQ[D]⟩/xQ_2⌋xSQ_2⌈xE_10[xBOP]⊃⟨xPG[(]xE_9[xBOP]∧⟨xPG[(]xE_3[xBOP]ε⟨xPG[{]xTV_[2]_3[L/D/xTV_3[xTV_[0]_4[L/A/xTV_4[xPD[}]]/xE_3[xE_4[xBOP]ε⟨xPG[{]xTV_[2]_5[L/C/xTV_5[xTV_[0]_6[L/A/xTV_6[xPD[}]]/xE_4[xPD[)]]/xE_9[xE_5[xBOP]ε⟨xPG[{]xTV_[2]_7[L/D/xTV_7[xTV_[2]_8[L/C/xTV_8[xPD[}]]/xE_5[xPD[)]/xE_10⟩/xSQ_2]/xE_13[xE_14[xQ_3⌊xTVQ[D]⟩/xQ_3⌋xSQ_3⌈xE_11[xBOP]⊃⟨xPG[(]xE_6[xBOP]ε⟨xPG[{]xTV_[3]_9[L/D/xTV_9[xTV_[0]_10[L/A/xTV_10[xPD[}]]/xE_6[xE_7[xBOP]ε⟨xPG[{]xTV_[3]_11[L/D/xTV_11[xTV_[0]_12[L/b/xTV_12[xPD[}]]/xE_7[xPD[)]/xE_11⟩/xSQ_3]/xE_14[xPD[)]/xE_16[xPD[)]/xE_17[xPD[)]/xE_18⟩/xSQ_0]◀≮╝╚x╝☜

✎☞╚➢[Do17⟩⌊a⌋⌈≡(!{a}~(⌊A⌋⌈~(ε{Aa})⌉))⌉▶xQ_0⌊xTVQ[a]⟩/xQ_0⌋xSQ_0⌈xE_6[xBOP]≡⟨xPG[(]xE_1[xTCAC[!]xCTX_1[xPG[{]xTV_[0]_0[D/a/xTV_0[xPD[}]]/xCTX_1]/xE_1[xE_5[xMOP]~⟨xPG[(]xE_4[xQ_1⌊xTVQ[A]⟩/xQ_1⌋xSQ_1⌈xE_3[xMOP]~⟨xPG[(]xE_2[xBOP]ε⟨xPG[{]xTV_[1]_1[L/A/xTV_1[xTV_[0]_2[L/a/xTV_2[xPD[}]]/xE_2[xPD[)]/xE_3⟩/xSQ_1]/xE_4[xPD[)]/xE_5[xPD[)]/xE_6⟩/xSQ_0]◀≮╝╚x╝☜

↳ENDLIBRARY↲

↳RULES↲

✎ A partir de la (première) formule de INPUT_CORPUS,
✎ détacher la première équivalence pour comparaison
✎ ... boucler sur les formules de LIBRARY_CORPUS...
✎ ... comparer la première équivalence de l'incription donnée avec l'inscription trouvée dans la librairie
✎ La comparaison se fait sur des formules analysées (encapsulées) qui n'ont pas nécessairement la
✎ même numérotation

✎ PREPARER LA PREMIERE EQUIVALENCE DE LA FORMULE EN ENTREE
✎ **********

✎ La chaîne soumise à la règle de détachement est placée dans la zone de travail, soit :
✎ ➢‹avant›▶‹zonedetravail›◀‹apres›≮⇨‹prem_equivalence›⇦➲‹sec_equivalence›➲

✎ copier dans une zone temporaire la formule donnée
☞➢‹avant›►‹formule›◄‹apres›≺⇒➢‹avant›►‹formule›◄‹apres›≺▷‹formule›◁☜✎[1]

✎ dépister et cerner une expression biconditionnelle dominante dans l'inscription donnée
✎ attention : on ne travaille que sur l'expression dominante
☞➢‹avant›►‹zone›◄‹apres›≺▷xE_‹i›[xBOP]≡[‹v›◁⇒
➢‹avant›►‹zone›◄‹apres›≺▷xE_‹i›[xBOP]≡[●→‹v›←●◁/KO/[‹i›]/mark1_form☜✎[2]
☞➢‹avant›►‹zone›◄‹apres›≺▷‹a›●→‹v›/xE_‹i›↳INTEGER↵›]←●◁/KO/[‹j›]/mark1_form⇒
➢‹avant›►‹zone›◄‹apres›≺▷‹a›●→‹v›←●/xE_‹i›]◁/KO/[‹j›][‹i›]/mark2_form☜✎[3]
☞➢‹avant›►‹zone›◄‹apres›≺▷‹v›◁/KO/[‹i›][‹i›]/mark2_form⇒
➢‹avant›►‹zone›◄‹apres›≺▷‹v›◁/OK/mark3_form☜✎[4]
✎ cerner le contenu de l'expression, donc le contenu des délimiteurs
☞➢‹avant›►‹zone›◄‹apres›≺▷‹a›●→xPG(‹v›xPD)]←●‹b›◁/OK/mark3_form⇒
➢‹avant›►‹zone›◄‹apres›≺▷‹a›xPG(●→‹v›←●xPD)]‹b›◁/OK/mark4_form☜✎[5]
✎ cerner l'expression de la première équivalence et cerner l'expression de la deuxième équivalence
☞➢‹avant›►‹zone›◄‹apres›≺▷‹a›●→xE_‹i›[‹v›←●‹b›◁/OK/mark4_form⇒
➢‹avant›►‹zone›◄‹apres›≺▷‹a›●→xE_‹i›[‹v›←●‹b›◁/KO/[‹i›]/mark5_form☜✎[6]
☞➢‹avant›►‹zone›◄‹apres›≺▷‹a›●→‹v›/xE_‹i›]xE_‹k›[‹b›←●‹z›◁/KO/[‹i›]/mark5_form⇒
➢‹avant›►‹zone›◄‹apres›≺▷‹a›●→‹v›/xE_‹i›]←●○→xE_‹k›[‹b›←○‹z›◁/KO/[‹k›]/mark6_form☜✎[7]
☞➢‹avant›►‹zone›◄‹apres›≺▷‹a›●→‹v›←●○→‹b›/xE_‹i›]←○‹z›◁/KO/[‹i›]/mark6_form⇒
➢‹avant›►‹zone›◄‹apres›≺▷‹a›●→‹v›←●○→‹b›/xE_‹i›]←○‹z›◁/OK/mark7_form☜✎[8]
✎ si la première expression est une généralisation, enlever l'encapsulation
☞➢‹avant›►‹zone›◄‹apres›≺▷‹a›●→xE_‹i›[xQ_‹j›⌊‹v›/xSQ_‹j›⌉/xE_‹i›]←●‹b›◁/OK/mark7_form⇒
➢‹avant›►‹zone›◄‹apres›≺▷‹a›●→xQ_‹j›⌊‹v›/xSQ_‹j›⌉←●‹b›◁/OK/mark7_form☜✎[9]
✎ si la deuxième expression est une généralisation, enlever l'encapsulation
☞➢‹avant›►‹zone›◄‹apres›≺▷‹a›○→xE_‹i›[xQ_‹j›⌊‹v›/xSQ_‹j›⌉/xE_‹i›]←○‹b›◁/OK/mark7_form⇒
➢‹avant›►‹zone›◄‹apres›≺▷‹a›○→xQ_‹j›⌊‹v›/xSQ_‹j›⌉←○‹b›◁/OK/mark7_form☜✎[10]

✎ effacer les marqueurs
☞➢‹avant›►‹zone›◄‹apres›≺▷‹v›◁/OK/mark7_form⇒➢‹avant›►‹zone›◄‹apres›≺▷‹v›◁☜✎[11]
✎ on sauvegarde la deuxième équivalence pour l'extraite
✎ si l'on trouve une expression qui permette le détachement
☞➢‹avant›►‹zone›◄‹apres›≺▷‹a›●→‹v›←●○→‹w›←○‹z›◁⇒
➢‹avant›►‹zone›◄‹apres›/SNDEQUIV=⌈‹w›⌋≺⇨‹v›⇦/DIFFERENTE☜✎[12]

✎ BOUCLER SUR LA LIBRAIRIE
✎ **********

✎ commuter vers LIBRARY_CORPUS
☞↳SWITCHCORPUS(↳LIBRARY_CORPUS↵)↵☜✎[13]
✎ se positionner à la première ligne de LIBRARY_CORPUS
☞↳CORPUSFIRST↵☜✎[14]

✎ >>> LIGNE_SUIVANTE <<<

✎ extraire la ligne de LIBRARY_CORPUS, le contenu est mis dans TEMPORARY_ZONE
☞↳GETCORPUSLINE↵☜✎[15]

✎ concaténation à la première équivalence de la formule courante de la librairie
✎ qui se trouve dans TEMPORARY_ZONE qui peut retourner aussi {<EOF>}
☞↳CONCAT(↳RM_RESULT_ZONE↵,↳RM_RESULT_ZONE↵,↳TEMPORARY_ZONE↵)↵☜✎[16]
✎ tester eof après concaténation, si eof : TEMPORARY_ZONE retourne {<EOF>}

✎ ALLER A >>> FIN<<<
☞‹a›/DIFFERENTE{<EOF>}⇒‹a›/EOF↳GOTORULE(26)↵☜✎[17]

✎ se positionner à la ligne suivante de LIBRARY_CORPUS
✎ (instruction mise ici pour ne pas déranger le statut de ifmatch_goto
✎ cette instruction ne décide pas si il y a exécution réussie / analyse de correspondances satisfaite
✎ d'un membre de droit)
☞↳CORPUSNEXT↵☜✎[18]

✎ éxécuter une règle d'affichage qui doit remettre le status "sequencematched" à true
✎ et positionner les séparateurs en fin : ➲➲
☞➢‹avant›►‹zone›◄‹apres›≺⇨‹prm›⇦/DIFFERENTE‹equivcompare›⇒
UNLOOP➢‹avant›►‹zone›◄‹apres›≺⇨‹prm›⇦➲‹equivcompare›➲☜✎[19]
☞UNLOOP➢‹avant›►‹zone›◄‹apres›≺⇨‹prm›⇦➲‹sec›➲⇒
➢‹avant›►‹zone›◄‹apres›≺⇨‹prm›⇦➲‹sec›➲/concat1☜✎[20]

✎ traiter la formule de la ligne de la librairie
☞➢‹avant›►‹zone›◄‹apres›≺⇨‹prm›⇦➲➢[‹id›]‹exp›►‹libform›◄‹y›≺‹x›➲/concat1⇒
➢‹avant›►‹zone›◄‹apres›/DETACH=[‹id›][‹exp›]≺⇨‹prm›⇦➲‹libform›➲☜✎[21]

✎ comparer les deux expressions prm et sec
☞↳EXECUTEGRAM(COMPARE)↲☜✎[22]

✎ tester si trouvé une équivalence

✎ ALLER A >>> FIN<<<
☞➢‹avant›▶‹zone›◀‹apres›
/SNDEQUIV=⁅‹sndEquivDetach›⁆/DETACH=⟦‹id›⟧⟦‹exp›⟧≺[<EQUIV>]⇨‹prm›⇦➲‹sec›➲⇒
➢‹avant›▶‹zone›◀‹apres›≺
▷⟦‹id›⟧⟦‹exp›⟧_/_‹sndEquivDetach›◁/EQUIVALENTE↳GOTORULE(26)↲☜✎[23]
✎ pas trouvé, nettoyer la formule
☞➢‹avant›▶‹zone›◀‹apres›/DETACH=⟦‹id›⟧⟦‹exp›⟧≺[<DIFF>]⇨‹prm›⇦➲‹sec›➲⇒
➢‹avant›▶‹zone›◀‹apres›≺⇨‹prm›⇦/DIFFERENTE☜✎[24]

✎ ALLER A >>> LIGNE_SUIVANTE <<<
☞↳IFMATCH_GOTORULE(15)↲☜✎ ligne de la librairie suivante [25]

✎ >>> FIN <<<

☞➢‹avant›▶‹zone›◀‹apres›≺▷‹w›◁/EQUIVALENTE⇒➢‹avant›▶‹zone›◀‹apres›≺▷‹w›◁☜✎[26]
☞➢‹avant›▶‹zone›◀‹apres›/SNDEQUIV=⁅‹reste›⁆◁≺⇨‹prm›⇦/EOF⇒
➢‹avant›▶‹zone›◀‹apres›≺▷DIFF◁☜✎[27]

☞↳STOP↲☜✎[28]
↳ENDRULES↲
↳ENDGRAM↲

GO_i de comparaison

↳GRAM↲
☞GRAMNAME⇒COMPARE☜
✎ Comparer deux "formules" tenant compte du plus haut niveau d'encapsulation
✎ Dès lors il s'agit de tester pour la première encapsulation : toutes les encapsulations possibles :
✎ xE, xT, xQ, xSQ, xTVx
✎ format d'entrée :
✎ ➢‹av.›▶‹zonedetravail›◀‹ap.›≺⇨‹prem_equivalence_donnee›⇦➲‹sec_equivalence_reference›➲
✎ où ‹prem_equivalence› et ‹sec_equivalence› sont les formules à comparer
✎ Les curseurs utilisés : ●→←●, ○→←○, ◊→←◊
✎ On cerne d'abord les encapsulations de la formule de gauche et ensuite celle de droite
✎ et cela de gauche à droite
✎ Pourquoi ne testons-nous pas tout d'un coup ? A cause des numéroteurs d'encapsulation !
✎ *** Attention : bic est considéré dans cette grammaire comme un binop et est traité comme tel ***

↳PARAMS↲
☞MAXGRAMLOOP⇒5☜
☞MAXRULELOOP⇒30☜
↳ENDPARAMS↲

↳SETS↲
✎ Variables
☞vp⇒"p","q","r","s"☜✎ variables propositionnelles
☞vns⇒"A","B","C","D"☜✎ variables de nom singulier
☞vng⇒"a","b"☜✎ variables de nom générique
☞vn⇒vns,vng☜✎ les symboles de variable de nom
☞vp_vn⇒vp,vns,vng☜✎ tous les symboles de variable

✎ Fonctions Propositionnelles
☞fp2⇒"f","g","𝓔"☜✎ fonctions propositionnelles à deux places
☞fp1⇒"h","𝓐"☜✎ fonctions propositionnelles à une place
☞fp⇒fp2,fp1☜✎ toutes les fonctions propositionnelles

☞vp_vn_fp⇒vp,vns,vng,fp☜✎ tous les symboles de termes qui peuvent être quantifiés

✎ Constantes, termes constants
☞bic⇒"≡"☜✎ biconditionnelle
☞cp0⇒"V","F","T"☜✎ constantes propositionnelles sans contexte
☞tc1⇒"~","¬","α","!","›","↓","𝓔","𝓐"☜✎ constantes unaires
☞tc2⇒"μ","Ц","ε","∧","∨","⊃","𝓔"☜✎ constantes avec contexte binaire
☞tcn⇒"Δ","δ"☜✎ constantes n_aires
☞tcmc⇒"τ","ω"☜✎ constantes multi-contextes
☞tcac⇒tc1,tc2,tcn,tcmc☜✎ les constantes avec contexte(s) sans BIC

☞mop⇒tc1,fp1☜🖉 opérateurs unaires
☞bop⇒tc2,bic,fp2☜🖉 opérateurs binaires
☞xop⇒tcn,tcmc☜🖉 opérateurs complexes, multi-contextes

🖉 Parenthèses gauches et délimiteurs gauches de contexte
☞pgA⇒"("☜
☞pgB⇒"["☜
☞pgC⇒"{"☜
☞pgD⇒"<"☜
☞pgE⇒"\"☜
☞pgF⇒"⟨"☜
🖉 ...
☞pg⇒pgA,pgB,pgC,pgD,pgE,pgF☜🖉 ...

🖉 Parenthèses droites et délimiteurs droits de contexte
☞pdA⇒")"☜
☞pdB⇒"]"☜
☞pdC⇒"}"☜
☞pdD⇒">"☜
☞pdE⇒"/"☜
☞pdF⇒"⟩"☜
🖉 ...
☞pd⇒pdA,pdB,pdC,pdD,pdE,pdF☜🖉 ...
↳ENDSETS↲

↳RULES↲
🖉 mise en forme initiale avec positionnement de curseurs
☞➢‹zone›≺⇨‹prm›⇦➲‹sec›➲⇒➢‹zone›≺⇨●→‹prm›←●⇦➲●→‹sec›←●➲/EQUIV/mark1☜🖉[1]

🖉 >>> FIN_ANALYSE <<<

🖉 vérifier si l'analyse comparative est terminée
☞➢‹zone›≺⇨‹a›●→←●‹b›⇦➲‹c›●→←●‹d›➲/EQUIV/‹mark›⇒
➢‹zone›≺⇨‹a›‹b›⇦➲‹c›‹d›➲/EQUIVALENTE☜🖉[2]

🖉 comparer (première / deuxième équivalence) les morceaux de formes
🖉 de gauche à droite et de la surface vers la profondeur

🖉 xE_‹i›⁝ :
☞➢‹zone›≺⇨‹a›●→xE_‹i›⁝‹prm›←●‹b›⇦➲‹sec›➲/EQUIV/mark1⇒
➢‹zone›≺⇨‹a›xE_‹i›⁝●→‹prm›←●‹b›⇦➲‹sec›➲/DIFF/mark2☜🖉[3]
☞➢‹zone›≺⇨‹prm›⇦➲‹a›●→xE_‹i›⁝‹sec›←●‹b›➲/DIFF/mark2⇒
➢‹zone›≺⇨‹prm›⇦➲‹a›xE_‹i›⁝●→‹sec›←●‹b›➲/EQUIV/mark1☜🖉[4]

🖉 ⁄xE_‹i›⁝ :
☞➢‹zone›≺⇨‹a›●→⁄xE_‹i›⁝‹prm›←●‹b›⇦➲‹sec›➲/EQUIV/mark1⇒
➢‹zone›≺⇨‹a›⁄xE_‹i›⁝●→‹prm›←●‹b›⇦➲‹sec›➲/DIFF/mark2☜🖉[5]
☞➢‹zone›≺⇨‹prm›⇦➲‹a›●→⁄xE_‹i›⁝‹sec›←●‹b›➲/DIFF/mark2⇒
➢‹zone›≺⇨‹prm›⇦➲‹a›⁄xE_‹i›⁝●→‹sec›←●‹b›➲/EQUIV/mark1☜🖉[6]

🖉 xQ_‹i›⌊ :
☞➢‹zone›≺⇨‹a›●→xQ_‹i›⌊‹prm›←●‹b›⇦➲‹sec›➲/EQUIV/mark1⇒
➢‹zone›≺⇨‹a›xQ_‹i›⌊●→‹prm›←●‹b›⇦➲‹sec›➲/DIFF/mark2☜🖉[7]
☞➢‹zone›≺⇨‹prm›⇦➲‹a›●→xQ_‹i›⌊‹sec›←●‹b›➲/DIFF/mark2⇒
➢‹zone›≺⇨‹prm›⇦➲‹a›xQ_‹i›⌊●→‹sec›←●‹b›➲/EQUIV/mark1☜🖉[8]

🖉 ⁄xQ_‹i›⌋ :
☞➢‹zone›≺⇨‹a›●→⁄xQ_‹i›⌋‹prm›←●‹b›⇦➲‹sec›➲/EQUIV/mark1⇒
➢‹zone›≺⇨‹a›⁄xQ_‹i›⌋●→‹prm›←●‹b›⇦➲‹sec›➲/DIFF/mark2☜🖉[9]
☞➢‹zone›≺⇨‹prm›⇦➲‹a›●→⁄xQ_‹i›⌋‹sec›←●‹b›➲/DIFF/mark2⇒
➢‹zone›≺⇨‹prm›⇦➲‹a›⁄xQ_‹i›⌋●→‹sec›←●‹b›➲/EQUIV/mark1☜🖉[10]

🖉 xSQ_‹i›⌈ :
☞➢‹zone›≺⇨‹a›●→xSQ_‹i›⌈‹prm›←●‹b›⇦➲‹sec›➲/EQUIV/mark1⇒
➢‹zone›≺⇨‹a›xSQ_‹i›⌈●→‹prm›←●‹b›⇦➲‹sec›➲/DIFF/mark2☜🖉[11]
☞➢‹zone›≺⇨‹prm›⇦➲‹a›●→xSQ_‹i›⌈‹sec›←●‹b›➲/DIFF/mark2⇒
➢‹zone›≺⇨‹prm›⇦➲‹a›xSQ_‹i›⌈●→‹sec›←●‹b›➲/EQUIV/mark1☜🖉[12]

🖉 ⁄xSQ_‹i›⌉ :
☞➢‹zone›≺⇨‹a›●→⁄xSQ_‹i›⌉‹prm›←●‹b›⇦➲‹sec›➲/EQUIV/mark1⇒
➢‹zone›≺⇨‹a›⁄xSQ_‹i›⌉●→‹prm›←●‹b›⇦➲‹sec›➲/DIFF/mark2☜🖉[13]
☞➢‹zone›≺⇨‹prm›⇦➲‹a›●→⁄xSQ_‹i›⌉‹sec›←●‹b›➲/DIFF/mark2⇒

➢‹zone›≺⇨‹prm›⇦⊃‹a›/xSQ_‹i›⌉●→‹sec›←●‹b›⊂/EQUIV/mark1☜🖉[14]

🖉 xCTX_‹i›[
☞➢‹zone›≺⇨‹a›●→xCTX_‹i›[‹prm›←●‹b›⇦⊃‹sec›⊂/EQUIV/mark1⇒
➢‹zone›≺⇨‹a›xCTX_‹i›[●→‹prm›←●‹b›⇦⊃‹sec›⊂/DIFF/mark2☜🖉[15]
☞➢‹zone›≺⇨‹prm›⇦⊃‹a›●→xCTX_‹i›[‹sec›←●‹b›⊂/DIFF/mark2⇒
➢‹zone›≺⇨‹prm›⇦⊃‹a›xCTX_‹i›[●→‹sec›←●‹b›⊂/EQUIV/mark1☜🖉[16]

🖉 /xCTX_‹i›]
☞➢‹zone›≺⇨‹a›●→/xCTX_‹i›]‹prm›←●‹b›⇦⊃‹sec›⊂/EQUIV/mark1⇒
➢‹zone›≺⇨‹a›/xCTX_‹i›]●→‹prm›←●‹b›⇦⊃‹sec›⊂/DIFF/mark2☜🖉[17]
☞➢‹zone›≺⇨‹prm›⇦⊃‹a›●→/xCTX_‹i›]‹sec›←●‹b›⊂/DIFF/mark2⇒
➢‹zone›≺⇨‹prm›⇦⊃‹a›/xCTX_‹i›]●→‹sec›←●‹b›⊂/EQUIV/mark1☜🖉[18]

🖉 xTV_[‹lien›]_‹i›[‹type›/‹p›/ :
☞➢‹zone›≺⇨‹a›●→xTV_[‹lien›]_‹i›[‹type›/‹p›↳IN(vp_vn)⇙›/‹prm›←●‹b›⇦⊃‹sec›⊂/EQUIV/mark1⇒
➢‹zone›≺⇨‹a›xTV_[‹lien›]_‹i›[‹type›/‹p›/●→‹prm›←●‹b›⇦⊃‹sec›⊂/DIFF/[‹p›]/mark2☜🖉[19]
☞➢‹zone›≺⇨‹prm›⇦⊃‹a›●→xTV_[‹lien›]_‹i›[‹type›/‹p›/‹sec›←●‹b›⊂/DIFF/[‹p›]/mark2⇒
➢‹zone›≺⇨‹prm›⇦⊃‹a›xTV_[‹lien›]_‹i›[‹type›/‹p›/●→‹sec›←●‹b›⊂/EQUIV/mark1☜🖉[20]

🖉 xTV_‹i›] :
☞➢‹zone›≺⇨‹a›●→xTV_‹i›]‹prm›←●‹b›⇦⊃‹sec›⊂/EQUIV/mark1⇒
➢‹zone›≺⇨‹a›xTV_‹i›]●→‹prm›←●‹b›⇦⊃‹sec›⊂/DIFF/mark2☜🖉[21]
☞➢‹zone›≺⇨‹prm›⇦⊃‹a›●→xTV_‹i›]‹sec›←●‹b›⊂/DIFF/mark2⇒
➢‹zone›≺⇨‹prm›⇦⊃‹a›xTV_‹i›]●→‹sec›←●‹b›⊂/EQUIV/mark1☜🖉[22]

🖉 xTVQ[‹p›] :
☞➢‹zone›≺⇨‹a›●→xTVQ[‹p›↳IN(vp_vn_fp)⇙›]‹prm›←●‹b›⇦⊃‹sec›⊂/EQUIV/mark1⇒
➢‹zone›≺⇨‹a›xTVQ[‹p›]●→‹prm›←●‹b›⇦⊃‹sec›⊂/DIFF/[‹p›]/mark2☜🖉[23]
☞➢‹zone›≺⇨‹prm›⇦⊃‹a›●→xTVQ[‹p›]‹sec›←●‹b›⊂/DIFF/[‹p›]/mark2⇒
➢‹zone›≺⇨‹prm›⇦⊃‹a›xTVQ[‹p›]●→‹sec›←●‹b›⊂/EQUIV/mark1☜🖉[24]

🖉 xTCAC[‹op›]
☞➢‹zone›≺⇨‹a›●→xTCAC[‹op›↳IN(tcac)⇙›]‹prm›←●‹b›⇦⊃‹sec›⊂/EQUIV/mark1⇒
➢‹zone›≺⇨‹a›xTCAC[‹op›]●→‹prm›←●‹b›⇦⊃‹sec›⊂/DIFF/[‹op›]/mark2☜🖉[25]
☞➢‹zone›≺⇨‹prm›⇦⊃‹a›●→xTCAC[‹op›]‹sec›←●‹b›⊂/DIFF/[‹op›]/mark2⇒
➢‹zone›≺⇨‹prm›⇦⊃‹a›xTCAC[‹op›]●→‹sec›←●‹b›⊂/EQUIV/mark1☜🖉[26]

🖉 xBOP⟩‹op›⟨ :
☞➢‹zone›≺⇨‹a›●→xBOP⟩‹op›↳IN(bop)⇙›⟨‹prm›←●‹b›⇦⊃‹sec›⊂/EQUIV/mark1⇒
➢‹zone›≺⇨‹a›xBOP⟩‹op›⟨●→‹prm›←●‹b›⇦⊃‹sec›⊂/DIFF/[‹op›]/mark2☜🖉[27]
☞➢‹zone›≺⇨‹prm›⇦⊃‹a›●→xBOP⟩‹op›⟨‹sec›←●‹b›⊂/DIFF/[‹op›]/mark2⇒
➢‹zone›≺⇨‹prm›⇦⊃‹a›xBOP⟩‹op›⟨●→‹sec›←●‹b›⊂/EQUIV/mark1☜🖉[28]

🖉 xMOP⟩‹op›⟨ :
☞➢‹zone›≺⇨‹a›●→xMOP⟩‹op›↳IN(mop)⇙›⟨‹prm›←●‹b›⇦⊃‹sec›⊂/EQUIV/mark1⇒
➢‹zone›≺⇨‹a›xMOP⟩‹op›⟨●→‹prm›←●‹b›⇦⊃‹sec›⊂/DIFF/[‹op›]/mark2☜🖉[29]
☞➢‹zone›≺⇨‹prm›⇦⊃‹a›●→xMOP⟩‹op›⟨‹sec›←●‹b›⊂/DIFF/[‹op›]/mark2⇒
➢‹zone›≺⇨‹prm›⇦⊃‹a›xMOP⟩‹op›⟨●→‹sec›←●‹b›⊂/EQUIV/mark1☜🖉[30]

🖉 xBOP_[‹lien›]⟩‹op›⟨ :
☞➢‹zone›≺⇨‹a›●→xBOP_[‹lien›]⟩‹op›↳IN(fp2)⇙›⟨‹prm›←●‹b›⇦⊃‹sec›⊂/EQUIV/mark1⇒
➢‹zone›≺⇨‹a›xBOP_[‹lien›]⟩‹op›⟨●→‹prm›←●‹b›⇦⊃‹sec›⊂/DIFF/[‹op›]/mark2☜🖉[31]
☞➢‹zone›≺⇨‹prm›⇦⊃‹a›●→xBOP_[‹lien›]⟩‹op›⟨‹sec›←●‹b›⊂/DIFF/[‹op›]/mark2⇒
➢‹zone›≺⇨‹prm›⇦⊃‹a›xBOP_[‹lien›]⟩‹op›⟨●→‹sec›←●‹b›⊂/EQUIV/mark1☜🖉[32]

🖉 xMOP_[‹lien›]⟩‹op›⟨ :
☞➢‹zone›≺⇨‹a›●→xMOP_[‹lien›]⟩‹op›↳IN(fp1)⇙›⟨‹prm›←●‹b›⇦⊃‹sec›⊂/EQUIV/mark1⇒
➢‹zone›≺⇨‹a›xMOP_[‹lien›]⟩‹op›⟨●→‹prm›←●‹b›⇦⊃‹sec›⊂/DIFF/[‹op›]/mark2☜🖉[33]
☞➢‹zone›≺⇨‹prm›⇦⊃‹a›●→xMOP_[‹lien›]⟩‹op›⟨‹sec›←●‹b›⊂/DIFF/[‹op›]/mark2⇒
➢‹zone›≺⇨‹prm›⇦⊃‹a›xMOP_[‹lien›]⟩‹op›⟨●→‹sec›←●‹b›⊂/EQUIV/mark1☜🖉[34]

🖉 xPG⟮‹pg›⟯ :
☞➢‹zone›≺⇨‹a›●→xPG⟮‹pg›↳IN(pg)⇙›⟯‹prm›←●‹b›⇦⊃‹sec›⊂/EQUIV/mark1⇒
➢‹zone›≺⇨‹a›xPG⟮‹pg›⟯●→‹prm›←●‹b›⇦⊃‹sec›⊂/DIFF/[‹pg›]/mark2☜🖉[35]
☞➢‹zone›≺⇨‹prm›⇦⊃‹a›●→xPG⟮‹pg›⟯‹sec›←●‹b›⊂/DIFF/[‹pg›]/mark2⇒
➢‹zone›≺⇨‹prm›⇦⊃‹a›xPG⟮‹pg›⟯●→‹sec›←●‹b›⊂/EQUIV/mark1☜🖉[36]

🖉 xPD⟮‹pd›⟯ :
☞➢‹zone›≺⇨‹a›●→xPD⟮‹pd›↳IN(pd)⇙›⟯‹prm›←●‹b›⇦⊃‹sec›⊂/EQUIV/mark1⇒
➢‹zone›≺⇨‹a›xPD⟮‹pd›⟯●→‹prm›←●‹b›⇦⊃‹sec›⊂/DIFF/[‹pd›]/mark2☜🖉[37]
☞➢‹zone›≺⇨‹prm›⇦⊃‹a›●→xPD⟮‹pd›⟯‹sec›←●‹b›⊂/DIFF/[‹pd›]/mark2⇒

➢‹zone›≺⇨‹prm›⇦➲‹a›xPD‹pd›●→‹sec›←●‹b›⊂/EQUIV/mark1[38]

ALLER A : >>> FIN_ANALYSE <<<
IFMATCH_GOTORULE(2) morceau de forme suivante [39]

➢‹zone›≺⇨‹prm›⇦➲‹sec›⊂‹x›/mark‹i›⇒➢‹zone›≺⇨‹prm›⇦➲‹sec›⊂/DIFF/fin[40]
➢‹zone›≺⇨‹prm›⇦➲‹sec›⊂/DIFF/‹x›⇒➢‹zone›≺[<DIFF>]⇨‹prm›⇦➲‹sec›⊂[41]
➢‹zone›≺⇨‹prm›⇦➲‹sec›⊂/EQUIVALENTE⇒➢‹zone›≺[<EQUIV>]⇨‹prm›⇦➲‹sec›⊂[42]
➢‹zone›≺‹res›⇨‹a›●→‹b›←●‹c›⇦➲‹sec›⊂⇒➢‹zone›≺‹res›⇨‹a›‹b›‹c›⇦➲‹sec›⊂[43]
➢‹zone›≺‹res›⇨‹prm›⇦➲‹a›●→‹b›←●‹c›⊂⇒➢‹zone›≺‹res›⇨‹prm›⇦➲‹a›‹b›‹c›⊂[44]

STOP[45]
ENDRULES
ENDGRAM

Traitement de la règle d'inférences de distribution des quantificateurs

GRAM
GRAMNAME⇒DISTRIB
Appliquer la règle de distribution à une et une seule généralisation, générer toutes les possibilités
Format d'entrée :
➢zone1DeFormuleEnEntree≺▷zone2DeTravailSurLaFormule◁➲zone3DeResultat⊂
On sait a priori que toute la formule est bien construite

INCLUDE
DEPX⇒"D:\TheseL\GramXDepouille\GX05Depouille.txt"
ENDINCLUDE

PARAMS
MAXGRAMLOOP⇒5
MAXRULELOOP⇒30
ENDPARAMS

SETS
zero_sy⇒"0"
ENDSETS

RULES

MISE EN FORME INITITALE
****
mise en forme initiale : copier la formule de la première zone dans la deuxième zone (de travail),
préparer la troisième zone (de résultat) et préparer les paramètres
➢‹avant›►‹formule›◄‹apres›≺⇒
➢‹avant›►‹formule›◄‹apres›≺▷◊→‹formule›←◊◁➲⊂
/GEN=[]/COMBITABLE=[]/TV=[x]/verifier_gen[1]

on se branche sur le corpus tampon
SWITCHCORPUS(BUFFER_CORPUS)[2]

VERIFIER QUE L'INSCRIPTION EST UNE GENERALISATION
****
l'inscription doit être une généralisation,
cerner le contenu de la généralisation contenue dans l'expression ou non (deuxième zone)
➢‹zone›≺▷◊→xE_‹i›INTEGER[‹v›/xE_‹i›]←◊◁➲⊂
/GEN=[]/COMBITABLE=[]/TV=[‹tvx›]/verifier_gen⇒
➢‹zone›≺▷xE_‹i›[◊→‹v›←◊/xE_‹i›]◁➲⊂
/GEN=[]/COMBITABLE=[]/TV=[‹tvx›]/verifier_gen[3]
➢‹zone›≺▷‹a›◊→xQ_‹i›INTEGER[‹v›/xSQ_‹i›]←◊‹b›◁➲⊂
/GEN=[]/COMBITABLE=[]/TV=[‹tvx›]/verifier_gen⇒
➢‹zone›≺▷◊→xQ_‹i›[‹v›/xSQ_‹i›]←◊‹b›◁➲⊂
/GEN=[‹i›]/COMBITABLE=[]/TV=[‹tvx›]/faire1_table[4]

CREATION DE LA TABLE COMBINATOIRE
****
cerner le quantificateur dominant dans la deuxième zone
➢‹zone1›≺▷‹a›◊→xQ_‹i›INTEGER[‹xq›/xQ_‹i›]‹xsq›←◊‹z›◁➲⊂
/GEN=[‹gen›]/COMBITABLE=[]/TV=[‹tvx›]/faire1_table⇒
➢‹zone1›≺▷‹a›◊→xQ_‹i›[◙→‹xq›←◙/xQ_‹i›]‹xsq›←◊‹z›◁➲⊂
/GEN=[‹gen›]/COMBITABLE=[]/TV=[‹tvx›]/faire2_table[5]

✎ dans la deuxième zone, compter le nombre de xTVQ dans le quantificateur
✎ qui va nous donner le niveau de la combinatoire
☞➢‹zone1›⋖▷‹a›◙→‹xq›←◙‹z›◁➲⊂/GEN=[‹gen›]/COMBITABLE=[]/TV=[‹tvx›]/faire2_table⇒
➢‹zone1›⋖▷‹a›◙→‹xq›←◙‹z›◁➲⊂
/GEN=[‹gen›]/COMBITABLE=[tot[↳**COUNT(‹xq›,┗xTVQ[‹tv›]┛)**↲]]/TV=[‹tvx›]/faire3_table☜✎[6][122]

✎ créer les tables par positions
☞➢‹zone1›⋖▷‹zone2›◁➲⊂/GEN=[‹gen›]/COMBITABLE=[tot[1]]/TV=[‹tvx›]/faire3_table⇒
➢‹zone1›⋖▷‹zone2›◁➲⊂
/GEN=[‹gen›]/**COMBITABLE=[tot[1]combi[1*1=[({1})]nbc[1]]**/TV=[‹tvx›]/faire4_table☜✎[7]
☞➢‹zone1›⋖▷‹zone2›◁➲⊂/GEN=[‹gen›]/COMBITABLE=[tot[2]]/TV=[‹tvx›]/faire3_table⇒
➢‹zone1›⋖▷‹zone2›◁➲⊂
/GEN=[‹gen›]/**COMBITABLE=[tot[2]combi[2*1=[({1})({2})]1*2=[({1}{2})]]nbc[3]]**
/TV=[‹tvx›]/faire4_table☜✎[8]
☞➢‹zone1›⋖▷‹zone2›◁➲⊂/GEN=[‹gen›]/COMBITABLE=[tot[3]]/TV=[‹tvx›]/faire3_table⇒
➢‹zone1›⋖▷‹zone2›◁➲⊂/GEN=[‹gen›]
/**COMBITABLE=[tot[3]combi[3*1=[({1})({2})({3})]3*2=[({1}{2})({1}{3})({2}{3})]1*3=[({1}{2}{3})]]nbc[7]]**
/TV=[‹tvx›]/faire4_table☜✎[9]
☞➢‹zone1›⋖▷‹zone2›◁➲⊂/GEN=[‹gen›]/COMBITABLE=[tot[4]]/TV=[‹tvx›]/faire3_table⇒
➢‹zone1›⋖▷‹zone2›◁➲⊂/GEN=[‹gen›]
/**COMBITABLE=[tot[4]combi[4*1=[({1})({2})({3}{4})]6*2=[({1}{2})({1}{3})({1}{4})({2}{3})({2}{4})({3}{4})]3*3=**
[({1}{2}{3})({1}{2}{4})({2}{3}{4})]1*4[({1}{2}{3}{4})]]nbc[13]]
/TV=[‹tvx›]/faire4_table☜✎[10]

✎ se donner un compteur/numéro de position de TVQ
☞➢‹zone1›⋖▷‹zone2›◁➲⊂/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/faire4_table⇒
➢‹zone1›⋖▷‹zone2›◁➲⊂/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/faire5_[cpt=1]_table☜✎[11]
✎ numéroter provisoirement les positions de TVQ du quantificateur dominant
☞➢‹zone1›⋖▷‹a›◙→‹b›xTVQ[‹tv›]‹c›←◙‹z›◁➲⊂/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tvx›]/faire5_[cpt=‹cpt›]_table⇒
➢‹zone1›⋖▷‹a›◙→‹b›**xTVQ_[‹cpt›]_**[‹tv›]‹c›←◙‹z›◁➲⊂/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tvx›]/**faire5_[cpt=↳PLUS(‹cpt›,1)↲]**_table☜✎[12]
✎ supprimer le compteur de TVQ
☞➢‹zone1›⋖▷‹zone2›◁➲⊂/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/faire5_[cpt=‹cpt›]_table⇒
➢‹zone1›⋖▷‹zone2›◁➲⊂/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/faire6_table☜✎[13]

✎ reporter les tv en fonction des positions

✎ >>> PROCHAIN_TV <<<

✎ rechercher et extraire le tv correspondant à la position
☞➢‹zone1›⋖▷‹a›◙→‹b›xTVQ_[‹pos›]_[‹extraittv›]‹c›←◙‹z›◁➲⊂/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tvx›]/faire6_table⇒
➢‹zone1›⋖▷‹a›◙→‹b›xTVQ[‹extraittv›]‹c›←◙‹z›◁➲⊂/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/**TV=[‹extraittv›]**/faire7_[pos=‹pos›]_table☜✎[14]
✎ reporter le tv dans la table combinatoire
☞➢‹zone1›⋖▷‹zone2›◁➲⊂/GEN=[‹gen›]
/COMBITABLE=[tot[‹tot›]combi[‹a›{‹pos›}‹b›]nbc[‹nbc›]]/**TV=[‹tv›]**/faire7_[pos=‹pos›]_table⇒
➢‹zone1›⋖▷‹zone2›◁➲⊂/GEN=[‹gen›]
/COMBITABLE=[tot[‹tot›]combi[‹a›**{•‹tv›}**‹b›]nbc[‹nbc›]]/TV=[‹tv›]/faire7_[pos=‹pos›]_table☜✎[15]
✎ réinitialiser
☞➢‹zone1›⋖▷‹zone2›◁➲⊂/GEN=[‹gen›]
/COMBITABLE=[‹combi›]/TV=[‹tvx›]/faire7_[pos=‹pos›]_table⇒
➢‹zone1›⋖▷‹zone2›◁➲⊂/GEN=[‹gen›]/COMBITABLE=[‹combi›]/TV=[‹tvx›]/faire6_table☜✎[16]

✎ aller à >>> PROCHAIN_TV <<<
☞↳IFMATCH_GOTORULE(14)↲☜✎ tv suivant [17]
✎ supprimer les curseurs
☞➢‹zone1›⋖▷‹a›◙→‹b›←◙‹z›◁➲⊂/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/faire6_table⇒
➢‹zone1›⋖▷‹a›‹b›‹z›◁➲⊂/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/init_table☜✎[18]

[122] Dans le chapitre 6. *GOi destinées aux logiques de Leśniewski,* certaines références sont faites à la re-grammaire rapportée ici. Afin de faciliter la lecture nous avons mis en gras certains éléments.

✎ INITIALISER LA COMBITABLE
✎ **********
✎ initialiser les compteurs de la « combitable »
☞➢‹zone1›≮▷‹zone2›◁➲⊂/GEN=[‹gen›]
/COMBITABLE=[tot[‹seg›]combi[‹a›*‹seg›=[‹b›]‹c›]nbc[‹nbc›]]/TV=[‹tvx›]/init_table⇒
➢‹zone1›≮▷‹zone2›◁➲⊂/GEN=[‹gen›]
/COMBITABLE=[tot[‹seg›]combi[**‹a›*‹seg›=■→[‹b›]←■**‹c›]nbc[‹nbc›]]/TV=[‹tvx›]/init1_table☜✎[19]
✎ mettre le premier tv à traiter au chaud et démarquer le tv qui devient le courant
☞➢‹zone1›≮▷‹zone2›◁➲⊂/GEN=[‹gen›]
/COMBITABLE=[tot[‹seg›]combi[‹a›◙→[‹b›({•‹tv›}‹d›)‹e›]←◙‹f›]nbc[‹nbc›]]/TV=[‹tvx›]/init1_table⇒
➢‹zone1›≮▷‹zone2›◁➲⊂/GEN=[‹gen›]
/COMBITABLE=[tot[‹seg›]combi[‹a›**■→[‹b›↕→({‹tv›}‹d›)←↕‹e›]←■**‹f›]nbc[‹nbc›]]/TV=[‹tv›]/ok☜✎[20]

✎ DECOMPOSER LA BICONDITIONNELLE PRIMITIVE
✎ ET PREPARATION DES NOUVEAUX « SOUS » QUANTIFICATEURS
✎ **********

✎ >>> NOUVEAU_SEGMENT <<<

✎ cerner le quantificateur dans la deuxième zone (de travail)
✎ et le copier dans la troisième zone, délimiter le sous-quantificateur
☞➢‹zone›≮▷‹a›◊→xQ_‹i›↳INTEGER⇙›⌊‹xq›/xQ_‹i›⌋‹xsq›←◊‹z›◁➲⊂/‹param›/ok⇒
➢‹zone›≮▷‹a›xQ_‹i›⌊**◊→‹xq›←◊**/xQ_‹i›⌋●→‹xsq›←●‹z›◁**➲◊→xQ_‹i›⌊‹xq›/xQ_‹i›⌋←◊⊂**
/‹param›/mark1/[‹i›]☜✎[21]
✎ dans la deuxième zone, cerner le sous-quantificateur,
✎ délimiter son contenu et copier le tout dans la troisième zone (de résultat)
☞➢‹zone›≮▷‹a›●→xSQ_‹i›⌈‹xSQcontenu›/xSQ_‹i›⌉←●‹z›◁➲‹temp›⊂/‹param›/mark1/[‹i›]⇒
➢‹zone›≮▷‹a›xSQ_‹i›⌈●→‹xSQcontenu›←●/xSQ_‹i›⌉‹z›◁
➲‹temp›xSQ_‹i›⌈●→‹xSQcontenu›←●/xSQ_‹i›⌉⊂/‹param›/mark2☜✎[22]
✎ dans la deuxième zone, vérifier que le sous-quantificateur est
✎ une fonction logique biconditionnelle primitive et la copier dans la troisième zone
☞➢‹zone›≮▷‹a1›●→xE_‹i›↳INTEGER⇙›[xBOP]≡◁‹v›/xE_‹i›]←●‹z›◁➲‹a2›●→‹xsq›←●‹b2›⊂
/‹param›/mark2⇒
➢‹zone›≮▷‹a1›xE_‹i›[xBOP]≡◁●→‹v›←●/xE_‹i›]‹z›◁**➲‹a2›xE_‹i›[xBOP]≡◁●→‹v›←●/xE_‹i›]**
‹b2›⊂/‹param›/mark3☜✎[23]
✎ dans la deuxième zone, cerner les délimiteurs et leur contenu et copier dans la troisième zone
☞➢‹zone›≮▷‹a1›●→xPG[(]‹v›xPD[)]←●‹z›◁➲‹a2›●→‹xsq›←●‹b2›⊂/‹param›/mark3⇒
➢‹zone›≮▷‹a1›xPG[(]**●→‹v›←●**xPD[)]‹z›◁➲‹a2›xPG[(]**●→‹v›←●**xPD[)]‹b2›⊂
/‹param›/mark4☜✎[24]
✎ dans la deuxième zone, cerner la première expression qui est,
✎ elle-même, contenue dans la fonction logique biconditionnelle primitive
✎ du sous-quantificateur traité et la transformer en généralisation dans la troisième zone
☞➢‹zone›≮▷‹a›●→xE_‹i›↳INTEGER⇙›[‹v›/xE_‹i›]‹w›←●‹z›◁
➲‹a2›●→‹xsq›←●‹b2›⊂/GEN=[‹numgen›]/‹param›/mark4⇒
➢‹zone›≮▷‹a›xE_‹i›[**●→‹v›←●**/xE_‹i›]**o→‹w›←o**‹z›◁
➲‹a2›xE_↳PLUS(‹numgen›,201)⇙[xQ_**↳PLUS(‹numgen›,201)⇙**
⌊●→←●/xQ_↳PLUS(‹numgen›,201)⇙⌋xSQ_↳PLUS(‹numgen›,201)⇙⌈xE_‹i›[‹v›/xE_‹i›]
/xSQ_**↳PLUS(‹numgen›,201)⇙⌉**/xE_**↳PLUS(‹numgen›,201)⇙]o→‹w›←o**
‹b2›⊂/GEN=[‹numgen›]/‹param›/mark5☜✎[25]
✎ dans la deuxième zone, vérifier la bonne formation de la deuxième expression,
✎ copie et transformation en généralisation dans la troisième zone
☞➢‹zone›≮▷‹a›○→xE_‹i›↳INTEGER⇙›[‹v›/xE_‹i›]←○‹z›◁
➲‹a2›○→‹exp2›←○‹b2›⊂/GEN=[‹numgen›]/‹param›/mark5⇒
➢‹zone›≮▷‹a›xE_‹i›[○→‹v›←○/xE_‹i›]‹z›◁
➲‹a2›xE_↳PLUS(‹numgen›,202)⇙[xQ_**↳PLUS(‹numgen›,202)⇙⌊o→←o**
/xQ_**↳PLUS(‹numgen›,202)⇙⌋**xSQ_**↳PLUS(‹numgen›,202)⇙⌈**xE_‹i›[‹v›/xE_‹i›]
/xSQ_**↳PLUS(‹numgen›,202)⇙⌉**/xE_**↳PLUS(‹numgen›,202)⇙]**‹b2›⊂
/GEN=[‹numgen›]/‹param›/ok☜✎[26]

✎ PROCESSUS DE DISTRIBUTION
✎ **********

✎ >>> TV_COURANT <<<

✎ TRAITER LA PREMIERE EXPRESSION
✎ initialiser le marqueur
☞➢‹zone1›≮▷‹zone2›◁➲‹zone3›⊂/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/ok⇒
➢‹zone1›≮▷‹zone2›◁➲‹zone3›⊂/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/**prm_distribue**☜✎[27]

✎ >>> TV_PRM_EXP <<<

✎ Est-ce que le tv courant est dans la première expression ?
✎ A partir du tv courant équivalent à TV=[‹tv›], dépister dans la première expression
✎ de la deuxième zone un tv équiforme
☞➢‹zone1›◁▷‹a›●→‹b›xTV_[‹lien↳INTEGER↵›]_‹i↳INTEGER↵›[‹x›/xTV_‹i›]‹c›←●‹z›◁
➲‹zone3›➲/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tv›]/prm_distribue⇒
➢‹zone1›◁▷‹a›●→‹b›■**xTV_tmp_**[‹lien›]_‹i›[‹x›/xTV_‹i›]‹c›←●‹z›◁➲‹zone3›➲
/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tv›]/**tmp=[{‹lien›}{‹x›}]**/prm_cherche1_distribue☜✎[28]
✎ si la règle précédente n'est pas exécutée, c'est qu'il ny a pas le tv cherché, --- quitter la boucle ---
☞➢‹zone1›◁▷‹zone2›◁➲‹zone3›➲/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/prm_distribue⇒
➢‹zone1›◁▷‹zone2›◁➲‹zone3›➲/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/**snd_distribue**☜✎[29]
✎ tester si le tv est équiforme
☞➢‹zone1›◁▷‹zone2›◁➲‹zone3›➲/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[**‹tv›**]/tmp=[{‹lien›}{‹type›/**‹tv›**}]/prm_cherche1_distribue⇒
➢‹zone1›◁▷‹zone2›◁➲‹zone3›➲/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[**‹tv›**]/tmp=[{‹lien›}{‹type›/**‹tv›**}]/**prm_cherche2_distribue**☜✎[30]
✎ le tv n'est pas équiforme, réinitialiser et --- aller en chercher un autre ---
☞➢‹zone1›◁▷‹a›●→‹b›◙xTV_tmp_‹c›←●‹z›◁➲‹zone3›➲/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tvx›]/tmp=[{‹lien›}{‹type›/‹tv›}]/prm_cherche1_distribue⇒
➢‹zone1›◁▷‹a›●→‹b›xTV_tmp_‹c›←●‹z›◁➲‹zone3›➲/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tvx›]/**prm_distribue**☜✎[31]
✎ le tv est équiforme, tester si le lieur est équivalent
☞➢‹zone1›◁▷‹a›●→‹b›◙xTV_tmp_‹c›←●‹z›◁➲‹zone3›➲/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tvx›]/tmp=[{‹gen›}{‹type›/‹tv›}]/prm_cherche2_distribue⇒
➢‹zone1›◁▷‹a›●→‹b›**xTV_fait_**‹c›←●‹z›◁➲‹zone3›➲/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tvx›]/**prm_trouve1_distribue**☜✎[32]
✎ le tv est équiforme, mais le lieur n'est pas équivalent, --- aller en chercher un autre ---
☞➢‹zone1›◁▷‹a›●→‹b›◙xTV_tmp_‹c›←●‹z›◁➲‹zone3›➲/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tvx›]/tmp=[{‹gen›}{‹type›/‹tv›}]/prm_cherche2_distribue⇒
➢‹zone1›◁▷‹a›●→‹b›xTV_tmp_‹c›←●‹z›◁➲‹zone3›➲/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tvx›]/**prm_distribue**☜✎[33]
✎ Si le tv est trouvé, est-ce que le tv est déjà inscrit dans le quantificateur
✎ de la première expression de la troisième zone,
✎ si oui passer à la deuxième expression, --- quitter la boucle ---
☞➢‹zone›◁▷‹zone2›◁➲‹a›●→‹b›xTVQ[‹tv›]‹c›←●‹d›➲/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[**‹tv›**]/prm_trouve1_distribue⇒
➢‹zone›◁▷‹zone2›◁➲‹a›●→‹b›xTVQ[‹tv›]‹c›←●‹d›➲/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tv›]/**snd_distribue**☜✎[34]
✎ si non, aller marquer le TV trouvé dans le quantificateur de la deuxième zone
✎ et l'inscrire dans la troisième et --- quitter la boucle ---
☞➢‹zone›◁▷‹a›◊→‹b›xTVQ[‹tv›]‹c›←◊‹z›◁➲‹a3›●→‹b3›←●‹c3›➲/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tv›]/prm_trouve1_distribue⇒
➢‹zone›◁▷‹a›◊→‹b›xTVQ_fait[‹tv›]‹c›←◊‹z›◁➲‹a3›●→‹b3›**xTVQ[‹tv›]**←●‹c3›➲/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tv›]/**snd_distribue**☜✎[35]

✎ Aller à >>> TV_PRM_EXP <<<
☞↳IFMATCH_GOTORULE(28)↵☜✎ tv suivant [36]
✎ supprimer tous les marqueurs xTV_tmp
☞➢‹zone1›◁▷‹a›xTV_tmp_‹c›➲/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/snd_distribue⇒
➢‹zone1›◁▷‹a›xTV_‹c›➲/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/snd_distribue☜✎[37]

✎ TRAITER LA DEUXIEME EXPRESSION

✎ >>> TV_SND_EXP <<<

✎ parcourir la DEUXIEME expression,
✎ aller chercher s'il existe le même tv dans la DEUXIEME expression qui a comme lien
✎ le quantificateur numéroté : ‹lien›, dépister le xTV courant
☞➢‹zone1›◁▷‹a›○→‹b›xTV_[‹lien↳INTEGER↵›]_‹i↳INTEGER↵›[‹x›/xTV_‹i›]‹c›←○‹z›◁➲‹zone3›
➲
/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tv›]/snd_distribue⇒
➢‹zone1›◁▷‹a›○→‹b›◙xTV_tmp_[‹lien›]_‹i›[‹x›/xTV_‹i›]‹c›←○‹z›◁➲‹zone3›➲
/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tv›]/tmp=[{‹lien›}{‹x›}]/snd_cherche1_distribue☜✎[38]
✎ si la règle précédente n'est pas exécutée, c'est qu'il n'y a pas le tv cherché, --- quitter la boucle ---
☞➢‹zone1›◁▷‹zone2›◁➲‹zone3›➲/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/snd_distribue⇒
➢‹zone1›◁▷‹zone2›◁➲‹zone3›➲/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/fin_distribue☜✎[39]

✎ tester si le tv est équiforme
☞➢‹zone1›⋖▷‹zone2›◁➲‹zone3›⊂/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tv›]/tmp=[{‹lien›}{‹type›/‹tv›}]/snd_cherche1_distribue⇒
➢‹zone1›⋖▷‹zone2›◁➲‹zone3›⊂/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tv›]/tmp=[{‹lien›}{‹type›/‹tv›}]/snd_cherche2_distribue☜✎[40]
✎ le tv n'est pas équiforme, réinitialiser, --- aller en chercher un autre ---
☞➢‹zone1›⋖▷‹a›○→‹b›◙xTV_tmp_‹c›←○‹z›◁➲‹zone3›⊂/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tvx›]/tmp=[{‹lien›}{‹type›/‹tv›}]/snd_cherche1_distribue⇒
➢‹zone1›⋖▷‹a›○→‹b›xTV_tmp_‹c›←○‹z›◁➲‹zone3›⊂/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tvx›]/snd_distribue☜✎[41]
✎ le tv est équiforme, tester si le lieur est équivalent
☞➢‹zone1›⋖▷‹a›○→‹b›◙xTV_tmp_‹c›←○‹z›◁➲‹zone3›⊂/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tvx›]/tmp=[{‹gen›}{‹type›/‹tv›}]/snd_cherche2_distribue⇒
➢‹zone1›⋖▷‹a›○→‹b›xTV_fait_‹c›←○‹z›◁➲‹zone3›⊂/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tvx›]/snd_trouve1_distribue☜✎[42]
✎ le tv est équiforme, mais le lieur n'est pas équivalent, --- aller en chercher un autre ---
☞➢‹zone1›⋖▷‹a›○→‹b›◙xTV_tmp_‹c›←○‹z›◁➲‹zone3›⊂/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tvx›]/tmp=[{‹gen›}{‹type›/‹tv›}]/snd_cherche2_distribue⇒
➢‹zone1›⋖▷‹a›○→‹b›xTV_tmp_‹c›←○‹z›◁➲‹zone3›⊂/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tvx›]/snd_distribue☜✎[43]
✎ si tout est ok, construire la DEUXIEME expression dans la troisième zone
✎ inscrire le tv en TVQ dans le quantificateur de la DEUXIEME expression de la troisième
✎ zone si et seulement s'il n'existe pas déjà
✎ vérifier si le tv existe déjà dans le quantificateur, --- quitter la boucle ---
☞➢‹zone1›⋖▷‹zone2›◁➲‹a3›○→‹b3›xTVQ[‹tv›]‹c3›←○‹d3›⊂/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tv›]/snd_trouve1_distribue⇒
➢‹zone1›⋖▷‹zone2›◁➲‹a3›○→‹b3›xTVQ[‹tv›]‹c3›←○‹d3›⊂/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tv›]/fin_distribue☜✎[44]
✎ si le xTVQ n'existe pas, l'inscrire, --- quitter la boucle ---
☞➢‹zone1›⋖▷‹zone2›◁➲‹a3›○→‹b3›←○‹c3›⊂/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tv›]/snd_trouve1_distribue⇒
➢‹zone1›⋖▷‹zone2›◁➲‹a3›○→‹b3›xTVQ[‹tv›]←○‹c3›⊂/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tv›]/fin_distribue☜✎[45]

✎ Aller à >>> TV_SND_EXP <<<
☞↳IFMATCH_GOTORULE(38)☍☜✎ tv suivant [46]
✎ initialiser le marqueur principal
☞➢‹zone1›⋖▷‹zone2›◁➲‹zone3›⊂/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/‹mark›_distribue⇒➢
‹zone1›⋖▷‹zone2›◁➲‹zone3›⊂/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/mark1_supcurs☜✎[47]
✎ supprimer tous les marqueurs xTV_tmp
☞➢‹zone1›⋖▷‹a›xTV_tmp_‹c›⊂/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/mark1_supcurs⇒
➢‹zone1›⋖▷‹a›xTV_‹c›⊂/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/mark1_supcurs☜✎[48]

✎ un bout de combinaison est terminé pour un tv donné
✎ supprimer le xTVQ dans le quantificateur dominant de la troisième zone, s'il existe
☞➢‹z1›⋖▷‹z2›◁➲‹a›◊→‹b›xTVQ[‹tv›]‹c›←◊‹e›⊂/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tv›]/mark1_supcurs⇒
➢‹z1›⋖▷‹z2›◁➲‹a›◊→‹b›‹c›←◊‹e›⊂/GEN=[‹gen›]
/COMBITABLE=[‹ct›]/TV=[‹tv›]/mark_combitable☜✎[49]
✎ si le quantificateur est déjà vide, ne pas s'en soucier mais ajuster le marqueur
☞➢‹z1›⋖▷‹z2›◁➲‹z3›⊂/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/mark1_supcurs⇒
➢‹z1›⋖▷‹z2›◁➲‹z3›⊂/GEN=[‹gen›]/COMBITABLE=[‹ct›]/TV=[‹tvx›]/mark_combitable☜✎[50]

✎ METTRE LA COMBITABLE ET LES COMPTEURS A JOUR
✎ **********
✎ Y a-t-il encore un tv à traiter pour ce segment ? Tester s'il reste un tv marqué dans la table.
✎ Un segment est entre ().Mettre aussi le nouveau tv au chaud. Et,

✎ Aller à >>> TV_COURANT <<<
☞➢‹zone1›⋖▷‹zone2›◁➲‹zone3›⊂/GEN=[‹gen›]
/COMBITABLE=[tot[‹seg›]combi[‹a›↕→(‹b›{•‹tv›}‹c›)←↕‹d›]nbc[‹nbc›]]/TV=[‹tvx›]/mark_combitable⇒
➢‹zone1›⋖▷‹zone2›◁➲‹zone3›⊂/GEN=[‹gen›]
/COMBITABLE=[tot[‹seg›]combi[‹a›↕→(‹b›{‹tv›}‹c›)←↕‹d›]nbc[‹nbc›]]/TV=[‹tv›]
/ok↳GOTORULE(27)☍☜✎[51]

✎ Il n'y a plus de tv pour ce segment entre (). Chercher un segment suivant et prendre le premier tv.
☞➢‹zone1›⋖▷‹zone2›◁⊃‹zone3›⊂/GEN=[‹gen›]/COMBITABLE=[tot[‹seg›]combi[‹a›
◙→[‹b›↕→(‹c›)←↕‹d›(‹e›{•‹tv›}‹f›)‹g›]←◙‹h›]nbc[‹nbc›]]/TV=[‹tvx›]/mark_combitable⇒
➢‹zone1›⋖▷‹zone2›◁⊃‹zone3›⊂/GEN=[‹gen›]/COMBITABLE=[tot[‹seg›]combi[‹a›
◙→[‹b›(‹c›)‹d›↕→(‹e›{‹tv›}‹f›)←↕‹g›]←◙‹h›]nbc[‹nbc›]]/TV=[‹tv›]/nettoyer☜✎[52]
✎ Il n'y plus de segment pour cette position entre [].
✎ Chercher une autre position, en diminuant le compteur tot
✎ 1) supprimer les curseurs de position et segment
☞➢‹zone1›⋖▷‹zone2›◁⊃‹zone3›⊂/GEN=[‹gen›]
/COMBITABLE=[tot[‹seg›]combi[‹a›◙→‹b›↕→‹c›←↕‹d›←◙‹e›]nbc[‹nbc›]]/TV=[‹tvx›]/mark_combitable
⇒➢‹zone1›⋖▷‹zone2›◁⊃‹zone3›⊂/GEN=[‹gen›]
/COMBITABLE=[tot[‹seg›]combi[‹a›‹b›‹c›‹d›‹e›]nbc[‹nbc›]]/TV=[‹tvx›]/mark1_combitable☜✎[53]
✎ 2) diminuer le compteur tot/position
☞➢‹zone1›⋖▷‹zone2›◁⊃‹zone3›⊂/GEN=[‹gen›]
/COMBITABLE=[tot[‹seg›]‹combi›]/TV=[‹tvx›]/mark1_combitable⇒
➢‹zone1›⋖▷‹zone2›◁⊃‹zone3›⊂/GEN=[‹gen›]
/COMBITABLE=[tot[↳MINUS(‹seg›,1)↲]‹combi›]/TV=[‹tvx›]/mark2_combitable☜✎[54]
✎ 3) tester si fin et sans mémoriser celui-ci
☞➢‹zone1›⋖▷‹zone2›◁⊃‹zone3›⊂/GEN=[‹gen›]
/COMBITABLE=[tot[‹seg↳IN(zero_sy)↲›]‹combi›]/TV=[‹tvx›]/mark2_combitable⇒
➢‹zone1›⋖▷‹zone2›◁⊃‹zone3›⊂/GEN=[‹gen›]
/COMBITABLE=[tot[‹seg›]‹combi›]/TV=[‹tvx›]/nettoyer☜✎[55]
✎ 4) cerner la nouvelle position
☞➢‹zone1›⋖▷‹zone2›◁⊃‹zone3›⊂/GEN=[‹gen›]
/COMBITABLE=[tot[‹seg›]combi[‹a›*‹seg›=[‹b›]‹c›]nbc[‹nbc›]]/TV=[‹tvx›]/mark2_combitable⇒
➢‹zone1›⋖▷‹zone2›◁⊃‹zone3›⊂/GEN=[‹gen›]/COMBITABLE=[tot[‹seg›]combi[‹a›*‹seg›=◙→[‹b›]←◙
‹c›]nbc[‹nbc›]]/TV=[‹tvx›]/mark3_combitable☜✎[56]
✎ 5) cerner un nouveau segment et un nouveau TV
☞➢‹zone1›⋖▷‹zone2›◁⊃‹zone3›⊂/GEN=[‹gen›]/COMBITABLE=[tot[‹seg›]combi[‹a›
◙→[‹b›({•‹tv›}‹d›)‹e›]←◙‹f›]nbc[‹nbc›]]/TV=[‹tvx›]/mark3_combitable⇒
➢‹zone1›⋖▷‹zone2›◁⊃‹zone3›⊂/GEN=[‹gen›]/COMBITABLE=[tot[‹seg›]combi[‹a›
◙→[‹b›↕→({‹tv›}‹d›)←↕‹e›]←◙‹f›]nbc[‹nbc›]]/TV=[‹tv›]/nettoyer☜✎[57]

✎ NETTOYER ET CHANGER DE SEGMENT
✎ **********
✎ supprimer les encapsulations de généralisations vides (où aucun tv n'a été distribué)
✎ dans la PREMIERE et DEUXIEME expression

✎ >>> SUP_QUANT <<<

✎ si un quantificateur n'a pas été distribué et n'a pas pris position dans un sous-quantificateur,
✎ supprimer l'encapsulation de quantification et sous-quantification correspondantes
✎ pour l'encapsulation de la DEUXIEME expression
☞➢‹z1›⋖▷‹z2›◁⊃‹a›xQ_‹i↳INTEGER↲›⌊○→←○‹z›⊂/GEN=[‹gen›]
/COMBITABLE=[‹combi›]/TV=[‹tvx›]/nettoyer⇒
➢‹z1›⋖▷‹z2›◁⊃‹a›◙‹z›⊂/GEN=[‹gen›]/COMBITABLE=[‹combi›]/TV=[‹tvx›]/[‹i›]/nettoyer_m1x☜✎[58]
✎ pour l'encapsulation de la PREMIERE expression
☞➢‹z1›⋖▷‹z2›◁⊃‹a›xQ_‹i↳INTEGER↲›⌊●→←●‹z›⊂/GEN=[‹gen›]
/COMBITABLE=[‹combi›]/TV=[‹tvx›]/nettoyer⇒
➢‹z1›⋖▷‹z2›◁⊃‹a›◙‹z›⊂/GEN=[‹gen›]/COMBITABLE=[‹combi›]/TV=[‹tvx›]/[‹i›]/nettoyer_m1x☜✎[59]
✎ supprimer la fin de l'encapsulation du quantificateur
☞➢‹z1›⋖▷‹z2›◁⊃‹a›◙⁄xQ_‹i↳INTEGER↲›⌋‹z›⊂/GEN=[‹gen›]
/COMBITABLE=[‹combi›]/TV=[‹tvx›]/[‹i›]/nettoyer_m1x⇒
➢‹z1›⋖▷‹z2›◁⊃‹a›◙‹z›⊂/GEN=[‹gen›]/COMBITABLE=[‹combi›]/TV=[‹tvx›]/[‹i›]/nettoyer_m2x☜✎[60]
✎ supprimer le début de l'encapsulation du sous-quantificateur
☞➢‹z1›⋖▷‹z2›◁⊃‹a›◙xSQ_‹i↳INTEGER↲›⌈‹z›⊂/GEN=[‹gen›]
/COMBITABLE=[‹combi›]/TV=[‹tvx›]/[‹i›]/nettoyer_m2x⇒
➢‹z1›⋖▷‹z2›◁⊃‹a›◙‹z›⊂/GEN=[‹gen›]/COMBITABLE=[‹combi›]/TV=[‹tvx›]/[‹i›]/nettoyer_m3x☜✎[61]
✎ supprimer la fin de l'encapsulation du sous-quantificateur
☞➢‹z1›⋖▷‹z2›◁⊃‹a›◙‹v›⁄xSQ_‹i↳INTEGER↲›⌉‹z›⊂/GEN=[‹gen›]
/COMBITABLE=[‹combi›]/TV=[‹tvx›]/[‹i›]/nettoyer_m3x⇒
➢‹z1›⋖▷‹z2›◁⊃‹a›◙→‹v›←◙‹z›⊂/GEN=[‹gen›]
/COMBITABLE=[‹combi›]/TV=[‹tvx›]/[‹i›]/nettoyer_m4x☜✎[62]
✎ supprimer le début de l'encapsulation de l'expression de la généralisation
☞➢‹z1›⋖▷‹z2›◁⊃‹a›xE_‹j↳INTEGER↲›◙→‹v›←◙‹z›⊂/GEN=[‹gen›]
/COMBITABLE=[‹combi›]/TV=[‹tvx›]/[‹i›]/nettoyer_m4x⇒
➢‹z1›⋖▷‹z2›◁⊃‹a›◙→‹v›←◙‹z›⊂/GEN=[‹gen›]
/COMBITABLE=[‹combi›]/TV=[‹tvx›]/[‹i›]/nettoyer_m5x☜✎[63]

✎ supprimer la fin de l'encapsulation de l'expression de la généralisation
☞➢‹z1›≺▷‹z2›◁➲‹a›◙→‹v›←◙/xE_‹i›[‹z›⊂/GEN=[‹gen›]
/COMBITABLE=[‹combi›]/TV=[‹tvx›]/[‹i›]/nettoyer_m5x⇒
➢‹z1›≺▷‹z2›◁➲‹a›‹v›‹z›⊂/GEN=[‹gen›]
/COMBITABLE=[‹combi›]/TV=[‹tvx›]/nettoyer☜✎[64]

✎ Aller à >>> SUP_QUANT <<<
☞↳IFMATCH_GOTORULE(58)⏎☜✎ quantificateur / généralisation suivant [65]

✎ si le quantificateur DOMINANT est vide supprimer la généralisation
☞➢‹z1›≺▷‹z2›◁➲◊→xQ_‹i›↳INTEGER⏎›⌊/xQ_‹i›⌋←◊‹z›⊂/GEN=[‹gen›]
/COMBITABLE=[‹combi›]/TV=[‹tvx›]/nettoyer⇒
➢‹z1›≺▷‹z2›◁➲‹z›⊂/GEN=[‹gen›]/COMBITABLE=[‹combi›]/TV=[‹tvx›]/nettoyer_m2x☜✎[66]
✎ supprimer l'encapsulation du sous-quantificateur DOMINANT
☞➢‹z1›≺▷‹z2›◁➲xSQ_‹i›⌈‹v›/xSQ_‹i›⌉⊂/GEN=[‹gen›]
/COMBITABLE=[‹combi›]/TV=[‹tvx›]/nettoyer_m2x⇒
➢‹z1›≺▷‹z2›◁➲‹v›⊂/GEN=[‹gen›]/COMBITABLE=[‹combi›]/TV=[‹tvx›]/nettoyer☜✎[67]

✎ supprimer les curseurs dans la troisième zone
☞➢‹z1›≺▷‹z2›◁➲‹a›◊→‹b›←◊‹c›⊂/‹param›/nettoyer⇒
➢‹z1›≺▷‹z2›◁➲‹a›‹b›‹c›⊂/‹param›/nettoyer☜✎[68]
☞➢‹z1›≺▷‹z2›◁➲‹a›●→‹b›←●‹c›⊂/‹param›/nettoyer⇒
➢‹z1›≺▷‹z2›◁➲‹a›‹b›‹c›⊂/‹param›/nettoyer☜✎[69]
☞➢‹z1›≺▷‹z2›◁➲‹a›○→‹b›←○‹c›⊂/‹param›/nettoyer⇒
➢‹z1›≺▷‹z2›◁➲‹a›‹b›‹c›⊂/‹param›/nettoyer☜✎[70]

✎ MEMORISER LE RESULTAT
✎ **********
✎ mémoriser le résultat dans buffer
☞➢‹z1›≺▷‹z2›◁➲‹z3›⊂/GEN=[‹gen›]
/COMBITABLE=[tot[‹seg›]combi[‹combi›]nbc[‹nbc›]]/TV=[‹tvx›]/nettoyer⇒
➢‹z1›≺▷‹z2›◁➲[‹nbc›]‹z3›⊂/GEN=[‹gen›]
/COMBITABLE=[tot[‹seg›]combi[‹combi›]nbc[‹nbc›]]/TV=[‹tvx›]/mem1_cpt☜✎[71]
☞‹x›/mem1_cpt⇒‹x›/mem☜✎[72]
☞↳APPENDBUFFERLINE⏎☜✎[73]
✎ reprendre le résultat et sélectionner la zone 3 de résultats
☞➢‹z1›≺▷‹z2›◁➲‹z3›⊂/‹param›/mem⇒➲‹z3›⊂☜✎[74]
✎ formater la formule pour envoi à DEPX
☞➲‹z3›⊂⇒➢▶‹z3›◀≺☜✎[75]
☞↳EXECUTEGRAM(DEPX)⏎☜✎[76]
✎ afficher le résultat
☞➢▶‹z3›◀≺▷‹res›◁⇒▷‹res›◁☜✎[77]
☞↳APPENDBUFFERLINE⏎☜✎[78]
✎ reprendre la ligne de résultat analysée dans buffer
☞↳CORPUSLAST⏎☜✎[79]
☞↳CORPUSPREVIOUS⏎☜✎[80]
☞↳GETCORPUSLINE⏎☜✎[81]
☞↳CONCAT(↳RM_RESULT_ZONE⏎,"",↳TEMPORARY_ZONE⏎)⏎☜✎[82]
✎☞‹a›⇒‹a›↳STOP⏎☜✎[]
☞↳CORPUSLAST⏎☜✎[83]

✎ diminuer le compteur de combinaisons
☞➢‹z1›≺▷‹z2›◁➲‹z3›⊂/GEN=[‹gen›]/COMBITABLE=[tot[‹seg›]combi[‹combi›]
nbc[‹nbc›]]/TV=[‹tvx›]/mem⇒
➢‹z1›≺▷‹z2›◁➲‹z3›⊂/GEN=[‹gen›]/COMBITABLE=[tot[‹seg›]combi[‹combi›]
nbc[↳MINUS(‹nbc›,1)⏎]]/TV=[‹tvx›]/mark1_combitable☜✎[84]
✎ tester le compteur de combinaisons
☞➢‹z1›≺▷‹z2›◁➲‹z3›⊂/GEN=[‹gen›]/COMBITABLE=[tot[‹seg›]combi[‹combi›]
nbc[‹nbc↳IN(zero_sy)⏎›]]/TV=[‹tvx›]/mark1_combitable⇒
➢‹z1›≺▷‹z2›◁➲‹z3›⊂/GEN=[‹gen›]/COMBITABLE=[tot[‹seg›]combi[‹combi›]
nbc[‹nbc›]]/TV=[‹tvx›]/sortie☜✎[85]

✎ recommencer un segment
✎ initialiser les marqueurs "fait" dans la zone 2
☞➢‹z1›≺▷‹a›_fait‹z›◁➲‹z3›⊂/GEN=[‹gen›]/COMBITABLE=[‹combi›]/TV=[‹tvx›]/mark1_combitable⇒
➢‹z1›≺▷‹a›‹z›◁➲‹z3›⊂/GEN=[‹gen›]/COMBITABLE=[‹combi›]/TV=[‹tvx›]/mark1_combitable☜✎[86]

✎ repositionner les curseurs de quantificateur dans la zone 2
☞➢‹z1›≺▷‹a›◊→‹xq›←◊‹z›◁➲‹z3›⊂/GEN=[‹gen›]
/COMBITABLE=[‹combi›]/TV=[‹tvx›]/mark1_combitable⇒
➢‹z1›≺▷◊→‹a›‹xq›‹z›←◊◁➲‹z3›⊂/GEN=[‹gen›]
/COMBITABLE=[‹combi›]/TV=[‹tvx›]/mark2_combitable☜✎[87]
✎ supprimer les curseurs restants dans la zone 2
☞➢‹z1›≺▷‹a›●→‹v›←●‹z›◁➲‹z3›⊂/GEN=[‹gen›]
/COMBITABLE=[‹combi›]/TV=[‹tvx›]/mark2_combitable⇒➢‹z1›≺
▷‹a›‹v›‹z›◁➲‹z3›⊂/GEN=[‹gen›]/COMBITABLE=[‹combi›]/TV=[‹tvx›]/mark3_combitable☜✎[88]
✎ supprimer les curseurs restants dans la zone 2
☞➢‹z1›≺▷‹a›○→‹v›←○‹z›◁➲‹z3›⊂/GEN=[‹gen›]
/COMBITABLE=[‹combi›]/TV=[‹tvx›]/mark3_combitable⇒➢‹z1›≺
▷‹a›‹v›‹z›◁➲‹z3›⊂/GEN=[‹gen›]/COMBITABLE=[‹combi›]/TV=[‹tvx›]/mark4_combitable☜✎[89]
✎ nettoyer la troisième zone, et

✎ Aller à >>> NOUVEAU_SEGMENT <<<
☞➢‹z1›≺▷‹z2›◁➲‹z3›⊂/GEN=[‹gen›]/COMBITABLE=[‹combi›]/TV=[‹tvx›]/mark4_combitable⇒
➢‹z1›≺▷‹z2›◁➲⊂/GEN=[‹gen›]/COMBITABLE=[‹combi›]/TV=[‹tvx›]/ok↳GOTORULE(21)↲☜✎[90]

✎ BOUCLER SUR LES RESULTATS
✎ **
☞↳SWITCHCORPUS(↳BUFFER_CORPUS↲)↲☜✎[91]
☞↳CORPUSFIRST↲☜✎[92]

✎ >>> PARCOURS_BUFFER <<<

☞↳GETCORPUSLINE↲☜✎[93]
✎ concaténation de TEMPORARY_ZONE (ligne de BUFFER_CORPUS)
✎ dans RM_RESULT_ZONE, qui peut retourner aussi {<EOF>}
☞↳CONCAT(↳RM_RESULT_ZONE↲,"",↳TEMPORARY_ZONE↲)↲☜✎[94]
✎ tester eof après concaténation, si eof : TEMPORARY_ZONE retourne {<EOF>} , et

✎ Aller à >>> FIN <<<
☞‹a›{<EOF>}⇒‹a›/EOF↳GOTORULE(100)↲☜✎[95]

✎ afficher la ligne trouvée
☞➢‹a›▶‹z1›◀‹b›≺▷‹z2›◁➲‹z3›⊂/‹param›/mem⇒DET:➢‹a›▶‹z1›◀‹b›≺➲‹z3›⊂☜✎[96]
☞▷[‹x›]‹z›◁⇒DEP:▷[‹x›]‹z›◁☜✎[97]
☞↳CORPUSNEXT↲☜✎[98]

✎ Aller à >>> PARCOURS_BUFFER <<<
☞↳IFMATCH_GOTORULE(93)↲☜✎ [99]

✎ >>> FIN <<<

☞INUTILE⇒NE RIEN FAIRE☜✎[100]

☞↳STOP↲☜✎[101]
↳ENDRULES↲
↳ENDGRAM↲

Traitement de la règle de substitution (protothétique)

GO_i de test de la grammaire de la règle d'inférences de substitution

↳GRAM↲
☞GRAMNAME⇒TESTSUBSTITU☜

↳INCLUDE↲
☞SUBSTITU⇒"D:\TheseL\GramXSubstitut\GXDCASubstitut.txt"☜
☞DEPX⇒"D:\TheseL\GramXDepouille\GX05Depouille.txt"☜
↳ENDINCLUDE↲

↳PARAMS↲
☞MAXGRAMLOOP⇒5☜
☞MAXRULELOOP⇒30☜
↳ENDPARAMS↲

↳RULES↲

✎ MISE EN FORME INITIALE
✎ **********
✎ aller chercher les 3 informations A, B et E dans INPUT_CORPUS
☞↳SWITCHCORPUS(↳INPUT_CORPUS↲)↲☜✎[1]
☞↳CORPUSFIRST↲☜✎[2]
☞↳GETCORPUSLINE↲☜✎ thèse B inscrite dans la biblithèque (zone 2a) [3]
☞↳CONCAT(↳RM_RESULT_ZONE↲,"",↳TEMPORARY_ZONE↲)↲☜✎[4]
☞‹a›{<EOF>}⇒ERREUR : ‹a›/EOF trouvé !↳STOP↲☜✎[5]
☞xQ_‹i›⌊‹v›⁄xSQ_‹i›⌉⇒➢▶◀≺➲▶xQ_‹i›⌊‹v›⁄xSQ_‹i›⌉◀▷◁➲//☜✎[6]
☞↳CORPUSNEXT↲☜✎[7]
☞↳GETCORPUSLINE↲☜✎ variable cible à substituer [8]
☞↳CONCAT(↳RM_RESULT_ZONE↲,↳RM_RESULT_ZONE↲,↳TEMPORARY_ZONE↲)↲☜✎[9]
☞‹a›{<EOF>}⇒ERREUR : ‹a›/EOF trouvé !↳STOP↲☜✎[10]
☞➢▶◀≺➲▶‹v›◀▷◁➲//‹vas›⇒➢▶◀≺➲▶‹v›◀‹vas›▷◁➲//☜✎[11]
☞↳CORPUSNEXT↲☜✎[12]
☞↳GETCORPUSLINE↲☜✎ expression E à substituer (zone2b) [13]
☞↳CONCAT(↳RM_RESULT_ZONE↲,↳RM_RESULT_ZONE↲,↳TEMPORARY_ZONE↲)↲☜✎[14]
☞‹a›{<EOF>}⇒ERREUR : ‹a›/EOF trouvé !↳STOP↲☜✎[15]
☞➢▶◀≺➲▶‹v›◀‹vas›▷◁➲//‹eas›⇒➢▶◀≺➲▶‹v›◀‹vas›▷‹eas›◁➲☜✎[15]
☞↳EXECUTEGRAM(SUBSTITU)↲☜✎[16]
☞➢▶◀≺➲▶‹z2a›◀‹vas›▷‹z2b›◁⁘‹substitution›⁖➲/subs_reussie⇒
➢➲≥‹z2a›≤‹vas›▷‹z2b›◁⁘‹substitution›⁖➲▶‹substitution›◀≺☜✎[17]
☞↳EXECUTEGRAM(DEPX)↲☜✎[18]
☞➢➲≥‹expB›≤‹vas›▷‹expE›◁⁘‹substitution›⁖➲▶‹X›◀≺▷‹subdep›◁⇒
➢➲≥‹expB›≤‹vas›▷‹expE›◁⁘‹subdep›⁖➲▶‹expB›◀≺☜✎[19]
☞↳EXECUTEGRAM(DEPX)↲☜✎[20]
☞➢➲≥‹expB›≤‹vas›▷‹expE›◁⁘‹subdep›⁖➲▶‹X›◀≺▷‹expBdep›◁⇒
➢➲≥‹expBdep›≤‹vas›▷‹expE›◁⁘‹subdep›⁖➲▶‹expE›◀≺☜✎[20]
☞↳EXECUTEGRAM(DEPX)↲☜✎[21]
☞➢➲≥‹depB›≤‹vas›▷‹expE›◁⁘‹subdep›⁖➲▶‹X›◀≺▷‹depE›◁⇒
➢➲‹depB›➲≥‹vas›≤▷‹depE›◁⁘‹subdep›⁖▶◀≺☜✎[22]

☞↳STOP↲☜✎[]
↳ENDRULES↲
↳ENDGRAM↲

GO_i de la règle d'inférences de substitution

↳GRAM↲
☞GRAMNAME⇒SUBSTITU☜
✎ Appliquer E, la règle de substitution à B pour obtenir A
✎ A une thèse B inscrite dans la bibliothèque du système,
✎ substituer une expression E aux termes équiformes
✎ Format d'entrée et de traitement :
✎ ➢▶‹zone1Entree›◀≺
✎ ➲▶‹zone2aTheseB›◀‹termAsubstituer›▷‹zone2bExpressionE›◁⁘‹zone2cTheseBsubstituee›⁖➲
✎ On sait a priori que toutes les formule sont bien construites

↳PARAMS↲
☞MAXGRAMLOOP⇒5☜
☞MAXRULELOOP⇒10☜
↳ENDPARAMS↲

↳SETS↲
✎ Variables
☞vp⇒"p","q","r","s"☜✎ variables propositionnelles
☞vns⇒"A","B","C","D"☜✎ variables de nom singulier
☞vng⇒"a","b"☜✎ variables de nom générique
☞vn⇒vns,vng☜✎ les symboles de variable de nom
☞vp_vn⇒vp,vns,vng☜✎ tous les symboles de variable

✎ Fonctions Propositionnelles
☞fp2⇒"f","g","ƨ"☜✎ fonctions propositionnelles à deux places
☞fp1⇒"h","ɲ"☜✎ fonctions propositionnelles à une place
☞fp⇒fp2,fp1☜✎ toutes les fonctions propositionnelles
☞vp_vn_fp⇒vp,vns,vng,fp☜✎ tous les symboles de termes qui peuvent être quantifiés

```
✎ Constantes, termes constants
☞bic⇒"≡"☜✎ biconditionnelle
☞cp0⇒"V","F","T"☜✎ constantes propositionnelles sans contexte
☞tc1⇒"~","¬","α","!","♭","↓","𝓔","𝒜"☜✎ constantes unaires
☞tc2⇒"μ","Ц","ε","∧","∨","⊃","𝓮"☜✎ constantes avec contexte binaire
☞tcx⇒"Δ","δ"☜✎ constantes n_aires
☞tcmc⇒"τ","ω"☜✎ constantes multi-contextes
☞tcac⇒tc1,tc2,tcx,tcmc☜✎ les constantes avec contexte(s) sans BIC
☞mop⇒tc1,fp1☜✎ opérateurs unaires
☞bop⇒tc2,bic,fp2☜✎ opérateurs binaires
☞xop⇒tcx,tcmc☜✎ opérateurs complexes, multi-contextes
☞tcn1⇒"!"☜✎ termes constants unaires pour la catégorie des noms
☞tcn2⇒"ε"☜✎ termes constants binaires pour la catégorie des noms
☞tcn⇒tcn1,tcn2☜✎ termes constants pour la catégorie des noms
☞op⇒bop,mop☜✎ termes constants unaires et binaires

✎ Parenthèses gauches et délimiteurs gauches de contexte
☞pgA⇒"("☜
☞pgB⇒"["☜
☞pgC⇒"{"☜
☞pgD⇒"<"☜
☞pgE⇒"⟮"☜
☞pgF⇒"⟨"☜
✎ ...
☞pg⇒pgA,pgB,pgC,pgD,pgE,pgF☜✎ ...

✎ Parenthèses droites et délimiteurs droits de contexte
☞pdA⇒")"☜
☞pdB⇒"]"☜
☞pdC⇒"}"☜
☞pdD⇒">"☜
☞pdE⇒"⟯"☜
☞pdF⇒"⟩"☜
✎ ...
☞pd⇒pdA,pdB,pdC,pdD,pdE,pdF☜✎ ...

☞zero_sy⇒"0"☜
☞moinsun_sy⇒"-1"☜
☞optype_sy⇒"B","M"☜
↳ENDSETS↲

↳RULES↲
✎ MISE EN FORME INITITALE
✎ *********************************
✎ mise en forme initiale : copier la formule de la zone 2a dans la zone 2c (de résultat)
☞➢‹zone1›≺➲►‹zone2aTheseB›◄‹termAsubstituer›▷‹zone2bExpressionE›◁⊂⇒
➢‹zone1›≺➲►‹zone2aTheseB›◄‹termAsubstituer›
▷‹zone2bExpressionE›◁ ❖‹zone2aTheseB›❖ ⊂/verifier_Bgen☜✎[1]

✎ VERIFIER QUE B A SUBSTITUER EST UNE GENERALISATION
✎ ***************************************************************************
✎ Condition 1. de la règle
☞➢‹zone1›≺➲►xQ_‹i›↳INTEGER↲›⌊‹v›⁄xSQ_‹i›⌉◄‹tas›▷‹zone2b›◁ ❖‹zone2c›❖ ⊂/verifier_Bgen⇒
➢‹zone1›≺➲►xQ_‹i›⌊‹v›⁄xSQ_‹i›⌉◄‹tas›▷‹zone2b›◁ ❖‹zone2c›❖ ⊂/est_Bgen☜✎[2]
✎ Condition 1. pas respectée
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ⊂/verifier_Bgen⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ⊂/[erreur: B n'est pas une généralisation]☜✎[3]

✎ VERIFIER DANS B QUE LE TERME/EXPRESSON A SUBSTITUER EST
✎ MENTIONNE DANS LA GENERALISATON
✎ ********************************************************************************************
☞➢‹z1›≺➲►‹a›xTVQ⟦‹tas›⟧‹z›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ⊂/est_Bgen⇒
➢‹z1›≺➲►‹a›xTVQ⟦‹tas›⟧‹z›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ⊂/tas_mention☜✎[4]
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ⊂/est_Bgen⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ⊂
/[erreur: le terme cible ‹tas› à substituer n'est pas dans B]☜✎[5]
```

✎ ANALYSER LE TERME VARIABLE CIBLE A SUBSTITUER
✎ **********
✎ Condition 6. de la règle de substitution
✎ est une variable propositionnelle, un terme variable (vp)
☞➢‹z1›≺➲▶‹z2a›◀‹tas›IN(vp)‹z2b›◁ ❖‹z2c›❖ ➲/tas_mention⇒
➢‹z1›≺➲▶‹z2a›◀‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[(TV)(VP)(S)]/cible_est_correct✎[6]
✎ est une variable de nom, un terme varaible (vn)
☞➢‹z1›≺➲▶‹z2a›◀‹tas›IN(vn)‹z2b›◁ ❖‹z2c›❖ ➲/tas_mention⇒
➢‹z1›≺➲▶‹z2a›◀‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[(TV)(VN)(N)]/cible_est_correct✎[7]
✎ est pas permis
☞➢‹z1›≺➲▶‹z2a›◀‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/est_Bgen⇒
➢‹z1›≺➲▶‹z2a›◀‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲
/[erreur: le terme ‹tas› substituable n'est pas un terme variable]✎[8]

✎ ANALYSER L'EXPRESSION E, SOURCE, A SUBSTITUER
✎ **********
✎ Condition 7. de la règle de substitution
✎ est une variable propositionnelle, un terme variable (vp)
☞➢‹z1›≺➲▶‹z2a›◀‹ts›▷‹es›IN(vp)‹z2c›❖ ➲/cible=[‹cible›]/cible_est_correct⇒
➢‹z1›≺➲▶‹z2a›◀‹ts›▷‹es›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[(TV)(VP)(S)]/src_est_correct✎[9]
✎ est une variable de nom, un terme variable (vn)
☞➢‹z1›≺➲▶‹z2a›◀‹ts›▷‹es›IN(vn)‹z2c›❖ ➲/cible=[‹cible›]/cible_est_correct⇒
➢‹z1›≺➲▶‹z2a›◀‹ts›▷‹es›◁ ❖‹z2c›❖ ➲/cible=‹cible›]/src=[(TV)(VN)(N)]/src_est_correct✎[10]
✎ est une constante propositionnelle (cp0)
☞➢‹z1›≺➲▶‹z2a›◀‹ts›▷‹es›IN(cp0)‹z2c›❖ ➲/cible=[‹cible›]/cible_est_correct⇒
➢‹z1›≺➲▶‹z2a›◀‹ts›▷‹es›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[(CP)(CP)(S)]/src_est_correct✎[11]
✎ est une fonction propositionnelle (fp), catégorie S
☞➢‹z1›≺➲▶‹z2a›◀‹ts›▷xE_‹i›[x‹optype›IN(optype_sy)‹z2c›OP_[‹lien›]‹op›IN(fp)‹z2c›
xPG[‹pg›]‹v›xPD[‹pd›]/xE_‹i›]◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/cible_est_correct⇒
➢‹z1›≺➲▶‹z2a›◀‹ts›▷xE_‹i›[x‹optype›OP_[‹lien›]‹op›
xPG[‹pg›]‹v›xPD[‹pd›]/xE_‹i›]◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[(EXP)(FP)(S)]/src_est_correct✎[12]
✎ est une expression (tcn), catégorie N
☞➢‹z1›≺➲▶‹z2a›◀‹ts›▷xE_‹i›[x‹optype›IN(optype_sy)‹z2c›OP‹op›IN(tcn)‹z2c›xPG[‹pg›]‹v›
xPD[‹pd›]/xE_‹i›]◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/cible_est_correct⇒
➢‹z1›≺➲▶‹z2a›◀‹ts›▷xE_‹i›[x‹optype›OP‹op›xPG[‹pg›]
‹v›xPD[‹pd›]/xE_‹i›]◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[(EXP)(‹optype›OP)(N)]/src_est_correct✎[13]
✎ est une expression (op), catégorie
S☞➢‹z1›≺➲▶‹z2a›◀‹ts›▷xE_‹i›[x‹optype›IN(optype_sy)‹z2c›OP‹op›IN(bop)‹z2c›xPG[‹pg›]
‹v›xPD[‹pd›]/xE_‹i›]◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/cible_est_correct⇒
➢‹z1›≺➲▶‹z2a›◀‹ts›▷xE_‹i›[x‹optype›OP‹op›xPG[‹pg›]
‹v›xPD[‹pd›]/xE_‹i›]◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[(EXP)(‹optype›OP)(S)]/src_est_correct✎[14]
✎ pas permis
☞➢‹z1›≺➲▶‹z2a›◀‹ts›▷‹es›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/cible_est_correct⇒
➢‹z1›≺➲▶‹z2a›◀‹ts›▷‹es›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]
/[erreur: catégorie de l'expression source pas conforme]✎[15]

✎ VERIFIER LES CATEGORIES DE L'EXPRESSION SOURCE ET DU TERME VARIABLE CIBLE
✎ **********
✎ Condition 3. de la règle de substitution
✎ vérifier l'équivalence entre les catégories
☞➢‹z1›≺➲▶‹z2a›◀‹tas›▷‹eas›◁ ❖‹z2c›❖ ➲
/cible=[(‹typec›)(‹tvx›)(‹cat›)]/src=[(‹typee›)(‹exp›)(‹cat›)]/src_est_correct⇒
➢‹z1›≺➲▶‹z2a›◀‹tas›▷‹eas›◁ ❖‹z2c›❖ ➲
/cible=[(‹typec›)(‹tvx›)(‹cat›)]/src=[(‹typee›)(‹exp›)(‹cat›)]/cat_est_correct✎[16]
✎ les catégorie ne sont pas équivalent
☞➢‹z1›≺➲▶‹z2a›◀‹tas›▷‹eas›◁ ❖‹z2c›❖ ➲/cible=[(‹typec›)(‹tvx›)(‹catc›)]
/src=[(‹typee›)(‹exp›)(‹cats›)]/src_est_correct⇒
➢‹z1›≺➲▶‹z2a›◀‹tas›▷‹eas›◁ ❖‹z2c›❖ ➲/cible=[(‹typec›)(‹tvx›)(‹catc›)]
/src=[(‹typee›)(‹exp›)(‹cats›)]/[erreur: les catégories ne sont pas équivalentes]✎[17]

✎ L'EXPRESSION E LIBRE POUR LA VARIABLE A SUBSTITUER
✎ **********
✎ Condition 4. de la règle de substitution ; traduit en re-règles de grammaire :
✎ - aucune mention de E ne doit être faite dans B
✎ - cela doit être testé pour les VP et VN et pas pour les constantes propositionnelles
✎ - pour les expressions, les tv de l'expression ne doivent pas être mentionnées (équiformes) dans le sous-quantificateur de B
✎ - les fp de E doivent aussi être libre dans B

✎ POUR UN TV SEUL A SUBSTITUER, VERIFIER QUE LE TV DE E NE SOIT DANS B
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹tas›◁⁘‹z2c›❖ ⊂/cible=[(‹typec›)(‹tvx›)(‹cat›)]
/src=[(TV)(‹exp›)(‹cat›)]/cat_est_correct⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹tas›◁⁘‹z2c›❖ ⊂/cible=[(‹typec›)(‹tvx›)(‹cat›)]
/src=[(TV)(‹exp›)(‹cat›)]/[erreur: ‹tas› de E pas libre dans B]☜✎[18]

✎ POUR TOUS LES TV A SUBSTITUER D'UNE EXPRESSION E,
✎ VERIFIER QU'AUCUN TV DE E NE SOIT DANS B

✎ >>> PROCHAIN_TV_E <<<

✎ pour E de type expression, parcourir B et vérifier mention ne soit fait des TV de E,
✎ soit vérifier les xTV de E dans le sous-quantificateur de B
✎ extraire pour commencer un TV de E
✎ pour le definiens : L dans le métaterme, pour le definiendum : D dans le métaterme
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xTV_[‹lien›↳INTEGER⇳]_‹i›↳INTEGER⇳[‹LD›/‹tv›/xTV_‹i›]‹z›◁⁘
‹z2c›❖ ⊂/cible=[(‹typec›)(‹tvx›)(‹cat›)]/src=[(EXP)(‹exp›)(‹cat›)]/cat_est_correct⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xTV_mark_[‹lien›]_‹i›[‹LD›/‹tv›/xTV_‹i›]‹z›◁⁘
‹z2c›❖ ⊂/cible=[(‹typec›)(‹tvx›)(‹cat›)]/src=[(EXP)(‹exp›)(‹cat›)]/tvE=[‹tv›]/tvE_libre☜✎[19]

✎ >>> PROCHAIN_TV_B <<<

✎ parcourir B avec le tv de E
☞➢‹z1›≺➲►‹a›xTV_[‹lien›↳INTEGER⇳]_‹i›↳INTEGER⇳[‹LD›/‹tv›/xTV_‹i›]‹z›◄‹tas›
▷‹eas›◁⁘‹z2c›❖ ⊂/cible=[‹cible›]/src=[‹src›]/tvE=[‹tvx›]/tvE_libre⇒
➢‹z1›≺➲►‹a›xTV_mark_[‹lien›]_‹i›[‹LD›/‹tv›/xTV_‹i›]‹z›◄‹tas
›▷‹eas›◁⁘‹z2c›❖ ⊂/cible=[‹cible›]/src=[‹src›]/tvE=[‹tvx›]/tvB=[‹tv›]/expE2_libre☜✎[20]
✎ tester si les tv sont équiformes, si oui erreur
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁⁘‹z2c›❖ ⊂/cible=[‹cible›]/src=[‹src›]/tvE=[‹tv›]/tvB=[‹tv›]/expE2_libre
⇒➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁⁘‹z2c›❖ ⊂/cible=[‹cible›]/src=[‹src›]/tvE=[‹tv›]/tvB=[‹tv›]
/[erreur: ‹tv› de E pas libre dans B]☜✎[21]
✎ initialiser le marqueur
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁⁘‹z2c›❖ ⊂/cible=[‹cible›]/src=[‹src›]/tvE=[‹tvx›]/tvB=[‹tvy›]
/expE2_libre⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁⁘‹z2c›❖ ⊂/cible=[‹cible›]/src=[‹src›]/tvE=[‹tvx›]/tvE_libre☜✎[22]

✎ Aller à >>> PROCHAIN_TV_B <<<
☞↳IFMATCH_GOTORULE(20)⇳☜✎ tv de B suivant [23]
✎ supprimer les mark de B
☞➢‹z1›≺➲►‹a›xTV_mark_‹z›◄‹tas›▷‹z2b›◁⁘‹z2c›❖ ⊂/cible=[‹cible›]/src=[‹src›]/tvE=[‹tvx›]/tvE_libre⇒
➢‹z1›≺➲►‹a›xTV_‹z›◄‹tas›▷‹z2b›◁⁘‹z2c›❖ ⊂/cible=[‹cible›]/src=[‹src›]/tvE=[‹tvx›]/tvE_libre☜✎[24]
✎ supprimer la mémorisation de tvE
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁⁘‹z2c›❖ ⊂/cible=[‹cible›]/src=[‹src›]/tvE=[‹tvx›]/tvE_libre⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁⁘‹z2c›❖ ⊂/cible=[‹cible›]/src=[‹src›]/cat_est_correct☜✎[25]

✎ Aller à >>> PROCHAIN_TV_E <<<
☞↳IFMATCH_GOTORULE(19)⇳☜✎ tv de B suivant [26]
✎ supprimer les mark de E
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xTV_mark_‹z›◁⁘‹z2c›❖ ⊂/cible=[‹cible›]/src=[‹src›]/cat_est_correct⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xTV_‹z›◁⁘‹z2c›❖ ⊂/cible=[‹cible›]/src=[‹src›]/cat_est_correct☜✎[27]

✎ POUR TOUS LES FP A SUBSTITUER D'UNE EXPRESSION E,
✎ VERIFIER QU'AUCUN FP DE E NE SOIT DANS B

✎ >>> PROCHAIN_FP_E <<<

✎ pour E de type expression FP, parcourir B et vérifier qu'aucune mention ne soit fait des FP de E,
✎ soit : vérifier les FP de E dans le sous-quantificateur de B
✎ extraire pour commencer un FP de E
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›x‹optype↳IN(optype_sy)⇳›OP_[‹lien›]⁞‹fp↳IN(fp)⇳›⁞‹z›◁⁘‹z2c›❖ ⊂
/cible=[(‹typec›)(‹tvx›)(‹cat›)]/src=[(‹typee›)(‹exp›)(‹cat›)]/cat_est_correct⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›x‹optype›OP_mark_[‹lien›]⁞‹fp›⁞‹z›◁⁘‹z2c›❖ ⊂
/cible=[(‹typec›)(‹tvx›)(‹cat›)]/src=[(‹typee›)(‹exp›)(‹cat›)]/fpE=[‹fp›]/fpE_libre☜✎[28]

✎ >>> PROCHAIN_FP_B <<<

✎ parcourir B avec le fp de E
☞➢‹z1›≺➲►‹a›x‹optype↳IN(optype_sy)⇳›OP_[‹lien›]⁞‹fp›⁞‹z›◄‹tas›▷‹eas›◁⁘‹z2c›❖ ⊂
/cible=[‹cible›]/src=[‹src›]/fpE=[‹fpx›]/fpE_libre⇒
➢‹z1›≺➲►‹a›x‹optype›OP_mark_[‹lien›]⁞‹fp›⁞‹z›◄‹tas›▷‹eas›◁⁘‹z2c›❖ ⊂
/cible=[‹cible›]/src=[‹src›]/fpE=[‹fpx›]/fpB=[‹fp›]/fpE2_libre☜✎[29]

tester si les fp sont équiformes, si oui erreur
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/fpE=[‹fp›]/tvB=[‹fp›]/fpE2_libre⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/fpE=[‹fp›]/tvB=[‹fp›]
/[erreur: ‹fp› de E pas libre dans B]☜[30]
initialiser le marqueur
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/fpE=[‹fpx›]/fpB=[‹fpy›]
/fpE2_libre⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/fpE=[‹fpx›]/fpE_libre☜[31]

Aller à >>> PROCHAIN_FP_B <<<
☞↳IFMATCH_GOTORULE(29)☍☜ fp de B suivant [32]
supprimer les mark de B
☞➢‹z1›≺➲►‹a›_mark_‹z›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/fpE=[‹fpx›]/fpE_libre⇒
➢‹z1›≺➲►‹a›_‹z›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/fpE=[‹fpx›]/fpE_libre☜[33]
supprimer la mémorisation de fpE
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/fpE=[‹tvx›]/fpE_libre⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/cat_est_correct☜[34]

Aller à >>> PROCHAIN_FP_E <<<
☞↳IFMATCH_GOTORULE(28)☍☜ fp de B suivant [35]
supprimer les mark de E
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›_mark_‹z›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/cat_est_correct⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›_‹z›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/cat_est_correct☜[36]
initialiser marqeur
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/cat_est_correct⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/libre_correct☜[37]

RENUMEROTER LES ENCAPSULATIONS SI E EST UNE EXPRESSION COMPLEXE
****
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xQ_‹i›⌊‹v›⁄xQ_‹i›⌋xSQ_‹i›⌈‹w›⁄xSQ_‹i›⌉‹z›◁ ❖‹z2c›❖ ➲
/cible=[‹cible›]/src=[(EXP)(‹tvs›)(‹cats›)]/libre_correct⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xQ◙_↳PLUS(‹i›,100)☍⌊‹v›⁄xQ_↳PLUS(‹i›,100)☍⌋xSQ◙_↳PLUS(‹i›,100)
☍⌈‹w›⁄xSQ_↳PLUS(‹i›,100)☍⌉‹z›◁ ❖‹z2c›❖ ➲
/cible=[‹cible›]/src=[(EXP)(‹tvs›)(‹cats›)]/libre_correct☜[38]
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xE_‹i›[‹v›⁄xE_‹i›]‹z›◁ ❖‹z2c›❖ ➲
/cible=[‹cible›]/src=[(EXP)(‹tvs›)(‹cats›)]/libre_correct⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xE◙_↳PLUS(‹i›,100)☍[‹v›⁄xE_↳PLUS(‹i›,100)☍]‹z›◁ ❖‹z2c›❖ ➲
/cible=[‹cible›]/src=[(EXP)(‹tvs›)(‹cats›)]/libre_correct☜[39]
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xTV_[‹lien›]_‹i›[‹v›⁄xTV_‹i›]‹z›
◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[(EXP)(‹tvs›)(‹cats›)]/libre_correct⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xTV◙_[‹lien›]_↳PLUS(‹i›,100)☍[‹v›⁄xTV_↳PLUS(‹i›,100)☍]‹z›
◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[(EXP)(‹tvs›)(‹cats›)]/libre_correct☜[40]
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›◙‹z›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[(EXP)(‹tvs›)(‹cats›)]/libre_correct⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›‹z›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[(EXP)(‹tvs›)(‹cats›)]/libre_correct☜[41]

SUBSTITUTION
****
En particulier, condition 2. de la règle de substitution
Plusieurs cas peuvent se présenter :
1) simple substitution d'un TV dans le quantificateur et le sous-quantificateur
2) substitution d'un CP (contante propositionnelle) seulement dans le sous-quantificateur MAIS
  avec suppression dans le quantificateur du terme à substitué
  il faudra supprimer la quantification si elle est vide
3) substitution d'une expression complexe constituée d'un MOP, d'un BOP ou d'un FP
  et où les tv et les fp devront être inscrits dans le quantificateur (pas les cp)
CAS 1. TV PAS TV, substitution d'un TV par un autre TV, sélection du cas dans les paramètres
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[(TV)(‹tvc›)(‹catc›)]/src=[(TV)(‹tvs›)(‹cats›)]
/libre_correct⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[(TV)(‹tvc›)(‹catc›)]/src=[(TV)(‹tvs›)(‹cats›)]
/subs1_tv_tv☜[42]
aller chercher dans le quantificateur de la zone 2c, le tas
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›xTVQ{‹tas›}‹z›❖ ➲/cible=[‹cible›]/src=[‹src›]/subs1_tv_tv⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›xTVQ{‹z2b›}‹z›❖ ➲/cible=[‹cible›]/src=[‹src›]/subs2_tv_tv☜[43]
parcourir le sous-quantificateur pour aller remplacer le tas par l'expression E à substituer
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›xTV_[‹lien›]_‹i›[‹LD›/‹tas›⁄xTV_‹i›]‹z›❖ ➲
/cible=[‹cible›]/src=[‹src›]/subs2_tv_tv⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›xTV_[‹lien›]_‹i›[‹LD›/‹z2b›⁄xTV_‹i›]‹z›❖ ➲
/cible=[‹cible›]/src=[‹src›]/subs2_tv_tv☜[44]

✎ nettoyer la formule
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹substitution›❖ ⊂/cible=[‹cible›]/src=[‹src›]/subs2_tv_tv⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹substitution›❖ ⊂/subs_reussie☜✎[45]

✎ CAS 2. TV PAR CP, substitution d'un TV par une CP, sélection du cas dans les paramètres
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ⊂/cible=[‹cible›]/src=[(CP)(‹tvs›)(‹cats›)]/libre_correct⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ⊂/cible=[‹cible›]/src=[(CP)(‹tvs›)(‹cats›)]/subs1_tv_cp☜✎[46]
✎ aller chercher dans le quantificateur de la zone 2c, le tas et le supprimer
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›xTVQ[‹tas›]‹z›❖ ⊂/cible=[‹cible›]/src=[‹src›]/subs1_tv_cp⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›‹z›❖ ⊂/cible=[‹cible›]/src=[‹src›]/subs2_tv_cp☜✎[47]
✎ parcourir le sous-quantificateur pour aller remplacer le tas par l'expression E à substituer, attention
✎ substituer aussi le type dans le métaterme
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›xTV_[‹lien›]_‹i›[L/‹tas›/xTV_‹i›]‹z›❖ ⊂
/cible=[‹cible›]/src=[‹src›]/subs2_tv_cp⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›xTV_[‹lien›]_‹i›[C/‹z2b›/xTV_‹i›]‹z›❖ ⊂
/cible=[‹cible›]/src=[‹src›]/subs2_tv_cp☜✎[48]
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›xTV_[‹lien›]_‹i›[D/‹tas›/xTV_‹i›]‹z›❖ ⊂
/cible=[‹cible›]/src=[‹src›]/subs2_tv_cp⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›xTV_[‹lien›]_‹i›[D/‹z2b›/xTV_‹i›]‹z›❖ ⊂
/cible=[‹cible›]/src=[‹src›]/subs2_tv_cp☜✎[49]
✎ supprimer la généralisation si le quantificateur est vide
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖xQ_‹i›⌊/xQ_‹i›⌋xSQ_‹i›⌈‹v›/xSQ_‹i›⌉❖ ⊂
/cible=[‹cible›]/src=[‹src›]/subs2_tv_cp⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹v›❖ ⊂/cible=[‹cible›]/src=[‹src›]/subs3_tv_cp☜✎[50]
✎ re-initialiser le marqueur
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ⊂/cible=[‹cible›]/src=[‹src›]/subs3_tv_cp⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ⊂/subs_reussie☜✎[51]

✎ CAS 3. TV PAR EXP, substitution d'un TV par une EXP de type BOP ou MOP,
✎ sélection du cas dans les paramètres
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ⊂/cible=[‹cible›]/src=[(EXP)(‹tvs›)(‹cats›)]/libre_correct⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ⊂/cible=[‹cible›]/src=[(EXP)(‹tvs›)(‹cats›)]/subs1_tv_exp☜✎[52]
✎ aller chercher dans le quantificateur de la zone 2c, le tas et le supprimer
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›xTVQ[‹tas›]‹z›❖ ⊂/cible=[‹cible›]/src=[‹src›]/subs1_tv_exp⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›‹z›❖ ⊂/cible=[‹cible›]/src=[‹src›]/subs2_tv_exp☜✎[53]

✎ CAS 3. INSCRIPTION D'UN TV DANS LE QUANTIFICATEUR

✎ >>> TV_E <<<

✎ parcourir les tv (pas les cp) de E dans la zone 2b pour les inscrire dans le quantificateur dominant
✎ cerner et extraire les tv
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xTV_[‹lien›]_‹i›[‹DL›/‹tv›/xTV_‹i›]‹z›◁ ❖‹z2c›❖ ⊂
/cible=[‹cible›]/src=[‹src›]/subs2_tv_exp⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›●→xTV_mark_[‹lien›]_‹i›[‹DL›/‹tv›/xTV_‹i›]←●‹z›◁ ❖‹z2c›❖ ⊂
/cible=[‹cible›]/src=[‹src›]/tv=[‹tv›]/subs3_tv_exp☜✎[54]
✎ inscrire le tv dans le quantificateur s'il n'est pas déjà inscrit
✎ Le xTVQ existe-t-il déjà dans la quantificateur ? Mémoriser le numéro du quantificateur
✎ pour mettre à jour le lien du tv
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖xQ_‹i›⌊‹a›xTVQ[‹tv›]‹b›/xQ_‹i›⌋‹z›❖ ⊂/cible=[‹cible›]
/src=[‹src›]/tv=[‹tv›]/subs3_tv_exp⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖xQ_‹i›⌊‹a›xTVQ[‹tv›]‹b›/xQ_‹i›⌋‹z›❖ ⊂/cible=[‹cible›]
/src=[‹src›]/lien=[‹i›]/subs4_tv_exp☜✎[55]
✎ inscrire le tv dans quantificateur et mémoriser le numéro du quantificateur
✎ pour mettre à jour le lien du tv
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖xQ_‹i›⌊‹v›/xQ_‹i›⌋‹z›❖ ⊂/cible=[‹cible›]
/src=[‹src›]/tv=[‹tv›]/subs3_tv_exp⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖xQ_‹i›⌊‹v›xTVQ[‹tv›]/xQ_‹i›⌋‹z›❖ ⊂/cible=[‹cible›]
/src=[‹src›]/lien=[‹i›]/subs4_tv_exp☜✎[56]
✎ mettre à jour le numéro du lieur du tv
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›●→xTV_mark_[‹ancienlien›]_‹i›[‹DL›/‹tv›/xTV_‹i›]←●‹z›◁ ❖‹z2c›❖ ⊂
/cible=[‹cible›]/src=[‹src›]/lien=[‹nouveaulien›]/subs4_tv_exp⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xTV_mark_[‹nouveaulien›]_‹i›[‹DL›/‹tv›/xTV_‹i›]‹z›◁ ❖‹z2c›❖ ⊂
/cible=[‹cible›]/src=[‹src›]/subs5_tv_exp☜✎[57]
✎ reinitialiser le marqueur
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ⊂/cible=[‹cible›]/src=[‹src›]/subs5_tv_exp⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ⊂/cible=[‹cible›]/src=[‹src›]/subs2_tv_exp☜✎[58]

Aller à >>> TV_E <<<
☞↳IFMATCH_GOTORULE(54)↲☜ tv de E suivant [59]
supprimer les mark
☞➤‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xTV_mark_‹z›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/subs2_tv_exp⇒
➤‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xTV_‹z›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/subs2_tv_exp☜[60]
re-initialiser le marqueur
☞➤‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/subs2_tv_exp⇒
➤‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/libre_correct☜[61]

CAS 3. INSCRIPTION D'UN FP DANS LE QUANTIFICATEUR

>>> FP_E <<<

Y a-t-il dans E des FP qui doivent être quantifiées ?
cerner et extraire les BOB ou MOP constitués de FP
☞➤‹z1›≺➲►‹z2a›◄‹tas›▷‹a›x‹optype↳IN(optype_sy)↲›OP_[‹lien›]‡‹fp↳IN(fp)↲›⦚‹z›◁ ❖‹z2c›❖ ➲
/cible=[‹cible›]/src=[‹src›]/libre_correct⇒
➤‹z1›≺➲►‹z2a›◄‹tas›▷‹a›●→x‹optype›OP_mark_[‹lien›]‡‹fp›⦚←●‹z›◁ ❖‹z2c›❖ ➲
/cible=[‹cible›]/src=[‹src›]/fp=[‹fp›]/subs1_tv_exp☜[62]

La fp existe-t-elle déjà dans la quantificateur ?
Mémoriser le numéro du quantificateur pour mettre à jour le lien du fp
☞➤‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖xQ_‹i›⌊‹a›xTVQ[‹fp›]‹b›/xQ_‹i›⌋‹z›❖ ➲
/cible=[‹cible›]/src=[‹src›]/fp=[‹fp›]/subs1_tv_exp⇒
➤‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖xQ_‹i›⌊‹a›xTVQ[‹fp›]‹b›/xQ_‹i›⌋‹z›❖ ➲
/cible=[‹cible›]/src=[‹src›]/lien=[‹i›]/subs2_tv_exp☜[63]

Inscrire la fp dans quantificateur et mémoriser le numéro du quantificateur
pour mettre à jour le lien du fp
☞➤‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖xQ_‹i↳INTEGER↲›⌊‹v›/xQ_‹i›⌋‹z›❖ ➲
/cible=[‹cible›]/src=[‹src›]/fp=[‹fp›]/subs1_tv_exp⇒
➤‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖xQ_‹i›⌊‹v›xTVQ[‹fp›]/xQ_‹i›⌋‹z›❖ ➲
/cible=[‹cible›]/src=[‹src›]/lien=[‹i›]/subs2_tv_exp☜[64]

mettre à jour le numéro du lieur du tv●→x‹optype›OP_mark_[‹ancienlien›]‡‹fp›⦚←●
☞➤‹z1›≺➲►‹z2a›◄‹tas›▷‹a›●→x‹optype›OP_mark_[‹ancienlien›]‡‹fp›⦚←●‹z›◁ ❖‹z2c›❖ ➲
/cible=[‹cible›]/src=[‹src›]/lien=[‹nouveaulien›]/subs2_tv_exp⇒
➤‹z1›≺➲►‹z2a›◄‹tas›▷‹a›x‹optype›OP_mark_[‹nouveaulien›]‡‹fp›⦚‹z›◁ ❖‹z2c›❖ ➲
/cible=[‹cible›]/src=[‹src›]/subs3_tv_exp☜[65]

reinitialiser le marqueur
☞➤‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/subs3_tv_exp⇒
➤‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/libre_correct☜[66]

Aller à >>> FP_E <<<
☞↳IFMATCH_GOTORULE(62)↲☜ fp de E suivant [67]
supprimer les marques
☞➤‹z1›≺➲►‹z2a›◄‹tas›▷‹a›_mark_‹z›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/libre_correct⇒
➤‹z1›≺➲►‹z2a›◄‹tas›▷‹a›_‹z›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/libre_correct☜[68]

INSERER E DANS B
se donner un compteur numéroteur d'encapsulation pour le substitut
☞➤‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/libre_correct⇒
➤‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/cnt=[0]/libre1_correct☜[69]

parcourir le sous-quantificateur pour aller remplacer le tas par l'expression E à substituer

>>> TV_SUIVANT <<<

cerner un xTV et l'extraire
☞➤‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›xTV_[‹lien›]_‹i↳INTEGER↲›[‹DL›/‹tv›/xTV_‹i›]‹z›❖ ➲
/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]/libre1_correct⇒
➤‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›◙→xTV●_[‹lien›]_‹i›[‹DL›/‹tv›/xTV_‹i›]←◙‹z›❖ ➲
/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]/tv=[‹tv›]/mark1☜[70]
plus de xTV
☞➤‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]/libre1_correct⇒
➤‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]/libre2_correct☜[71]
si tas et tv identique alors substituer
☞➤‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›◙→‹v›←◙‹z›❖ ➲/cible=[‹cible›]
/src=[‹src›]/cnt=[‹cnt›]/tv=[‹tas›]/mark1⇒
➤‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›○○→‹z2b›←○○‹z›❖ ➲/cible=[‹cible›]
/src=[‹src›]/cnt=[↳PLUS(‹cnt›,1)↲]/renum1☜[72]

✎ renuméroter le substitut
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xQ_‹i›⌊‹v›⁄xQ_‹i›⌋xSQ_‹i›⌈‹w›⁄xSQ_‹i›⌉‹z›◁ ❖‹z2c›❖ ➲
/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]/renum1⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xQ◙_↳PLUS(‹i›,‹cnt›)↲⌊‹v›⁄xQ_↳PLUS(‹i›,‹cnt›)↲⌋
xSQ◙_↳PLUS(‹i›,‹cnt›)↲⌈‹w›⁄xSQ_↳PLUS(‹i›,‹cnt›)↲⌉‹z›◁ ❖‹z2c›❖ ➲
/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]/renum1☜✎[73]
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xE_‹i›[‹v›⁄xE_‹i›]‹z›◁ ❖‹z2c›❖ ➲
/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]/renum1⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xE◙_↳PLUS(‹i›,‹cnt›)↲[‹v›⁄xE_↳PLUS(‹i›,‹cnt›)↲]‹z›◁ ❖‹z2c›❖ ➲
/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]/renum1☜✎[74]
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xTV_[‹lien›]_‹i›[‹v›⁄xTV_‹i›]‹z›
◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]/renum1⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›xTV◙_[‹lien›]_↳PLUS(‹i›,‹cnt›)↲[‹v›⁄xTV_↳PLUS(‹i›,‹cnt›)↲]‹z›
◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]/renum1☜✎[75]
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›◙‹z›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]/renum1⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹a›‹z›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]/renum1☜✎[76]
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]/renum1⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]/libre1_correct☜✎[77]

✎ si pas identique
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›◙→‹v›←◙‹z›❖ ➲/cible=[‹cible›]
/src=[‹src›]/cnt=[‹cnt›]/tv=[‹tvx›]/mark1⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›‹v›‹z›❖ ➲/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]/libre1_correct☜✎[78]

✎ Aller à >>> TV_SUIVANT <<<
☞↳IFMATCH_GOTORULE(70)↲☜✎ tv suivant [79]
✎ supprimer les ●
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›●‹z›❖ ➲/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]
/libre‹mark›_correct⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›‹z›❖ ➲/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]
/libre‹mark›_correct☜✎[80]
✎ supprimer les ○○→ ←○○
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›○○→‹v›←○○‹z›❖ ➲/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]
/libre‹mark›_correct⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹a›‹v›‹z›❖ ➲/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]
/libre‹mark›_correct☜✎[81]
✎ re-initialiser le marqueur et nettoyer
☞➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/cible=[‹cible›]/src=[‹src›]/cnt=[‹cnt›]/libre‹mark›_correct⇒
➢‹z1›≺➲►‹z2a›◄‹tas›▷‹z2b›◁ ❖‹z2c›❖ ➲/subs_reussie☜✎[82]

☞↳STOP↲☜✎[83]
↳ENDRULES↲
↳ENDGRAM↲

Traitement de la directive d'extensionalité

↳GRAM↲
☞GRAMNAME⇒EXTENS☜
✎ "L'inscription A est une thèse résultante de la directive d'extensionalité relativement à ce que
✎ contient actuellement le système, si et seulement si : ..."
✎ Il s'agit principalement de vérifier des éléments de bonne formation et d'aller dans
✎ la bibliothèque du système pour vérifier les catégories syntaxico-sémantiques

↳PARAMS↲
☞MAXGRAMLOOP⇒5☜
☞MAXRULELOOP⇒10☜
↳ENDPARAMS↲

↳SETS↲
✎ Fonctions Propositionnelles
☞fp2⇒"f","g","Ψ"☜✎ fonctions propositionnelles à deux places
☞fp1⇒"h","α","β"☜✎ fonctions propositionnelles à une place
☞fp⇒fp2,fp1☜✎ toutes les fonctions propositionnelles
☞optype_sy⇒"M","B"☜✎ opérateur unaire ou binaire
↳ENDSETS↲

↳RULES↲
MISE EN FORME INITITALE
****
mise en forme initiale : copier la formule A de la zone 1b dans la zone 2
‹zone1a›►‹zone1b›◄‹zone1c›≺⇒
‹zone1a›►‹zone1b›◄‹zone1c›≺➲‹zone1b›➲/cond1_ko[1]
ATTENTION : SUPPRESSION DE LA ZONE 1b POUR LES TESTS
‹zone1a›►‹zone1b›◄‹zone1c›≺⇒‹zone1a›►◄‹zone1c›≺➲‹zone1b›➲/cond1_ko[1]

1. L'INSCRIPTION A EST UNE GENERALISATION
****
renuméroter l'encapsulation de la généralisation A
‹zone1›≺➲xQ_‹i›↳INTEGER↲›⌊‹v›/xQ_‹i›⌋xSQ_‹i›⌈‹w›/xSQ_‹i›⌉➲/cond1_ko⇒
‹zone1›≺➲xQ_0⌊▷‹v›◁/xQ_0⌋xSQ_0⌈►‹w›◄/xSQ_0⌉➲/lien=[‹i›]/cond1.1_ko[2]
renuméroter les liens à xQ_0
‹zone1›≺➲‹a›[‹i›]‹z›➲/lien=[‹i›]/cond1.1_ko⇒‹zone1›≺➲‹a›[0]‹z›➲/lien=[‹i›]/cond1.1_ko[3]
‹zone1›≺➲‹z›➲/lien=[‹i›]/cond1.1_ko⇒‹zone1›≺➲‹z›➲/cond2_ko[4]

2. LE SOUS-QUANTIFICATEUR DE A EST UNE EXPRESSION BICONDITIONNELLE PRIMITIVE
****
... et renuméroter l'encapsulation
‹z1›≺➲‹a›►xE_‹i›[xBOP≡⦃xPG[(]‹v›xPD[)]]/xE_‹i›]◄‹z›➲/cond2_ko⇒
‹z1›≺➲‹a›xE_1[xBOP≡⦃xPG[(]►‹v›◄xPD[)]]/xE_1]‹z›➲/cond3_ko[5]

3. LA PREMIERE ET LA DEUXIEME EQUIVALENCE DE SQ DE A EST UNE GENERALISATION
****
première équivalence, renuméroter l'encapsulation
‹z1›≺➲‹a›►xQ_‹i›↳INTEGER↲›⌊‹v›/xQ_‹i›⌋xSQ_‹i›⌈‹w›/xSQ_‹i›⌉‹b›◄‹z›➲/cond3_ko⇒
‹z1›≺➲‹a›►●→xQ_1⌊‹v›/xQ_1⌋xSQ_1⌈‹w›/xSQ_1⌉←●‹b›◄‹z›➲/lien=[‹i›]/cond3.1_ko[6]
renuméroter les liens à xQ_1
‹zone1›≺➲‹a›●→‹b›[‹i›]‹c›←●‹z›➲/lien=[‹i›]/cond3.1_ko⇒‹zone1›≺➲‹a›●→‹b›[1]‹c›←●‹z›➲/li
en=[‹i›]/cond3.1_ko[7]
‹zone1›≺➲‹z›➲/lien=[‹i›]/cond3.1_ko⇒‹zone1›≺➲‹z›➲/cond3.2_ko[8]
deuxième équivalence, renuméroter l'encapsulation
‹z1›≺➲‹a›►●→‹b›←●xQ_‹i›↳INTEGER↲›⌊‹v›/xQ_‹i›⌋xSQ_‹i›⌈‹w›/xSQ_‹i›⌉‹c›◄
‹z›➲/cond3.2_ko⇒
‹z1›≺➲‹a›►●→‹b›←●○→xQ_2⌊‹v›/xQ_2⌋xSQ_2⌈‹w›/xSQ_2⌉←○‹c›◄
‹z›➲/lien=[‹i›]/cond3.3_ko[9]
renuméroter les liens à xQ_2
‹zone1›≺➲‹a›○→‹b›[‹i›]‹c›←○‹z›➲/lien=[‹i›]/cond3.3_ko⇒
‹zone1›≺➲‹a›○→‹b›[2]‹c›←○‹z›➲/lien=[‹i›]/cond3.3_ko[10]
‹zone1›≺➲‹z›➲/lien=[‹i›]/cond3.3_ko⇒‹zone1›≺➲‹z›➲/cond4_ko[11]

4. DANS LE Q DE A ON TROUVE DES VARIABLES DE TYPE FP DE MEME CATEGORIE
SYNTAXICO-SEMANTIQUE ET CONNUES DU SYSTEME
****
on ne devrait trouver que 2 xTVQ dans le contenu de Q de A
‹zone1›≺➲‹a›▷xTVQ[‹v1›]xTVQ[‹v2›]◁‹z›➲/cond4_ko⇒
‹zone1›≺➲‹a›▷xTVQ[‹v1›]xTVQ[‹v2›]◁‹z›➲/V1=[[‹v1›]]V2=[[‹v2›]]/cond4.1_ko[12]

EXECUTER DES REGLES / UNE RE-GRAMMAIRE QUI LIVRENT LA CATEGORIE DE V1 ET V2
pour les tests, simuler et forcer une réponse
‹zone1›≺➲‹zone2›➲/V1=[[‹v1›]]V2=[[‹v2›]]/cond4.1_ko⇒
‹zone1›≺➲‹zone2›➲/V1=[[‹v1›]][S/S/<-->]V2=[[‹v2›]][S/S/<-->]/cond4.2_ko[13]
‹zone1›≺➲‹zone2›➲/V1=[[‹v1›]][‹cat›]V2=[[‹v2›]][‹cat›]/cond4.2_ko⇒
‹zone1›≺➲‹zone2›➲/cond5_ko[14]

5. LE SQ DE LA PREMIERE ET DEUXIEME EQUIVALENCE DE A EST UNE BICONDITIONNELLE
****
première équivalence, cerner le contenu du SQ, renuméroter l'encapsulation de l'expression
‹z1›≺➲‹a›●→‹b›xSQ_1⌈xE_‹i›↳INTEGER↲›[xBOP≡⦃xPG[(]‹v›xPD[)]]/xE_‹i›]/xSQ_1⌉←●‹z›➲
/cond5_ko⇒
‹z1›≺➲‹a›●→‹b›xSQ_1⌈xE_2[xBOP≡⦃xPG[(]◙→‹v›←◙xPD[)]]/xE_2]/xSQ_1⌉←●‹z›➲
/cond5.1_ko[15]
deuxième équivalence
‹z1›≺➲‹a›○→‹b›xSQ_2⌈xE_‹i›[xBOP≡⦃xPG[(]‹v›xPD[)]]/xE_‹i›]/xSQ_2⌉←○‹z›➲/cond5.1_ko⇒
‹z1›≺➲‹a›○→‹b›xSQ_2⌈xE_3[xBOP≡⦃xPG[(]◙→‹v›←◙xPD[)]]/xE_3]/xSQ_2⌉←○‹z›➲
/cond6_ko[16]


6. LES DEUX ARGUMENTS DES BIC DES SQ DE LA PREMIERE
ET DEUXIEME EQUIVALENCE SONT DES FP
première équivalence, premier argument (F1), extraire le type de fonction (unaire, binaire) ;
les lieurs doivent être dans le Q dominant, soit : numéro 0
renuméroter l'encapsulation de l'expression
première équivalence, deuxième argument (F2), renuméroter l'encapsulation de l'expression
première équivalence, deuxième argument (F2), tester le type de fonction (binaire, unaire) et de lien
deuxième équivalence, premier argument (F3), le lieur doit être = 2,
extraire le type de fonction (unaire, binaire) et la fonction propositionnelle
renuméroter l'encapsulation de l'expression
deuxième équivalence, deuxième argument (F4), tester le type de fonction (binaire, unaire)
deuxième équivalence, deuxième argument, tester la fonction propositionnelle
7. FP DE F1 ET F2 SONT LIES PAR Q DE A
premier fp
deuxième fp
effacer les marqueurs
8. TV DE F1 ET F2 SONT LIES PAR Q DE LA PREMIERE EQUIVALENCE DE A
premier tv
deuxième tv
effacer les marqueurs

9. FP DE F3 ET FP DE F4 ONT LIEURS DANS Q0 DE A
ET FP DE DEUXIEME EQUIVALENCE DANS Q2
****
le fp de F3
☞➢‹z1›≺➲‹a›○→‹b›xTV_[0]_‹i›[L/‹fp›/xTV_‹i›]‹e›←○‹z›⊂/cond9_ko⇒
➢‹z1›≺➲‹a›○→‹b›x◙TV_[0]_‹i›[◙L/‹fp›/xTV_‹i›]‹e›←○‹z›⊂/fp=[‹fp›]/cond9.1_ko☜[35]
☞➢‹z1›≺➲‹a›▷‹b›xTVQ{‹fp›IN(fp)}‹c›◁‹z›⊂/fp=[‹fp›]/cond9.1_ko⇒
➢‹z1›≺➲‹a›▷‹b›xTVQ{‹fp›}‹c›◁‹z›⊂/cond9.2_ko☜[36]
le fp de F4
☞➢‹z1›≺➲‹a›○→‹b›xTV_[0]_‹i›[L/‹fp›/xTV_‹i›]‹e›←○‹z›⊂/cond9.2_ko⇒
➢‹z1›≺➲‹a›○→‹b›x◙TV_[0]_‹i›[◙L/‹fp›/xTV_‹i›]‹e›←○‹z›⊂/fp=[‹fp›]/cond9.3_ko☜[37]
☞➢‹z1›≺➲‹a›▷‹b›xTVQ{‹fp›}‹c›◁‹z›⊂/fp=[‹fp›]/cond9.3_ko⇒
➢‹z1›≺➲‹a›▷‹b›xTVQ{‹fp›}‹c›◁‹z›⊂/cond9.4_ko☜[38]
effacer les marqueurs
☞➢‹z1›≺➲‹a›◙‹z›⊂/cond9.4_ko⇒➢‹z1›≺➲‹a›‹z›⊂/cond9.4_ko☜[39]
☞➢‹z1›≺➲‹a›⊂/cond9.4_ko⇒➢‹z1›≺➲‹a›⊂/cond10_ko☜[40]

10. LES TV F1, F2, F3 ET F4 SONT DES CATEGORIES SYNTAXICO-SEMANTIQUE
CONNUES DU SYSTEME
****
les termes de F1 et F2 sont déjà vérifiés dans les règles de la condition 4.
et sont de la catégorie des propositions (S)
PREMIERE EQUIVALENCE DE A, FP DE F1
cerner le fp de F1, pour un MOP
☞➢‹z1›≺➲‹a›●→‹b›xMOP_[0]{‹fp›}xPG[‹pg›]‹v›xPD[‹pd›]‹c›←●‹z›⊂/cond10_ko⇒
➢‹z1›≺➲‹a›●→‹b›x◙MOP_[0]{‹fp›}xPG[‹pg›]‹v›xPD[‹pd›]‹c›←●‹z›⊂/catSS={F1=[[S/S]{‹fp›}[‹pg›
-‹pd›]}/cond10.1_ko☜[41]
cerner le fp de F1, pour un BOP
☞➢‹z1›≺➲‹a›●→‹b›xBOP_[0]{‹fp›}xPG[‹pg›]‹v›xPD[‹pd›]‹c›←●‹z›⊂/cond10_ko⇒
➢‹z1›≺➲‹a›●→‹b›x◙BOP_[0]{‹fp›}xPG[‹pg›]‹v›xPD[‹pd›]‹c›←●‹z›⊂
/catSS={F1=[[S/SS]{‹fp›}[‹pg›--‹pd›]}/cond10.1_ko☜[42]
PREMIERE EQUIVALENCE DE A, FP DE F2
cerner le fp de F2, pour un MOP
☞➢‹z1›≺➲‹a›●→‹b›xMOP_[0]{‹fp›}xPG[‹pg›]‹v›xPD[‹pd›]‹c›←●‹z›⊂/catSS={‹F1›}/cond10.1_ko⇒
➢‹z1›≺➲‹a›●→‹b›x◙MOP_[0]{‹fp›}xPG[‹pg›]‹v›xPD[‹pd›]‹c›←●‹z›⊂
/catSS={‹F1›F2=[[S/S]{‹fp›}[‹pg›-‹pd›]}/cond10.2_ko☜[43]
cerner le fp de F2, pour un MOP
☞➢‹z1›≺➲‹a›●→‹b›xBOP_[0]{‹fp›}xPG[‹pg›]‹v›xPD[‹pd›]‹c›←●‹z›⊂/catSS={‹F1›}/cond10.1_ko⇒
➢‹z1›≺➲‹a›●→‹b›x◙BOP_[0]{‹fp›}xPG[‹pg›]‹v›xPD[‹pd›]‹c›←●‹z›⊂
/catSS={‹F1›F2=[[S/SS]{‹fp›}[‹pg›--‹pd›]}/cond10.2_ko☜[44]

vérifier que F1 et F2 sont de la même catégorie
☞➢‹z1›≺➲‹a›⊂/catSS={F1=[[‹cat›]{‹fp1›}[‹ctx1›]}F2=[[‹cat›]{‹fp2›}[‹ctx2›]}/cond10.2_ko⇒
➢‹z1›≺➲‹a›⊂/catSS={F1=[[‹cat›]{‹fp1›}[‹ctx1›]}F2=[[‹cat›]{‹fp2›}[‹ctx2›]}/cond10.3_ko☜[45]
vérifier que F1 et F2 ont le même contexte
☞➢‹z1›≺➲‹a›⊂/catSS={F1=[[‹cat1›]{‹fp1›}[‹ctx›]}F2=[[‹cat2›]{‹fp2›}[‹ctx›]}/cond10.3_ko⇒
➢‹z1›≺➲‹a›⊂/catSS={F1=[[‹cat1›]{‹fp1›}[‹ctx›]}F2=[[‹cat2›]{‹fp2›}[‹ctx›]}/cond10.4_ko☜[46]

DEUXIEME EQUIVALENCE DE A, FP DE F3
cerner le fp de F3 qui ne peut être qu'un MOP
☞➢‹z1›≺➲‹a›○→‹b›xMOP_[2]{‹fp›}xPG[‹pg›]‹v›xPD[‹pd›]‹c›←○‹z›⊂
/catSS={F1=[[‹cat1›]{‹fp1›}[‹ctx1›]}F2=[[‹cat2›]{‹fp2›}[‹ctx2›]}
➢‹z1›≺➲‹a›○→‹b›x◙MOP_[2]{‹fp›}xPG[‹pg›]‹v›xPD[‹pd›]‹c›←○‹z›⊂
/catSS={F1=[[‹cat1›]{‹fp1›}[‹ctx1›]}F2=[[‹cat2›]{‹fp2›}[‹ctx2›]}
F3=[[S/(‹cat1›)]{‹fp›}[‹pg›-‹pd›]}/cond10.5_ko☜[47]
cerner le fp de F4 qui ne peut être qu'un MOP
☞➢‹z1›≺➲‹a›○→‹b›xMOP_[2]{‹fp›}xPG[‹pg›]‹v›xPD[‹pd›]‹c›←○‹z›⊂
/catSS={‹F1›F2=[[‹cat2›]{‹fp2›}[‹ctx2›]}‹F3›}/cond10.5_ko⇒
➢‹z1›≺➲‹a›○→‹b›x◙MOP_[2]{‹fp›}xPG[‹pg›]‹v›xPD[‹pd›]‹c›←○‹z›⊂
/catSS={‹F1›F2=[[‹cat2›]{‹fp2›}[‹ctx2›]}‹F3›F4=[[S/(‹cat2›)]{‹fp›}[‹pg›-‹pd›]}/cond10.6_ko☜[48]

vérifier que F3 et F4 sont de la même catégorie
☞➢‹z1›≺➲‹a›⊂/catSS={‹F1F2›F3=[[‹cat›]{‹fp3›}[‹ctx3›]}F4=[[‹cat›]{‹fp4›}[‹ctx4›]}/cond10.6_ko⇒
➢‹z1›≺➲‹a›⊂/catSS={‹F1F2›F3=[[‹cat›]{‹fp3›}[‹ctx3›]}F4=[[‹cat›]{‹fp4›}[‹ctx4›]}
/cond10.7_ko☜[49]
vérifier que F3 et F4 ont le même fp
☞➢‹z1›≺➲‹a›⊂/catSS={‹F1F2›F3=[[‹cat3›]{‹fp›}[‹ctx3›]}F4=[[‹cat4›]{‹fp›}[‹ctx4›]}/cond10.7_ko⇒
➢‹z1›≺➲‹a›⊂/catSS={‹F1F2›F3=[[‹cat3›]{‹fp›}[‹ctx3›]}F4=[[‹cat4›]{‹fp›}[‹ctx4›]}
/cond10.8_ko☜[50]

✎ vérifier que F3 et F4 ont le même contexte
☞➢‹z1›≺➲‹a›⊂/catSS=⟦‹F1F2›F3=❛[‹cat3›]⟧‹fp3›⟦[‹ctx›]❜F4=❛[‹cat4›]⟧‹fp4›⟦[‹ctx›]❜⟧/cond10.8_ko⇒
➢‹z1›≺➲‹a›⊂/catSS=⟦‹F1F2›F3=❛[‹cat3›]⟧‹fp3›⟦[‹ctx›]❜F4=❛[‹cat4›]⟧‹fp4›⟦[‹ctx›]❜⟧
/cond10.9_ko☜✎[51]
✎ effacer les marqueurs
☞➢‹z1›≺➲‹a›◙‹z›⊂/catSS=⟦‹cat›⟧/cond10.9_ko⇒➢‹z1›≺➲‹a›‹z›⊂/catSS=⟦‹cat›⟧/cond10.9_ko☜✎[52]

✎ VERIFIER DANS LA BIBLOTHEQUE QUE LES CATEGORIES ONT ETE DEFINIES
✎ **SIMULATION (exécution d'une grammaire externe)**
☞➢‹z1›≺➲‹a›⊂/catSS=⟦‹cat›⟧/cond10.9_ko⇒➢‹z1›≺➲‹a›⊂/catSS=⟦‹cat›/ok⟧/cond10.10_ko☜✎[53]
☞➢‹z1›≺➲‹a›⊂/catSS=⟦‹cat›/ok⟧/cond10.10_ko⇒➢‹z1›≺➲‹a›⊂/catSS=⟦‹cat›/ok⟧/cond11_ko☜✎[54]

✎ 11. F1 ET F2, CATEGORIE C1/C2...CN, F3 ET F4 S/(C1/C2...CN)
✎ **********
☞➢‹z1›≺➲‹a›⊂/catSS=⟦F1=❛[S/‹cat›]⟧‹fp1›⟦[‹ctx1›]❜‹F2F3F4›/ok⟧/cond11_ko⇒
➢‹z1›≺➲‹a›⊂/catSS=⟦F1=❛[S/‹cat›]⟧‹fp1›⟦[‹ctx1›]❜‹F2F3F4›/ok⟧/cond11.1_ko☜✎[55]
☞➢‹z1›≺➲‹a›⊂/catSS=⟦‹F1F2›F3=❛[S/(‹cat›)]⟧‹fp3›⟦[‹ctx3›]❜‹F4›/ok⟧/cond11.1_ko⇒
➢‹z1›≺➲‹a›⊂/catSS=⟦‹F1F2›F3=❛[S/(‹cat›)]⟧‹fp3›⟦[‹ctx3›]❜‹F4›/ok⟧/cond11.2_ko☜✎[56]
☞➢‹z1›≺➲‹a›⊂/catSS=⟦‹cat›⟧/cond11.2_ko⇒➢‹z1›≺➲‹a›⊂/nettoyer☜✎[57]

✎ nettoyer
☞➢‹z1›≺➲‹a›▷‹b›◁‹c›⊂/nettoyer⇒➢‹z1›≺➲‹a›‹b›‹c›⊂/nettoyer☜✎[58]
☞➢‹z1›≺➲‹a›▶‹b›◀‹c›⊂/nettoyer⇒➢‹z1›≺➲‹a›‹b›‹c›⊂/nettoyer☜✎[59]
☞➢‹z1›≺➲‹a›●→‹b›←●‹c›⊂/nettoyer⇒➢‹z1›≺➲‹a›‹b›‹c›⊂/nettoyer☜✎[60]
☞➢‹z1›≺➲‹a›○→‹b›←○‹c›⊂/nettoyer⇒➢‹z1›≺➲‹a›‹b›‹c›⊂/nettoyer☜✎[61]

✎ mettre le status
☞➢‹z1›≺➲‹a›⊂/nettoyer⇒➢‹z1›≺➲‹a›⊂/extens_principe_ok☜✎[62]
☞➢‹z1›≺➲‹a›⊂/‹cond›_ko⇒➢‹z1›≺➲‹a›⊂/extens_principe_ok☜✎[63]

☞↳STOP↲☜✎[64]
↳ENDRULES↲
↳ENDGRAM↲

Génération d'une bibliothèque

↳GRAM↲
☞GRAMNAME⇒GENLIB☜
✎ générer les formule analysées pour une utilisation en bibliothèque

↳INCLUDE↲
☞PREANALYSE⇒"D:\TheseL\GramXAnalyse\GX01PreAnalyse.txt"☜
☞PROCDEF⇒"D:\TheseL\GramXAnalyse\GX02ProcDef.txt"☜
↳ENDINCLUDE↲

↳PARAMS↲
☞MAXGRAMLOOP⇒2☜
☞MAXRULELOOP⇒50☜
↳ENDPARAMS↲

↳RULES↲
☞↳SWITCHCORPUS(↳INPUT_CORPUS↲)↲☜✎[1]
☞↳CORPUSFIRST↲☜✎[2]

✎ >>> ENTRER_FORMULE <<<

✎ extraire la ligne de INPUT_CORPUS, le contenu est mis dans TEMPORARY_ZONE
☞↳GETCORPUSLINE↲☜✎[3]
✎ concaténation de TEMPORARY_ZONE (ligne de INPUT_CORPUS)
✎ dans RM_RESULT_ZONE, qui peut retourner aussi {<EOF>}
☞↳CONCAT(↳RM_RESULT_ZONE↲,"[entre]",↳TEMPORARY_ZONE↲)↲☜✎[4]
✎ tester eof après concaténation, si eof : TEMPORARY_ZONE retourne {<EOF>}

✎ Aller à >>> TRAITER BUFFER <<<
☞‹a›{<EOF>}⇒‹a›/EOF↳GOTORULE(16)↲☜✎[5]

✎ formater la formule depuis INPUT_CORPUS
✎ préparer la structure : ➢‹avant›▶‹formule›◀‹apres›≺
☞[entre]⌊‹a›⌉⟦‹id›⟧⇒➢⟦‹id›⟧⌊‹a›⌉▶⌊‹a›⌉◀≺/ok☜✎[6]
☞[entre]≡‹a›⟦‹id›⟧⇒➢⟦‹id›⟧≡‹a›▶≡‹a›◀≺/ok☜✎[7]

```
✐ PREANALYSE
✐ l'analyse se fait entre les ►...◄
☞➢‹a›►‹b›◄‹c›≮/ok⇒➢‹a›►‹b›◄‹c›≮☜✐[8]
☞↳EXECUTEGRAM(PREANALYSE)↲☜✐[9]

✐ PROCEDURE DEFINITOIRE
✐ l'analyse de procédure définitoire se fait entre les ►...◄
☞↳EXECUTEGRAM(PROCDEF)↲☜✐[10]

✐ MEMORISER LE RESULTAT DANS BUFFER
☞↳APPENDBUFFERLINE↲☜✐[11]

✐ mettre un marqueur de réussite
☞➢‹a›►‹b›◄‹c›≮⇒➢‹a›►‹b›◄‹c›≮/ok☜✐[12]

✐ ☞↳DISPOSERESULT↲☜✐[13]
☞INUTILE⇒NE RIEN FAIRE☜✐[13]

☞↳CORPUSNEXT↲☜✐[14]

✐ Aller à >>> ENTRER_FORMULE <<<
☞↳IFMATCH_GOTORULE(5)↲☜✐ [15]

✐ >>> TRAITER BUFFER <<<
☞↳DISPOSERESULT↲☜✐[16]
☞↳SWITCHCORPUS(↳BUFFER_CORPUS↲)↲☜✐[17]
☞↳CORPUSFIRST↲☜✐[18]

✐ >>> LIGNE_BUFFER_SUIVANTE <<<
☞↳GETCORPUSLINE↲☜✐[19]
✐ concaténation de TEMPORARY_ZONE (ligne de INPUT_CORPUS)
✐ dans RM_RESULT_ZONE, qui peut retourner aussi  {<EOF>}
☞↳CONCAT(↳RM_RESULT_ZONE↲,"BUFFER:",↳TEMPORARY_ZONE↲)↲☜✐[20]

✐ Aller à >>> FIN <<<
☞‹a›{<EOF>}⇒‹a›/EOF↳GOTORULE(25)↲☜✐[21]
☞BUFFER:➢‹a›►‹b›◄‹c›≮⇒➢‹a›►‹b›◄‹c›≮☜✐ [22]
☞↳CORPUSNEXT↲☜✐[23]

✐ Aller à >>> LIGNE_BUFFER_SUIVANTE <<<
☞↳IFMATCH_GOTORULE(19)↲☜✐ [24]

✐ >>> FIN <<<

☞INUTILE⇒NE RIEN FAIRE☜✐[25]

☞↳STOP↲☜✐[26]
↳ENDRULES↲
↳ENDGRAM↲
```

En nous donnant l'extrait de *corpus d'entrée* suivant :

```
⌊qpr⌋⌈≡(≡(≡(pr)≡(qp))≡(rq))⌉⦗Ap1⦘
⌊qpr⌋⌈≡(≡(p≡(qr))≡(≡(pq)r))⌉⦗Ap2⦘
⌊pq⌋⌈≡(μ(pq)≡(p≡(pq)))⌉⦗Dp1⦘
⌊pq⌋⌈≡(Ц(pq)≡(q≡(pq)))⌉⦗FGp1⦘
⌊pq⌋⌈≡(τ(pq)≡(≡(pq)≡(pq)))⌉⦗Dp2⦘
⌊p⌋⌈≡(α(p)≡(p⌊q⌋⌈≡(qq)⌉))⌉⦗Dp3⦘
⌊p⌋⌈≡(~(p)≡(p⌊q⌋⌈q⌉))⌉⦗Dp4⦘
⌊pq⌋⌈≡(ω(pq)¬(≡(pq)))⌉⦗Dp5⦘
⌊p⌋⌈≡(τ(p)≡(pp))⌉⦗Dp6⦘
⌊pqr⌋⌈≡(δ(pqr)≡(r¬(≡(pq))))⌉⦗Dp7⦘
≡(F⌊p⌋⌈p⌉)⦗Dp8⦘
≡(V~(F))⦗Dp9⦘
≡(T⌊p⌋⌈≡(pp)⌉)⦗Dp10⦘
⌊qpr⌋⌈≡(ω⟍pq⟋(r)≡(~(≡(pq))r))⌉⦗Dp11⦘
⌊qpr⌋⌈≡(ω[p]⟍q⟋(r)≡(~(≡(pq))r))⌉⦗Dp12⦘
⌊pqg⌋⌈≡(Δ[gpq]≡(p~(g(pq))))⌉⦗Dp13⦘
⌊pq⌋⌈≡(∧(pq)⌊h⌋⌈≡(p≡(h(p)h(q)))⌉)⌉⦗Dp14⦘
⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉⦗Dp15⦘
⌊pq⌋⌈≡(∨(pq)~(∧(~(p)~(q))))⌉⦗Dp16⦘
```

Nous obtenons les résultats :

GENLIB, 1, 22, 1:
➢⦚Ap1⦛⌊qpr⌋⌈≡(≡(≡(pr)≡(qp))≡(rq))⌉►xQ_0⌊xTVQ⦃p⦄xTVQ⦃q⦄xTVQ⦃r⦄⁄xQ_0⌋xSQ_0⌈xE_5⦃xBOP⦄≡⦃xPG⦃(⦄xE_4⦃xBOP⦄≡⦃xPG⦃(⦄xE_1⦃xBOP⦄≡⦃xPG⦃(⦄xTV_⟦0⟧_1⦃L/p⁄xTV_1⦄xTV_⟦0⟧_2⦃L/r⁄xTV_2⦄xPD⦃)⦄⁄xE_1⦄xE_2⦃xBOP⦄≡⦃xPG⦃(⦄xTV_⟦0⟧_3⦃L/q⁄xTV_3⦄xTV_⟦0⟧_4⦃L/p⁄xTV_4⦄xPD⦃)⦄⁄xE_2⦄xPD⦃)⦄⁄xE_4⦄xE_3⦃xBOP⦄≡⦃xPG⦃(⦄xTV_⟦0⟧_5⦃L/r⁄xTV_5⦄xTV_⟦0⟧_6⦃L/q⁄xTV_6⦄xPD⦃)⦄⁄xE_3⦄xPD⦃)⦄⁄xE_5⦄⁄xSQ_0⌉◄≺

GENLIB, 1, 22, 2:
➢⦚Ap2⦛⌊qpr⌋⌈≡(≡(p≡(qr))≡(≡(pq)r))⌉►xQ_0⌊xTVQ⦃p⦄xTVQ⦃q⦄xTVQ⦃r⦄⁄xQ_0⌋xSQ_0⌈xE_5⦃xBOP⦄≡⦃xPG⦃(⦄xE_3⦃xBOP⦄≡⦃xPG⦃(⦄xTV_⟦0⟧_1⦃L/p⁄xTV_1⦄xE_1⦃xBOP⦄≡⦃xPG⦃(⦄xTV_⟦0⟧_2⦃L/q⁄xTV_2⦄xTV_⟦0⟧_3⦃L/r⁄xTV_3⦄xPD⦃)⦄⁄xE_1⦄xPD⦃)⦄⁄xE_3⦄xE_4⦃xBOP⦄≡⦃xPG⦃(⦄xE_2⦃xBOP⦄≡⦃xPG⦃(⦄xTV_⟦0⟧_4⦃L/p⁄xTV_4⦄xTV_⟦0⟧_5⦃L/q⁄xTV_5⦄xPD⦃)⦄⁄xE_2⦄xTV_⟦0⟧_6⦃L/r⁄xTV_6⦄xPD⦃)⦄⁄xE_4⦄xPD⦃)⦄⁄xE_5⦄⁄xSQ_0⌉◄≺

…

GENLIB, 1, 22, 18:
➢⦚Dp15⦛⌊pq⌋⌈≡(⊃(pq)~(∧(p~(q))))⌉►xQ_0⌊xTVQ⦃p⦄xTVQ⦃q⦄⁄xQ_0⌋xSQ_0⌈xE_5⦃xBOP⦄≡⦃xPG⦃(⦄xE_1⦃xTCAC⦃⊃⦄xCTX_1⟦xPG⦃(⦄xTV_⟦0⟧_0⦃D/p⁄xTV_0⦄xTV_⟦0⟧_0⦃D/q⁄xTV_0⦄xPD⦃)⦄⁄xCTX_1⟧⁄xE_1⦄xE_4⦃xMOP⦄~⦃xPG⦃(⦄xE_3⦃xBOP⦄∧⦃xPG⦃(⦄xTV_⟦0⟧_1⦃L/p⁄xTV_1⦄xE_2⦃xMOP⦄~⦃xPG⦃(⦄xTV_⟦0⟧_2⦃L/q⁄xTV_2⦄xPD⦃)⦄⁄xE_2⦄xPD⦃)⦄⁄xE_3⦄xPD⦃)⦄⁄xE_4⦄xPD⦃)⦄⁄xE_5⦄⁄xSQ_0⌉◄≺

GENLIB, 1, 22, 19:
➢⦚Dp16⦛⌊pq⌋⌈≡(∨(pq)~(∧(~(p)~(q))))⌉►xQ_0⌊xTVQ⦃p⦄xTVQ⦃q⦄⁄xQ_0⌋xSQ_0⌈xE_6⦃xBOP⦄≡⦃xPG⦃(⦄xE_1⦃xTCAC⦃∨⦄xCTX_1⟦xPG⦃(⦄xTV_⟦0⟧_0⦃D/p⁄xTV_0⦄xTV_⟦0⟧_0⦃D/q⁄xTV_0⦄xPD⦃)⦄⁄xCTX_1⟧⁄xE_1⦄xE_5⦃xMOP⦄~⦃xPG⦃(⦄xE_4⦃xBOP⦄∧⦃xPG⦃(⦄xE_2⦃xMOP⦄~⦃xPG⦃(⦄xTV_⟦0⟧_1⦃L/p⁄xTV_1⦄xPD⦃)⦄⁄xE_2⦄xE_3⦃xMOP⦄~⦃xPG⦃(⦄xTV_⟦0⟧_2⦃L/q⁄xTV_2⦄xPD⦃)⦄⁄xE_3⦄xPD⦃)⦄⁄xE_4⦄xPD⦃)⦄⁄xE_5⦄xPD⦃)⦄⁄xE_6⦄⁄xSQ_0⌉◄≺

Enchaînement des GOi d'analyses, de détermination de catégories, de calculs propositionnels, de conversion et de dépouillement

⦚GRAM⦛
☞GRAMNAME⇒TOUT☜
🖉 exécuter toutes les grammaires d'analyse pour une formule donnée

⦚INCLUDE⦛
☞PREANALYSE⇒"D:\TheseL\GramXAnalyse\GX01PreAnalyse.txt"☜
☞PROCDEF⇒"D:\TheseL\GramXAnalyse\GX02ProcDef.txt"☜
☞DEPX⇒"D:\TheseL\GramXDepouille\GX05Depouille.txt"☜
☞CATEGORIE⇒"D:\TheseL\GramXCategorie\GX03Categorie.txt"☜
☞CALCULUS⇒"D:\TheseL\GramXCalculus\GX04Calculus.txt"☜
☞CONVERTXC⇒"D:\TheseL\GramXConvert\GX06Convert.txt"☜
☞DEPC⇒"D:\TheseL\GramCDepouille\GC06Depouille.txt"☜
⦚ENDINCLUDE⦛

⦚PARAMS⦛
☞MAXGRAMLOOP⇒2☜
☞MAXRULELOOP⇒5☜
⦚ENDPARAMS⦛

⦚RULES⦛
🖉 entrer la formule depuis INPUT_CORPUS
🖉 préparer la structure : ➢‹avant›►‹formule›◄‹apres›≺
☞⌊‹a›⇒➢⌊‹a›►⌊‹a›◄≺☜🖉[1]
☞≡‹a›⇒➢≡‹a›►≡‹a›◄≺☜🖉[2]

🖉 PREPARER ET EXECUTER L'ANALYSE DES TERMINAUX
🖉 ET CONSTRUCTION DES TERMES ET EXPRESSIONS
🖉 l'analyse se fait entre les ►...◄
☞⦚EXECUTEGRAM(PREANALYSE)⦛☜🖉[3]
🖉 VERIFIER LA PROCEDURE DEFINITOIRE, TRIER LES VARIABLES DANS
🖉 LES DIFFERENTES EXPRESSIONS, ETABLIR LES LIENS ENTRE LIEURS DES
🖉 QUANTIFICATEURS ET VARIABLES LIEES
🖉 l'analyse de procédure définitoire se fait entre les ►...◄
☞⦚EXECUTEGRAM(PROCDEF)⦛☜🖉[4]

✎ DEPOUILLER
✎ le dépouillement se fait entre les curseurs : ▷...◁ pour une structure :
✎ ➢‹avant›►‹formule›◄‹apres›≺▷‹résultat›◁
☞↳EXECUTEGRAM(DEPX)↲☜✎[5]
✎ on copie le résultat après l'expression initiale : ➢‹avant›__/__‹resultat›►‹formule›◄‹apres›≺
☞➢‹avant›►‹v›◄‹apres›≺▷‹res›◁⇒➢‹avant›__/__‹res›►‹v›◄‹apres›≺☜✎[6]

✎ DETERMINER ET INSCRIRE LES CATEGORIES SYNTAXICO-SEMANTIQUES
☞↳EXECUTEGRAM(CATEGORIE)↲☜✎[7]

✎ CALCULER LES TABLES DE VERITE
☞↳EXECUTEGRAM(CALCULUS)↲☜✎[8]

✎ CONVERTIR DE LA VERSION CONTEXTUELLE VERS LA FORME CATEGORIELLE
☞↳EXECUTEGRAM(CONVERTXC)↲☜✎[9]

✎ DEPOUILLER LA FORME CATEGORIELLE
☞➢‹avant›►‹v›◄‹apres›≺▷‹w›◁⇒➢‹avant›►‹w›◄‹apres›≺☜✎[10]
☞↳EXECUTEGRAM(DEPC)↲☜✎[11]
☞➢‹avant›►‹v›◄‹apres›≺▷‹w›◁⇒➢‹avant›►‹w›◄‹apres›≺☜✎[11]

☞↳STOP↲☜✎[]
↳ENDRULES↲
↳ENDGRAM↲

Remplacement des éléments morphosyntaxiques et symboliques par des formules en langue naturelle

↳GRAM↲
☞GRAMNAME⇒CONDISUBEX☜
✎ le but de cet exemple rudimentaire de re-grammaire est de remplacer (substituer) :
✎ les variables du quantificateur, les termes variables du definiendum et du definiens par :
✎ p : « il fait beau temps » ; ~p : « il fait mauvais temps »
✎ q : « on sort le chien » ; ~q : « on ne sort pas le chien »
✎ les termes constants par :
✎ ⊃ : « si…alors »
✎ ∧ : « et »
✎ une expression négative (cE
✎ cE_‹i›cMOP~cPG(‹contenu›cPD)/cE_‹i› : « il est faut de dire que… »
✎ le quantificateur universel :
✎ ∀ : « Quel que soit… »

↳INCLUDE↲
☞PREANALYSE⇒"D:\TheseL\GramXAnalyse\GX01PreAnalyse.txt"☜
☞PROCDEF⇒"D:\TheseL\GramXAnalyse\GX02ProcDef.txt"☜
☞CONVERTXC⇒"D:\TheseL\GramXConvert\GX06Convert.txt"☜
↳ENDINCLUDE↲

↳PARAMS↲
☞MAXGRAMLOOP⇒50☜
☞MAXRULELOOP⇒50☜
↳ENDPARAMS↲

↳RULES↲
✎ L'inscription doit être encapsulée par ►◄
☞►‹a›◄⇒►‹a›◄/ok1☜✎[1]
✎ L'inscription doit être une généralisation.
☞►⌊‹a›⌋⌈‹b›⌉◄/ok1⇒►⌊‹a›⌋⌈‹b›⌉◄/ok2☜✎[2]
✎Mettre dans le format standard, sauvegarder l'expression en version contextuelle brute entre : ➢…►
☞►⌊‹a›⌋⌈‹b›⌉◄/ok2⇒➢⌊‹a›⌋⌈‹b›⌉►⌊‹a›⌋⌈‹b›⌉◄≺☜✎[3]

✎ PREPARER ET EXECUTER L'ANALYSE DES TERMINAUX ET ENCAPSULATION DES TERMES
✎ ET DES EXPRESSIONS
✎ l'analyse se fait entre les ►...◄
☞↳EXECUTEGRAM(PREANALYSE)↲☜✎[4]

✎ VERIFIER LA PROCEDURE DEFINITOIRE, TRIER LES VARIABLES, ETABLIR LES LIENS
✎ ENTRE LIEURS ET TERMES VARIABLES
✎ l'analyse de procédure définitoire se fait entre les ►...◄
☞↳EXECUTEGRAM(PROCDEF)↲☜✎[5]

✎ CONVERTIR VERS LA FORME CATEGORIELLE
✎ si on convertit la forme contextuelle vers la forme catégorielle,
✎ c'est parce que sa structure syntaxique est plus proche de la langue française
☞↳EXECUTEGRAM(CONVERTXC)✍☜✎[6]

✎ REORGANISER LA CHAINE
☞➢‹avant›►‹x›◄‹apres›≺▷‹w›◁⇒➢‹avant›►‹w›◄‹apres›≺☜✎[7]

✎ Insertions / remplacements des variables propositionnelles
✎ Dans une version plus générique, cette partie et les suivantes, concernant les remplacements
✎ eux-mêmes et donnés ici en re-règles devraient être inscrits dans le corpus d'entrée
☞➢‹avant›►‹a›cTVQ[**p**]‹b›◄‹apres›≺⇒➢‹avant›►‹a›cTVQ[**il fait beau temps**]‹b›◄‹apres›≺☜✎[8]
☞➢‹avant›►‹a›cTVQ[**q**]‹b›◄‹apres›≺⇒➢‹avant›►‹a›cTVQ[**on sort le chien**]‹b›◄‹apres›≺☜✎[9]

✎ REMPLACEMENTS DES TERMES
✎ Remplacer les variables propositionnelles mise en négation dans une expression (du definiens)

✎ >>> NEG_Q <<<

✎ Cerner le contenu d'une expression avec négation
☞➢‹avant›►‹a›cE_‹i›[cMOP]~[cPG[(]‹b›cPD[)]]/cE_‹i›]‹c›◄‹apres›≺⇒
➢‹avant›►‹a›cE●_‹i›[cMOP]~[cPG[(]→‹b›←cPD[)]]/cE_‹i›]‹c›◄‹apres›≺/m1☜✎[10]
✎ Tester si la négation porte sur un terme
☞➢‹avant›►‹a›→cTV_[‹x›]_‹i›[L/**q**/cTV_‹i›]←‹b›◄‹apres›≺/m1⇒
➢‹avant›►‹a›→cTV_[‹x›]_‹i›[L/**on ne sort pas le chien**/cTV_‹i›]←‹b›◄‹apres›≺/m2☜✎[11]
✎ si oui, supprimer l'encapsulation
☞➢‹avant›►‹a›cE●_‹i›[cMOP]~[cPG[(]→‹b›←cPD[)]]/cE_‹i›]‹c›◄‹apres›≺/m2⇒
➢‹avant›►‹a›‹b›‹c›◄‹apres›≺☜✎[12]
✎ si non, supprimer les curseurs et recommencer
☞➢‹avant›►‹a›→‹b›←‹c›◄‹apres›≺/m1⇒➢‹avant›►‹a›‹b›‹c›◄‹apres›≺☜✎[13]

✎ Aller à >>> NEG_Q <<<
☞↳IFMATCH_GOTORULE(10)✍☜✎[14]
✎ supprimer les possibles marqueurs restants
☞➢‹avant›►‹a›cE●_‹b›◄‹apres›≺⇒➢‹avant›►‹a›cE_‹b›◄‹apres›≺☜✎[15]
✎ Remplacer la variable propositionnelle q dans une expression affirmative (du definiens)
☞➢‹avant›►‹a›cTV_[‹i›]_‹j›[‹type›/**q**/cTV_‹j›]‹b›◄‹apres›≺⇒
➢‹avant›►‹a›cTV_[‹i›]_‹j›[‹type›/**on sort le chien**/cTV_‹j›]‹b›◄‹apres›≺☜✎[16]

✎ >>> NEG_P <<<

☞➢‹avant›►‹a›cE_‹i›[cMOP]~[cPG[(]‹b›cPD[)]]/cE_‹i›]‹c›◄‹apres›≺⇒
➢‹avant›►‹a›cE●_‹i›[cMOP]~[cPG[(]→‹b›←cPD[)]]/cE_‹i›]‹c›◄‹apres›≺/m1☜✎[17]
☞➢‹avant›►‹a›→cTV_[‹x›]_‹i›[L/**p**/cTV_‹i›]←‹b›◄‹apres›≺/m1⇒
➢‹avant›►‹a›→cTV_[‹x›]_‹i›[L/**il fait mauvais temps**/cTV_‹i›]←‹b›◄‹apres›≺/m2☜✎[18]
☞➢‹avant›►‹a›cE●_‹i›[cMOP]~[cPG[(]→‹b›←cPD[)]]/cE_‹i›]‹c›◄‹apres›≺/m2⇒
➢‹avant›►‹a›‹b›‹c›◄‹apres›≺☜✎[19]
☞➢‹avant›►‹a›→‹b›←‹c›◄‹apres›≺/m1⇒➢‹avant›►‹a›‹b›‹c›◄‹apres›≺☜✎[20]

✎ Aller à >>> NEG_P <<<
☞↳IFMATCH_GOTORULE(17)✍☜✎[21]
☞➢‹avant›►‹a›cE●_‹b›◄‹apres›≺⇒➢‹avant›►‹a›cE_‹b›◄‹apres›≺☜✎[22]
☞➢‹avant›►‹a›cTV_[‹i›]_‹j›[‹type›/**p**/cTV_‹j›]‹b›◄‹apres›≺⇒
➢‹avant›►‹a›cTV_[‹i›]_‹j›[‹type›/**il fait beau temps**/cTV_‹j›]‹b›◄‹apres›≺☜✎[23]

✎ REMPLACEMENT DU QUANTIFICATEUR
☞➢‹avant›►cQ_‹i›[(∀‹n›)/cQ_‹i›]‹z›◄‹apres›≺⇒
➢‹avant›►cQ_‹i›[[**Quel que soit**]‹n›/cQ_‹i›]‹z›◄‹apres›≺/mark1☜✎[24]
☞➢‹avant›►cQ_‹i›[[‹quantu›]‹a›cTVQ[‹t›]‹b›/cQ_‹i›]‹z›◄‹apres›≺/mark1⇒
➢‹avant›►cQ_‹i›[[‹quantu›]‹a›"‹t›", ‹b›/cQ_‹i›]‹z›◄‹apres›≺/mark1☜✎[25]
☞➢‹avant›►cQ_‹i›[[‹quantu›]‹tvq›/cQ_‹i›]‹z›◄‹apres›≺/mark1⇒
➢‹avant›►‹quantu›‹tvq›‹z›◄‹apres›≺/mark2☜✎[26]

✎ REMPLACEMENT DANS LE DEFINIENDUM DU SOUS-QUANTIFICATEUR
✎ cerner le contenu du sous-quantificateur avec des curseurs
☞➢‹avant›►‹Q›cSQ_‹i›[‹a›/cSQ_‹i›]◄‹apres›≺/mark2⇒
➢‹avant›►‹Q›➲⇨‹a›⇦⊂◄‹apres›≺/mark3☜✎[27]
✎ cerner le contenu de l'expression du sous-quantificateur avec les mêmes curseurs
☞➢‹avant›►‹a›➲⇨cE_‹i›[cPG[(]‹cont›cPD[)]]/cE_‹i›]⇦⊂◄‹apres›≺/mark3⇒
➢‹avant›►‹a›➲⇨‹cont›⇦⊂◄‹apres›≺/mark4☜✎[28]
✎ cerner le definiendum et le definiens du sous-quantificateur avec les mêmes curseurs

☞➢‹avant›►‹a›➲⇨cE_‹i›[cPG[(]‹cont›cPD[)]]/cE_‹i›]cBOP}≡{‹b›⇦⊂◄‹apres›≮/mark4⇒
➢‹avant›►‹a›⇨‹cont›⇦cBOP}≡{➲‹b›⊂◄‹apres›≮/mark5☜🖉[29]

🖉 traiter le terme constant du definiendum
🖉 dans une version générique on traiterait les cas de termes constants multi-contextes…
☞➢‹avant›►‹Q›⇨‹tv1›cBOP}⊃{‹tv2›⇦cBOP}≡{➲‹diens›⊂◄‹apres›≮/mark5⇒
➢‹avant›►‹Q›⇨si }‹tv1›{ alors }‹tv2›{⇦cBOP}≡{➲‹diens›⊂◄‹apres›≮/mark6☜🖉[30]
☞➢‹avant›►‹Q›⇨‹tv1›cBOP}∧{‹tv2›⇦cBOP}≡{➲‹diens›⊂◄‹apres›≮/mark5⇒
➢‹avant›►‹Q›⇨}‹tv1›{ et }‹tv2›{⇦cBOP}≡{➲‹diens›⊂◄‹apres›≮/mark6☜🖉[31]
☞➢‹avant›►‹Q›⇨‹a›}cTV_[‹i›]_‹j›[‹t›/‹prop›/cTV_‹j›]{‹z›⇦cBOP}≡{➲‹diens›⊂◄‹apres›≮/mark6⇒
➢‹avant›►‹Q›⇨‹a›‹prop›‹z›⇦cBOP}≡{➲‹diens›⊂◄‹apres›≮/mark6☜🖉[32]
☞➢‹avant›►‹Q›⇨‹a›⇦cBOP}≡{➲‹diens›⊂◄‹apres›≮/mark6⇒
➢‹avant›►‹Q› on définit : "‹a›", par la définition : ➲‹diens›⊂◄‹apres›≮☜🖉[33]

🖉 REMPLACEMENT DANS LE DEFINIENS DU SOUS-QUANTIFICATEUR

🖉 >>> EXP_T_T <<<

☞➢‹avant›►‹a›➲‹b›cE_‹i›[cPG[(]‹c›cPD[)]]/cE_‹i›]‹z›⊂◄‹apres›≮⇒
➢‹avant›►‹a›➲‹b›cE●_‹i›[cPG[(]→‹c›←cPD[)]]/cE_‹i›]‹z›⊂◄‹apres›≮/m1☜🖉[34]
☞➢‹avant›►‹a›→cTV_[‹x›]_‹i›[L/‹p›/cTV_‹i›]cBOP}‹bop›{cTV_[‹y›]_‹k›[L/‹q›/cTV_‹k›]←‹b›◄
‹apres›≮/m1⇒
➢‹avant›►‹a›→‹p› }‹bop›{ ‹q›←‹b›◄‹apres›≮/m2☜🖉[35]
🖉 Dans une version générique on traiterait non seulement l'ensemble des termes constants binaires,
🖉 mais aussi unaires et n-aires
☞➢‹avant›►‹a›→‹b›}∧{‹c›←‹d›◄‹apres›≮/m2⇒➢‹avant›►‹a›→‹b›et‹c›←‹d›◄‹apres›≮/m3☜🖉[36]
☞➢‹avant›►‹a›→‹b›}∨{‹c›←‹d›◄‹apres›≮/m2⇒➢‹avant›►‹a›→‹b›ou‹c›←‹d›◄‹apres›≮/m3☜🖉[37]
☞➢‹avant›►‹a›cE●_‹i›[cPG[(]→‹b›←cPD[)]]/cE_‹i›]‹c›◄‹apres›≮/m3⇒
➢‹avant›►‹a›‹b›‹c›◄‹apres›≮☜🖉[38]
☞➢‹avant›►‹a›→‹b›←‹c›◄‹apres›≮/m1⇒➢‹avant›►‹a›‹b›‹c›◄‹apres›≮☜🖉[39]

🖉 Aller à >>> EXP_T_T <<<
☞↳IFMATCH_GOTORULE(34)↲☜🖉[40]
☞➢‹avant›►‹a›cE●_‹b›◄‹apres›≮⇒➢‹avant›►‹a›cE_‹b›◄‹apres›≮☜🖉[41]

🖉 NEGATION D'UNE EXPRESSION

🖉 >>> NEG_EXP <<<

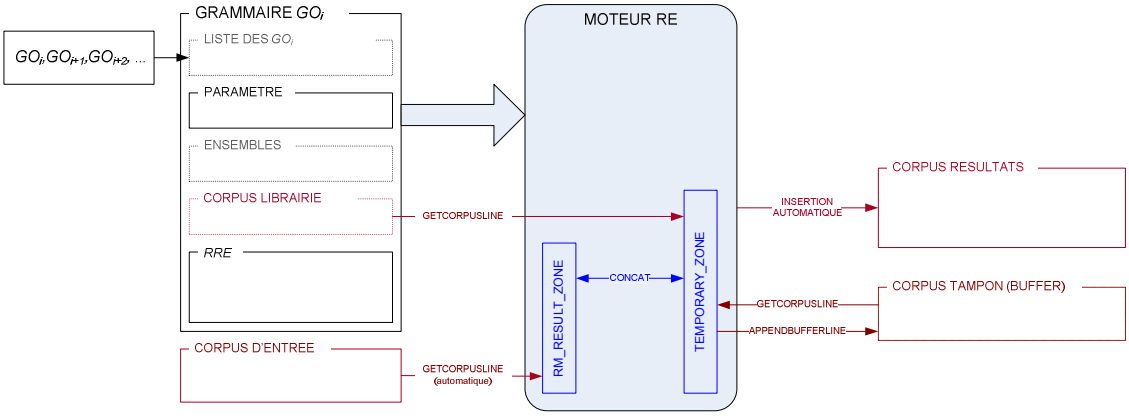
☞➢‹avant›►‹a›➲‹b›cE_‹i›[cMOP}~{cPG[(]‹c›cPD[)]]/cE_‹i›]‹z›⊂◄‹apres›≮⇒
➢‹avant›►‹a›➲‹b› il est faux de dire que ‹c›‹z›⊂◄‹apres›≮☜🖉[42]

🖉 Supprimer les curseurs inutiles
☞➢‹avant›►‹a›➲‹b›⊂◄‹apres›≮⇒➢‹avant›►‹a›‹b›◄‹apres›≮☜🖉[43]

☞↳STOP↲☜🖉[44]
↳ENDRULES↲
↳ENDGRAM↲

Manuel utilisateur du logiciel prototype RE V4.4

Structures internes



L'interface utilisateur

On lance le logiciel à partir d'une icône de raccourci installée sur le bureau :

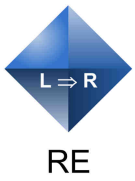


Figure 68. Icône de lancement

La fenêtre principale, RE V4.4 s'affiche :

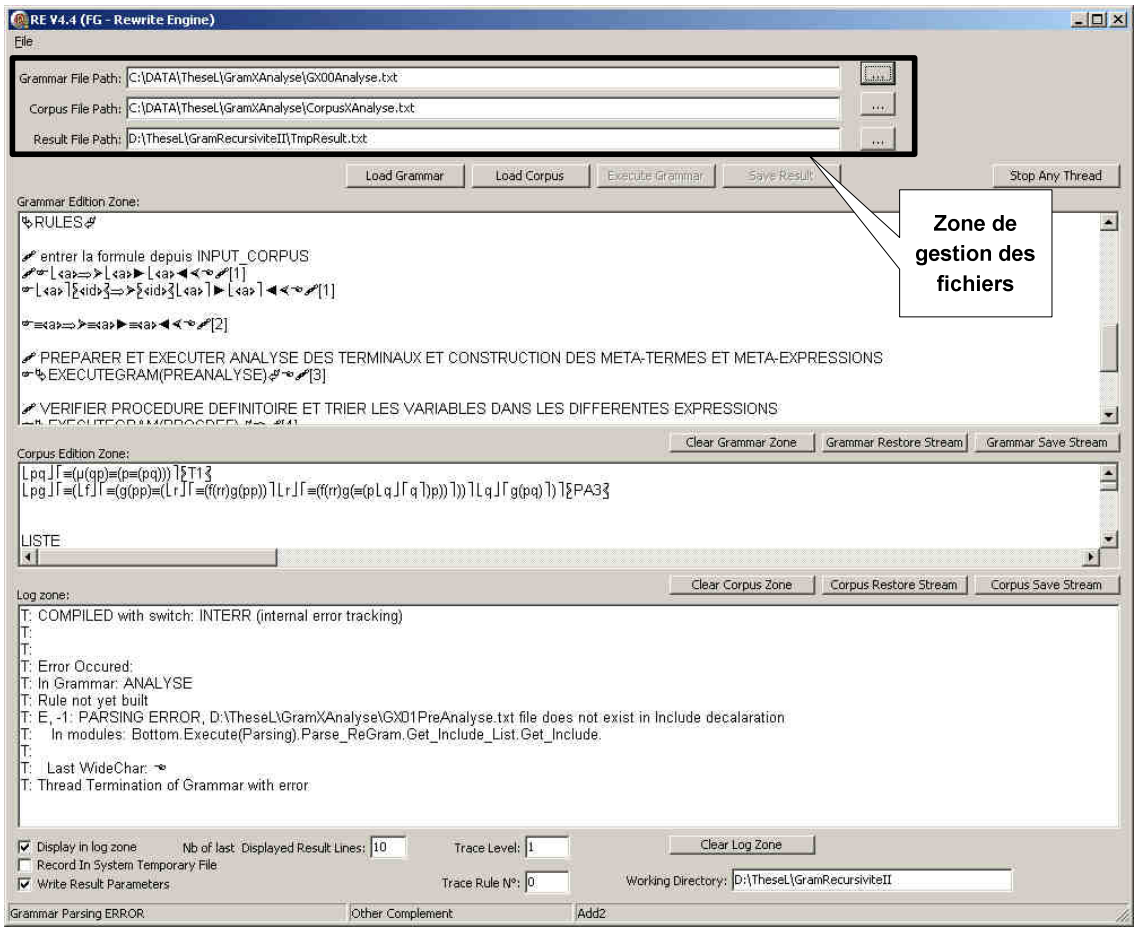


Figure 69. Zone de gestion des fichiers

Le premier champ, *Gramar File Path*, indique au re-système où se trouve et où on sauvegardera la re-grammaire sur laquelle on travaille. Le deuxième, *Corpus File Path*, détermine l'endroit où se trouve la corpus (d'entrée). Le troisième, *Result File Path*, spécifie où vont se sauvegarder les résultats d'une exécution. A ces trois champs sont associés à un bouton qui permet de rechercher un fichier dans le système Windows.

Chacun de ces champs correspond à une zone de travail ou zone d'édition :

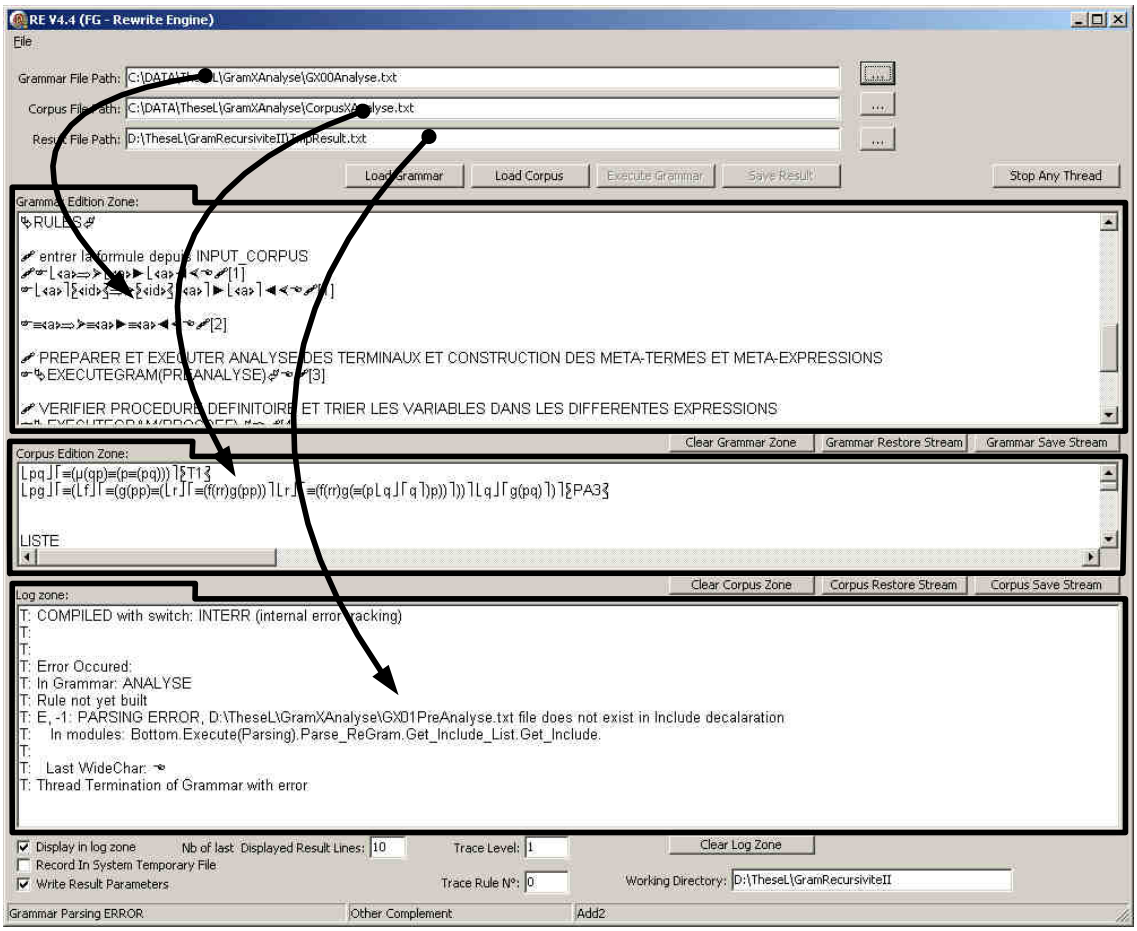


Figure 70. Les zones d'édition et d'affichage

Dans la zone, *Grammar Edition Zone*, on saisit, modifie, supprime, insère, coupe, copie et colle le texte ou des portions de texte de re-grammaires. La deuxième zone, *Corpus Edition Zone*, offre les mêmes fonctions que la première, mais destinée à l'édition des lignes du corpus d'entrée. La zone, *Log zone*, est une zone réservée à l'affichage des erreurs, du traçage des processus et des résultats de la compilation ou de l'exécution de la re-grammaire.

A chacune de ces zones est associé un ensemble de boutons :

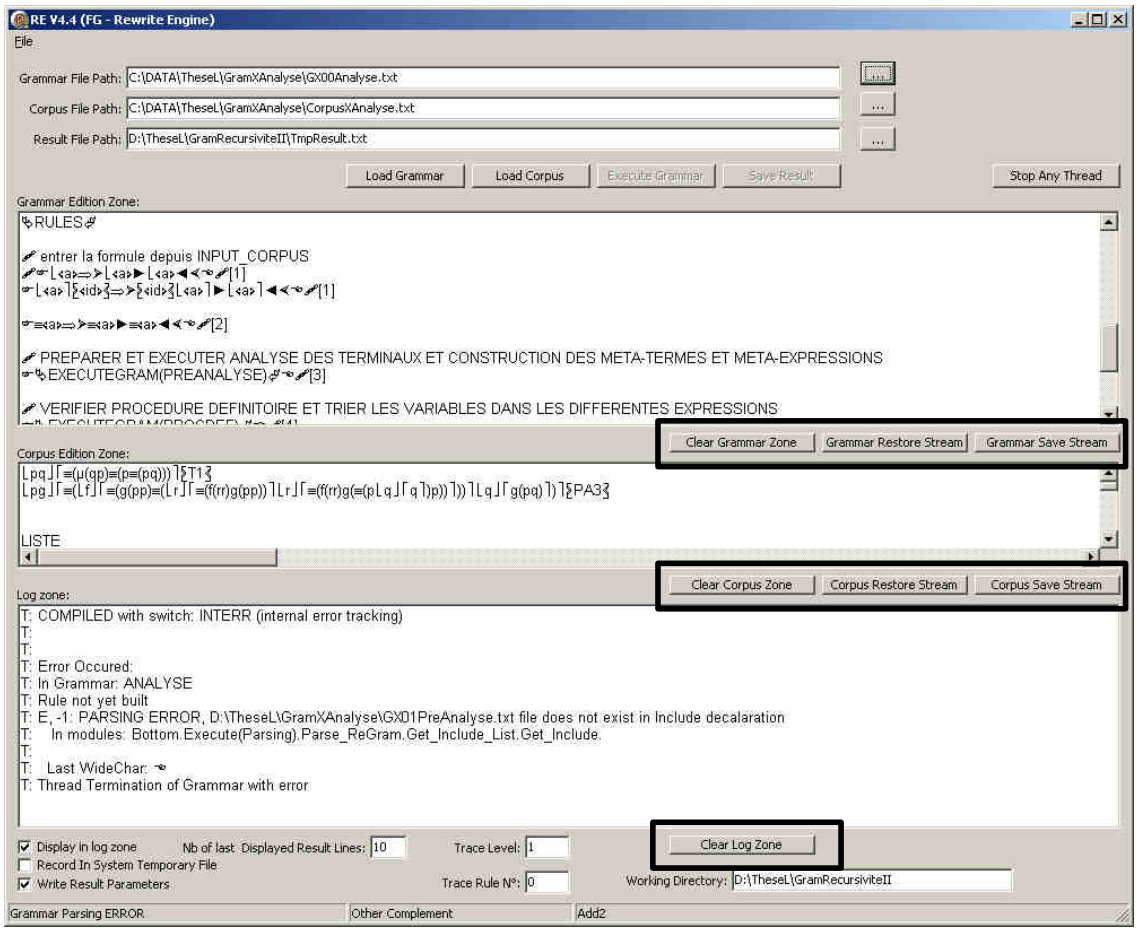


Figure 71. Les boutons associés aux zones d'édition ou d'affichage

Trois boutons sont associés aux deux zones d'édition des re-grammaires et du corpus d'entrée. Un bouton destiné à effacer tout le contenu de la zone : *Clear Grammar Zone* ou *Clear Corpus Zone*. Un bouton qui affiche dans la zone le contenu du fichier qui est indiqué dans les champs de la zone de gestion des fichiers : *Grammar Restore Stream* ou *Corpus Restore Stream*. Le troisième bouton permet de sauvegarder dans le fichier qui est indiqué dans le champ correspondant, l'état du contenu de la zone d'édition. La zone d'affichage, *Log zone* n'a qu'un bouton permettant d'effacer son contenu *Clear Log Zone*.

Dans la zone des cinq boutons d'interactions :

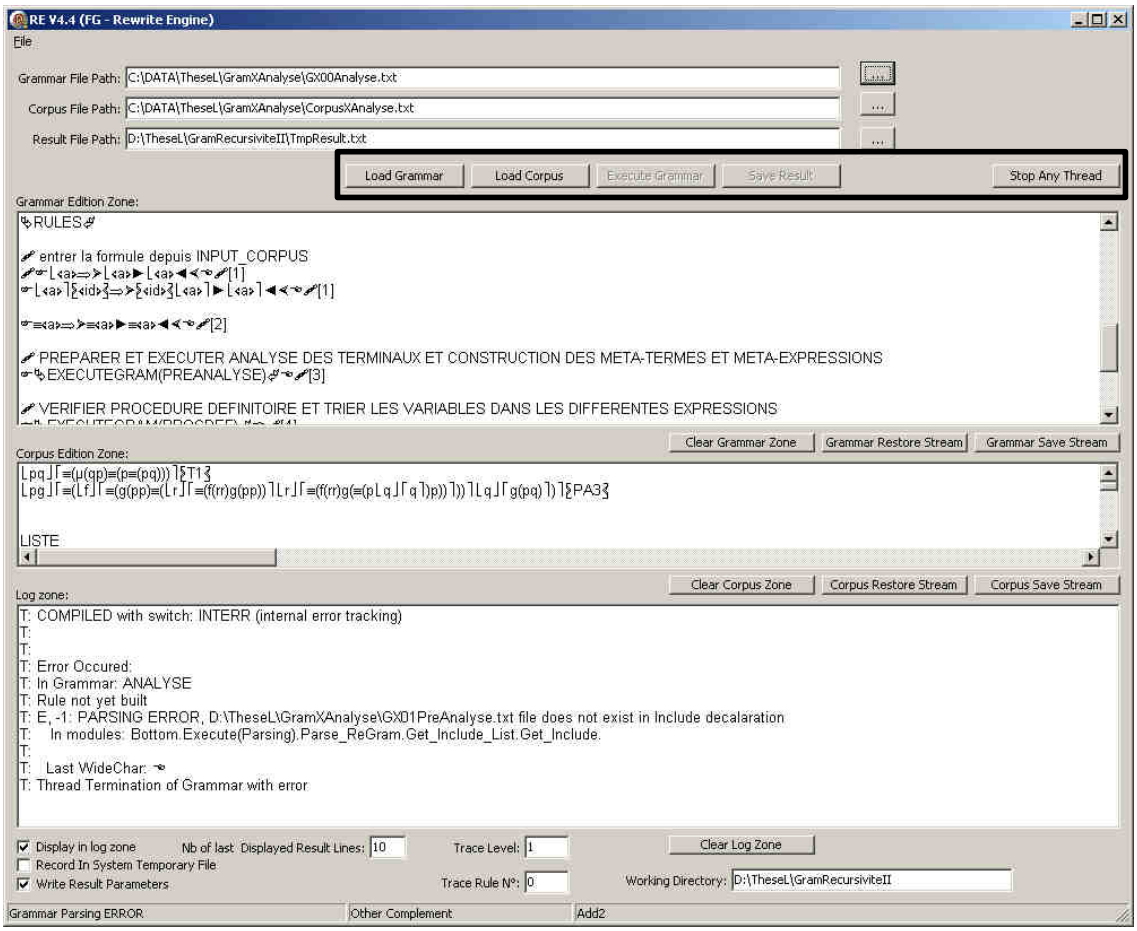


Figure 72. Les boutons d'interactions

Le bouton, *Load Grammar* a pour fonction de compiler une re-grammaire et d'afficher les résultats dans la zone : *Log zone*. Si la zone d'édition, *Grammar Edition Zone* est vide, le re-système charge cette zone avec le fichier indiqué dans le champ correspondant de la zone de gestion de fichier et compile ensuite la re-grammaire chargée.

Si la compilation réussit, il active le bouton, *Load Corpus*, qui a pour fonction de charger dans le re-système le corpus d'entrée. Si la zone, *Corpus Edition Zone* est vide, le logiciel commence par charger dans la zone le fichier indiqué dans le champ de gestion de fichier correspondant.

Si la compilation de la re-grammaire et le chargement du corpus réussissent, le bouton *Execute Grammar* est activé. Il a pour fonction d'exécuter la re-grammaire qui a été chargée au préalable par la compilation. Selon les paramètres d'exécution, l'exécution affiche une erreur, enregistre des résultats temporaires dans un fichier externe, affiche les résultats dans la zone, *Log zone*.

Si l'exécution réussit le bouton *Save result* s'active. Il permet de sauvegarder, non pas ce qui est affiché dans la zone Log zone, mais le résultat interne de l'exécution de chaque re-règle dans le fichier dont le nom est donnée dans le champ *Result File Path*.

Un dernier bouton, *Stop Any Thread* stoppe toutes les actions en cours du re-système.

Une dernière zone de paramètres est prévue principalement pour le dépannage d'erreurs dans une re-grammaire ainsi que pour la mise au point du logiciel :

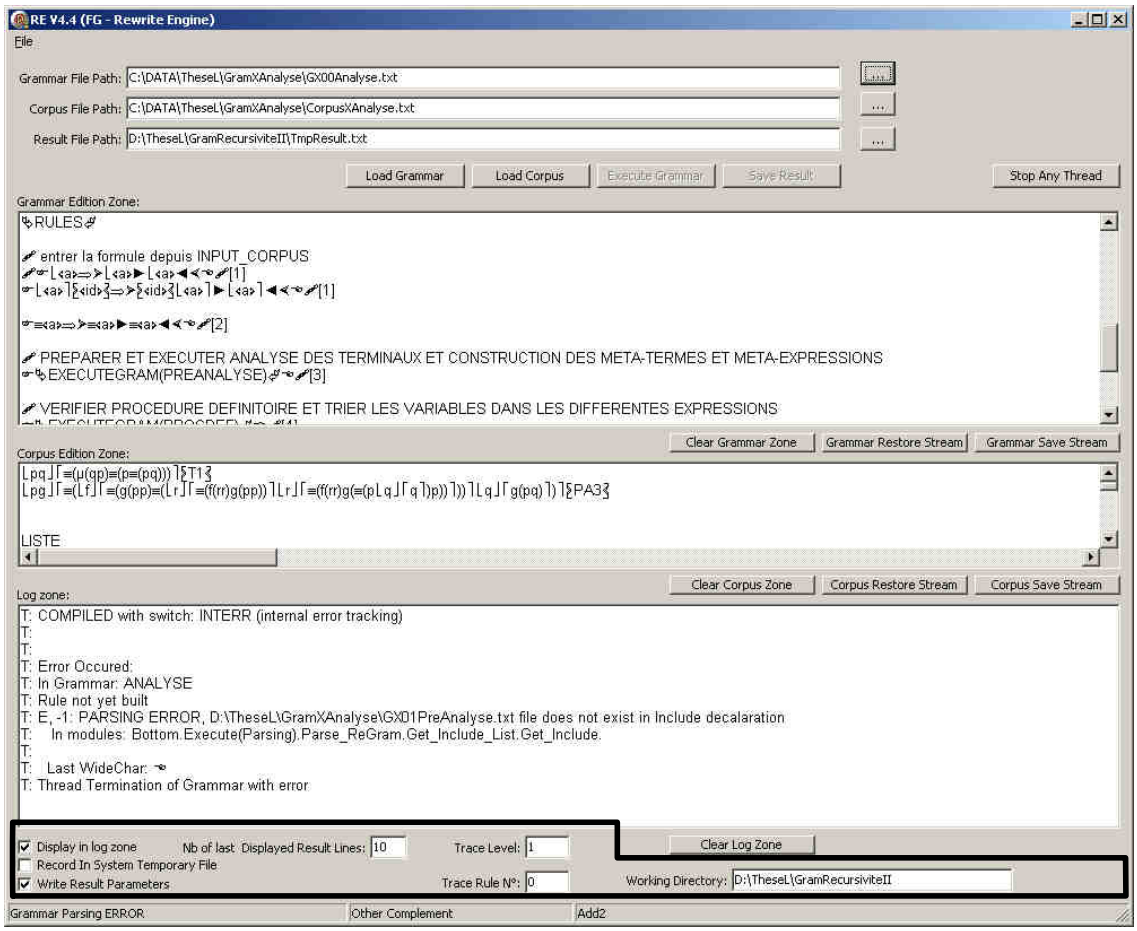


Figure 73. Zone de paramètres

L'affichage des caractères *unicodes* est très lourd dans les fenêtres Windows. L'affichage d'une grosse liste de résultats peut prendre plusieurs minutes, alors que de sauvegarder directement ces données dans un fichier ne prend que quelque secondes. On peut donc inhiber cet affichage en cliquant dans la case à cocher, *Dispaly in log zone*, ce qui active la case à cocher, *Record in System Temporary File*. Ce fichier se nomme *SysTmpResultFile.txt* et s'enregistrera dans le répertoire indiqué dans le champ *Working Directory*.

Lors de l'affichage des résultats des re-règles (de la réécriture des membres de droite), on peut décider d'avoir ou non en entête de la ligne le nom de la re-grammaire exécutée, le numéro de la re-règle et les nombre d'itérations. C'est pratique pour le dépannage et le développement. C'est plus gênant si l'on veut réutiliser les résultats comme *bibliothèque*, par exemple. On peut donc décider d'activer ou pas la case à cocher *Write Result Paramaters*.

Le champ numérique *Nb of Last Displayed Result Lines* indique au re-système combien de lignes (des dernières) il doit afficher dans la zone *Log zone*. Si l'on veut que tous les résultats s'affichent dans cette zone, on saisit : *all* dans le champ.

Le champ *Trace Level* indique le niveau de détail que l'on veut obtenir du traçage du processus. Introduire un chiffre plus grand ou égal à 3 dans cette zone affiche une partie de la traduction des re-règles en structure interne. Ce paramètre est en principe réservé au développement du logiciel. Il en va de même pour le champ *Trace Rule N°* qui stoppe le re-système à l'exécution de la re-règle saisie dans ce champ et cela pour autant que le logiciel ait été construit avec les paramètres qui le permettent.

* * *