UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN MATHÉMATIQUES ET INFORMATIQUE
APPLIQUÉES

PAR

MOHAMED HASSANI

VERS UN MODÈLE FORMEL APPLICATIF POUR UNE NOUVELLE INGÉNIERIE DE LA LANGUE ET DE L'INFORMATION

MAI 2016

Résumé

La lecture et l'analyse du texte, qui sont à la base de tout traitement en ingénierie de la langue et de l'information, sont un processus dynamique qui admet plusieurs points de vue, qui peut mener à différentes compréhensions et qui doit être réalisé selon des objectifs multiples.

Dans ce projet, nous développons une architecture pour l'ingénierie de la langue et de l'information qui sera flexible, modulaire, consistante et cohérente, et où chaque tâche sera représentée par une fonction autonome et indépendante du reste des fonctions de l'architecture. Les processus d'analyse seront des assemblages de telles fonctions. Un cadre théorique formel basé sur le modèle des systèmes applicatifs typés et de la logique combinatoire est proposé. Il permettra la construction de multiples processus d'analyse de textes selon des objectifs déterminés. Ce cadre donnera également la possibilité de systématiser la vérification de la cohérence de la séquence dans l'assemblage des fonctions d'un processus d'analyse donné.

Remerciements

Je tiens à remercier mon directeur de recherche Ismail Biskri pour son accueil bienveillant et ses précieux conseils tout au long de la réalisation de ce projet de recherche.

Une pensée particulière est adressée à tous les membres de ma famille qui m'ont soutenu financièrement et moralement pour réaliser mon projet d'études.

Je remercie également Boucif Amar Bensaber professeur à l'Université du Québec à Trois-Rivières, ainsi que tous mes amis et mes collègues du laboratoire LAMIA qui ont bien voulu partager leurs connaissances et répondre patiemment et aimablement à mes nombreuses questions.

Table des matières

Liste des figures

Page

Liste des tableaux

Page

Liste des abréviations, des sigles et des acronymes

ACP	Analyse en composantes principales
ALG	Groupe d'apprentissage automatisé
API	Application programming interface
CIL	Common Intermediate Language
CLI	interface de ligne de commande
CLR	Common Language RuntimE
D2K	Data to Knowledge
Expr	Expression
FN	Forme normale
GATE	General Architecture for Text Engineering
GAU	La grammaire applicative universelle
GCCA	Grammaire catégorielle combinatoire applicative
GNU	GNU's Not UNIX
GPL	Licence publique générale
GUI	Graphical user interface
ISO	Organisation internationale de normalisation
KNIME	Konstanz Information Miner
LAMIA	Laboratoire de mathématiques et d'informatique appliquées
MIR	Music Information Retrieval
ML	Machine learning
NAILI	Nouvelle Architecture de l'Ingénierie de la Langue et de l'Information
NCSA	National Center for Supercomputing Applications
NGS	séquençage de prochaine génération
OMT	Object Modeling Technique
PHP	Hypertext Preprocessor
POA	Programmation orientée aspect
POO	Programmation orientée objet
RFID	Radio Fréquence Identification
SDK	Software development kit
SE	Système expert
T2k	Text-To-Knowledge
TALN	Traitement automatique des langues naturelles

UIUC	Université de l'Illinois à Urbana-Champaign
UQTR	Université du Québec à Trois-Rivières
UTF-8	Universal Character Set Transformation Format - 8 bits
WEKA	Waikato Environment for Knowledge Analysis
WPF	Windows Presentation Foundation
XAML	eXtensible Application Markup Language
XML	Extensible Markup Language

Chapitre 1

INTRODUCTION

L'ingénierie de la langue et de l'information est un domaine inévitable qui interagit avec une panoplie de domaines (informatique, social, économique, scientifique, politique, culturel, etc.). Cette multidisciplinarité prend de plus en plus d'ampleur avec la croissance de la masse d'informations et des connaissances. Celles-ci, structurées ou non, sont présentes partout : dans les corpus, les bases de données textuelles et les différents types de réseaux sociaux généralistes (comme Twitter, Facebook, Myspace...), professionnels, de partage, de service, de rencontre, politique ou de géolocalisation. L'ingénierie de la langue et de l'information se base sur la modélisation symbolique, par le recours à des règles ou des statistiques du langage naturel de l'informatique. Elle étudie des approches, des méthodes et des modèles qui permettent de décrire ou d'acquérir des langues et des connaissances contenues dans des textes, soit au moyen de patrons grammaticaux, soit par des modèles empiriques et numériques.

Depuis les débuts de la linguistique informatique (1950) [1], nous pouvons citer les recherches et travaux des scientifiques œuvrant dans l'amélioration de ce domaine, mais nous pouvons aussi dire qu'il y a peu d'entre eux qui en dénoncent les limites. Par exemple, la réalisation de certains logiciels concepteurs, lesquels implémentent un certain modèle théorique d'un domaine quelconque, mais qui n'offrent pas de possibilités d'améliorations en permettant les mises à jour régulières. Il se peut que ce modèle demeure non reformulé de manière applicative comme désiré, ou bien que son implémentation ne réponde pas à certaines règles imposées par le chercheur ou l'analyste. Ceci dévalorise le modèle en question ou mène à une acquisition de valeur ajoutée lors de son exploitation, ou encore entraîne une utilisation non complète de ce logiciel. Cette problématique rétrécit à la fois l'émergence de l'ingénierie de la langue à de nouveaux domaines et son ouverture

en matière d'échange et de collaboration avec d'autres modèles de pensée provenant de différentes disciplines comme l'intelligence artificielle, les technologies langagières, la psychologie, la sémiologie, la logique, la philosophie, la terminologie, l'ontologie, etc.

De nombreuses solutions ont été proposées à ce genre de problème par le développement de systèmes experts et plateformes de logiciels. Celles-ci sont toutefois insatisfaisantes, car elles comportent certaines limites, qui seront présentées dans le présent travail. L'une d'entre elles provient de la liberté du code source de l'outil, ce qui limite sa réutilisation; une autre découle de la méthodologie de mise en place du paradigme de programmation suivi et les langages de programmation utilisés, ce qui impose une complexité à différents degrés. Une autre limite résulte de l'inclusion des modèles linguistiques non extensibles, ce qui limite encore l'évolution partielle du système. Par ailleurs, la conception n'est généralement pas établie pour permettre une interaction fluide avec d'autres logiciels, de plus, il est difficile d'ajouter de nouvelles fonctionnalités. Finalement, certains outils et la formation en permettant l'utilisation manquent de documentation, ce qui constitue un obstacle non négligeable pour les utilisateurs.

C'est sur cela que nous réfléchissons dans ce travail de recherche, afin de développer une nouvelle ingénierie de langue et de l'information pour une lecture et analyse du texte qui se base sur l'assurance d'un processus dynamique, lequel devra être réalisé selon des objectifs multiples, admettant en même temps plusieurs opinions et qui mènera à de nouvelles compréhensions.

Ce rapport détaillera notre projet de recherche en quatre grandes parties. La première partie porte sur l'état de l'art, contient une introduction au traitement des langues naturelles et de l'information, traite des systèmes experts avant d'étudier ceux qui existent en présentant leurs avantages et leurs limites, et présente des paradigmes de programmation et certains langages de programmation qui nous seront utiles. La deuxième

partie comportera des concepts de base et les outils fonctionnels, dont la logique combinatoire, les grammaires applicatives et cognitives, la grammaire catégorielle combinatoire applicative (GCCA) et le langage de programmation. La troisième partie sera consacrée à introduire le passage de la théorie vers l'application. Enfin, la quatrième partie inclura les sections d'implémentation, ainsi qu'une discussion des résultats.

Chapitre 2

ÉTAT DE L'ART

2.1 Introduction

Ce chapitre a pour objectif la présentation du contexte de notre projet. Dans un premier temps, les concepts utilisés dans le domaine du traitement des langues naturelles et de l'information seront introduits. Ensuite, sa relation avec l'analyse linguistique et son applicabilité sera discutée. Deuxièmement, les systèmes experts et leurs architectures seront présentés, puis la troisième partie discutera des avantages et limites des systèmes existants. Enfin, dans la quatrième partie, nous verrons une brève présentation des paradigmes de programmation, avant de voir quelques langages de programmation d'interfaçage qui seront utiles pour ce présent projet.

2.2 Le traitement des langues naturelles et de l'information

Le traitement des langues naturelles (TALN) est un champ multidisciplinaire qui regroupe la linguistique, l'informatique, la logique, les mathématiques et la philosophie du langage et du raisonnement [1].

Chaque société humaine a au moins une langue. Actuellement, on peut compter plus de 6 000 langues parlées [2], quoique beaucoup de langues sont en voie de disparition, faute de locuteurs. Bien que le vocabulaire puisse être acquis continuellement, toute personne normale est capable de tenir une conversation courante dans sa langue maternelle, et ce, vers l'âge de cinq ans, avant la maîtrise du raisonnement.

Les langues naturelles sont les langues humaines utilisées couramment; elles sont, en quelque sorte, des créations collectives spontanées auxquelles on ne peut attribuer de date de naissance précise. Elles s'opposent principalement aux langues formelles ou langues artificielles que sont notamment les langages de programmation informatiques ou la logique mathématique. Parmi les langues naturelles se trouvent également les créations linguistiques intentionnelles comme l'espéranto ou le volapuk, qui sont des langues internationales ou autres signes qui sont dédiés à être utilisés pour la communication. L'objet des sciences du langage est l'étude scientifique des langues naturelles. Les linguistes essaient de suivre les fonctionnements et universaux communs de ces langues ainsi que les structures qu'elles partagent. Ces linguistes ne sont pas nécessairement multilingues; ils cherchent à comprendre les règles qui régissent les langues, plutôt qu'à multiplier les connaissances qu'ils ont de certaines d'entre elles [3].

L'informatique joue un rôle de plus en plus important, à travers le domaine du traitement automatique du langage naturel (TALN) dans la linguistique. L'informatique ne se limite pas à la simple utilisation d'ordinateurs et de programmes. Le TALN est en fait l'héritier d'une longue tradition mathématique et logique de modélisation du calcul. On peut dire aussi que les fondements de l'informatique sont doubles : coder les données en 0 et 1 des éléments discrets et coder les traitements par des algorithmes.

C'est de cette manière qu'il est possible de s'approcher du traitement automatique du langage. Introduire une démarche informatique dans un domaine fait appel aux mêmes questions : Quelles sont les données pertinentes de ce domaine? Quels sont les traitements pertinents de ce domaine? Comment les coder? [3]

Pour maîtriser une langue, nous sommes invités à manipuler, nécessairement, de nombreuses données, et à mettre en œuvre de nombreux traitements. Les linguistes les ont progressivement mis à jour et caractérisés, alors que les informaticiens ont

progressivement contribué à les modéliser. Ces interactions font l'objet de la naissance du TALN [3].

L'informatique apporte non seulement une perspective nouvelle pour la démocratisation et le développement de la communication linguistique par l'outil informatique, ce qui a eu pour conséquence une multiplication exponentielle des corpus linguistiques, mais elle apporte aussi des outils de traitement et de gestion de masses de données qui sont incomparablement supérieurs à ceux que peut développer un humain. De ces apports, les sciences du langage peuvent et doivent dorénavant profiter de l'informatique pour conduire leurs recherches et descriptions linguistiques.

2.2.1 L'analyse linguistique et le TALN

L'analyse est un concept partagé par plusieurs matières. Dans notre domaine, nous nous limitons à la signification linguistique, philosophique, logique, mathématique et numérique. Le sens linguistique de l'analyse repose sur une matière première d'exemples réels ou fictifs, grammaticaux ou fautifs. Le sens philosophique repose sur des idées et des concepts. Le sens logique, comme les mathématiques, se base sur les systèmes symboliques suffisamment cohérents pour exprimer formellement le traitement des phénomènes langagiers et les expliquer. Le sens numérique se concentre sur l'exploitation des récurrences et cooccurrences dans les corpus pour créer des patrons de sens ou signification. Autrement dit, l'analyse se base sur un empirique réel non observable par l'homme [1]. Cela montre donc que l'analyse linguistique répond spécialement aux besoins des spécialistes en traitement automatique plus qu'à ceux des autres scientifiques mentionnés plus haut.

2.2.2 L'application et le TALN

Le Larousse définit « application » comme étant l'action d'appliquer quelque chose, de poser ou d'étendre une chose sur une autre pour qu'elle y adhère; ou encore, il

la définit par l'action d'employer quelque chose à une fin déterminée ou de le mettre en pratique. En effet, l'application, dans le domaine du traitement automatique des langues naturelles, nécessite le travail des chercheurs (leurs actions) afin de mettre en place une analyse informatique permettant de tester la robustesse de la théorie. De là émane la constatation d'un échange entre l'application et l'analyse permettant l'enrichissement de chacune.

Aussi, les études effectuées par les chercheurs afin de comprendre les problèmes en matière de recherche sont renforcées par l'application qui mène à découvrir de nouvelles problématiques à résoudre théoriquement. Ces études respectent un cadre disciplinaire ou interdisciplinaire bien défini pour pouvoir, en utilisant l'application, répondre concrètement à des besoins au niveau de la lecture assistée par ordinateur, de la prédiction, de la prise de décision, de la recherche de l'information, du traitement des langues, etc. [1].

2.3 Les systèmes experts

2.3.1 Définition

Selon J.C. Pomerol, un système expert (SE) est un outil informatique d'intelligence artificielle, lequel permet la simulation du savoir-faire d'un spécialiste dans un domaine bien déterminé, grâce à l'exploitation d'un certain nombre de connaissances fournies explicitement par des experts du domaine. La simulation du savoir-faire d'un spécialiste dans un domaine bien déterminé, grâce à l'exploitation d'un certain nombre de connaissances fournies explicitement par des experts du domaine.

Par l'utilisation d'un SE, il est possible de modéliser le raisonnement d'un expert, de manipuler des connaissances sous une forme déclarative, d'en faciliter l'acquisition, la

modification et la mise à jour et de produire des explications sur la façon dont sont obtenus les résultats d'une expertise [4].

Dans certains domaines, comme le traitement automatique des langues naturelles, le développement d'applications ou la gestion de l'espace, un système expert peut être un outil d'aide à la décision parce qu'il permet de tenir compte de variables quantitatives et qualitatives pour construire une base de connaissance. Ainsi, la structuration du savoir de façon logique, son organisation pour construire un modèle de simulation et la proposition de réponses prospectives sont possibles.

L'utilisation d'un système expert est indépendante de la connaissance concernée; c'est-à-dire qu'il peut être utilisé par des non-experts dans le domaine de la connaissance ou la technique de modélisation. Et ceci, sans considérer explicitement la spatialisation des phénomènes. Aussi, elle offre une satisfaction aux utilisateurs, relativement au contexte, vis-à-vis de leurs analyses et de leurs prises de décision tant qu'ils ne connaissent pas mieux.

2.3.2 L'architecture des systèmes experts

L'architecture des systèmes experts est caractérisée par :

- Une base de connaissance qui est à la fois le savoir-faire et l'expertise nécessaires pour résoudre un problème. Les unités de raisonnement sont représentées par des règles libellées de la manière suivante,

  Si "situation" alors "action"

où "situation" désigne l'hypothèse de la règle et "action" désigne la conclusion.

- Une base de fait, c'est la mémoire qui regroupe à tout instant les informations connues sur le problème en cours, qui s'enrichit au fur et à mesure par les réponses de l'utilisateur, c'est un moteur d'inférences qui se compose des algorithmes chargés d'exploiter les connaissances et les faits pour en tirer un raisonnement. Il exploite les connaissances en effectuant des déductions logiques et, à partir des mêmes hypothèses que les experts, il propose des résultats identiques [4].

En fonction de la situation courante désignée par la base de fait, le moteur d'inférences choisit les connaissances de la base à utiliser et leurs enchaînements et il construit le raisonnement adapté au cas en question par la répétition d'un cycle de choix des règles à utiliser et à déclencher parmi les règles détectées pertinentes.

Les règles pertinentes se définissent en fonction des algorithmes de résolution inclus dans le moteur; ceux utilisés plus couramment sont le chaînage avant (partant des faits pour en rechercher les conséquences) et le chaînage arrière (partant des conclusions envisagées pour vérifier si les conditions sont réunies).

2.4 L'étude des plateformes existantes

Nous avons étudié cinq systèmes qui œuvrent dans le domaine de traitement de données et de l'information en offrant des fonctionnalités de modélisation. L'étude de l'existant permettra d'établir une base de connaissance en termes d'avantages et de limites afin d'en profiter pour mener à bien notre projet de recherche. Ces plateformes sont D2K/T2k, GATE, WEKA, KNIME et Pipeline Pilot.

2.4.1 Avantages et limites de D2K/T2K

La plateforme D2k/T2k est une suite d'outils d'apprentissage automatique mise au point par le National Center for Supercomputing Applications (NCSA) du Groupe

d'apprentissage automatisé (ALG) à l'Université de l'Illinois à Urbana-Champaign (UIUC) [5]. L'idée de base de l'apprentissage automatique (machine learning, ou ML) est de donner à un programme ML une collection d'éléments au sein de certains ensembles connus de propriétés, puis le ML cherche à «apprendre» les règles, afin de savoir pourquoi les articles appartiennent à un ensemble particulier. L'exemple classique est le pourriel (*spam e-mail*). Dans ce cas, il suffit de donner au programme Machine Learning (ML) une collection de courriels qui ont été catégorisés comme pourriels ou non. Le programme ML analyse ensuite les caractéristiques de chaque groupe et arrive avec un modèle pour la reconnaissance de courriels comme pourriel ou non. Grâce à ce modèle, neuf courriels non lus peuvent être analysés de manière similaire et le programme ML décide si oui ou non c'est un pourriel [5].

Cette plateforme est construite en Java et c'est une solution dépendante. Ses composants sont des modules qu'on peut classer en six catégories sur lesquelles on se base pour créer de nouveaux modules, et ce, par l'héritage [6].

Les fonctionnalités que D2K présente sont comme suit:

Extensions des API existantes

- Fournit la capacité de connecter, par voie de la programmation, des modules et des propriétés définies.
- Permet aux applications entraînées par D2K à être développées.
- Fournit la capacité de faire une pause et de redémarrer un processus.

Développement distribué amélioré

- Permet aux modules en entrée d'être exécuté à distance.
- Utilise les services Jini pour rechercher des ressources distribuées.

- Comprend l'interface pour spécifier la mise en page de l'exécution d'un processus distribué.

Processeur Statut Overlay

- Affiche l'utilisation des ressources informatiques distribuées.

Mise au point distribuée

Gestionnaire de ressources

- Fournit un mécanisme pour traiter des structures de données sélectionnées comme si elles sont stockées dans la mémoire globale.
- Offre un espace de mémoire accessible à partir de plusieurs modules exécutés localement ou à distance.

Ensemble de traitement / de services Web [7]

L'un des avantages de cette solution est que ces fonctionnalités peuvent être associées à d'autres solutions pour traiter différents types de données comme la musique. Des chercheurs de l'entreprise One Llama Media ont travaillé sur D2K / T2K / m2K, leur objectif étant de construire un système de la recherche d'information musicale e (Music Information Retrieval ou MIR) qui combine la puissance des programmes basés audio d'apprentissage automatique avec la connaissance sociale de mélomanes, créant ainsi un environnement de découverte musicale approfondi et de gestion. De fait, avec l'utilisation de ce système, il est possible de profiter des apports des systèmes intelligents intégrables avec d'autres systèmes en réunissant les aspects technologiques et l'expérience humaine [5].

Malgré toutes les fonctionnalités de cette plateforme, elle demeure limitée par la nécessité d'avoir les connaissances de ces outils diversifiés pour pouvoir réussir

l'assemblage des modules et, dans certains cas, connaître le code de programmation [8]. Elle rend difficile son intégration ou son utilisation.

2.4.2 Avantages et limites de GATE

GATE est une plateforme qui répond aux besoins de l'analyse de texte, la technologie sémantique ou de la transformation générale de la langue. Les avantages d'une solution à base de GATE se manifestent dans son offre d'outils mis en place dans un seul paquet. Ces sources de données et de traitement sont libres (*open source*, documenté, maintenu), extensibles (avec des API bien définies et points de terminaison de service Web) et réutilisables (le code est présent dans plus de demandes que tout autre système similaire). Ce système est développé dans les normes ISO (TC37), des référentielles sémantiques standards d'Ontotext, ce qui rend l'utilisation facile des applications et processus. Le fait que le système a de nombreuses composantes *Plug-and Play* et que les utilisateurs peuvent voir où les ensembles de règles ou l'ontologie ou les algorithmes en forme, leur permette d'avoir une idée de la façon dont le système fonctionne [9].

Limites

Les services de système GATE ne sont pas destinés à une large catégorie d'utilisateurs, mais ils les offrent directement aux développeurs qui connaissent bien son infrastructure et peuvent réaliser des codes permettant son intégration. Ainsi, son utilisation reste limitée aux spécialistes de la programmation informatique. Les linguistes ou scientifiques qui travaillant sur les données devront s'efforcer d'apprendre les langages des outils de GATE et leurs efforts ne seront pas consacrés en la résolution de leurs propres problèmes.

2.4.3 Avantages et limites de WEKA

WEKA (Waikato Environment for Knowledge Analysis) signifie, en français, environnement Waikato pour l'analyse de connaissances; c'est une plateforme d'apprentissage automatique. Développée en Java à l'Université de Waikato, Nouvelle-Zélande. WEKA est un logiciel libre disponible sous la licence publique générale GNU (GPL) [10].

Les principaux avantages de WEKA sont:

- WEKA est disponible en source libre et est gratuit sous la licence publique générale GNU,

- Sa portabilité, car il est entièrement implémenté en Java et donc fonctionne sur plusieurs plateformes modernes, et en particulier sur presque tous les systèmes d'exploitation actuels,

- WEKA contient une collection complète de préprocesseurs de données et de techniques de modélisation ;

- WEKA supporte plusieurs outils d'exploration de données standards, en particulier des préprocesseurs de données, des agrégateurs de données (*data clustering*), des classificateurs statistiques, des analyseurs de régression, des outils de visualisation, et des outils d'analyse discriminante [11].

Limites

Comparativement à l'environnement de développement multiplateforme Rstudio pour R, un langage de programmation utilisé pour le traitement de données et l'analyse

statistique, WEKA n'est pas capable de faire de l'exploration de données multirelationnelles, mais il existe des logiciels tiers pour convertir une collection de tables de base de données liées entre elles en une table unique adaptée au traitement par WEKA. Même si elle n'est pas si difficile à apprendre, l'interface graphique n'est pas aussi bien documentée ou intuitive que Rstudio [12]. WEKA propose trois façons d'utiliser le logiciel : l'interface graphique, une API Java, et une interface de ligne de commande (CLI).

En plus du problème de la documentation pour WEKA, la manipulation des ensembles de données est beaucoup plus facile avec R qu'avec WEKA.

Certaines fonctionnalités de cette plateforme ne sont pas facilement manipulables comparativement à un logiciel concurrent. Par exemple, faire une analyse qui combine l'analyse en composantes principales (ACP) avec la classification hiérarchique. Ce fut un processus très facile à mettre en place dans la R, mais qui s'avéré trop difficile pour WEKA. Alors que WEKA propose des modules à la fois pour l'ACP et le *clustering*, l'utilisateur a des difficultés à les combiner sans avoir à recourir à l'écriture de code Java personnalisé.

WEKA GUI fournit plusieurs panneaux intégrés 'visualisation', mais ceux-ci sont très limités, surtout lorsqu'on les compare à ce qui peut être fait avec les progiciels R comme Ggplot. La visualisation des données est l'une des grandes forces de R. Dans WEKA l'approche à la visualisation se concentre sur la compréhension du comportement des algorithmes de l'intelligence artificielle, plutôt que des ensembles de données.

2.4.4 Avantages et limites de KNIME

Le système Knime est une plateforme d'analyse de données qui permet à l'utilisateur de construire un flot de traitement de données et de comparer et de modifier des algorithmes de fouille de données ou de classification. Il dispose d'une interface utilisateur graphique pour combiner les nœuds. Les collections de nœuds sont connues comme des extensions. KNIME est basé sur Eclipse, la plateforme libre et Java [13].

Au départ, il y avait très peu de nœuds de chimie pour une utilisation avec KNIME. Cependant l'addition d'un nouveau menu de programmation Java, beaucoup de nœuds ont été ajoutés.

KNIME utilise une méthodologie de flux de travail, selon laquelle, lorsque la tâche 1 est terminée, les données sont transmises au large à la tâche 2, laquelle est terminée avant que les données soient remises à la tâche 3 et ainsi de suite [14].

Le traitement table par table de KNIME offre des avantages : les multiples itérations sur les mêmes données (important pour de nombreuses données sur les algorithmes d'extraction), la capacité de toujours afficher les résultats intermédiaires sur les connexions entre les nœuds, même après que le flux de travail a été exécuté, et la capacité de redémarrer le flux de travail à tout nœud intermédiaire. La pénalité consiste cependant en la nécessité de stocker les données quelque part, mais il est plus facile à mettre en cache les données à la fin de chaque tâche. Dans un pipeline de données, une cache de toutes les données peut être ajoutée en tant que composant « finir ici et reprendre ».

Enfin, même si KNIME est libre et moins cher, il est moins facile à utiliser. Par exemple, l'outil KNIME Analytics Platform est libre et gratuit avec la possibilité

d'acquérir des extensions selon le besoin. Cependant, la facilité d'utilisation est un critère subjectif, et la familiarité avec un autre système peut avoir une incidence sur elle [14].

KNIME Server ajoute les fonctionnalités complètes basées sur le serveur, y compris l'exécution à distance et prévue des flux de modélisation, un référentiel du flux de travail, des données partagées dans un référentiel géré, méta-nœuds partagés ou les flux de travail pour la réutilisation de composants, et l'accès au navigateur Web. L'interface du navigateur expose des flux de travail pour les utilisateurs qui peuvent entrer les paramètres définis pour un flux de travail et ainsi obtenir un résultat affiché dans le navigateur [13].

2.4.5 Avantages et limites de Pipeline Pilot

Pipeline Pilot est une plateforme qui permet aux utilisateurs de composer graphiquement des protocoles, en utilisant de différentes composantes configurables pour des opérations telles que la récupération de données, la manipulation, le filtrage informatique, et l'affichage. Donc, ce système permet la création scientifique graphique pour optimiser le processus de l'innovation en recherche, augmenter l'efficacité opérationnelle et de réduire les coûts à la fois pour la recherche et pour les technologies de l'information. Il permet également d'automatiser l'analyse scientifique des données, afin de donner la possibilité aux utilisateurs dans toute l'entreprise d'explorer, de visualiser et de présenter les résultats de recherche [15].

Java fait partie de la base de Pipeline Pilot et les programmeurs peuvent créer de nouveaux composants avec les composants API de Java ou de développer de nouvelles applications clientes, à l'encontre de SDK de Java. En outre, Pipeline Pilot dispose de son propre langage de script (pour les non-programmeurs); par ailleurs, il a beaucoup plus de technologie chimio-informatique intégrée et le script est plus concis.

Pipeline, comme Pipeline Pilot, se caractérise par une méthodologie de flux de travail où la tâche 1 est terminée sur le composant 1 et les données sont transmises à la tâche 2. La tâche 1 peut alors commencer sur le composant suivant. Cette transmission explique l'existence des flux de données de la tâche 1 vers la tâche 2. Le processus peut évoluer sans impact sur la mémoire, et l'efficacité est acquise si une opération en aval peut être commencée sur certains enregistrements pendant qu'une opération en amont travaille toujours sur les autres.

Dans certains marchés, ils ne font pas concurrences aux autres logiciels; Pipeline Pilot n'est pas conçu pour les applications non scientifiques; il a plutôt un rôle distinct à jouer au sein du portfolio de logiciels de Accelrys.

Enfin, les utilisateurs disent que Pipeline Pilot est très cher mais facile à utiliser.

Cet outil offre aux spécialistes de la flexibilité, car des algorithmes ont été implémentés et déployés comme des composants dans l'environnement de forage de données où les scientifiques ont été capables d'avoir une grande flexibilité chimique dans la définition de leurs propres flux de travail. Ces flux de travail permettent l'implémentation rapide de tâches complexes et l'automatisation de leurs exécutions. Les composants d'analyse de Pipeline Pilot peuvent être couplés avec des algorithmes faits maison ou d'autres algorithmes provenant de vendeurs tiers pour permettre de multiples possibilités quant à l'implémentation de tâches dans des protocoles [16].

Initialement, Pipeline Pilot a été utilisé principalement dans le domaine de la chimio-informatique. Il est maintenant utilisé dans d'autres domaines scientifiques comme le séquençage de prochaine génération (NGS) et l'imagerie. Aussi, grâce à la possibilité de l'évolution, Pipeline Pilot peut être utilisé pour les déploiements de masse de données provenant de ces domaines, ce qui est pratique pour les entreprises qui souhaitent l'utiliser.

L'un des avantages de Pipeline Pilot est d'offrir plusieurs options de sauvegarde et d'enregistrement de données. Cependant, la sauvegarde présente quelques désavantages, comme dans certains exemples d'applications avec une masse de données et surtout lors de l'utilisation d'un outil de sauvegarde intégrable dans Pipeline Pilot [15].

Par ailleurs, l'outil Ftrees Writer, qui n'agit pas dans le domaine de traitement de texte, mais celui de molécules, permet d'écrire un jeu de données de molécules aux FTrees spéciales dans un format de fichier de base de données Pipeline Pilot. Alternativement, il est possible de stocker les molécules avec le descripteur de *Feature Tree* utilisant l'éditeur de cache *Cache Writer* ou au format SD molécule en utilisant l'éditeur de SD 'SD Writer'. Nous pouvons voir les avantages et les inconvénients de chaque format énumérés dans le tableau ci-dessous :

Tableau 1 les avantages et les inconvénients de chaque format de stockage [17].

Reader/Writer type:	**FTrees**	**Cache**	**SD**
Write Speed (secs)	26	6	3
File Size (MB)	58	36	113
Read Speed (secs)	17	5	3
User Defined File Location	Yes	No	Yes
ASCII Format File	No	No	Yes

Les valeurs indiquées ici sont pour le traitement d'un petit ensemble de données molécule typique contenant un peu plus de 33 000 molécules. Les temps sont ceux rapportés dans le pilote Pipeline GUI. Il suffit donc de comparer le temps relatif de chaque processus [17].

« User Defined File location » se traduit par « l'emplacement du fichier défini par l'utilisateur ». Autrement dit, l'utilisateur entre dans l'emplacement où le fichier doit être stocké sur le disque. Le fichier est alors indépendant de Pipeline Pilot et peut être déplacé ou supprimé, etc. à l'extérieur de Pipeline Pilot. La cache (« Cache ») est un type de stockage interne spécial dans Pipeline Pilot. À noter que la documentation Pipeline Pilot mentionne que:

Dès que les caches sont créées avec des champs User Only *et* All Users, *ils peuvent être accessibles par de multiples tâches d'exécution. L'utilisateur doit savoir que des problèmes peuvent survenir si deux tâches essaient d'écrire en même temps dans la même mémoire cache partagée. Aussi, pour gagner de l'espace sur le disque, l'utilisateur est responsable de la suppression de ces caches lorsqu'il termine, en utilisant le composant* Delete Cache.

Par contre, il existe des protocoles de stockages recommandés pour éviter les problèmes liés au temps de stockage et de l'espace. La figure suivante représente un de ces protocoles pour Ftrees dans Pipeline Pilot [17].

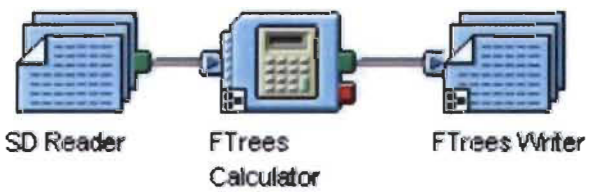


Figure 1 Protocole de stockages recommandé pour FTrees dans Pipeline Pilot [17]

2.4.6 Les limites communes des plateformes étudiées

Le tableau suivant regroupe septes limites des plateformes que nous avons étudiées: D2K/T2K, GATE, WEKA, KNIME et Pipeline Pilot. Ce dernier nous aidera à déterminer les limites communes de ces solutions existantes.

Tableau 2 Les limites communes des plateformes étudiées

Plateformes / Limites	D2K/T2K	GATE	WEKA	KNIME	Pipeline Pilot
Intégration déficile	●	●	●		●
Connaissances préalables des fonctionnalités offertes	●			●	●
La catégorie d'utilisateurs n'est pas large	●	●			●
Nécessite des connaissances des languages de programmation	●	●	●	●	●
Manque de documentation	●		●		
La visualisation et l'exploration de données sont limitées			●		
Problèmes liés au stockage de données				●	●

Nous déduisons d'après l'étude des plateformes existantes et le tableau des limites communes que toutes ces plateformes ne peuvent être utilisées qu'avec des connaissances préalables de leurs outils et/ou fonctionnalités offertes. Puis l'intégration demeure très difficile pour D2K/T2K, GATE, WEKA et Pipeline Pilot. La connaissance de certains langages de programmation comme Java demeure un prérequis pour utiliser la majorité de ces plateformes. Ainsi, la catégorie des utilisateurs se limite pour les informaticiens et les spécialistes du domaine surtout pour les plateformes D2K/T2K, GATE et Pipeline Pilot. Le manque de documentation s'ajoute à son rôle comme limite pour D2K/T2K et WEKA ce qui nuit à l'apprentissage pour exploiter les services de ces plateformes. Les opérations de stockage de données dans KNIME et Pipeline Pilot sont aussi limitées avec la visualisation et l'exploration de données dans WEKA, ceux-ci engendrent une difficulté pour les utilisateurs dans la manipulation des données.

2.5 Les paradigmes de programmation

Il existe plusieurs méthodes de programmation dans le domaine informatique. Aussi, plusieurs langages de programmation permettent de répondre aux besoins de traitement de l'information, et ce, depuis l'apparition des ordinateurs. Un langage est une façon d'implanter des règles. Les paradigmes de programmation que nous allons survoler dans cette section aideront à réaliser une conception réussie des différentes couches de notre projet. Effectivement, lors de la construction de notre système, nous devons nous appuyer sur des outils existants et réutiliser certaines bases.

Cette section présentera trois paradigmes afin de pouvoir analyser et mieux connaitre leurs lignes directrices. Ces dernières sont des règles dictant la manière dont les programmes doivent être formulés dans un langage de programmation implémentant le paradigme choisi. Les paradigmes que nous étudierons sont : la programmation orientée agent, la programmation orientée objet, la programmation orientée aspect paradigme et la programmation fonctionnelle.

2.5.1 Programmation orientée objet

La programmation orientée objet (POO) vise à structurer les objets pour rendre les applications plus modulables, c'est-à-dire réutilisables et extensibles (concept d'héritage).

Ce sont les besoins en termes de logiciels qui ont mené à cette méthode de programmation, car ils changent rapidement et ces derniers sont de plus en plus présents dans la vie de tous les jours. Les logiciels deviennent plus complexes et l'effort de conception de ceux-ci est en constante croissance. Les techniques de programmation devaient donc s'adapter. La POO et ses techniques de modélisation, comme OMT (*Object*

Modeling Technique), permettent d'enrichir la programmation procédurale en y ajoutant des concepts essentiels comme l'encapsulation, l'abstraction et le polymorphisme [18].

Parmi les avantages de la programmation orientée objet :

- Elle facilite l'organisation, la réutilisation, la méthode plus intuitive, permet l'héritage, facilite la correction, donne des projets plus faciles à gérer. L'intérêt principal de la POO demeure dans la possibilité de décrire par le code des actions à réaliser de façon linéaire, mais par des ensembles cohérents appelés objets.

- Elle est facilement concevable, car son code décrit des entités comme il en existe dans le monde réel. Dans un modèle à objets, toute entité du monde réel est un objet, et réciproquement, tout objet représente une entité du monde réel.

- Elle permet en quelque sorte de factoriser le code en ensembles logiques.

- Les informations concernant un domaine étant centralisées en objets, il est possible de sécuriser le programme par la gestion d'accès à ces objets aux autres parties du programme.

- La POO permet de diviser la complexité en partageant deux parties, une pour la conception des objets et l'autre pour l'utilisation de ces objets dans les programmes.

La POO montre quelques limites dans certains cas. Prenons, par exemple, un programme gérant des commandes client en objet. Si nous souhaitons supprimer un client

de notre application, nous devons vérifier si une commande n'est pas en cours pour celui-ci, sachant qu'une commande doit être obligatoirement liée à un client. Cela engendre une contrainte d'intégrité des données. Dans notre problème, et avec la logique POO, ni l'objet commande, ni l'objet client ne peut effectuer cette opération, car ce n'est pas de leur responsabilité. Il ne serait pas logique de demander à une commande de supprimer un client, ni à un client de vérifier qu'une commande est en cours. D'une autre manière, pour respecter la logique POO, l'objet client ne doit pas contenir de commandes.

La dispersion de code est une autre limite de la POO. L'un des désavantages de ce phénomène est la gestion des traces. Par exemple, si vous voulez faire un affichage à chaque appel de méthode, ou d'instanciation d'objet, vous serez obligé d'ajouter une ligne de code avant chacun de ces points du programme. Il y a donc une copie fréquente de codes dans l'ensemble du programme pour ajouter cette seule fonctionnalité de gestion de traces.

Les principaux concepts de la POO sont :

Objet et classe: le concept d'objet associe dans une même entité informatique les données et les traitements définis au sein d'une classe. Chaque objet est une instance de classe et est défini par ses attributs et par des actions appelées méthodes. Alors, un objet est une entité avec un état variable en fonction des messages qu'il reçoit. Les opérations/actions/messages appelés méthodes que l'on peut effectuer sur un objet définissent son interface [19].

L'encapsulation: les détails d'un objet ne sont accessibles que par ses méthodes visibles, mais ses attributs sont cachés. Alors, toute modification demeure locale, ce qui accroit la modularité et l'évolutivité d'un programme [19].

L'héritage: il consiste en un mécanisme de transmission des propriétés d'une classe vers une sous-classe. Les notions de classe mère et classe dérivée en découlent. Une classe dérivée hérite des propriétés de la classe mère, mais peut cependant avoir ses propres propriétés. Ainsi, le type d'héritage peut être simple ou multiple selon qu'une classe hérite d'une seule classe ou de plusieurs classes [19].

Polymorphisme : il est possible de redéfinir une opération héritée pour pouvoir lui donner une implémentation différente afin de tenir compte des propriétés locales non héritées. La même signature d'une opération que celle de la classe mère est redéfinie, cependant, elle s'applique aux instances de la classe dérivée. Alors, une opération redéfinie peut prendre différentes implémentations, en fonction du type d'objet auquel elle s'applique. [19]

2.5.2 Programmation orientée aspect

La programmation orientée aspect (POA) est un paradigme de programmation qui permet de traiter séparément les préoccupations transversales, qui relèvent souvent de la technique (aspect en anglais) et des préoccupations métier, qui constituent le cœur d'une application.

Ce paradigme propose deux principaux concepts d'aspects permettant l'encapsulation de ces préoccupations transversales et de tissage pour intégrer le code des aspects et le code métier. Pour détailler le fonctionnement général de la programmation orientée aspect, nous l'avons illustré dans la figure suivante. Les solutions construites en utilisant la POO peuvent être représentées par une architecture composée d'un code métier (fonctionnel) qui utilise le code technique (non fonctionnel). La POA propose d'encapsuler le fonctionnel dans des aspects qui seront tissés par la suite avec le code non fonctionnel afin d'aboutir à une nouvelle solution contenant la fonctionnalité technique [20].

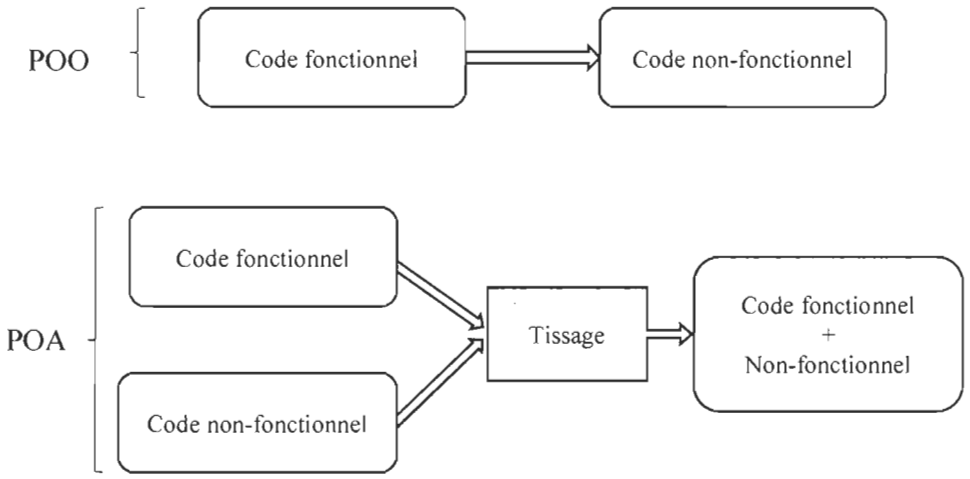


Figure 2 le fonctionnement de la POA

La programmation orientée aspect est capable de résoudre la problématique des fonctionnalités transversales, et ce, par la mise en place d'une la fonctionnalité de vérification d'intégrité des données avec l'aide d'un aspect, ce que la POO ne peut pas faire dans le premier cas cité dans la section précédente.

Pour le phénomène de la dispersion de code présenté dans la section précédente comme deuxième limite, la programmation orientée aspect est capable de résoudre ce problème, en ajoutant un aspect qui capture les appels de méthodes dont le code Greffon (*advice*) serait l'affichage de la trace. Cela est de plus très pratique, car il peut non seulement être implémenté facilement et rapidement dans un programme, mais aussi facilement supprimable.

La POA a son jargon spécifique, composé des concepts suivants :

- **Aspect :** un module définissant des greffons et leurs points d'activation,

- **Greffon (*advice*) :** désigne une méthode définissant le code non fonctionnel qui sera exécuté lors de certains points de jonction définis par un point de coupe. C'est un programme qui sera activé à un certain point d'exécution du système, précisé par un point de jonction [20],

- **Tissage ou tramage (*weaving*) :** insertion statique ou dynamique dans le système logiciel de l'appel aux greffons,

- **Point de coupe, d'action, de coupure ou de greffe (*pointcut*) :** désigne un ensemble de points de jonction et permet éventuellement l'accès à certaines valeurs dans le contexte d'exécution de ces points de jonction. Autrement dit, c'est un endroit du logiciel où est inséré un greffon par le tisseur d'aspect;

- **Point de jonction ou d'exécution (*join point*) :** point précis et bien défini dans l'exécution d'un programme où il est possible d'exécuter du code transversal [20]. Autrement dit, un endroit spécifique dans le flot d'exécution du système, où il est valide d'insérer un greffon,

- **Considérations entrecroisées ou préoccupations transversales (*cross-cutting concerns*) :** mélange, au sein d'un même programme, de sous-programmes distincts couvrant des aspects techniques séparés [21].

2.5.3 Programmation fonctionnelle

Principalement, la programmation fonctionnelle permet de concevoir des programmes sous forme de fonctions mathématiques pouvant être composées entre elles. Les programmes écrits par la programmation impérative sont organisés en instructions et

ils produisent des effets de bords, alors que les programmes fonctionnels se basent sur des expressions dont la valeur est le résultat du programme. En effet, dans un langage fonctionnel, il existe l'effet secondaire et non pas l'effet de bord. La traduction d'un algorithme par un programme fonctionnel consiste en une expression Expr. Cette expression Expr est régie par des règles de réécriture du code : la réduction consiste en le remplacement d'une partie de programme fonctionnel par une autre partie de programme selon une règle de réécriture bien définie. Ce processus de réduction sera répété jusqu'à l'obtention d'une expression irréductible (aucune partie ne peut être réécrite). L'expression Expr* ainsi obtenue est appelée forme normale (FN) de Expr et constitue la sortie du programme.

Le λ-calcul a été introduit par Church; c'est la base théorique commune à tous les langages fonctionnels. Il se base sur trois concepts : les variables, l'abstraction fonctionnelle, qui permet de construire des fonctions et l'application de fonctions. Avec ce langage, il est possible d'exprimer des fonctions d'ordre supérieur dont le résultat lui-même est une fonction, et par conséquent applicable à d'autres fonctions. Ce langage est l'exemple le plus simple de langage fonctionnel et toutes les études effectuées sur ce langage se reporteront naturellement dans toute la classe des langages fonctionnels (ML, Haskell, Scheme). Même pour les langages non fonctionnels, le λ-calcul permet de fournir une compréhension de certains phénomènes, comme les appels de procédures et la récursion.

Les concepts de base de la programmation fonctionnelle sont:

Immutabilité : c'est le fait de s'assurer de la valeur des variables du début à la fin de nos actions. Cela permet d'écrire un code plus robuste et plus stable, c'est-à-dire moins de *bugs* et moins de maintenance. Une fonction ne doit pas modifier les variables qui lui sont passées en paramètre [22].

Les fonctions d'ordre supérieur : fonctions qui possèdent une ou plusieurs fonctions en paramètre, et permet de renvoyer une fonction.

Les lambdas : aussi appelées fonctions anonymes; sont des fonctions utilisées de manière ponctuelle et n'effectuant généralement qu'une opération. Certains langages, comme JavaScript ou PHP sont encore verbeux sur l'utilisation des lambdas. Java, Ruby ou Scala exploitent toute la puissance des lambdas pour avoir un code à la fois concis, clair et robuste.

Récursivité : dans la programmation fonctionnelle, lorsqu'une fonction peut appeler elle-même pour effectuer un traitement, elle appelée récursive. Ce type d'algorithme a pour objectif réduire le code et le rendre plus lisible.

2.6 Les langages de programmation pour l'interface utilisateur

2.6.1 Le langage XAML

XAML est un langage de balisage déclaratif utilisé pour écrire des applications *Windows Presentation Foundation* (WPF). Aussi, il simplifie la création de l'interface utilisateur pour une application .NET Framework. Les éléments d'une interface utilisateur peuvent être visibles dans les balises XAML déclaratives, ce qui permet de séparer la définition de l'interface utilisateur de la logique au moment de l'exécution en utilisant des fichiers *code-behind*, puis de les joindre aux balises via des définitions de classe partielles. Le code XAML représente directement l'instanciation d'objets dans un ensemble spécifique de types de stockage défini dans des assemblages. Il est différent de la plupart des autres langages de balisage, qui sont en général interprétés sans un tel lien direct à un système de types de stockage. XAML crée un flux de travail dans lequel des parties éloignées peuvent travailler ensemble sur l'interface utilisateur et la logique d'une application, en utilisant des outils totalement différents [23].

Les fichiers XAML sont des fichiers XML qui disposent généralement d'une extension *.XAML* .Les fichiers peuvent être encodés par tout encodage XML, mais l'encodage en tant qu'UTF-8 est typique.

2.6.1.1 Les concepts de bases du XAML

Éléments d'objet : permet de déclarer une instance d'un type. Ce type est défini dans les assemblages, lesquels fournissent les types de stockage à une technologie qui utilise XAML en tant que langage [23].

Attribut (propriétés) : les propriétés d'un objet peuvent être exprimées sous la forme d'attributs de l'élément objet. Une syntaxe d'attribut nomme la propriété définie dans la syntaxe d'attribut, suivie par l'opérateur d'assignation (=). La valeur d'un attribut est toujours spécifiée sous la forme d'une chaîne entre guillemets. La Figure 3 est un exemple d'utilisation de l'attribut. À noter que les éléments objet *StackPanel* et *Button* sont tous les deux mappés au nom d'une classe qui est définie par WPF et fait partie des assemblages WPF [23].

```
<StackPanel>
  <Button Content="Cliquez ici"/>
</StackPanel>
```

Figure 3 Exemple de création d'un élément objet avec une propriété

Élément de propriété : parfois il est impossible d'utiliser la syntaxe d'attribut avec certaines propriétés d'un élément objet, car l'objet ou les informations nécessaires pour définir la valeur de la propriété ne peuvent pas être exprimées de manière appropriée par un guillemet et une simple chaîne de syntaxe d'attribut. Dans ces cas-là, une syntaxe

différente, connue sous le nom de syntaxe d'élément de propriété, peut être utilisée [23]. En utilisant l'élément de propriété, la réécriture du code l'exemple de la Figure 3 devient comme suit :

```
<Button>
 <Button.Content>
        Cliquez ici
 </Button.Content>
</Button>
```

Figure 4 Exemple de l'utilisation de l'élément propriété

Collection : une des optimisations que le langage XAML offre pour produire des balises plus explicites. Cette optimisation permet de citer le fait que si une propriété particulière utilise un type de collection, alors les éléments déclarés dans les balises en tant qu'éléments enfants dans la valeur de cette propriété font partie de la collection. Dans ce cas, une collection d'éléments objets enfants représente la valeur qui est définie dans la propriété de la collection [23].

Éléments racine XAML et espaces de noms XAML : un fichier XAML contient un seul élément racine, qui sera à la fois un fichier XML de forme correcte et un fichier XAML valide. L'exemple suivant montre l'élément racine d'un fichier XAML type pour une page WPF, avec l'élément racine Page. L'élément racine contient également les attributs xmlns et xmlns:x. L'attribut xmlns indique spécifiquement l'espace de noms XAML par défaut.

Évènements et *code-behind* XAML : la plupart des applications WPF sont composées de *code-behind* et de balises XAML. On peut utiliser Microsoft Visual Basic ou C# pour écrire un fichier *code-behind* avec un langage CLR. Lorsqu'un fichier XAML

est compilé pour balisage en tant que partie de modèles d'application et de programmation WPF, l'emplacement du fichier *code-behind* XAML pour un fichier XAML est identifié en spécifiant un espace de noms et une classe en tant qu'attribut x:Class de l'élément racine du XAML [23].

Au niveau de l'application, le mécanisme de base pour ajouter un comportement pour un élément objet consiste à utiliser un évènement existant de l'élément classe et d'écrire un gestionnaire spécifique pour cet évènement appelé lorsque cet évènement est déclenché au moment de l'exécution. Le nom de l'évènement et le nom du gestionnaire à utiliser sont spécifiés dans la balise, tandis que le code qui implémente le gestionnaire est défini dans le *code-behind* [23].

2.6.2 Le langage CLR

Common Language Runtime (CLR) est un composant de machine virtuelle de la plateforme .NET. Il s'agit de l'implémentation du standard *Common Language Infrastructure* (CLI) qui définit l'environnement d'exécution des codes de programmes. Le CLR fait tourner un type de code à octets nommé *Common Intermediate Language* (CIL). Le compilateur à la volée transforme le code CIL en code natif spécifique au système d'exploitation. Le CLR fonctionne sur des systèmes d'exploitation Windows [24].

Le CLR se compose de la partie *Common Type System* (CTS), *Common Language Specification* (CLS), *Metadata* et *Virtual Execution System* (VES). Le CLR prend en charge le langage C#, F#, VB.Net et *PowerShell* [24].

2.7 Conclusion

L'introduction au TALN et les systèmes experts nous ont permis de nous situer et cadrer notre travail. En outre, voir les différents systèmes existants nous aidera à anticiper les outils convenables pour éviter certains désavantages et minimiser la faiblesse du système, soit en matière de développement, soit en matière de complexité de leur exploitation. Les paradigmes et langages de programmation vus dans ce chapitre apportent de nouvelles connaissances et techniques permettant de bien choisir une méthode spécifique pour réaliser notre projet. De fait, le choix des paradigmes a été relatif à la complexité du problème; c'est pour cela qu'une partie sera basée sur le paradigme fonctionnel et l'autre partie, surtout, sera basée sur la programmation orientée objet. Plus particulièrement, avec cette littérature, nous serons capables de déterminer les outils théoriques et fonctionnels, et d'une autre manière, de préciser notre méthodologie de recherche dans le chapitre suivant.

Chapitre 3

CONCEPTS ET OUTILS FONCTIONNELS

3.1 Introduction

Ce chapitre contient deux parties sur le traitement des langues naturelles. La première est consacrée à introduire la logique combinatoire, puis la deuxième présente les grammaires applicative et cognitive, en commençant par les grammaires catégorielles, suivies du calcul de Lambek et le travail proposé par Shaumyan: la grammaire applicative universelle. Enfin, nous présenterons brièvement la grammaire catégorielle combinatoire applicative (GCCA).

3.2 La logique combinatoire

En 1924, Schönfinkel a introduit la notion de combinateur, et en 1958 Curry et Feys ont repris et ont étendu cette notion. La logique combinatoire a été développée pour analyser logiquement les paradoxes (Russell) et la notion de substitution. Elle est aussi en rapport avec le λ-calcul de Church (1941). De nos jours, ces deux systèmes sont les moyens utilisés par les spécialistes en informatique pour analyser les propriétés sémantiques des langages de programmation de haut niveau comme Java, OCaml, et Python.

Dans la terminologie de Curry « β-réduction » est une règle spécifique qui définit l'action d'un combinateur sur un argument. D'une autre manière, cette règle permet l'établissement d'une relation indépendante de la signification des arguments entre une expression avec un combinateur et une expression sans combinateur équivalente à la

première [25]. Autrement dit, les combinateurs sont des opérateurs abstraits qui construisent, à partir des opérateurs plus élémentaires, des opérateurs plus complexes.

Les opérateurs et les opérandes sont de différents types. La théorie des types de Church [1941] construit les types fonctionnels des opérateurs à partir d'un ensemble de types de base, ce qui permet de classer tous les opérateurs en différents types et d'en organiser l'architectonie [25].

Pour éviter certains paradoxes, comme le paradoxe réanalysé dans l'ouvrage de Curry et Feys en 1958 [26], la logique combinatoire introduit différents types d'opérateurs et d'opérandes, ce qui a pour effet de limiter la liberté dans la production de certaines expressions, en particulier les expressions paradoxales qui ne sont pas pour autant éliminées comme étant « sans signification », mais qui acquièrent un statut particulier, par exemple un statut non propositionnel [27] [26].

Les combinateurs élémentaires utilisés dans le domaine du traitement des langues naturelles sont :

Le combinateur d'effacement K ;
Le combinateur de substitution (distribution) S ;
Le combinateur d'identité I ;
Le combinateur de composition B ;
Le combinateur de duplication W ;
Le combinateur de coordination φ ;
Le combinateur de distribution ψ ;
Le combinateur de changement de type C* ;
Le combinateur de permutation C ;
Et les combinateurs complexes.

Ces combinateurs sont caractérisés par une règle β-réduction [2], présentée ci-dessous.

3.2.1 Le Combinateur d'effacement K :

Ce combinateur prend deux arguments en entrées et rend le premier comme résultat. D'une manière formelle, le combinateur K permet d'effacer l'opérande fictif x dans l'expression combinatoire K fx :

K fx → f

3.2.2 Le combinateur de distribution S

Le combinateur logique S permet de former un opérateur complexe SXY en associant ce combinateur à X un opérateur binaire et à Y un opérateur unaire. Cet opérateur a été utilisé par Steedman. En appliquant ce combinateur à un opérande X, il agit comme suit:

SXYZ → XZ(YZ)

3.2.3 Le combinateur identité I

En associant ce combinateur à un opérateur P, le combinateur d'identité **I** n'a aucun effet. Ce combinateur est caractérisé par la règle β-réduction suivante:

IP → P

D'une autre manière, **I** permet d'exprimer quelques propriétés intrinsèques. Prenons, par exemple, P étant quelconque, le converse de son converse se réduit à P.

Le combinateur d'identité peut être défini par le combinateur complexe SKK.

SKK P = **K**P (**K**P)
= **K**P
= P

3.2.4 Le combinateur de composition B

Prenons **B**XY, un opérateur complexe formé par l'application du combinateur **B** à deux opérateurs X et Y, l'action de **B**XY sur un argument Z est traduite par la règle β-réduction suivante:

BXYZ → X (YZ)

En d'autres termes, le combinateur logique **B** se comporte comme une composition de deux fonctions X et Y.

3.2.5 Le combinateur de duplication W

Ce combinateur permet de produire un combinateur complexe **W**X, et ceci par l'application du combinateur **W** à un opérateur X. Cet opérateur agit sur un opérande Y comme suit:

(**W** X) Y → X Y Y

Le combinateur duplicateur **W** est généralement utilisé dans le cas des prédicats réfléchis (se-raser, se-laver, se-blesser,...).

Exemple d'application 1:

Pour illustrer l'utilisation du combinateur **W** en linguistique pour les verbes réfléchis, on commence par une expression applicative préfixée, c'est-à-dire l'opérateur précède toujours l'opérande. Dans cet exemple, lave est un opérateur qui s'applique à son opérande Mohamed; ensuite, l'ensemble du résultat de cette application (lave Mohamed) s'applique à son opérande Mohamed, donc (lave Mohamed) Mohamed:

Mohamed se lave → Mohamed lave Mohamed.

0. Mohamed + lave + Mohamed .....intreprétaion
1. Mohamed lave Mohamed = (lave Mohamed) Mohamed..... hypothèse
2. W lave Mohamed..... W introduction
3. [se-lave = déf W lave] ..... loi grammaticale

Exemple d'application 2 :

Prenons l'exemple du prédicat *se-raser* et appliquons le à Mohamed.

L'expression résultante sera *se-raser Mohamed.*

Posons *se-rase* = ***W rase,*** le remplacement de cette expression combinatoire dans l'expression *se-rase Mohamed* donne ***W rase Mohamed.***

La réduction de cette nouvelle expression combinatoire donne comme résultat l'expression: *rase Mohamed Mohamed.*

Ces deux propositions *se-rase Mohamed* et *rase Mohamed Mohamed* sont sémantiquement équivalentes [2].

3.2.6 Le combinateur de coordination Φ

Ce combinateur permet de construire un opérateur complexe composé, par exemple, de F, G et H des opérateurs quelconques, qui agit sur un opérande x de la manière suivante :

$$\Phi \; F\, G\, H\, x \rightarrow F\,(Gx)\,(Hx)$$

Exemple d'application:

Prenons l'opérateur (+) d'addition. '**Φ** + gh' est un opérateur complexe qui représente la somme des deux fonctions unaires g et h. Ceci est montré par :

$$(\Phi + g\, h)\,(a) \rightarrow +\,(g\, a)\,(h\, a)$$

Et si les fonctions g et h sont binaires, la somme des deux fonction est représentée par l'opérateur complexe '**Φ** (**Φ** +) gh'. En effet :

$$(\Phi\,(\Phi +)\, gh)\, a\, b \rightarrow (\Phi +)\,(g\, a)\,(h\, a)\, b$$
$$\rightarrow +\,(g\, a\, b)\,(h\, a\, b)$$

3.2.7 Le combinateur de distribution ψ

En appliquant ce combinateur sous les mêmes conditions citées ci-dessus, le combinateur ψ donne le résultat qui suit la règle β-réduction suivante:

$$\psi\, F\, G\, H\, x \rightarrow F\,(G\, H)\,(G\, x)$$

3.2.8 Le combinateur de changement de type C*

Comme son nom l'indique, le combinateur **C*** permet de changer type (type *raising*) des opérandes et des opérateurs; autrement dit, cela change le statut d'un opérateur en opérande et inversement. Ce combinateur est caractérisé par la règle βréduction suivante:

C* X Y → Y X

Autrement dit, si X un opérateur a pour opérande Y, alors sous l'action de **C***, X devient opérande de l'opérateur (**C*** Y) [27].

3.2.9 Le combinateur de permutation C

Le combinateur de permutation **C** permet d'exprimer la commutativité d'une opération. L'association de ce combinateur à un opérateur quelconque X produit un opérateur complexe '**C** X'. Ce combinateur se traduit par la règle β-réduction suivante:

(**C** X) Y F → X F Y

Pour décrire des verbes symétriques comme *épouser, rencontrer, croiser, etc.*, on utilise ce combinateur de permutation.

Il existe aussi d'autres prédicats qui ne sont pas symétriques, mais on trouve des prédicats de correspondance.

Exemple d'application : verbe symétrique

Dans le cas de l'expression : *Ali épouse Sara,* nous allons montrer que cette proposition est équivalente à *Sara épouse Ali.*

L'expression combinatoire relative à *Ali épouse Sara* est *épouse Ali Sara.*

Posons : épouse = C *épouse.*

épouse Ali Sara ~ C épouse Ali Sara

L'application de la règle de réduction associée au combinateur C nous donne :

épouse Sara Ali.

Ali épouse Sara ~ Sara épouse Ali.

Exemple d'application : prédicats non symétriques

Prenons deux prédicats P_2 (est-supérieur-ou-égal-à) et Q_0(est-inférieur-ou-égal-à) appariés, on a comme résultats :

C $P_2 \equiv Q_0$ et **C** $Q_0 \equiv P_2$

C (est-inférieur-ou-égal-à) ≡ (est-supérieur-ou-égal-à)

3.2.10 Les combinateurs complexes

L'association de plusieurs combinateurs élémentaires permet de produire des combinateurs complexes.

Ces combinateurs ont une action déterminée par l'application successive des combinateurs élémentaires de gauche à droite. Autrement dit, cette action est spécifiée par un schéma de règle[1], Par exemple ' **B B Φ** ', '**W W W**',…

L'exemple suivant [28] montre l'application du combinateur complexe BCC aux opérandes x, y, z. La réduction de BCC se fait d'abord par la réduction de B, puis de C, puis de C.

B C C x y z
C (**C** x) y z
(**C**x)z y
xyz

3.2.11 Puissance d'un combinateur :

À la manière des combinateurs complexes, il existe un cas particulier de combinateurs complexes : $\mathbf{B^2}$, $\mathbf{B^3}$, $\mathbf{B^4}$,..., $\mathbf{B^n}$, $\mathbf{W^2}$, $\mathbf{W^3}$, $\mathbf{W^4}$,..., $\mathbf{W^n}$, $\mathbf{C^2}$, $\mathbf{C^3}$, $\mathbf{C^4}$,..., $\mathbf{C^n}$,...

Ces combinateurs sont définis par :

« Si χ est un combinateur régulier, alors χ^n itère n fois l'action du combinateur χ tel que : $\chi^1 = \chi$ et $\chi^n = \mathrm{B}\ \chi^n\ \chi^{n-1}$ » [28].

Et ceci avec la supposition que :

$$\chi^1 \equiv \chi \ ; \ \chi^{n+1} \equiv \chi \ . \chi^{1\ n}$$

---

[1] Spécifié par une β-règle de réduction

Exemples :

$\mathbf{B^2}$ w x y z → w (x y z) ;

$\mathbf{B^3}$ t w x y z → t (w x y z) ;

$\mathbf{W^3}$ x y → x y y y ;

3.2.12 Combinateurs à distance :

Un autre cas particulier de combinateurs complexes, aussi appelé combinateurs différés, se distingue : les combinateurs dont l'action se fait à distance [25].

Ces combinateurs sont définis par :

Si χ est un combinateur, alors $\chi_{(n)}$ diffère son action de n pas et agit comme χ à partir du (n+1)ième opérande :

$$\chi_{(n)} \equiv \mathbf{B}^k \chi_{(0)}$$

Exemples :

$\mathbf{B^2}$ **C** t w x y z → **C**(t w x) y z → t w x z y

B $\mathbf{K^2}$ t w x y z → $\mathbf{K^2}$ (t w) x y z → t w z

3.3 Les grammaires applicatives et cognitives

3.3.1 Les grammaires catégorielles

Les grammaires catégorielles ont été introduites par le philosophe Husserl (1913) en présentant la différence entre les expressions catégorématiques[2] et les expressions syncatégorématiques[3]. Il s'agit là d'une opposition traditionnelle depuis les Grecs. On ne peut pas percevoir une signification complète avec ces expressions sans conjoindre les significations d'autres parties du discours. [28]

En 1922, le logicien Lesniewski a expliqué sa conception des catégories sémantiques esquissée en reprenant la tradition des catégories aristotéliciennes, des parties du discours de la grammaire traditionnelle et des catégories de signification développée par Husserl. Il a retenu deux types d'expressions : les noms et les propositions.

Pour le premier type, les noms sont des expressions linguistiques qui servent à désigner des entités objectales ou des classes de telles entités. Et pour le deuxième, les propositions sont des expressions linguistiques qui sont construites par une énonciation visant à représenter un état de choses comme objet intentionnel.

Les grammaires catégorielles sont des systèmes formels, dits classiques. Elles comportent un ensemble de catégories divisées en deux classes : les catégories de base et les différentes catégories d'opérateurs, également appelées foncteurs. Ainsi, elles assignent des catégories aux unités (lexicales et morphèmes) du dictionnaire. La bonne connexion syntaxique est assurée par la simplification des catégories d'une manière

---

[2] Les expressions qui sont pourvues d'un sens, elles apparaissent sous formes de syntagmes nominaux ou d'énoncés.

[3] Les expressions qui ne sont pas pourvues d'un sens complet

successive. Une phrase est une unité linguistique qui a une bonne connexion syntaxique qui se simplifie en une catégorie de phrase.

Donc, on peut définir une grammaire catégorielle par la donnée d'une assignation des catégories à ses unités linguistiques. La simplification des catégories revient à effectuer un calcul inférentiel sur les catégories et non pas sur les unités linguistiques. En considérant chaque catégorie comme un type, on peut donc conceptualiser les grammaires catégorielles comme des systèmes inférentiels de types. [25] [28]

Système inférentiel de types : considérant les types de base et les types dérivés, l'ensemble des types est défini comme suit :

(T0) les types de base sont des types ;

(T1) Si Y et Z sont des types alors Y/Z et Y\Z sont des types ;

(T2) Si Y et Z sont des types, Y-Z est un type.

Une entité e (unité linguistique par exemple) qui est de type Y, sera notée par : [Y : e]

Les types dérivés Y/Z et Y\Z se lisent respectivement ''Y sur Z'' et ''Y sous Z''. L'interprétation de ces types dérivés est fonctionnelle : ils représentent le type d'un opérateur qui a pour argument une entité de type Z et qui donne pour résultat une entité de type Y. L'interprétation fonctionnelle se déduit des deux règles applicatives (avant et arrière) suivantes :

[Y/Z : f] , [Z : a] [Z : a] , [Y\Z : f]

(**A>**) ----------> ; (**A<**) ----------<

[Y : f - a] [Y : a - f]

On donne deux règles d'inférences sur les types, directement associées aux règles applicatives précédentes :

Y/Z - Z Z - Z\Y

(**AB>**) ----------> ; (**AB<**) ----------<

Y Y

Les règles signifient que des deux types contigus et présents dans cet ordre dans la prémisse, on en infère le type de la conclusion.

La concaténation des types, notée '-', peut s'interpréter comme une concaténation ou un produit cartésien. [28]

3.3.2 Le calcul de Lambek

J. Lambek a fait apparaitre, dans ses articles en 1958 et 1961, le calcul de Lambek sur les types des grammaires catégorielles. Il s'agit en fait d'une extension des analyses de Bar-Hillel et Ajdukiewicz permettant de résoudre des problèmes syntaxiques insolubles dans le cadre des grammaires catégorielles classiques [27].

Ce calcul permet de gérer les réductions entre les types et certains changements de types. Le changement de type a été proposé pour l'analyse du pronom. Aussi, Lambek a introduit un ensemble de règles et de notations permettant de faciliter l'analyse des phrases d'un langage naturel [2].

Le système décrit par Lambek se compose des types primitifs S et N, deux opérateurs constructeurs / et \ et un opérateur de concaténation représenté par le symbole (+). Pour deux expressions el et e2 de types respectifs X et Y, la concaténation de ces deux expressions sera notée : el + e2 et sera de type X + Y, et aura une règle de réduction « → ». Selon cette règle, la notation X → Y signifie que le type X se réduit au type Y. De même, la notation X ↔ y signifie à la fois que le type de X se réduit au type Y et que le type Y se réduit au type X.

Le système de calcul de types de Lambek se base sur les règles suivantes [2] :

- La réflexivité:

X → X

- L'associativité:

(X Y) Z → X (Y Z)

X (Y Z) → (X Y) Z

- La transitivité :

Si X Y et Y Z Alors: X → Z

- Si X Y → Z Alors X → Z / Y
- Si X Y → Z Alors Y → Z \ X
- Si X → Z / Y Alors X Y → Z
- Si Y → Z \ X Alors X + Y → Z

Les relations entre types peuvent être faites par les théorèmes suivants :

- $X \rightarrow (X\ Y)\ IY$
- $(Z\ I\ Y)\ Y \rightarrow Z$
- $Y \rightarrow Z \setminus (Z / Y)$
- $(Z\ I\ Y)\ \ (Y / X) \rightarrow (Z / X)$
- $Z\ I\ Y \rightarrow (Z / X) / (Y / X)$
- $(Y \setminus X) / Z \rightarrow (Y / X) \setminus Z$
- $(X / Y) / Z \rightarrow X / (Z\ \ Y)$
- Si $X \rightarrow X'$ et $Y \rightarrow Y'$ Alors $X\ \ Y \rightarrow X'\ \ Y'$
- Si $X \rightarrow X'$ et $Y \rightarrow Y'$ Alors $X \setminus Y' \rightarrow X' / Y$

Ces théorèmes sont déduits des règles ci-dessus.

3.3.3 La grammaire applicative universelle de Shaumyan (GAU) :

Shaumyan propose dans ses travaux (1967, 1977, 1987, 1998) une distinction entre deux niveaux de représentation et d'analyse d'une langue : le niveau phénotype et le niveau génotype. Ses travaux se basent sur une opposition avec les hypothèses instaurées par les grammaires de réécriture. Le principal défi relevé par Shaumyan est de mettre en œuvre une grammaire applicative universelle [2]. La figure suivante décrit le modèle de ce genre de grammaire et permet de faire la distinction entre le niveau phénotype et le niveau génotype.

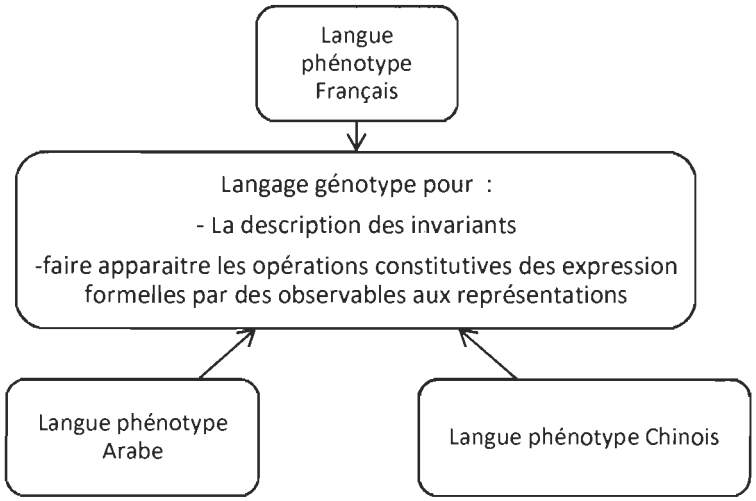


Figure 5: Le modèle de la Grammaire Applicative Universelle (GAU) de Shaumyan (1987)

Le niveau phénotypique : communément connu sous les noms de « niveau de surface » ou encore « niveau observable », ce niveau regroupe l'ensemble des représentations concaténées. Il permet d'évoquer tous les traits spécifiques d'une langue donnée à travers la description de l'ordre de ses mots, sa syntaxe ou encore tous ses traits morphologiques [2].

Le niveau génotypique : niveau abstrait tel que décrit par Shaumyan. Ce niveau permet de retracer les invariantes sémiotiques, principaux constituants des langages naturels. La description de ces invariants se fait sous forme d'opérations et de relations.

La GAU a pour objectif de :

- Reconstruire un système sémiotique composé principalement des unités et opérations qui seraient des entités et des opérations universelles constitutives des systèmes de représentation,

- Étudier les propriétés formelles du système sémiotique et analyser sa structure mathématique,

- Typer les langues naturelles en utilisant la structure du système sémiotique invariant[4] et les différentes règles d'encodage du système universel dans les différents systèmes particuliers dans le temps et l'espace,

- Rechercher les règles qui encodent le génotype dans les différentes langues naturelles observées (les phénotypes observés) et

- Étudier les lois générales sous-jacentes aux systèmes sémiotiques, qu'ils soient langues naturelles ou langues artificielles. [27]

3.3.4 La grammaire catégorielle combinatoire applicative

La grammaire catégorielle combinatoire applicative (GCCA) consiste à relier le phénotype au génotype par l'utilisation d'un système formel. Les règles de la grammaire catégorielle combinatoire de Steedman introduisent les combinateurs **B**, **C***, **S** dans la séquence syntagmatique. Cette introduction permet de passer d'une structure concaténée à une structure applicative.

Plus concrètement, la GCCA établit une association canonique entre les règles catégorielles combinatoires de Steedman et les combinateurs de la logique combinatoire de Curry. Les règles catégorielles auront pour rôle de vérifier la correction syntaxique des énoncés. L'introduction des combinateurs dans la chaîne syntagmatique dans un premier temps conduit à la construction des expressions applicatives dans un second temps avec

---

[4] Le génotype

la réduction des combinateurs. L'association des règles catégorielles avec les combinateurs permet de démontrer que les deux approches et les deux niveaux de représentations des langues proposés par S.K. Shaumyan et Jean-Pierre Desclés peuvent être complémentaires. Aussi, Ismaïl Biskri et J.P. Desclés soutenaient dans leur travail[5] le principe que la grammaire catégorielle combinatoire agit sur des structures concaténées, alors que les combinateurs agissent sur des structures applicatives.

Avant de citer quelques exemples d'applications des traitements basés sur la GCCA, le tableau suivant regroupe les règles de la GCCA, de changement de type et de composition :

Tableau 3 Les règles de la GCCA

Règles d'application	[X/Y : u1] - [Y : u2] ----------> ; [X : (u1 u2)] [Y : u1] - [X\Y : u2] <---------- [X : (u2 u1)]
Règles de changement de type	[X : u] ----------> T ; [Y/(Y\X) : (C* u)] [X : u] <---------- [Y \ (Y/X) : (C* u)] [X : u] ----------> Tx ; [Y/(Y/X) : (C* u)] [X : u] <---------- [Y \ (Y\X) : (C* u)]

[5] Ismaïl BISKRI Jean-Pierre DESCLES. « Du Phénotype au Génotype : La Grammaire Catégorielle Combinatoire Applicative ».Université de Paris-Sorbonne.

Règles de composition	[X/Y : u1]-[Y/Z : u2] [Y\Z : u1]-[X\Y : u2] ----------->B ; ----------<B [X/Z : (B u1 u2)] [X\Z : (B u2 u1)] [X/Y : u1]-[Y\Z : u2] [Y/Z : u1]-[X\Y : u2] ----------->B ; ----------<B [X\Z : (B u1 u2)] [X/Z : (B u2 u1)]

Un traitement complet basé sur la grammaire catégorielle combinatoire applicative s'effectue en deux grandes étapes. La première étape est consacrée à la vérification de la bonne connexion syntaxique et la construction de structures prédicatives avec des combinateurs introduits à certaines positions de la chaîne syntagmatique, alors que la seconde consiste à utiliser les règles de β-réduction des combinateurs pour former une structure prédicative sous-jacente à l'expression phénotypique. L'expression obtenue est applicative et appartient au langage génotype.

La GCCA engendre des processus qui associent une structure applicative à une expression concaténée du phénotype. Puis, il reste à éliminer les combinateurs de l'expression obtenue afin de construire la « forme normale » (au sens technique de la β-réduction), qui exprime l'interprétation sémantique fonctionnelle. Ce calcul s'effectue entièrement dans le génotype [29].

Le traitement proposé est donc considéré comme une compilation dont les étapes sont résumées dans la figure suivante.

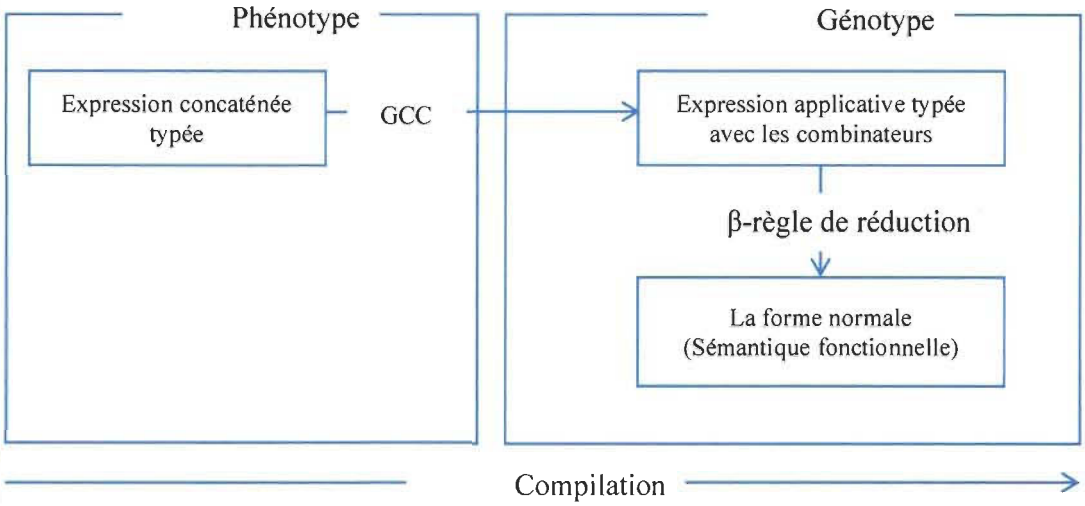


Figure 6 les étapes d'un traitement complet basé sur la grammaire catégorielle combinatoire applicative

Exemple d'application : une abeille butine la fleur.

1.	[N/N : *une*]-[N : *abeille*]-[(S\N)/N : *butine*]-[N/N : *la*]-[N : *fleur*]	
2.	[N : (*une abeille*)]-[(S\N)/N : *butine*]-[N/N : *la*]-[N : *fleur*]	(>)
3.	[S/(S\N) :(**C*** (*une abeille*))]-[(S\N)/N: *butine*]-[N/N: *la*]-[N: *fleur*]	(>**T**)
4.	[S/N : (**B** (**C*** (*une abeille)) butine)*]-[N/N : *la*]-[N : *fleur*]	(>**B**)
5.	[S/N : (**B** (**B** (**C*** (*une abeille)) butine) la)*]-[N : *fleur*]	(>**B**)
6.	[S : ((**B** (**B** (**C*** (*une abeille)) butine) la) fleur)]	(>)
7.	((**B** (**B** (**C*** (une abeille)) butine) la) fleur)	
8.	((**B** (**C*** (une abeille)) butine) (la fleur))	**B**
9.	((**C*** (*une abeille*)) (*butine* (*la fleur*)))	**B**
10.	((*butine* (*la fleur*)) (*une abeille*)))	**C***
11.	butine (la fleur) (une abeille)	

Dans la première étape de cet exemple, des types catégoriels ont été assignés aux unités linguistiques. À la deuxième étape, la règle (>) est appliquée aux unités linguistiques *une* et abeille. À la troisième étape, une règle de changement de type (>T) a été affectée pour construire un opérateur (C* *(une abeille))* à partir de l'opérande *(une abeille).* Cet opérateur est composé avec l'opérateur *butine* à la quatrième étape par une opération de composition (>B) pour former un opérateur complexe (B (C* *(une abeille)) butine).* Dans les cinquième et sixième étapes, deux autres opérations ont été appliquées : (>B) et (>), ce qui donne (B (B (C* *(une abeille)) butine) la) fleur*). Et de la septième étape à la dixième étape, une réduction a été effectuée sur les combinateurs dans le génotype pour construire l'interprétation sémantique fonctionnelle sous-jacente à l'expression phénotypique en entrée.

Il est donc possible de voir, dans le présent travail, une continuité des grammaires et la naissance de nouvelles règles permettant de faire une analyse couvrant de plus une variété de cas d'agencements. Par conséquent, l'interprétation d'une chaîne de traitement constituera le résultat de ses opérations primitives sous-jacentes et la façon dont ces opérations sont organisées en conséquence du principe de composition. L'ensemble des chaînes de traitement composé devient un ensemble de théorèmes pour le système formel proposé. Le système est en soi inférentiel. Il procède par réductions successives de catégories applicatives affectées à des opérations concernées par la composition.

3.4 Conclusion

La logique combinatoire était la première partie qu'on peut considérer parmi les moyens utilisés actuellement par les informaticiens pour analyser les propriétés sémantiques des langages de programmation de haut niveau.

Après avoir vu la logique combinatoire, nous sommes passés à la grammaire applicative, qui est un modèle d'analyse qui joue le rôle de l'interface entre la syntaxe et la sémantique. Ce modèle permet également de produire une analyse qui vérifie la correction syntaxique des énoncés et d'engendrer automatiquement des structures prédicatives qui rendent compte de l'interprétation sémantique fonctionnelle des énoncés pour un noyau du français.

En résumé, ce module se caractérise par le calcul permettant de vérifier la correction syntaxique, calcul qui se poursuit par une construction de l'interprétation sémantique fonctionnelle, qui elle-même s'applique par des méthodes syntaxiques applicatives (la réduction des combinateurs). Dans le chapitre suivant nous proposerons une interprétation et une application de cette théorie via un modèle formel et un rapprochement applicatif de cette partie.

Chapitre 4

DE LA THÉORIE VERS L'APPLICATION

4.1 Introduction

Dans ce chapitre nous détaillerons notre propre modèle sur lequel nous nous reposerons pour passer à l'applicatif. Ce modèle s'ajoute à plusieurs modèles théoriques qui ont été établies pour résoudre la problématique dans le domaine du traitement des langues naturelles. Dans la deuxième section, nous procéderons à un passage de la théorie vers l'application. Après cela, nous verrons quelques cas d'enchainements possibles dans notre modèle. Dans la cinquième section, nous passerons de la théorie vers l'application par des algorithmes résolvant la problématique. Finalement, nous présenterons une étude de cas pratique.

4.2 Le modèle formel pour le traitement des langues naturelles et de l'information

4.2.1 Présentation du modèle

Notre modèle proposé possède des fondements formels solides (grammaires applicatives et logique combinatoire). L'un des principaux enjeux de ce formalisme est d'assurer une ferme possibilité de composition des différents modules dans différentes chaînes de traitement. En outre, il permet de composer une infinité de modules.

Nous décrirons dans cette section la notion de modules, comment se fait la création des chaînes de traitement, ainsi que les règles permettant l'analyse des différents cas d'agencements.

4.2.2 Les modules

Un module peut être représenté par une opération qui s'applique sur une ou plusieurs entités objectales d'un type donné et retourne d'autres entités objectales d'un autre type. Il est noté par [M1: Fxy], M1 identifiant le module et Fxy exprimant son type comme le montre la figure suivante :

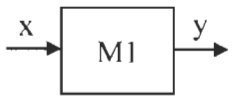


Figure 7 Représentation d'un module

Un module agit comme une fonction mathématique qui prend des arguments, applique un traitement spécifique et donne un résultat.

Chaque module est indépendant et peut être vu comme une boîte noire : nous sommes seulement intéressés à la fonction générale qu'il accomplit. Il est possible d'avoir des modules à plusieurs entrées et une sortie: Par exemple, M2 à deux entrées x1 et x2, comme représenté graphiquement dans la Figure 8 et M3 à trois entrées x1 x2 x3 et sortie y, représenté dans la Figure 9.

Figure 8 Représentation d'un module à deux entrées

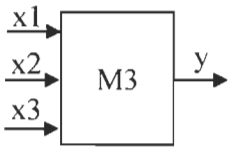


Figure 9 Représentation d'un module à trois entrées

Les modules doivent aussi avoir la capacité de communiquer entre eux à l'aide d'un protocole. Chaque module doit avoir un type catégoriel pour refléter la façon dont il agit sur ses opérandes. En effet, le type du module M2 de la Figure 8 est exprimé par Fx1Fx2y, et le type du Module M3 de la Figure 9 est exprimé par Fx1Fx2Fx3y.

4.2.3 Chaîne de traitement

Une séquence de modules, que nous appelons chaîne de traitement, est un agencement de modules. Les trois règles principales pour définir une séquence de traitement sont:

- la séquence doit contenir au moins un module,
- la séquence doit être syntaxiquement correcte et
- les aspects sémantiques de la séquence, qui sont de la responsabilité de l'ingénieur de la langue (un ingénieur de la langue est tout chercheur ou un développeur qui a des intérêts dans l'ingénierie de la langue. Ça peut être un chercheur en informatique, ou un linguiste, un terminologue, un philosophe, etc.), doivent s'assurer que les modules choisis servent les objectifs de la chaîne de traitement.

On peut définir une chaîne de traitement plus clairement par une séquence intégrée de modules de calcul dédiés à des traitements spécifiques, mis ensemble selon un objectif de processus déterminé par l'ingénieur de la langue.

La chaîne de traitement devra permettre la composition des modules. Par conséquent, il est essentiel de répondre à deux questions fondamentales :

- Étant donné un ensemble de modules, quelles sont les dispositions admissibles qui conduisent à des chaînes de traitement cohérentes (l'exactitude syntaxique)?

- Étant donnée une chaîne de traitement cohérente, comment pouvons-nous automatiser (autant que possible) son évaluation (dans le sens de sa calculabilité)?

4.2.4 Types

Les types catégoriels sont générés récursivement à partir de types de base et au moyen d'un opérateur « F » comme suit :

- Les types de base sont des types.

- Si x et y sont des types, alors Fxy est un type.

- Les prémisses dans chaque règle sont typées « modules connectés » et les résultats sont des expressions applicatives typées (des modules) avec une introduction éventuelle d'un combinateur.

- Ces expressions applicatives permettent l'interprétation des chaînes de traitement.

- Les types de modules dans les prémisses permettront de valider l'application des règles, et donc d'accepter ou de rejeter la connexion des modules.

4.2.5 Contraintes d'agencement

Pour connecter deux modules, il est nécessaire de vérifier l'égalité entre le type sortie du premier module avec le type d'entrée du deuxième module. Pour que l'agencement soit accepté, ces types doivent être égaux, sinon l'agencement sera rejeté. Les sous-sections suivantes représentent des exemples d'agencements rejetés et des exemples acceptés.

4.2.5.1 Exemples d'agencements rejetés

Prenons comme exemple, le module M1 avec type Fxy et M2 avec type Fx1Fx2y de la Figure 8. Nous ne pouvons pas connecter M1 avec M2 pour créer une chaîne de traitement, car le type y de la sortie de M1 est différent des deux entrées x1 et x2 de M2. La chaîne de traitement de la figure suivante n'est pas acceptée par le modèle étudié.

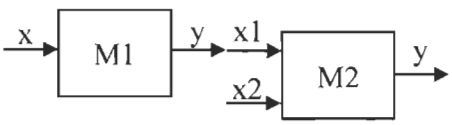


Figure 10 Exemple 1 de connexion rejetée

L'agencement représenté dans la figure suivante est rejeté. En effet, nous ne pouvons pas connecter M2 avec M1 parce que les types y (type de sortie de M2) et x (type d'entrée de M1) sont différents.

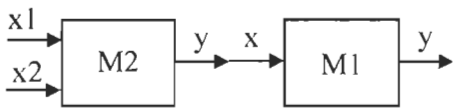


Figure 11 Exemple 2 de connexion rejetée

4.2.5.2 Exemples d'agencements acceptés

Pour connecter le module M1 avec type Fxy et M4 avec type Fyz, la connexion entre M1 et M4 peut se faire correctement si le module M1 est en entrée du module M4, car le type de la sortie de M1 est égale au type en entrée de M4 qui est y. La chaîne de traitement de la figure suivante est acceptée par le modèle étudié.

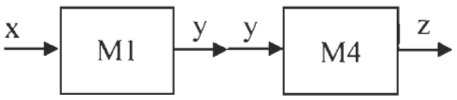


Figure 12 Exemple 1 de connexion acceptée

La Figure 13 est un autre exemple d'agencement accepté entre le module M5 avec type Fy1x1 et le module M3 avec type Fx1x2x3y. La connexion entre M5 et M3 est correctement établie, car le type de la sortie du module M5 est égal au type en entrée de

M3 qui est x1. La chaîne de traitement de la figure suivante est validée par notre modèle. Par la suite, il sera possible d'exécuter cet agencement.

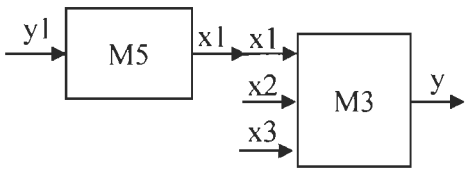


Figure 13 Exemple 2 de connexion acceptée

4.2.5.3 Agencement constitué d'un module connecté avec plusieurs modules

Prenons l'agencement construit par le module M5 avec type Fy1x1, le module M2 avec type Fx1Fx2Fx3y et le module M3 avec type Fx1Fx2y (Figure 15 Connexion de plusieurs sorties identiques à une entrée). Notre modèle ne permet pas de construire un agencement à base de ces trois modules, même si la sortie x1 du module M5 est identique à l'entrée x1 de M2 et M3. Donc l'agencement de la figure suivante est rejeté par notre modèle.

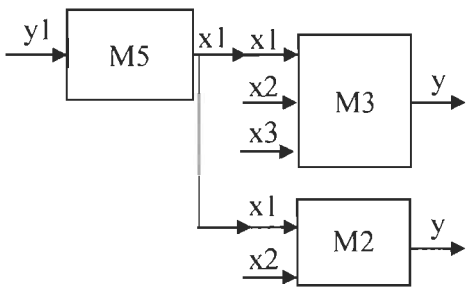


Figure 14 Exemple de connexion d'un module avec plusieurs modules

4.2.5.4 Agencement constitué de plusieurs modules connectés avec un seul module

Nous pouvons remarquer dans la figure suivante que le module M3 et M1 ont la même sortie, qui est de type y et le module M4 a une entrée de type y. Cependant, il est possible, dans ce cas, de connecter soit le M3 avec le module M4, soit le M1 avec M4. Donc, les trois modules ne peuvent pas être connectés de cette façon et notre système ne le permettra pas.

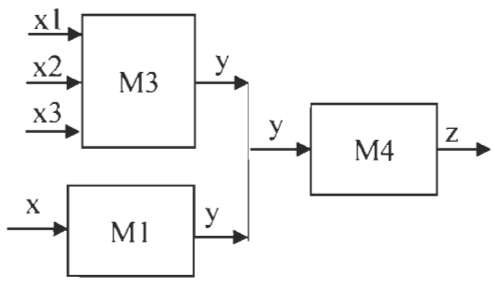


Figure 15 Connexion de plusieurs sorties identiques à une entrée

Ce genre d'agencement ne sera pas exécuté par le système.

4.3 Rapprochement applicatif de la théorie

4.3.1 Les chaînes de traitement et le principe applicatif

Théoriquement, dans la logique combinatoire, nous évoluons dans un système applicatif, et à ce titre, nous pouvons considérer nos modules comme des opérateurs dont les entrées seraient leurs arguments et dont les sorties seraient le résultat de l'application de l'opérateur à ces arguments. En outre, grâce au développement continuel des grammaires applicatives, nous pouvons les considérer comme des boites noires avec un traitement quelconque à l'intérieur, lesquelles peuvent agir les unes sur les autres.

Le principe applicatif permet de passer à une fonction avec une variable. En effet, un module qui recevrait plusieurs entrées ou arguments sera vu comme une fonction qui prendra le premier argument pour former une fonction complexe laquelle, à son tour, prendra la deuxième fonction pour former une autre fonction et ainsi de suite afin de retourner un résultat en sortie. Tout cela engendre l'introduction d'un combinateur selon la règle appliquée.

Prenons, par exemple, quatre modules M1, M2, M3 et M4, lesquels sont correctement connectés, comme le montre la Figure 16, et qui admettent une analyse complète par notre modèle. Le résultat de la chaîne de traitement construite par ces modules est représenté dans la Figure 17.

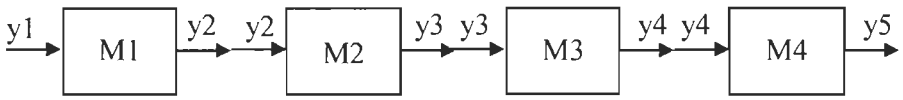


Figure 16 La chaîne de traitement et le principe applicatif

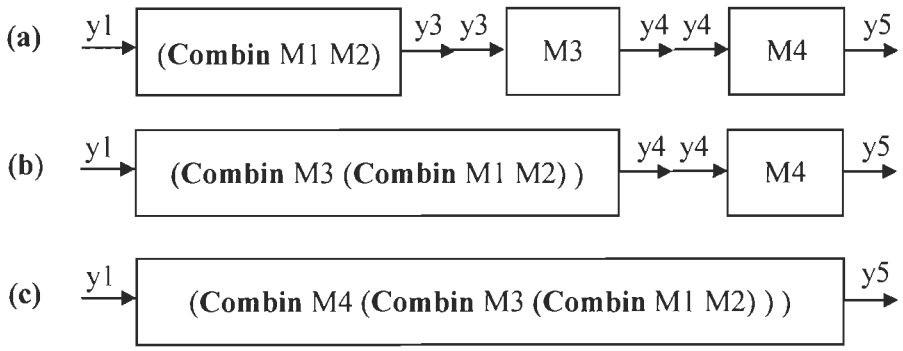


Figure 17 Exemple de traitement d'une chaîne

Dans cet exemple, **Combin** est utilisé pour représenter un combinateur quelconque introduit par la règle appliquée pour combiner les modules : il peut être le combinateur **B, C, W, S** ou autre.

Pour cet exemple, l'analyse a été exécutée en trois étapes **(a)** **(b)** et **(c)**. L'étape **(a)** est la combinaison du module M1 et M2 pour donner un nouveau module avec sa nouvelle expression (**Combin** M1 M2). L'étape **(b)** est le résultat de la combinaison du module (**Combin** M1 M2) avec et le module M3 pour donner à son tour (**Combin** M3 (**Combin** M1 M2)). Finalement, l'étape **(c)** est le résultat de la combinaison du dernier module M4 avec le résultat de l'étape **(b)** qui est la fin du traitement : (**Combin** M3 (**Combin** M3 (**Combin** M1 M2))).

4.3.2 Rapprochement des règles de la GCCA

Pour faire le traitement d'une langue, les grammaires catégorielles ne considèrent pas l'utilisation de règles de réécriture pour construire des dérivations afin de reconnaître les langages possibles ou de générer tous les expressions de cette langue. Cependant, elles reposent sur des fondements logiques, c'est pour cela qu'elles permettent de distinguer plusieurs niveaux de représentations des langues, dont la structure de l'observable[6] et la structure du construit[7] [30]. Elles conceptualisent, de ce fait, notre modèle comme un système d'arrangement de «modules» linguistiques dont certains fonctionnent comme des opérateurs, alors que d'autres fonctionnent comme des opérandes. Même si l'ordre applicatif n'apparaît pas directement au niveau de l'observable, la dichotomie opératoire analyse, dans une chaîne de traitement, certains modules comme des opérateurs et d'autres comme des opérandes.

La GCCA est une variante de la grammaire catégorielle combinatoire de Steedman. Elle associe les règles catégorielles combinatoires à des opérateurs de la logique combinatoire. L'application d'une règle combinatoire implique l'introduction d'un combinateur dans la chaîne syntagmatique [30]. Notre modèle se base sur les règles

---

[6] Une structure concaténée ou structure de surface

[7] Une structure opérateur-opérande permettant l'interprétation sémantique fonctionnelle

offertes par la GCCA pour aboutir à une solution; ces règles sont exprimées par des prémisses et des résultats. Les prémisses, dans chaque règle, sont typées, et sont en réalité des modules connectés, alors que les résultats sont des expressions applicatives typées. Dans notre travail, ce sont des modules avec une introduction conditionnelle d'un combinateur. Les types de modules dans les prémisses permettront de valider l'application des règles, et donc d'accepter ou de rejeter la connexion des modules.

Les paragraphes suivants montreront les règles utilisées dans notre projet, avec des exemples d'application sur les modules et leurs réductions pour arriver à l'écriture formelle de l'expression de chaque cas étudié.

4.3.2.1 Règle applicative

Dans le cas de traitement d'un seul module M1 avec type Fxy (voir Figure 18), nous appliquons cette règle sur l'entrée X de type x de M1 pour calculer le résultat Y de type y. La règle est définie par :

$$\frac{[X : x] + [M1 : Fxy]}{[Y : y]} \quad \textbf{(R1)}$$

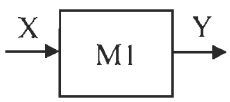


Figure 18 Cas d'application de la règle applicative

4.3.2.2 Règles de composition

Règle de composition (1)

Dans le cas d'une chaîne de traitement composée de deux modules M1:Fxy et M2:Fyz connectés en série de la façon montrée dans la Figure 19, nous appliquons cette règle pour calculer le résultat de cette chaîne. La règle est définie par :

[M1 : Fxy] + [M2 : Fyz]
----------B **(R2)**
[(B M2 M1) : Fxz]

La Figure 20 montre un cas d'agencement pour lequel on applique cette règle pour avoir en résultat le module complexe (BM2 M1) de type Fxz.

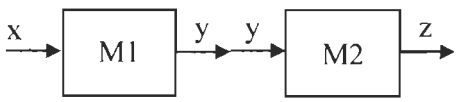


Figure 19 Cas d'application de la règle de composition

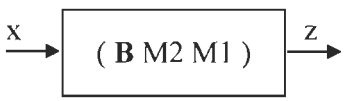


Figure 20 Résultat de l'application de la règle de composition

Règle de composition (2)

Un autre cas d'agencement montre un module à plusieurs entrées, enchaîné avec un autre à une seule sortie. Par exemple, dans l'association de M1:FwFxy et M2:Fyx de la manière de la Figure 21, la règle qui s'appliquera se définie par :

[M1 : FwFxy] + [M2 : Fyz]

(2) ----------B2 (**R3**)

[(B2 M2 M1) : FwFxz]

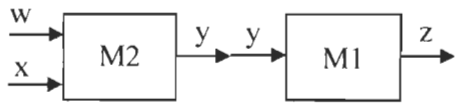


Figure 21 Cas d'application de la règle de composition (2)

Le schéma suivant montre le résultat de l'application de la règle de composition (2) qui est un module complexe (**B2** M2 M1) de type Fxz.

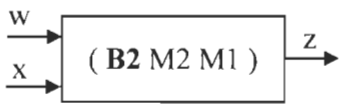


Figure 22 Résultat de l'application de la règle de composition

4.3.2.3 Règle de composition distributive

Dans le cas d'une chaîne de traitement composée de deux modules M1:Fxy et M2:FyFxz, comme montré dans la Figure 23, il est possible de constater que les deux modules M1 et M2 ont une entrée identique x. Il faut alors appliquer la règle de composition distributive pour calculer le résultat de cette chaîne. Cette règle est définie par :

[M1 : Fxy] + [M2: FxFyz]

----------S **(R4)**

[(S M2 M1) : Fxz]

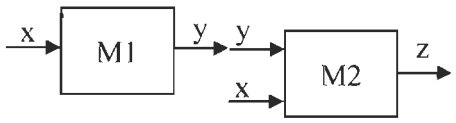


Figure 23 Cas d'application de la règle de composition distributive

La figure suivante montre un cas d'agencement pour lequel on a appliqué la règle de composition distributive donnant comme résultat le module complexe (SM2 M1) de type Fxz.

Figure 24 Résultat de l'application de la règle de composition distributive

4.3.2.4 Règle de permutation

Règle de permutation (1)

Cette règle agit sur la deuxième entrée du module en traitement; c'est ce qui la différencie des autres règles discutées précédemment. Ainsi, elle change l'ordre des entrées permettant le traitement de la chaîne par d'autres règles. Cette règle est définie par :

[M1 : FxFyz]

(1) ----------C **(R5)**

[(C M1) : FyFxz]

Par exemple, un simple agencement nous aidera à étudier l'action de cette règle. Prenons M1 : FxFy et M2 : FxFyz, enchaînés de la manière représentée par la figure suivante. L'application de la règle de permutation (1) agit seulement sur le module M2 en changeant sans type de FxFyz à FyFxz et avec une introduction du combinateur C. Le résultat est représenté dans figure suivante :

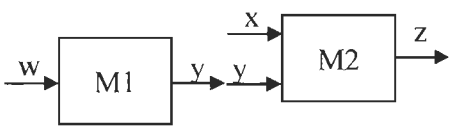


Figure 25 Cas d'application de la règle de permutation (1)

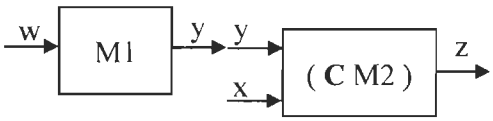


Figure 26 Résultat de l'application de la règle de permutation (1)

Règle de permutation (2)

Dans un autre cas d'agencement, où un module à plusieurs entrées est enchaîné avec un autre à sa troisième entrée, montre l'association de M1:Fxy et M3 : FtFxFyz, représenté dans la Figure 27. La règle s'appliquera seulement sur le module M3. Elle est définie par :

[M3 : Fx1Fx2Fx3y]

(2) ----------C# **(R6)**

[(C (C2 M1)) : Fx3Fx1Fx2y]

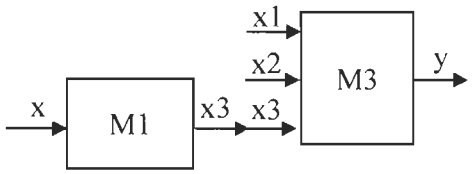


Figure 27 Cas d'application de la règle de permutation (2)

Le schéma suivant montre le résultat de l'application de la règle de permutation (2) qui est un module complexe **C** (**C2** M3) de type Fx3Fx1Fx2y, enchaîné à sa première entrée avec M1.

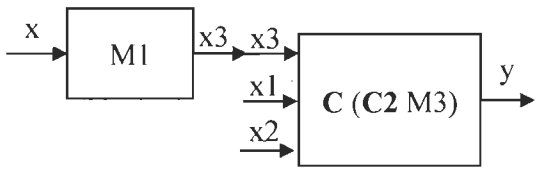


Figure 28 Résultat de l'application de la règle de permutation (2)

4.3.2.5 Règle de duplication

Dans certains cas de modules qu'on veut traiter et qui ont deux entrées identiques de type x, ces modules seront transformés pour avoir une seule entrée de type x avec introduction du combinateur **W**. Prenons l'exemple du module M1 : FxFxy, représenté dans la figure suivante. Il a deux entrées de type x, donc l'application de la règle de

duplication génère un module avec expression : (W M1) et type Fxy. La définition de la règle est comme suit :

[M1: FxFxy]
----------W (**R7**)
[(W M1) : Fxy]

La figure est un exemple simple d'application de cette règle sur le module M1 : FxFxy, représenté dans l'étape (1) avec son résultat (étape 2).

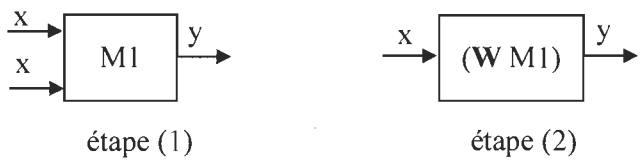


Figure 29 Application de la règle de duplication sur M1:FxFxy

4.4 Cas possibles d'agencement de plusieurs modules

4.4.1 Agencement en série

Si nous souhaitons construire une chaîne de traitement composé de n modules, de M1 jusqu'au Mn, la chaîne de traitement ressemblera à la représentation suivante :

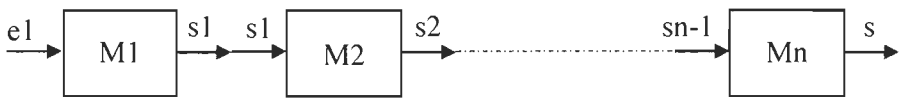


Figure 30 Agencement en série

Le traitement des chaînes en série sera effectué en utilisant la règle de composition (1).

4.4.2 Agencements en parallèle

La construction parallèle des chaînes peut se faire de la manière représentée dans la figure suivante et peut prendre de 1 à n modules :

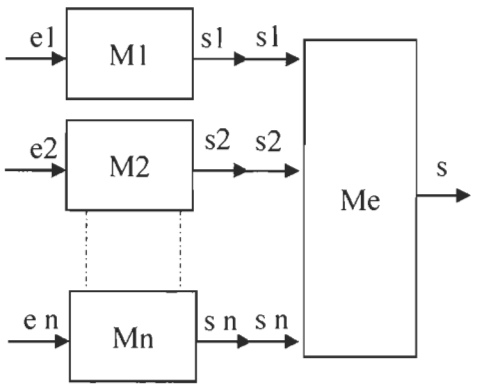


Figure 31 Agencement en parallèle

Dans la Figure 31, **Me** est un module à n entrées de type s1 à sn connecté avec *n* modules de M1 à Mn à leurs sorties de s1 à s*n* respectivement. Cet agencement est analysable avec les règles de composition et les règles (**R1, R2, R3, R4, R5, R6, R7**) présentées dans les pages 77 et 80. Un cas d'application, pour trois modules connectés avec un seul module, sera présenté dans la section étude de cas, à la page 94.

4.4.3 Agencements variés

On peut aussi construire des agencements avec des modules connectés parallèlement et/ou en série, en respectant bien sûr les contraintes de connectivité entre ces derniers. Leur traitement impliquera l'application des certaines règles, ou de toutes les règles de la GCCA, selon le cas de chaque étape du traitement.

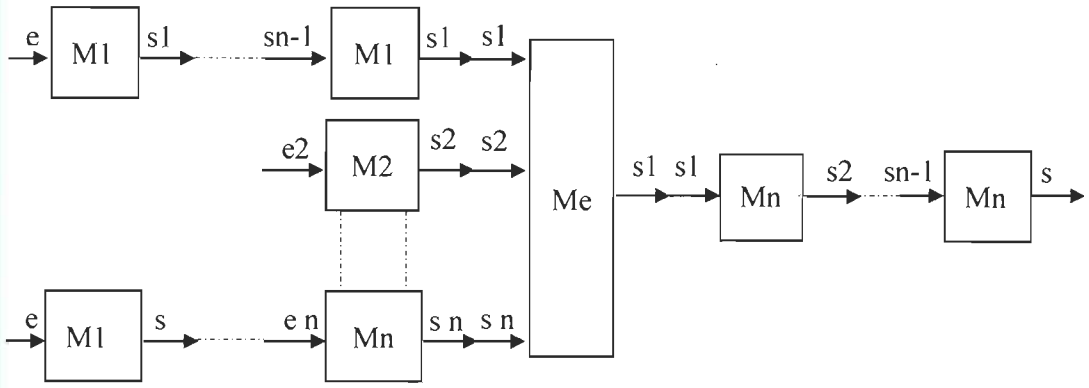


Figure 32 Agencement varié

Dans la Figure 32, **Me** est un module à n entrées de type s1 à sn connecté avec *n* modules de M1 à Mn à leurs sorties de s1 à s*n* respectivement et en même temps connecté avec *n* modules en série à sa sortie. Cet agencement est analysable avec les règles de composition et les règles (**R1, R2, R3, R4, R5, R6, R7**) présentées dans les pages 77 et 80.

4.5 De la théorie vers l'applicatif par les algorithmes

4.5.1 Algorithme du caractère exécutable d'un module

Brièvement, cet algorithme consiste à valider le caractère exécutable d'un module passant par son indice i en entrée. Un module est non exécutable s'il vérifie la condition qu'au moins un de ces prédécesseurs n'est pas encore traité; par conséquent, il sera marqué comme non exécutable. Si le module concerné n'a pas de prédécesseur, -1 est retourné pour signifier qu'il est exécutable.

Variable

Marques de type tableau de booléens

Pred de type liste d'entier

J et i entier

Début AlgoNonTraitable (i)

Pour j allant de 0 à longueur de Pred

Si Pred [j]= i et Marques[j] = faux

Retourner j

Sinon retourner -1

Fin Si

Fin pour

Fin

Nous verrons que cet algorithme sera utilisé comme partie de l'algorithme nommé AlgorithmeTM pour le traitement de modules, afin de faciliter la complexité du problème de lecture et de compréhension des algorithmes, sachant que les tableaux Marques et Pred ont été déjà initialisés et remplis automatiquement.

Exemple d'application de l'algorithme :

Pour l'exemple représenté dans la figure suivante notre algorithme marquera le premier module M1 : Fay comme non exécutable et le module M2 : FyFxz comme exécutable.

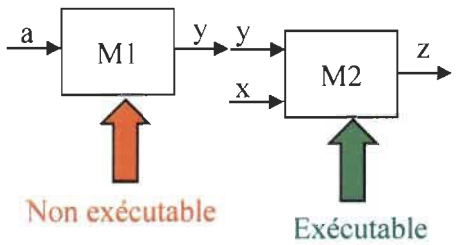


Figure 33 Exemple d'application de l'algorithme AlgoNonTraitable

Cet exemple montre que le module M2 qui va être exécuté par notre programme parce qu'il a un prédécesseur.

4.5.2 Algorithme de traitement de modules : AlgorithmeTM

Cette partie de notre algorithme a été écrite de manière récursive. Le traitement commence par une vérification du module concerné pour vérifier s'il est traité ou non, ce qui permettra d'aboutir à l'analyse et à l'évaluation de la règle qui le gère, et cela, en respectant la condition que le module en traitement ne doit pas avoir un prédécesseur. Cet algorithme ne passe les tests, bien évidemment, qu'après avoir fait appel à l'algorithme AlgoNonTraitable.

Si le module vérifie cette condition, il sera traité; c'est-à-dire que sa règle sera évaluée, et appliquée si nécessaire, ce qui fera appel à un autre algorithme défini par EvaluerRegle. Si le module en question a un autre module à sa sortie (il a un module suivant), EvaluerRegle sera appliqué, suivi d'un appel récursif en appliquant AlgorithmeTM sur le module suivant.

Par la suite, nous passerons à la vérification de l'égalité : Moduletraitee = Predecess. Autrement dit, si le module en question a un module à son entrée (appelé le module prédécesseur), ce module ne sera pas marqué « module de départ » par la mise à jour de la valeur de ModuleDepart à « faux ». Finalement, on pourra faire un appel à AlgorithmeTM, mais cette fois avec Predecess en entrée.

Variable

Marque de type tableau de booléens
Predecess de type entier
Moduletraitee de type entier
ModuleDepart de type entier

Début AlgorithmeTM (ModuleDepart)

Faire Marque = vrai

Faire Moduletraitee = AlgoModuleNonTraitable

Si Moduletraitee = -1

    Si Suiv = -1

        Faire EvaluerRegle(ModuleDepart)

        Faire arrêter le traitement

    Sinon

        Faire EvaluerRegle(ModuleDepart)

Faire AlgorithmeTM(Suiv)

Fin Si

Sinon Si Moduletraitee = Predecess

Faire Marque de ModuleDepart = Faux

Faire AlgorithmeTM(Predecess)

Fin Si

Fin Si

Fin

Nous remarquons que cet algorithme fait appel aux résultats de l'algorithme précédant pour définir la position des modules constituant la chaîne de traitement. Et par la suite, il permet de déterminer la règle à appliquer pour chaque module.

Exemple d'application de l'algorithme :

La première étape dans cet exemple AlgorithmeTM marque le premier module M1 comme non exécutable et le module M2 comme exécutable en utilisant l'algorithme AlgoModuleNonTraitable. La deuxieme étape est la détermination de la règle à appliquer pour le module M2 qui est la règle de composition R2.

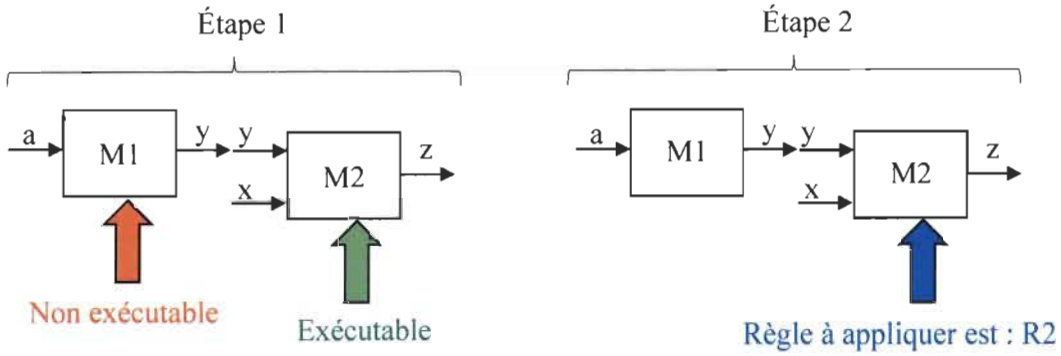


Figure 34 Exemple d'application de l'algorithme AlgorithmeTM

Cet exemple montre que c'est le module M2 qui va être exécuté par notre programme parce qu'il a un prédécesseur et la règle qui le gère est R2.

4.5.3 Algorithme de traitement la chaîne de modules AlgorithmeTC

Variable

Marque de type tableau de booléens
ModuleEntree de type Liste
TailleTypeE
GrapheChaine de type Matrice

Début AlgorithmeTC (GrapheChaine)

Tant que GrapheChaine ≠ Nulle faire

Pour chaque élément de Predecceur Faire

Ajouter le type à ModuleEntree

Fin Pour

Fin Tant que

Tant que ModuleEnEntree ≠ 0 faire

Si Type du ModuleChoisie [0] = dernier Element ModuleEntree

Si TailleTypeE ≤ 2

Faire appliquer la règle de composition a pour introduire le combinateur B sur l'expression du module choisi

Faire décrémenter ModuleEnEntrée

Faire appliquer la règle de composition a sur le type du Module choisit

Sinon

Faire appliquer la règle de composition a pour introduire le combinateur B^2 sur l'expression du Module choisi

Faire décrémenter ModuleEnEntrée

Faire appliquer la règle de composition à B^2 sur le type du Module choisit

Fin Si

Sinon Si les éléments 0 et 1 du type du module choisit = l'avant dernier et le dernier élément de ce type

Faire appliquer la règle de *S* sur l'expression

Faire appliquer la règle de composition a pour introduire le combinateur *S* sur l'expression du Module choisi

Faire décrémenter ModuleEnEntrée

Sinon Si le dernier élément du type du module En entrée = l'avant dernier et le dernier élément du type du Module choisi

Faire appliquer la règle de *C* sur l'expression

Faire appliquer la règle de composition a pour introduire le combinateur C sur l'expression du Module choisi

Sinon Si le troisième élément du type du module choisi = le dernier élément du type du Module en entrée

Faire appliquer la règle de *S* sur l'expression

Faire appliquer la règle de composition a pour introduire le combinateur S sur l'expression du Module choisi

Fin si

Si Type du module Choisi = Type du Module en entrée

Faire appliquer la règle de W sur l'expression

Faire appliquer la règle de composition a pour introduire le combinateur W sur l'expression du Module choisi

Fin Si

Faire Marque traitée = vrai

Fin tant que

Fin Tant que

Fin

Exemple d'application de l'algorithme :

Prenons le même agencement donné comme exemple pour les deux algorithmes précédents. Dans un premier temps, cet algorithme détermine le caractère exécutable et la règle à appliquer pour chaque module comme montré dans la figure suivante.

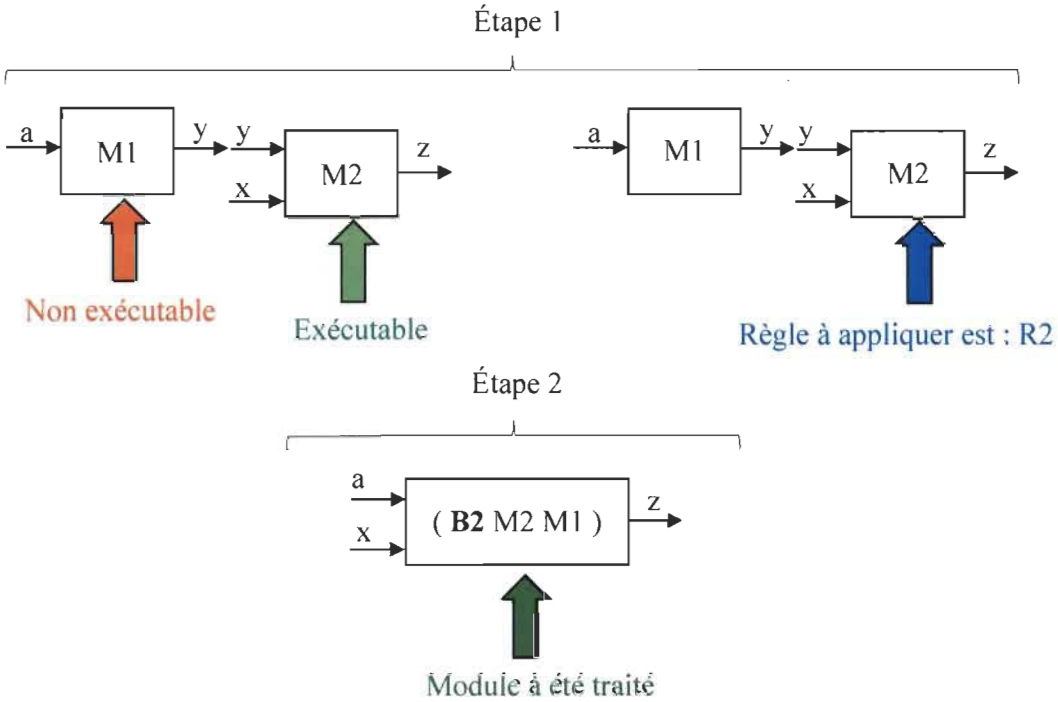


Figure 35 Exemple d'application de l'algorithme AlgorithmeTC

La deuxième étape consiste à appliquer la règle **R2** déterminée pour le module M2. Donc la règle de composition a été appliquée pour introduire le combinateur B sur l'expression du module M2, puis M1 a été désigné comme module en entrée de M2 et finalement générer l'expression **B** M2 M1 et le type FaFyz pour le nouveau module généré.

4.6 Étude de cas

Le cas étudié dans cette section représente un exemple d'application de notre modèle et les algorithmes pour passer de la théorie à l'application. La première étape est la construction d'une chaîne basée sur des modules que notre plateforme offrira aux

utilisateurs. Puis, nous allons créer l'arbre d'exécution et le marquage des modules selon nos algorithmes. Après cela, les règles seront appliquées selon l'ordre de l'arbre d'exécution. Vers la fin, on trouvera comme résultat la chaîne réduite avec des combinateurs introduits.

La construction du traitement de la chaîne :

La figure suivante est construite à base de cinq modules, qui sont M1 : Fyz, M2 : Fxy, M3 : Fza, M4 : Fyz et M5 : FaFzy. La construction se fait en branchant la sortie M1 avec l'entrée z de M3, en respectant l'égalité entre la sortie et l'entrée connectées des deux modules, et avec le même principe en interconnectant tous les modules utilisés.

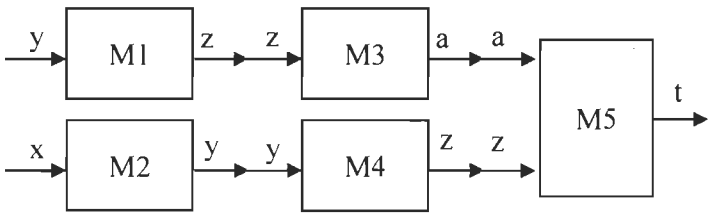


Figure 36 Cas étudié

Notre algorithme détermine le module prédécesseur et le module suivant de chaque module de la chaîne. De fait, le traitement commencera par réduire M1 et M3, ce qui produit le module (**B** M3 M1) de type Fya, puis ce module avec M5 pour créer (**B** M5 (**B** M3 M1)) de type FyFzt. La troisième étape est la composition de M2 et M4, qui donne (**B** M4 M2), comme montré dans la Figure 39. Dans ces trois premières étapes, c'est la règle de composition (1) qui s'applique. La règle de permutation s'applique dans la quatrième étape (Figure 40 pour donner en résultat le module complexe (**C** (**B** M5 (**B** M3 M1)) avec type FzFyt. La cinquième étape établit la composition des deux modules complexes restants pour donner un module avec expression finale d'un module (**C** (**B** M5 (**B** M3

M1)) (**B** M4 M2))) et type FzFyt. Les figures suivantes représentent les étapes du traitement pour le cas étudié.

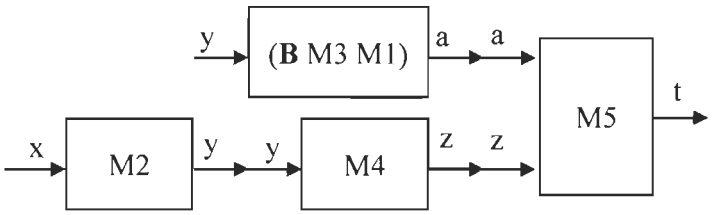


Figure 37 Étape 1 du traitement

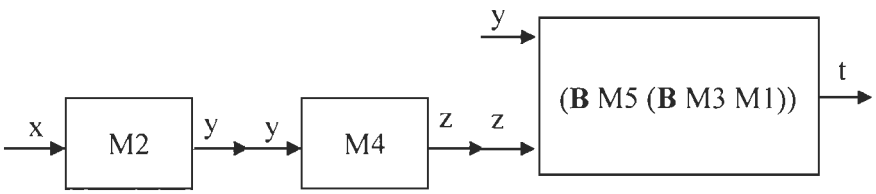


Figure 38 Etape 2 du traitement

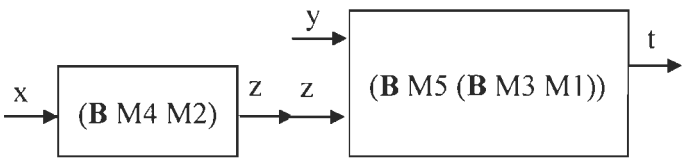


Figure 39 Étape 3 du traitement

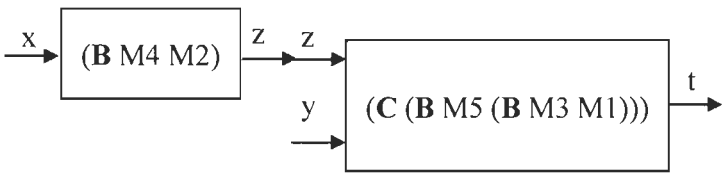


Figure 40 Étape 4 du traitement

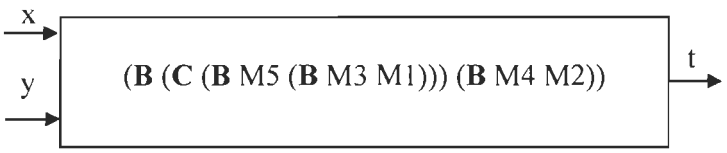


Figure 41 Étape 5 du traitement

4.7 Conclusion

Ce chapitre définissait le modèle théorique de base sur lequel nous avons appliqué un modèle applicatif. De fait, nous avons pu déceler les cas possibles de construction et en même temps déterminer les règles qui les régissent. Aussi, nous avons expliqué ce passage de la théorie vers l'applicatif algorithmiquement et donné un exemple d'application de ce dernier.

Après ce passage de la théorie vers l'application et l'étude de cas, il ne reste qu'à commencer la concrétisation du modèle par l'implantation informatique. Cette dernière validera le modèle applicatif et sa robustesse, qui reposent sur les outils fonctionnels présentés au chapitre précédant, comme la logique combinatoire et les grammaires applicatives.

Chapitre 5

IMPLÉMENTATION ET EXPÉRIMENTATIONS

5.1 Introduction

Le modèle applicatif et les algorithmes du chapitre 3 représentent une infrastructure aidant à mettre en place l'architecture, ainsi que les outils présentés au chapitre 2, qui sont une base pour la mise au point de notre système NAILI (Nouvelle Architecture de l'Ingénierie de la Langue et de l'Information).

Dans ce chapitre, nous présenterons son architecture, ainsi que les éléments de l'implémentation. Enfin, nous discuterons des expérimentations en quatre points : cas reconnus avec succès, cas reconnus qui n'auraient pas dus être reconnus (surgénération), cas non reconnus avec succès et cas non reconnus qui auraient dus être reconnus (sous-génération).

5.2 Architecture du système

En se basant sur l'architecture des systèmes experts et nos besoins en matière de programmation du modèle présenté dans les chapitres précédents, notre système se caractérise par une architecture répartie en trois grands modules :

- Le premier regroupe les composants programmés fonctionnellement. Plus précisément, c'est une bibliothèque de fonctions programmées en F#, permettant l'analyse et le traitement de la chaîne donnée en entrée,

- le deuxième est la partie de l'interface utilisateur composée de classes et de modules permettant l'affichage des modules et la base pour la construction de chaîne et

- le troisième regroupe les sous-programmes de vérification et de préparation des entrées de la partie fonctionnelle. Ainsi, il contient les programmes sous format XAML assurant l'intermédiation entre le fonctionnel et la présentation.

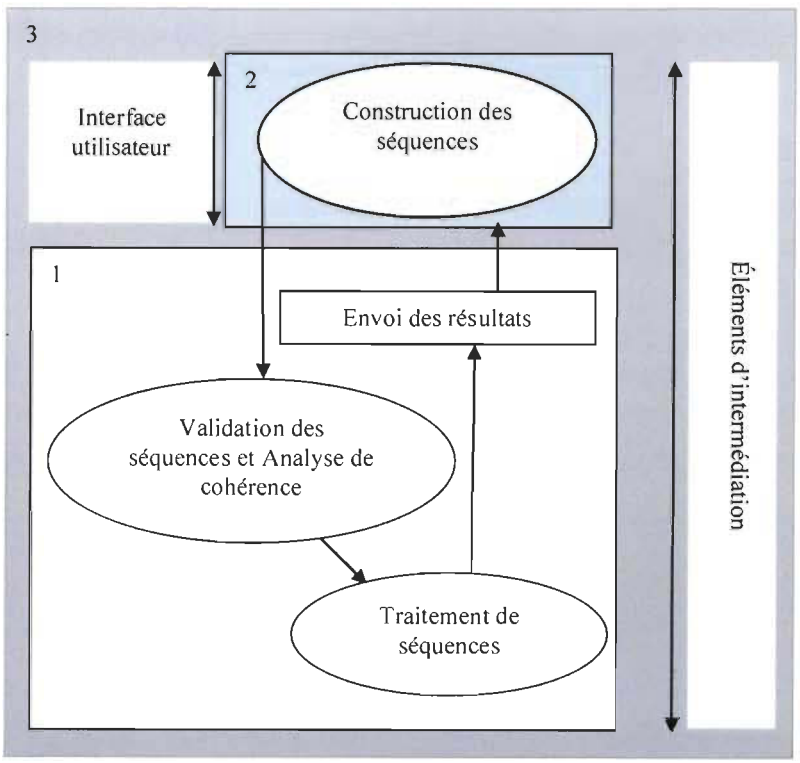


Figure 42 Architecture globale du système

5.3 Implémentation

5.3.1 Outils et langages choisis pour l'implémentation

Afin d'implémenter le module de présentation, nous avons utilisé une bibliothèque de visualisation graphique open-source appelée GraphX, qui prend en charge différents algorithmes de mise en page et des caractéristiques hautement personnalisables. Les bibliothèques principales de GraphX peuvent être utilisées à la fois en C # et VB .NET en utilisant WPF, parce que GraphX est un produit gratuit et ouvert à la collaboration [31].

Pour programmer le module fonctionnel, nous avons utilisé le langage F#, car il est construit en un paradigme purement fonctionnel et présente plusieurs avantages, lesquels sont démontrés dans l'Annexe C. Tous les outils constituant notre modèle applicatif ont une caractéristique fonctionnelle.

5.3.2 Choix et représentation des modules

Les modules sont représentés graphiquement comme une boite contenant un traitement ou une fonction quelconque, ainsi qu'un type d'entrée et un type de sortie. Le choix des modules utilisés dans notre système assure de multiples possibilités de compositions de chaînes, car leurs types permettent de les interconnecter de plusieurs façons et, en même temps, ils présentent une complexité au niveau de branchement puisque nous devons respecter les contraintes de connectivité ainsi que la cohérence des constructions de chaînes de traitement. Bien évidemment, ce choix permettra d'évaluer la robustesse de notre modèle. Ainsi, nous avons choisi sept modules, dont quatre avec une entrée et deux à deux entrées, en plus d'un module à trois entrées, qui sont listés dans le tableau suivant avec leurs types :

Tableau 4 La représentation des modules du système

Module	Type	Module	Type
	Fzx		FtFzu
	Fxy		FzFaFzu
	Fyz		FaFzt
	Fza		

5.3.3 Les interfaces

NAILI se compose de trois interfaces utilisateurs. La première contient une page d'accueil (voir figure suivante) présentant le projet avec un guide d'utilisateur et expose le lien d'accès pour commencer les assemblages de modules ou les chaînes de traitement et un lien redirigeant vers les pages du guide de l'utilisateur.

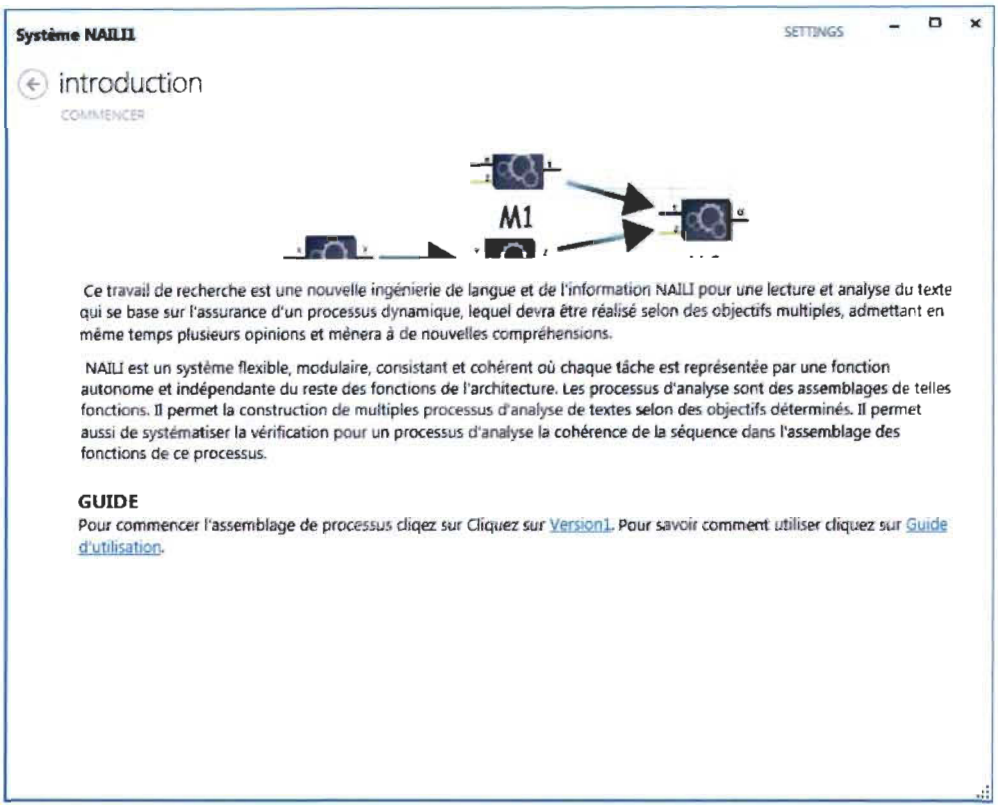


Figure 43 Interface de présentation du projet

La deuxième est l'interface principale (voir Figure 44), où il est possible de choisir les modules depuis la barre d'outils, les assembler dans la zone d'assemblage pour créer une chaîne de traitement selon le besoin de l'utilisateur, lancer le traitement et voir le résultat de l'analyse avec une possibilité d'imprimer les résultats ou les enregistrer sous format XPS ou PDF. L'enregistrement comprend la chaîne construite avec le résultat du traitement et les expressions des modules utilisés. En bas de la partie droite de la zone d'assemblage, un outil de zoom est disponible pour agrandir ou diminuer la taille du schéma construit et aussi pour le déplacer entièrement, afin d'assurer un affichage convenable. Le système interagit avec l'utilisateur par la barre d'interaction, où il sera indiqué l'état du traitement et quelques informations sur les éléments utilisés et les messages importants.

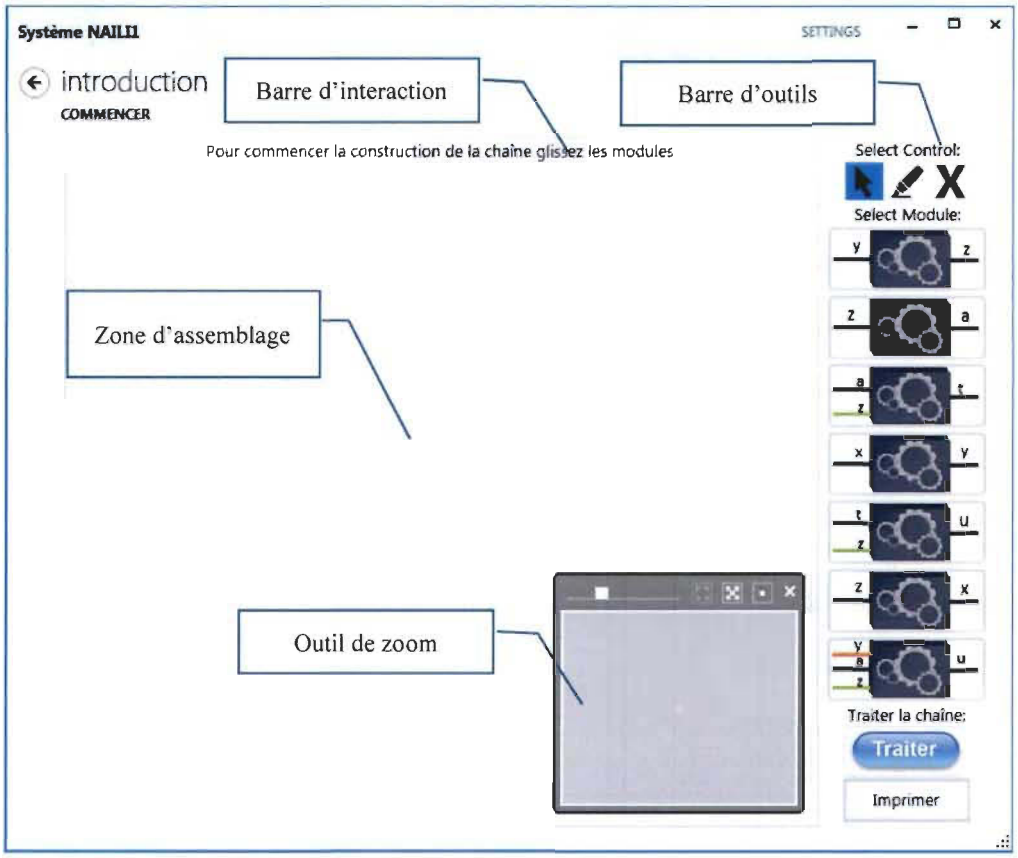


Figure 44 Interface principale

Finalement, une troisième interface comporte le guide d'utilisation; elle décrit l'utilisation du système et comment procéder à l'assemblage, ainsi que d'autres informations facilitant l'exploitation du système. Nous avons mis le guide d'utilisateur en annexe B, que nous considérons comme une première documentation du système.

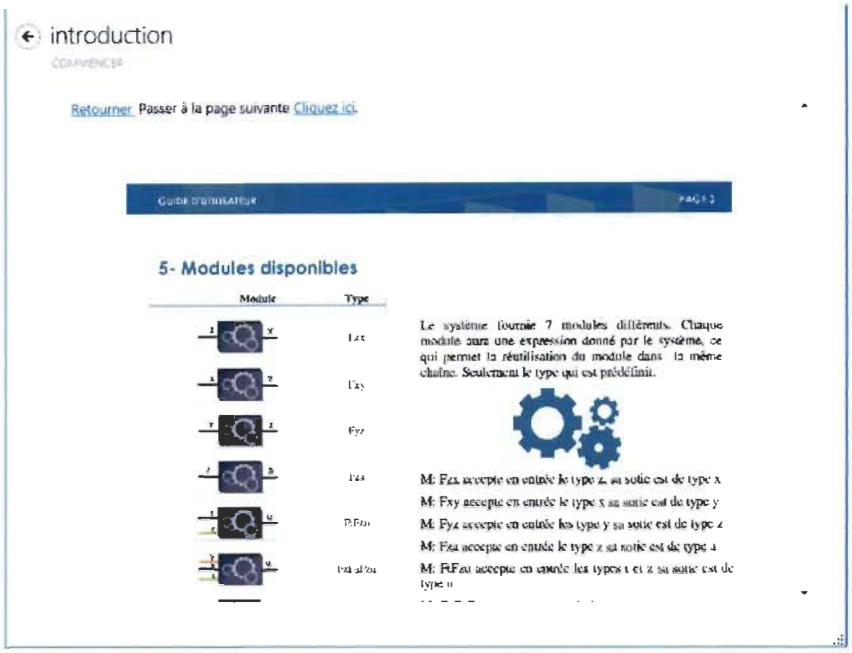


Figure 45 Interface du guide d'utilisation

5.3.4 Exemple de construction

Dans la construction suivante, nous constatons que les modules utilisés sont ordonnés de la manière dont ils sont ajoutés à la zone d'assemblage. Autrement dit, nous avons mis M1 avant M2 et après M3 jusqu'à M7, comme le montre la numérotation automatique. De plus, la connexion de ces derniers a été faite de la manière suivante : M1 avec M4, M4 avec M7, puis M2 avec M5, M5 avec M7, M3 avec M6 et M6 avec 7. La figure suivante montre graphiquement la chaîne construite :

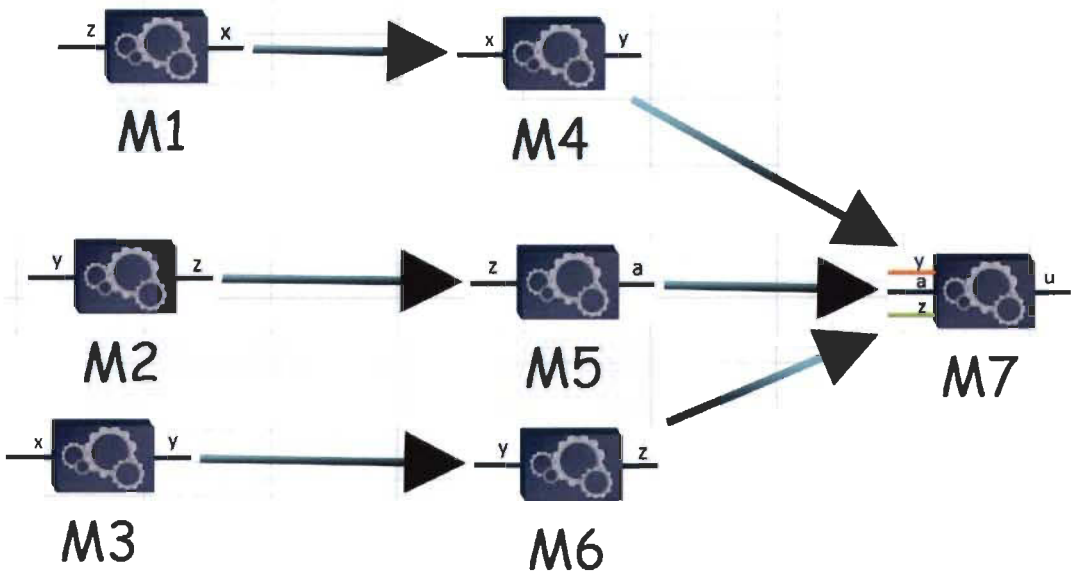


Figure 46 Exemple de construction d'une chaîne

5.3.5 L'ordre d'exécution

La chaîne de traitement de l'exemple précédent (Figure 46) peut être représentée par l'arbre à sept sommets, de M1 à M7. Pour en sortir un arbre d'exécution linéaire, il faut supprimer, dans un premier temps, les feuilles. Nous pouvons remarquer dans l'application de notre algorithme que l'application de règles commence par le module qui a un obligatoirement un module précédent, c'est pour cette raison nous aurons besoin d'éliminer les feuilles de l'arbre. Autrement dit, il faut enlever les nœuds qui n'ont pas de fils. Dans un deuxième temps, il faut les marquer (les numéroter) en respectant l'ordre de suppression (voir Figure 47). Après le parcours de cet arbre, l'ordre d'exécution peut être déterminé (voir Figure 48).

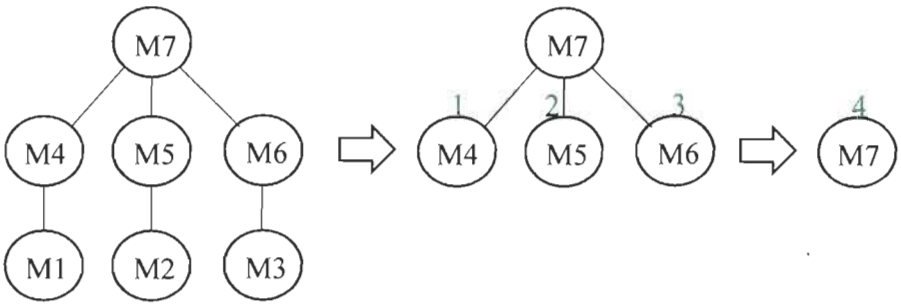


Figure 47 L'arbre initial de la chaîne de traitement

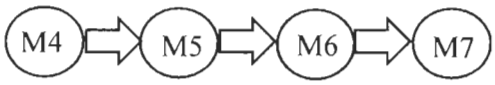


Figure 48 L'ordre d'exécution

Le traitement commence donc par l'évaluation des règles à appliquer puis à exécuter au gré de l'ordre d'exécution de la Figure 48 : M4 suivi de M5, puis M6 et finalement M7.

5.3.6 L'enregistrement des étapes de traitement

Chaque étape de traitement d'une chaîne s'enregistre dans les fichiers tests des résultats de traitements. Prenons l'exemple de construction précédant; l'enregistrement de son traitement comprend huit étapes :

1. (**B** M4 M1)
2. (**B** M5 M2)
3. (**B** M6 M3)
4. (**B** M7 (**B** M4 M1))
5. (**C** (**B** M7 (**B** M4 M1)))
6. (**B** (**C** (**B** M7 (**B** M4 M1))) (**B** M5 M2))
7. (**C** (**B** (**C** (**B** M7 (**B** M4 M1))) (**B** M5 M2)))
8. (**B** (**C** (**B** (**C** (**B** M7 (**B** M4 M1))) (**B** M5 M2))) (**B** M6 M3))

L'emplacement prédéfini du fichier des enregistrements est : « C:\resultat ». Voici un extrait de ce dernier :

```
(B M2 M1)[Y; A]
(B M4 (B M2 M1))[Y; Z; I]
(S (B M4 (B M2 M1)) M3)[Y; I]
(B M6 (S (B M4 (B M2 M1)) M3))[Y; Z; U]
(S (B M6 (S (B M4 (B M2 M1)) M3)) M5)[Y; U]
------------- Fin de l'enregistrement ---------------
(B M3 M1)[Z; Z; I]
(B (B M3 M1) M2)[Y; Z; I]
(B2 M5 (B (B M3 M1) M2))[Y; Z; Z; ... ]
(S (B2 M5 (B (B M3 M1) M2)) M4)[Y; Z; U]
------------- Fin de l'enregistrement ---------------
(B M3 M1)[Z; Z; I]
(B (B M3 M1) M2)[Y; Z; I]
(B M4 M6)[X; Z]
(B2 M5 (B (B M3 M1) M2))[Y; Z; Z; ... ]
(C (B2 M5 (B (B M3 M1) M2)))[Z; Y; Z; ... ]
(B (C (B2 M5 (B (B M3 M1) M2))) (B M4 M6))[X; Y; Z; ... ]
------------- Fin de l'enregistrement ---------------
(B M3 M1)[Z; Z; I]
```

Figure 49 Extrait du fichier des enregistrements

5.4 Expérimentations

5.4.1 Cas reconnus avec succès

D'après les tests de fonctionnement du système développé, nous avons listé des cas reconnus avec succès. Il s'agit d'agencements supportés par notre modèle, qui doivent être reconnus par le système NAILI. Après les avoir traités par ce dernier, nous avons obtenu une expression applicative typée avec les combinateurs corrects; de plus, cela a donné le résultat attendu. L'exemple de construction étudié dans la section d'implémentation est parmi les cas reconnus avec succès, de même que l'agencement suivant, entre M1 : Fyz et M2 : Fza, est réussi, car il respecte toutes les restrictions du modèle sur lequel nous nous basons. Enfin, le système a donné le résultat attendu, comme le montre la figure suivante.

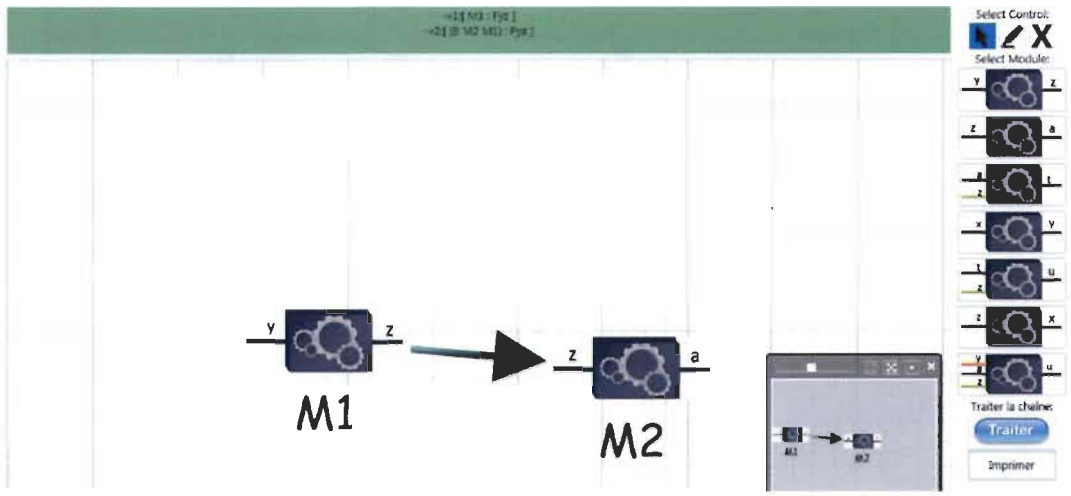


Figure 50 Exemple reconnu avec succès

Nous avons effectué plus de trente tests réussis (voir Annexe A, laquelle contient également certains enregistrements). Ils sont classés en deux sortes; la première regroupe les agencements en série et la deuxième regroupe les agencements en parallèle. Parmi les agencements en série, nous présentons, dans la Figure 51, dix modules construisant une chaîne valide, reconnue par le modèle étudié, et après sa modélisation par NAILI, son traitement a été correctement analysé, comme le montre son résultat. L'expression applicative du module en résultat est (**B** M10 (**B** M9 (**B** M8 (**B** M7 (**B** M6 (**B** M5 (**B** M4 (**B** M3 (**B** M2 M1)))))))) et son type est Fxy. Les étapes enregistrées sur le fichier de résultats, pour ce cas, sont les suivantes :

1. (**B** M2 M1): Fxy
2. (**B** M3 (**B** M2 M1)): Fxx
3. (**B** M4 (**B** M3 (**B** M2 M1))): Fxy
4. (**B** M5 (**B** M4 (**B** M3 (**B** M2 M1)))): Fxz
5. (**B** M6 (**B** M5 (**B** M4 (**B** M3 (**B** M2 M1))))): Fxx
6. (**B** M7 (**B** M6 (**B** M5 (**B** M4 (**B** M3 (**B** M2 M1)))))): Fxy

7. (**B** M8 (**B** M7 (**B** M6 (**B** M5 (**B** M4 (**B** M3 (**B** M2 M1)))))): Fxz
8. (**B** M9 (**B** M8 (**B** M7 (**B** M6 (**B** M5 (**B** M4 (**B** M3 (**B** M2 M1)))))): Fxx
9. (**B** M10 (**B** M9 (**B** M8 (**B** M7 (**B** M6 (**B** M5 (**B** M4 (**B** M3 (**B** M2 M1))))))): Fxy

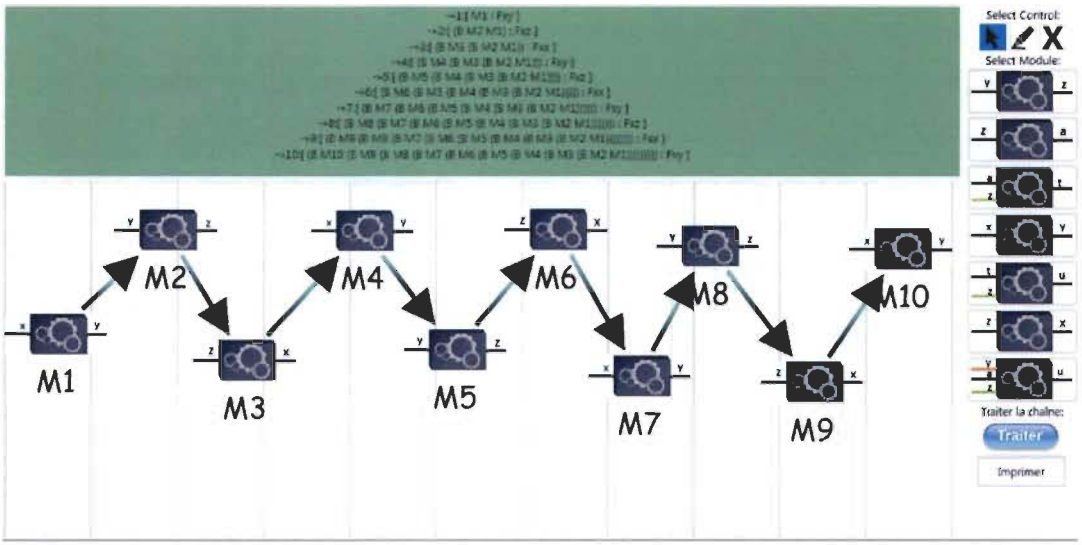


Figure 51 Exemple d'un agencement en série reconnu avec succès

Pour les agencements en parallèle, nous présentons dans la Figure 52 une chaîne valide construite par six modules, dont deux ont deux entrées; cette chaîne a été correctement analysée : le bon résultat a été obtenu. L'expression applicative typée avec les combinateurs de ce cas est (**S** (**B** M6 (**S** (**B** M4 (**B** M2 M1)) M3)) M5) et son type est Fyu. Les étapes enregistrées sur le fichier de résultats, pour ce cas, sont les suivantes :

1. (**B** M2 M1) : Fya
2. (**B** M4 (**B** M2 M1)) : FyFzt
3. (**S** (**B** M4 (**B** M2 M1)) M3) : Fyt
4. (**B** M6 (**S** (**B** M4 (**B** M2 M1)) M3)) : FyFzu
5. (**S** (**B** M6 (**S** (**B** M4 (**B** M2 M1)) M3)) M5): Fyu

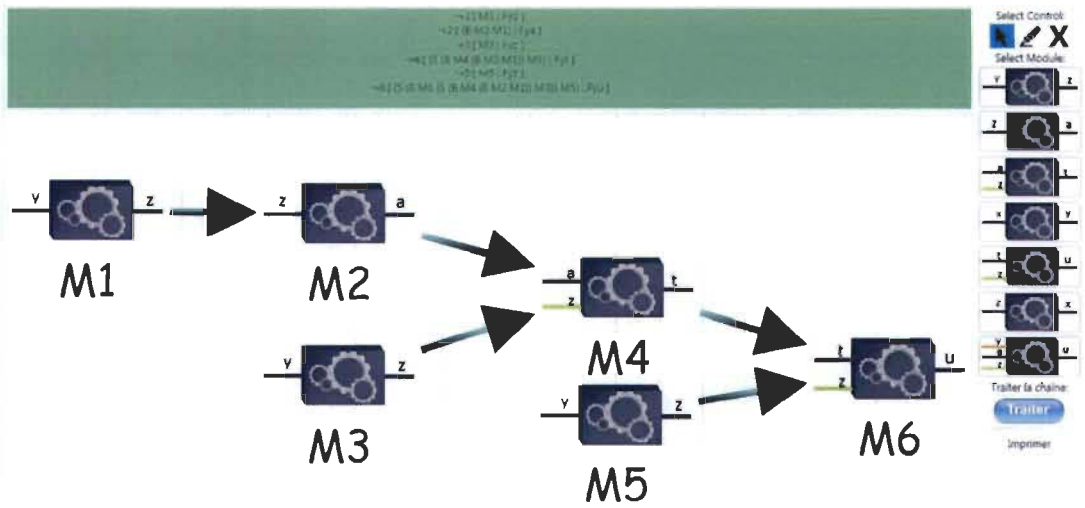


Figure 52 Exemple 1 d'un agencement en parallèle reconnu avec succès

Un autre exemple de cas de chaîne de traitement, présenté dans la Figure 53, contient un module de type FyFaFzu et cinq autres modules; le premier module a été testé et son résultat permet de le classer parmi les agencements en parallèle reconnu avec succès. Cette chaîne a pour résultat un module complexe de type FxFzu et une expression applicative typée avec les combinateurs suivants : (**B** (**S** (**B** M7 (**B** M4 M2)) M5) (**B** M6 M3)). Les étapes de traitement sont :

(**B** M4 M2) : Fzy
(**B** M6 M3) : Fxz
(**B** M7 (**B** M4 M2)) : FzFaFzu
(**S** (**B** M7 (**B** M4 M2)) M5) : FzFzu
(**B** (**S** (**B** M7 (**B** M4 M2)) M5) (**B** M6 M3)) : FxFzu

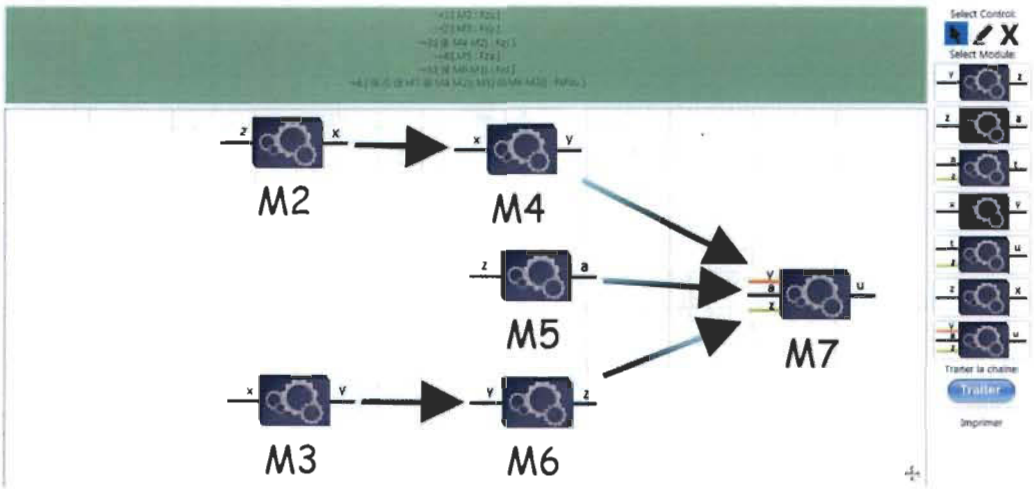


Figure 53 Exemple 2 d'un agencement en parallèle reconnu avec succès

5.4.2 Cas reconnus qui n'auraient pas dus être reconnus: surgénération

Lors du développement du système, dans sa partie fonctionnelle, il était possible de lui donner en entrée une chaîne de traitement qui n'est pas valide en termes de connectivité. Autrement dit, il est possible de traiter une chaîne qui est mal construite et qui ne répond pas aux restrictions de notre système présenté à la deuxième section du chapitre 3. Prenons, par exemple, M1 : Fxy, M2 : Fxy et M3 : Fxy. Il était possible de construire une chaîne à base de ces trois modules avec entrée et sortie identique en les connectant en série même si x est différent de y, ce qui n'est pas permis par le modèle.

Alors, l'utilisateur ne peut pas construire des chaînes non valides et un message d'erreur sera dans la barre d'interaction. C'est une implémentation des contraintes de connectivité présentée lors de la description du modèle.

5.4.3 Cas non reconnus avec succès

La gestion de la connectivité au moment de la construction de la chaîne ne permet pas de créer un assemblage non reconnu par notre modèle. Par exemple, connecter un module M1 : Fyz avec un module M2 : Fzy demeure impossible et un message d'erreur s'affiche pour indiquer l'incompatibilité entre ces deux modules. Avec cette gestion préalable, il est possible d'éviter de traiter ce genre d'agencement. Dans la Figure 54, le système ne reconnait pas la chaîne et ne permet pas sa construction; ainsi, notre modèle de base ne couvre pas son traitement.

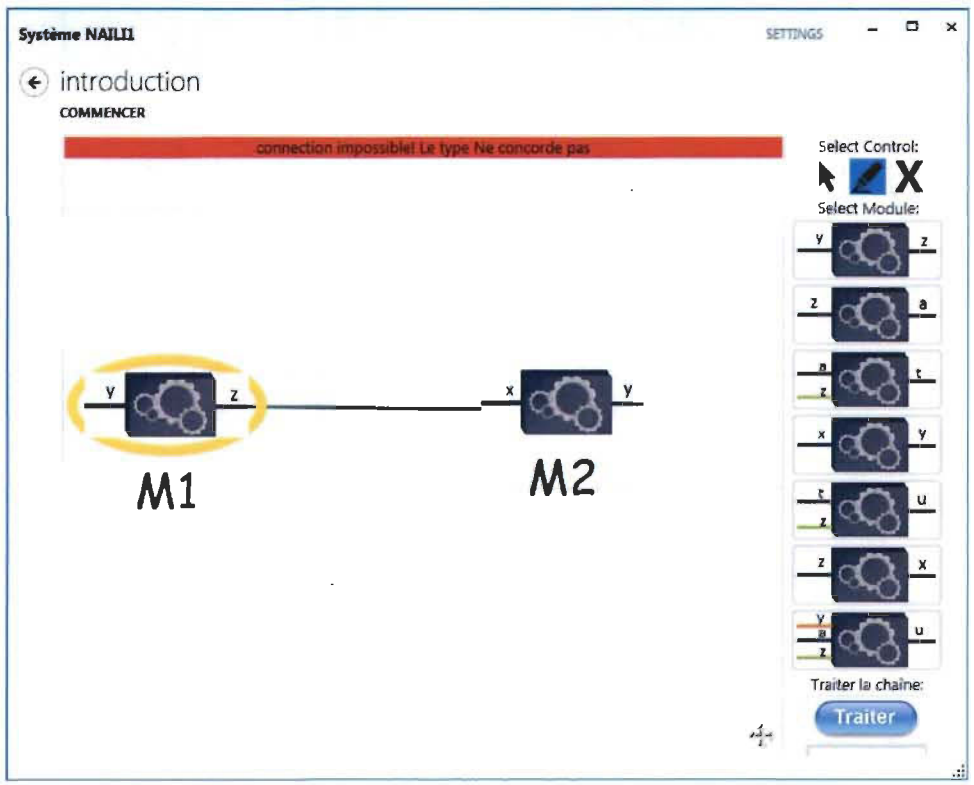


Figure 54 Exemple 1 d'un cas non reconnu avec succès

Le cas suivant est également non reconnu parce que le type x n'est pas disponible en entrée du module M5 : FyFaFzu. Après une tentative de connecter M4 avec M5, un message indiquant que le type ne concorde pas s'affiche dans la zone d'interaction.

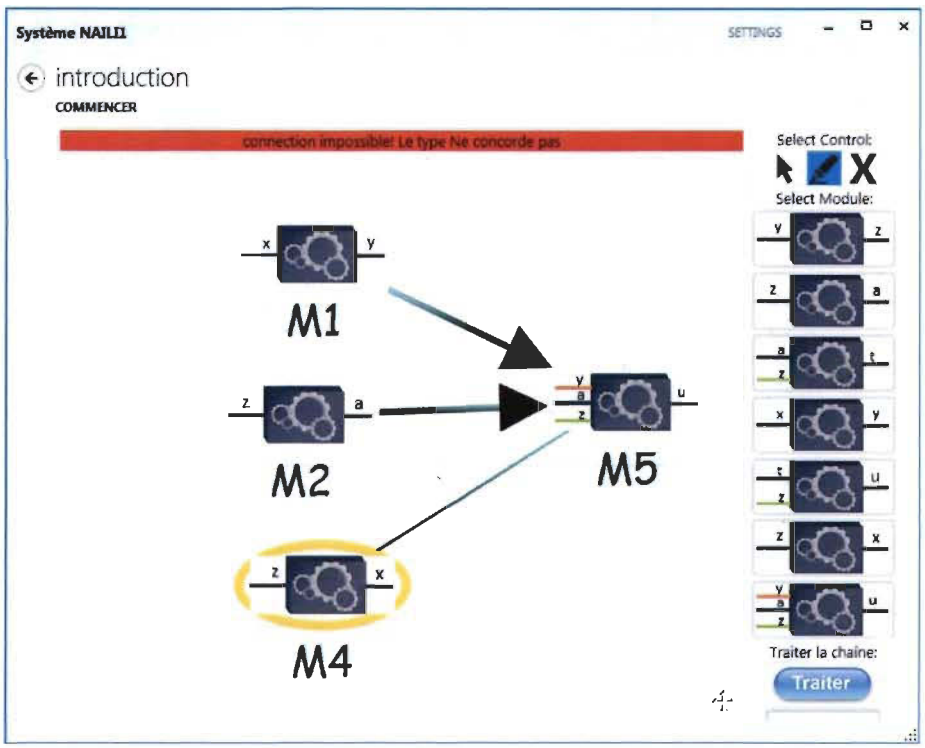


Figure 55 Exemple 2 d'un cas non reconnu avec succès

Il existe d'autres cas non reconnus par le modèle et par le système dans lesquels nous essayons de connecter un module quelconque avec un autre qui est déjà associé avec un troisième module, même si leurs types permettent une connexion valide. Dans de tels cas, on parle d'un problème de cardinalité : le nombre maximal d'associations possible est dépassé. En voici un exemple :

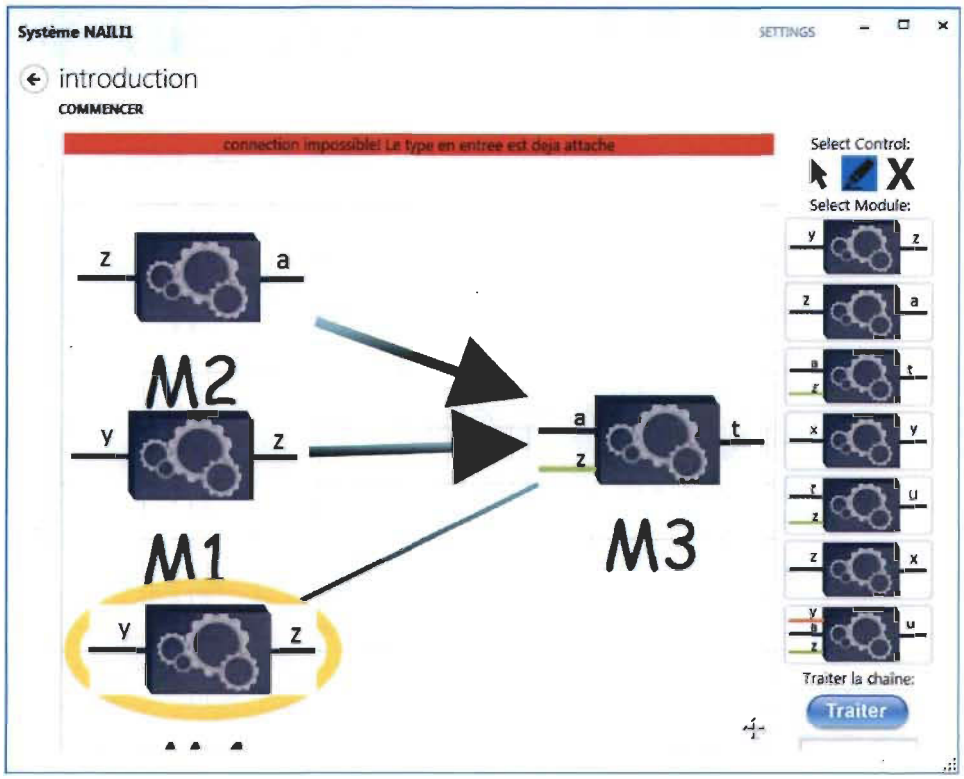


Figure 56 Exemple 3 d'un cas non reconnu avec succès

5.4.4 Cas non reconnus qui auraient dû être reconnus: sous-génération

Le problème de sous-génération peut être éprouvé lorsque le système n'est pas capable de reconnaitre les cas reconnus par le modèle étudié. En faisant les tests finaux, nous n'avons pas rencontré des cas avec ce phénomène, surtout lorsque change l'ordre de disposition de modules et de leurs interconnexions. En raison de ce changement d'ordre, certaines restrictions de notre modèle sont prises en compte alors qu'elles sont nécessaires pour que le système puisse bien créer le graphe à partir de la chaîne à traiter. Les restrictions de notre modèle jouent un rôle très important pour éviter le problème de sous-génération.

5.5 Conclusion

Ce cinquième chapitre nous a permis d'expliquer comment le système a été développé afin de répondre aux objectifs préalablement déterminés. De fait, nous avons vu la spécification du système, les choix des outils et une vue globale des interfaces et l'utilisation de cette dernière. Les expérimentations constituaient la partie de finalisation, vérification, validation du fonctionnement de notre architecture, pour terminer avec une discussion des cas, lesquels ont été classifiés en quatre classes.

En dressant un bilan des expérimentations, les cas reconnus avec succès reflètent la bonne évaluation des règles à appliquer, le bon fonctionnement du modèle, ainsi que le rôle des règles des grammaires applicatives dans l'analyse de l'information, de la validation et de la cohérence des séquences de fonctions. Le fait de ne pas rencontrer des cas non reconnus montrent que les restrictions sont bel et bien respectées par la plateforme développée et qu'ils sont valides pour éviter les générations erronées de l'expression applicative des séquences traitées. Les enregistrements des étapes nous montrent, par la comparaison des expérimentations identiques, les différences de résultat de traitement après certains changements, soit de l'ordre, soit de la disposition des modules lors de la construction des chaînes. En effet, grâce à ces enregistrements, nous avons pu démontrer qu'il n'y a pas des cas de sous-génération et les cas de surgénération en analysant les résultats avant et après les modifications mentionnées.

Chapitre 6

CONCLUSION ET PERSPECTIVES

L'ensemble des travaux de lecture dirigée, d'analyses, et de l'étude comparative des systèmes établis a un lien direct avec la résolution de la problématique. Ceux-ci forment une base enrichissante pour penser à une théorie innovante et sa mise en place. En outre, le passage de la théorie vers l'application donne naissance à une résolution de la problématique initiale qui a été clairement mentionnée au début du rapport. Avec tout cela et le paradigme fonctionnel, nous avons pu mettre en œuvre le système NAILI en gardant un intérêt supérieur à la flexibilité, à la cohérence, à la consistance et à la modularité. En outre, les composants offerts par le système représentent un ensemble d'exemples de modules produisant une complexité dans l'analyse des agencements crées avec ces modules; cela nous a aidés à tester les puissances des grammaires applicatives et de la logique combinatoire sur lesquelles nous nous sommes basés pour concevoir NAILI.

Le système NAILI admet les mises à jour non seulement au niveau architectural, mais aussi de l'expansion de ses fonctionnalités. Présentement, il a une architecture modulaire, simple et compréhensible, qui peut à tout moment être étendue en lui intégrant de nouveaux modules. Les composants du système actuel ont été développés avec une limite de sept modules pour commencer sa première version, et si nous souhaitons modifier ces composants ou ajouter d'autres composants avec d'autres types, il est toujours possible de le faire, et ce, grâce à la modularité du système. En effet, sa partie de la logique est séparée de la partie de la présentation.

Suite au bilan des expérimentations effectuées pour évaluer notre travail de recherche, nous avons constaté qu'il sera possible de rajouter quelques améliorations au niveau des algorithmes et du modèle, entre autres parce que la stratégie d'exécution du

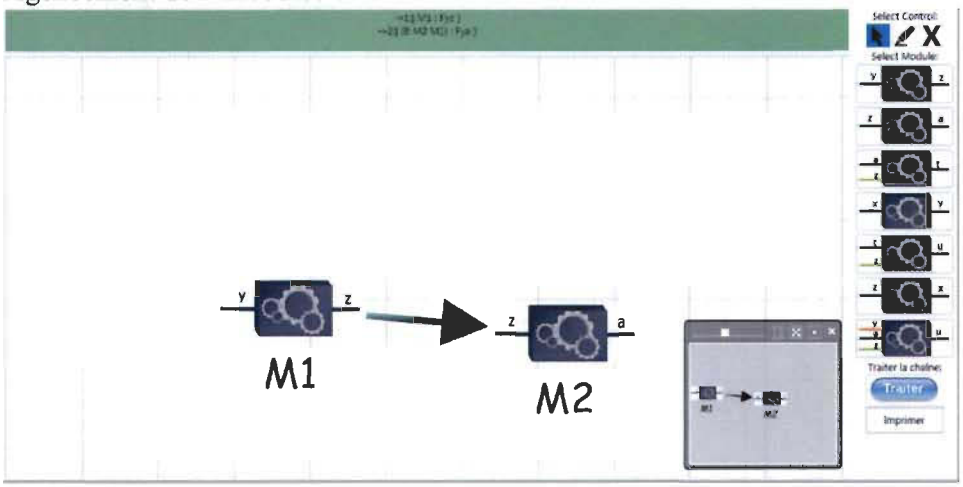
traitement développé dans NAILI est ouverte aux choix de l'utilisateur, ce qui peut engendrer la nécessité d'implémenter la suite des règles de la GCCA présentées dans le tableau 3. En effet, c'est avec les expérimentations qu'il est possible de détecter ce besoin. Enfin, nous pensons qu'une définition de l'ordre d'exécution des règles aidera à augmenter la consistance du modèle, car elle s'avère importante dans certains cas, puisqu'il arrive rarement des cas où deux règles sont applicables en même temps. Donc, une règle devra être appliquée avant l'autre. Dans notre modèle théorique, il y a une priorité d'application à suivre entre ces règles, selon l'ordre qu'elles sont présentées au rapport.

En résumé, le fait que notre architecture ne se limite pas juste à l'analyse des séquences de fonctions assemblées, nous sommes convaincus qu'elle est applicable dans l'internet des objets, qui est actuellement en émergence, parce qu'il est composé d'éléments de spécification variés, de différents types comme les puces RFID, de solutions de nommage ou de middlewares, etc. En outre, pour offrir de nouvelles fonctionnalités, ces composants peuvent être assemblés entre eux, ce qui créera un besoin d'analyse de leurs assemblages.

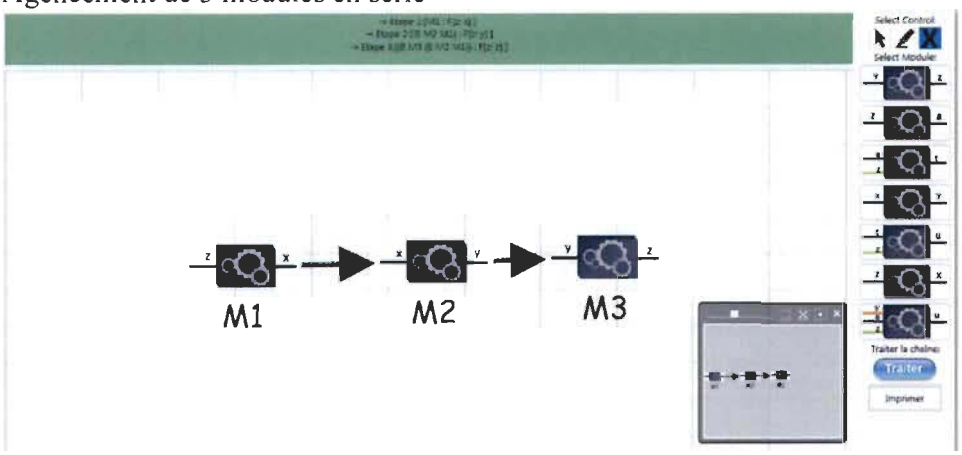
Annexe A : Expérimentations enregistrées

Cas reconnus avec succès : agencements en série

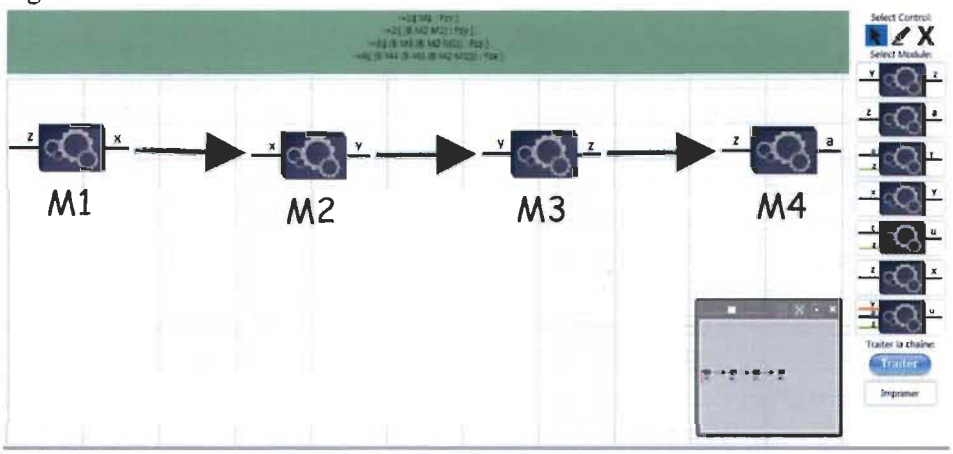
1. Agencement de 2 modules en série



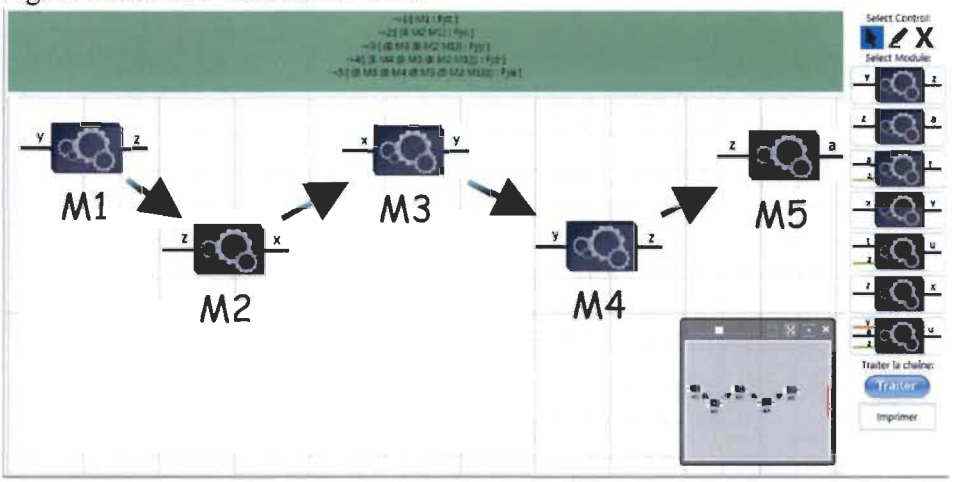
2. Agencement de 3 modules en série



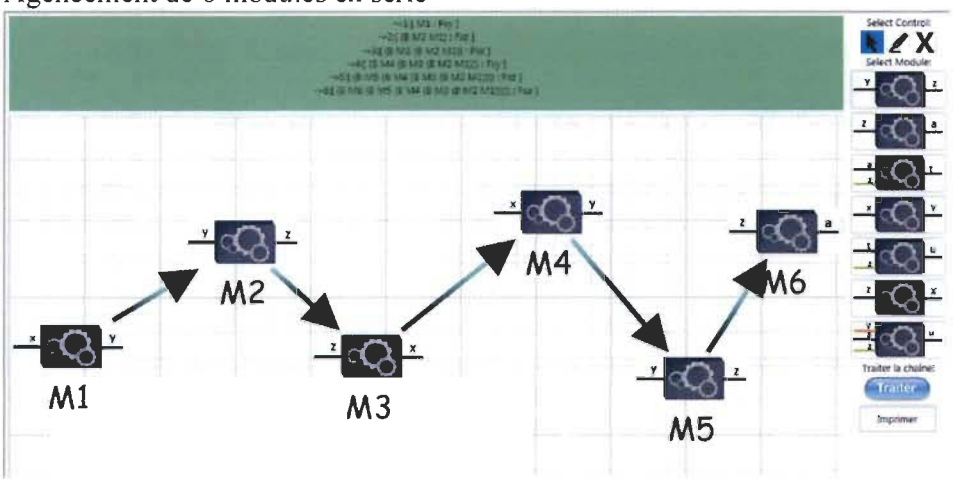
3. Agencement de 4 modules en série



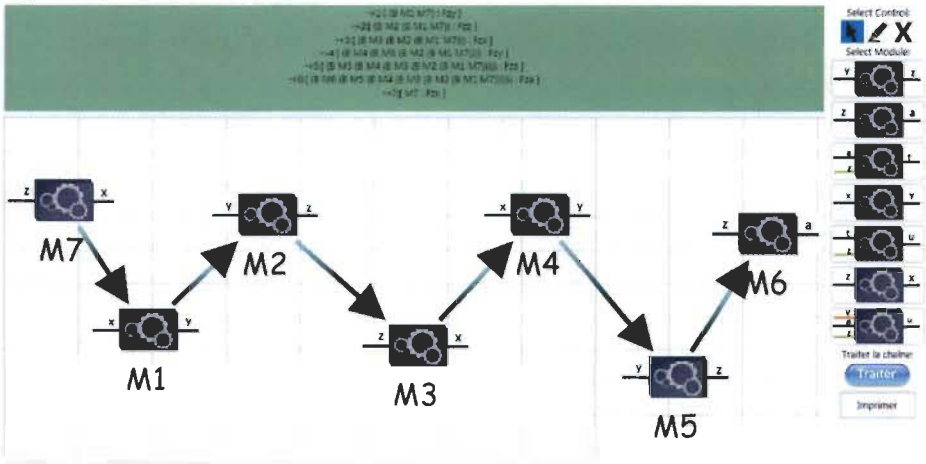
4. Agencement de 5 modules en série



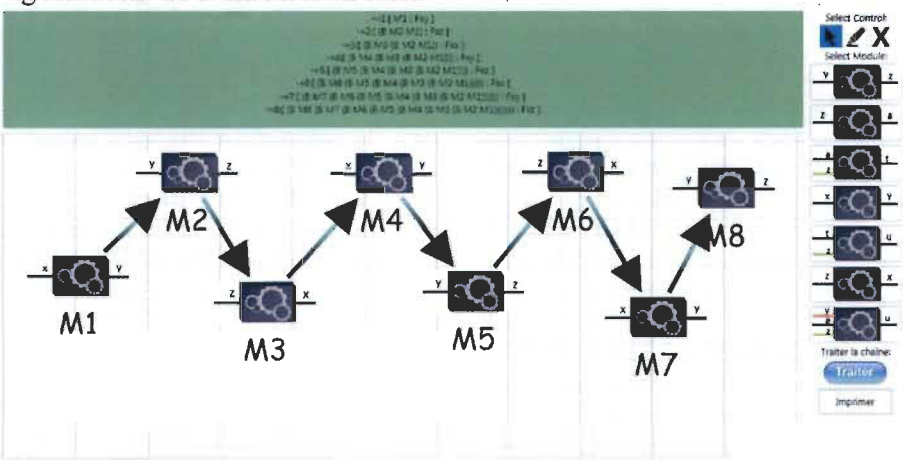
5. Agencement de 6 modules en série



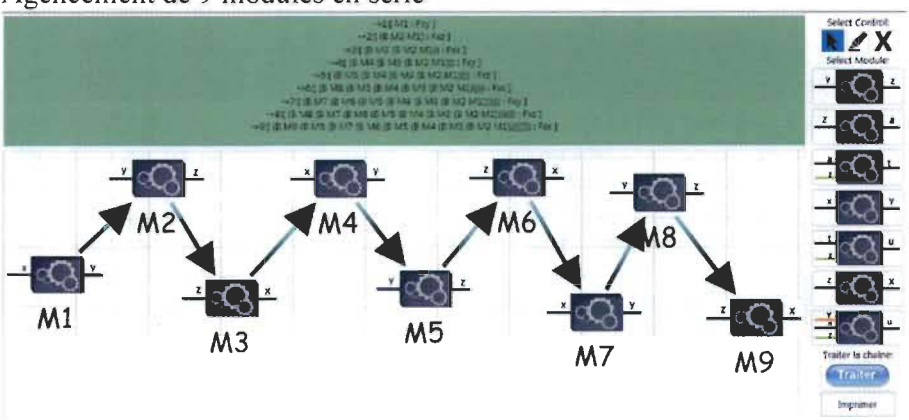
6. Agencement de 7 modules en série



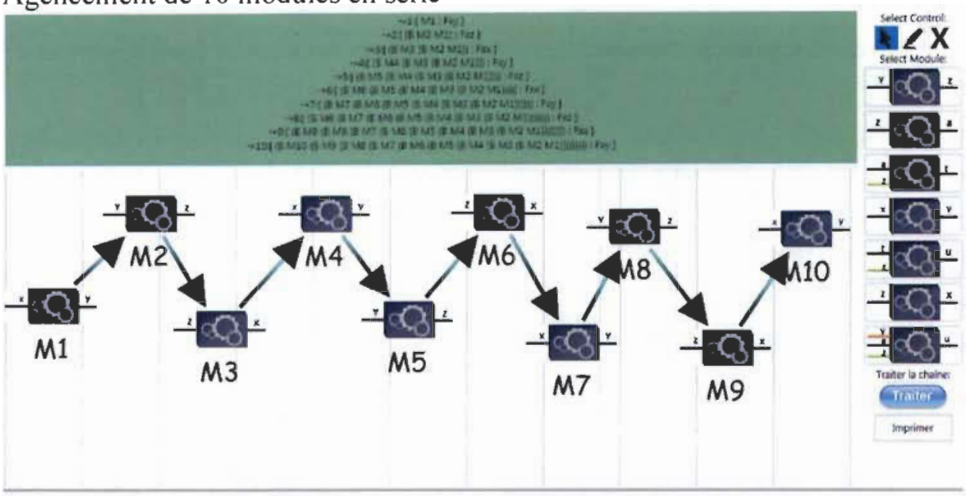
7. Agencement de 8 modules en série



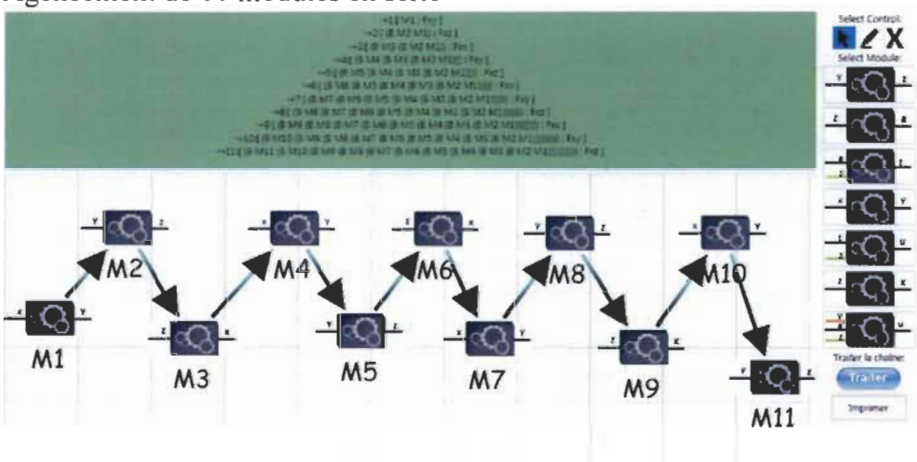
8. Agencement de 9 modules en série



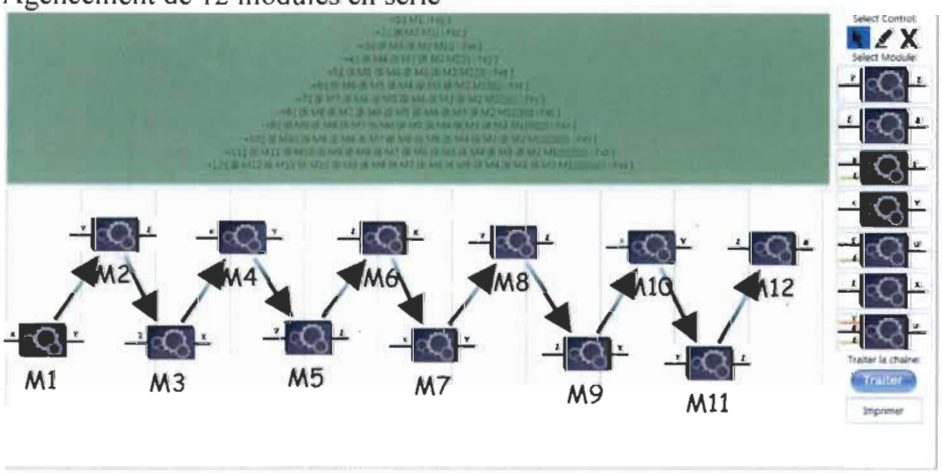
9. Agencement de 10 modules en série



10. Agencement de 11 modules en série

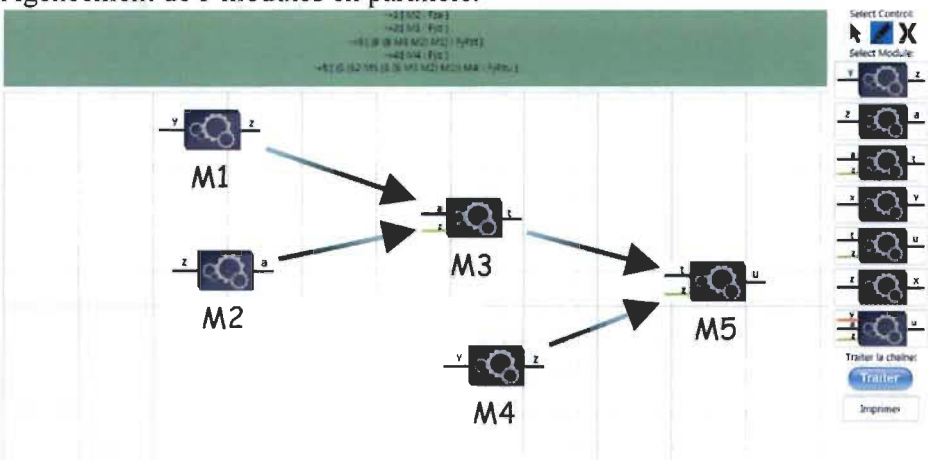


11. Agencement de 12 modules en série

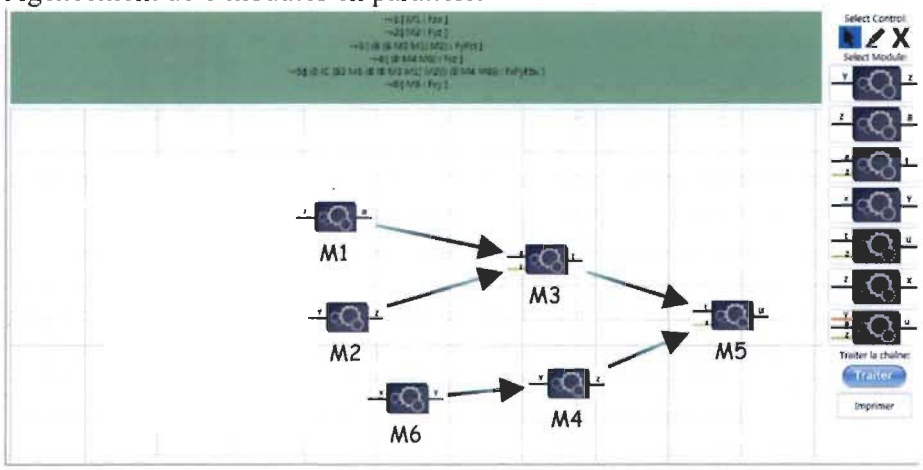


Cas reconnus avec succès : agencements en parallèle

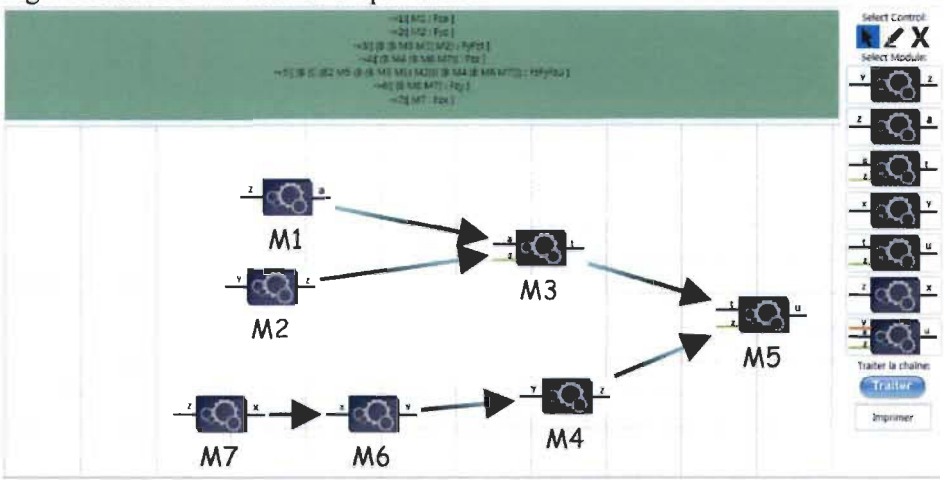
12. Agencement de 5 modules en parallèle.



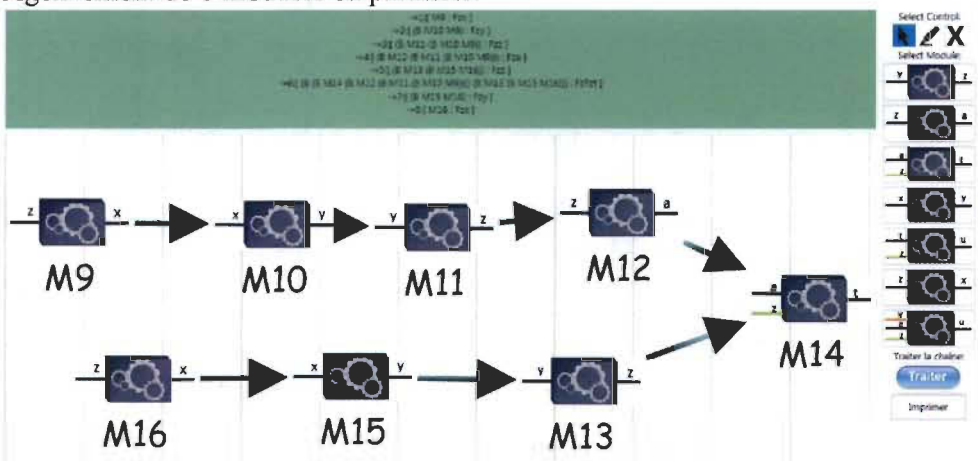
13. Agencement de 6 modules en parallèle.



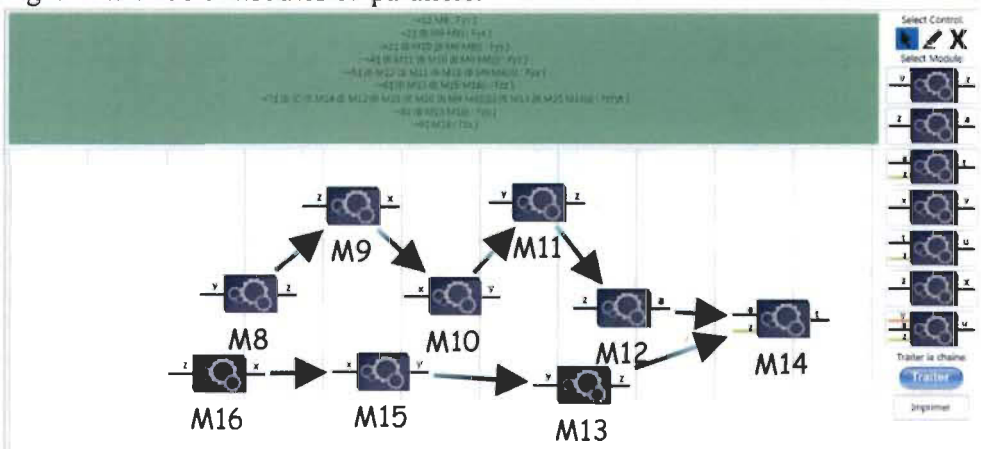
14. Agencement de 7 modules en parallèle.



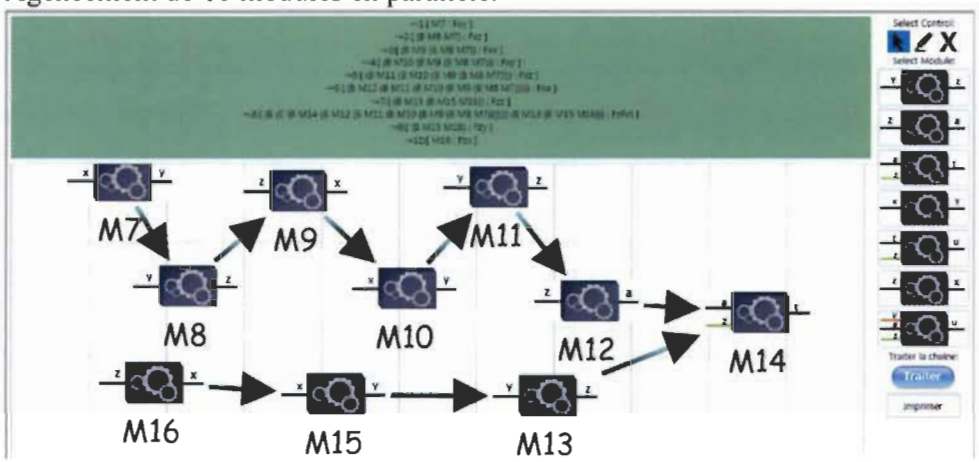
15. Agencement de 8 modules en parallèle.



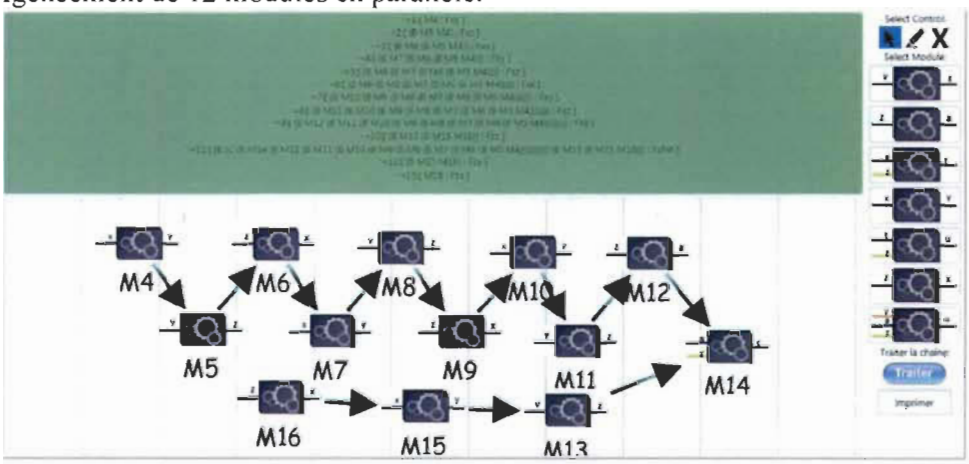
16. Agencement de 8 modules en parallèle.



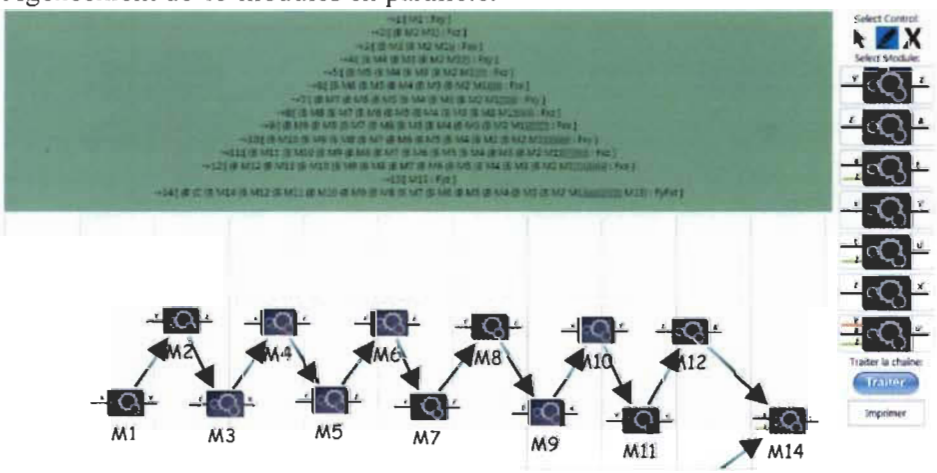
17. Agencement de 10 modules en parallèle.



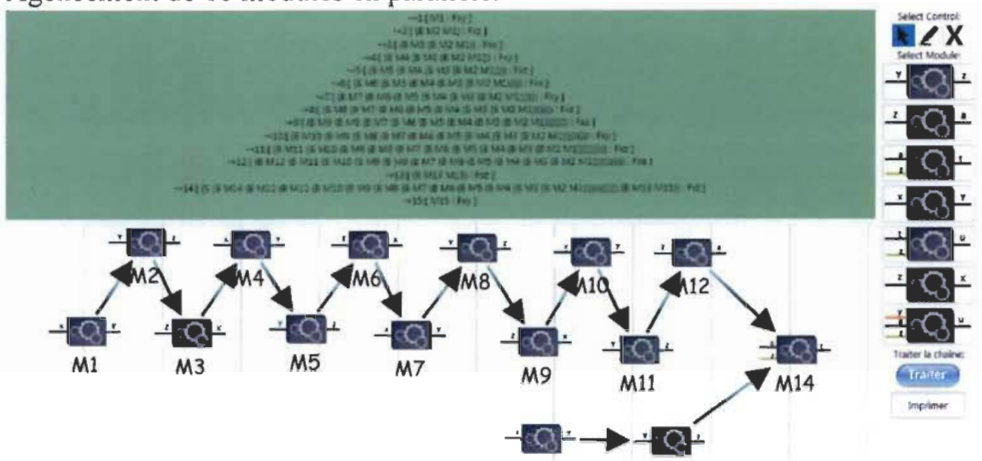
18. Agencement de 12 modules en parallèle.



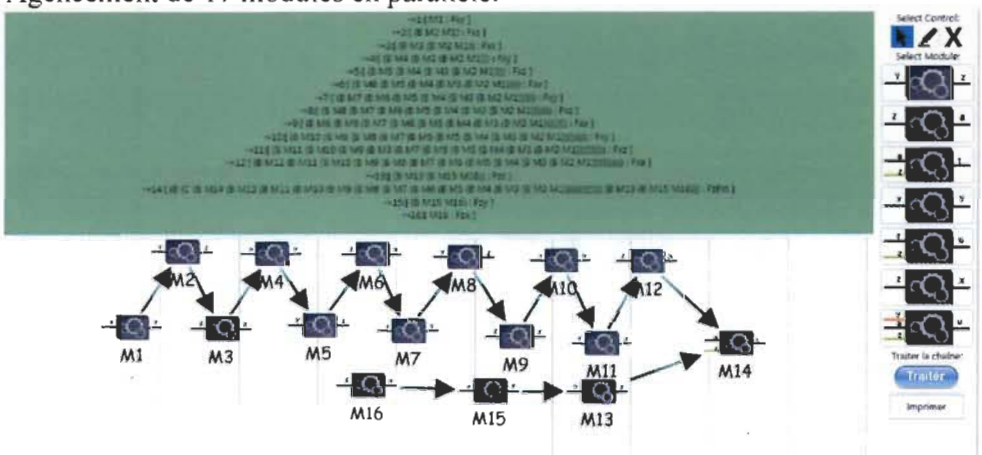
19. Agencement de 15 modules en parallèle.



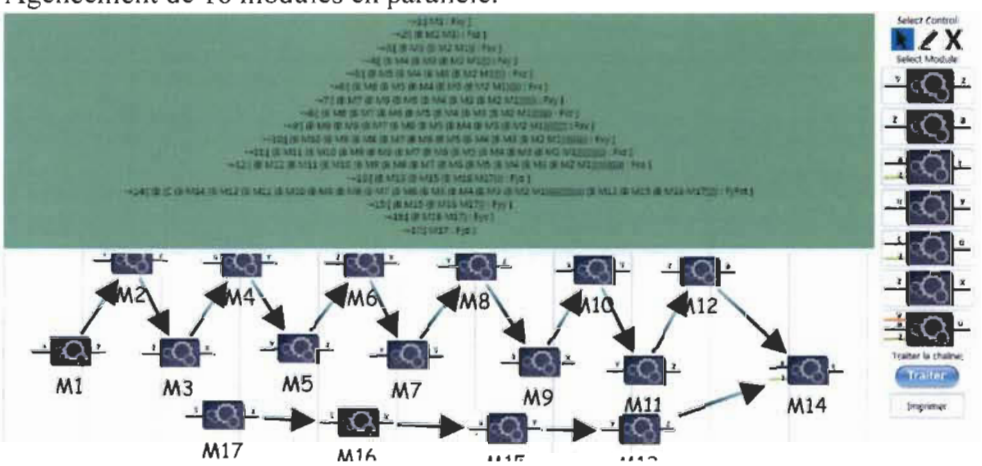
20. Agencement de 16 modules en parallèle.



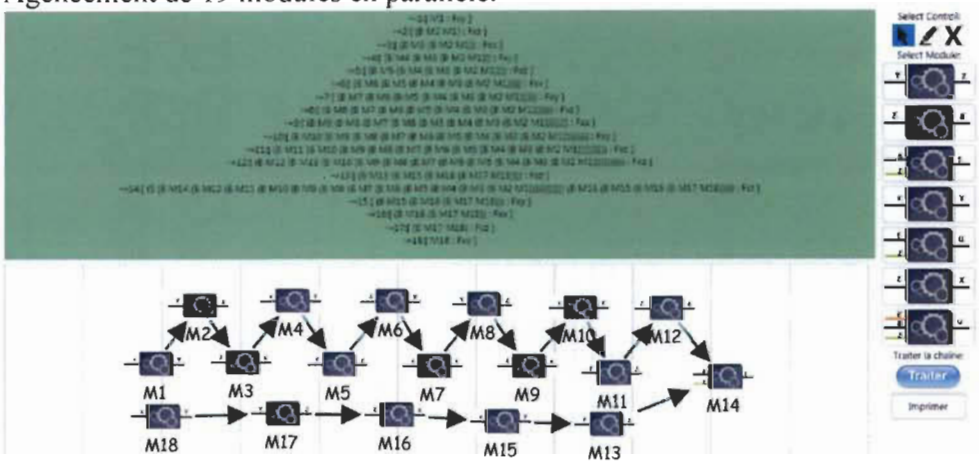
21. Agencement de 17 modules en parallèle.



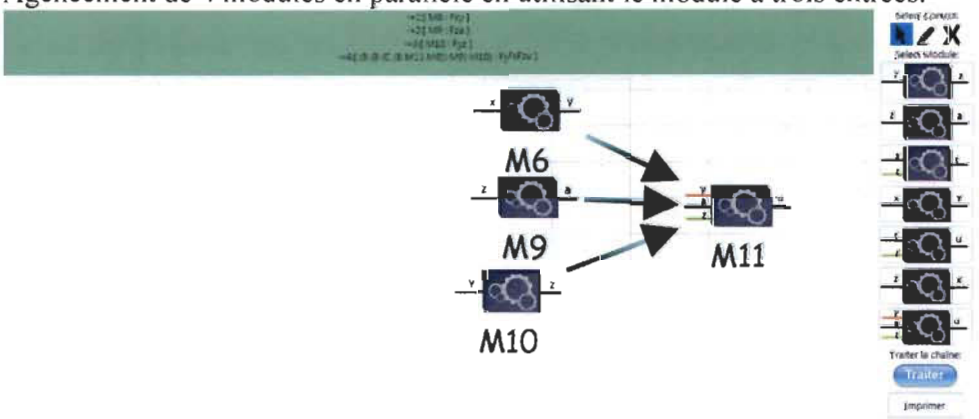
22. Agencement de 18 modules en parallèle.



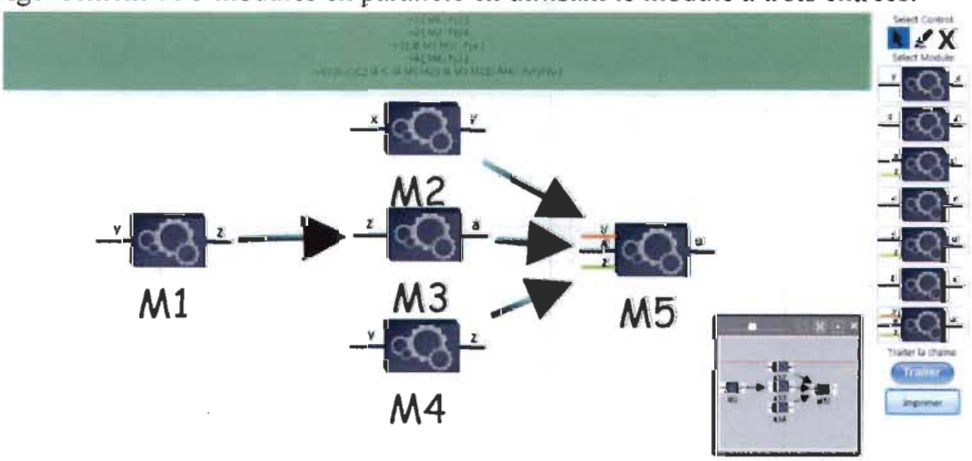
23. Agencement de 19 modules en parallèle.



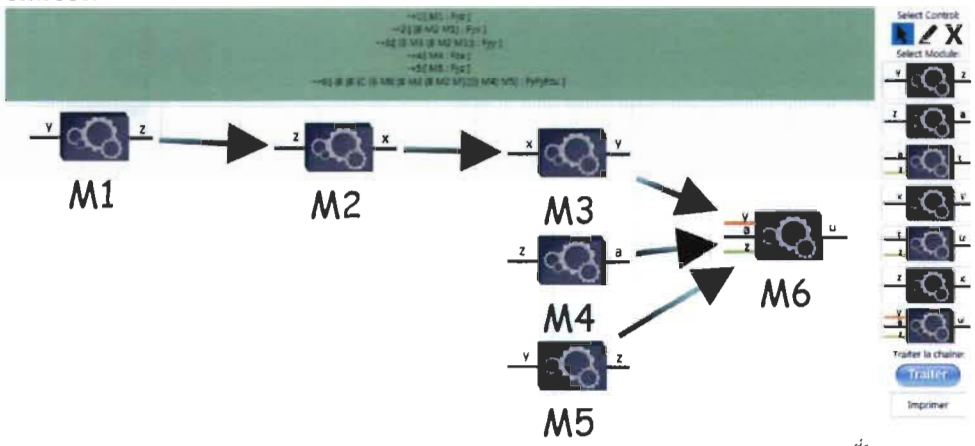
24. Agencement de 4 modules en parallèle en utilisant le module à trois entrées.



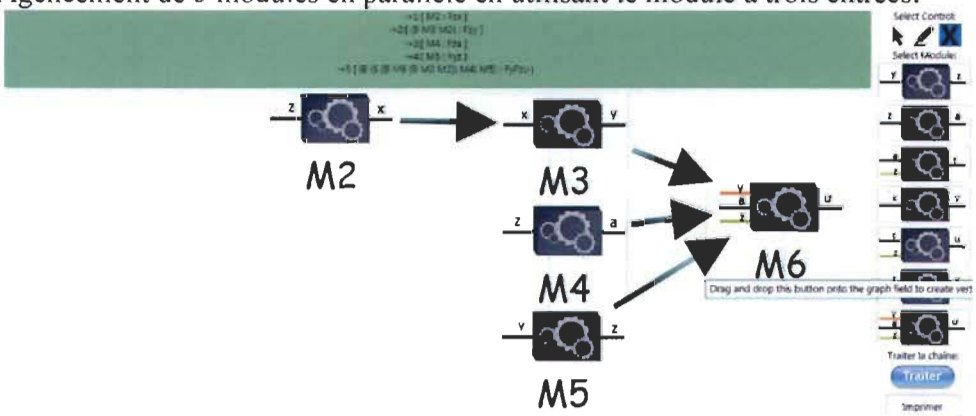
25. Agencement de 5 modules en parallèle en utilisant le module à trois entrées.



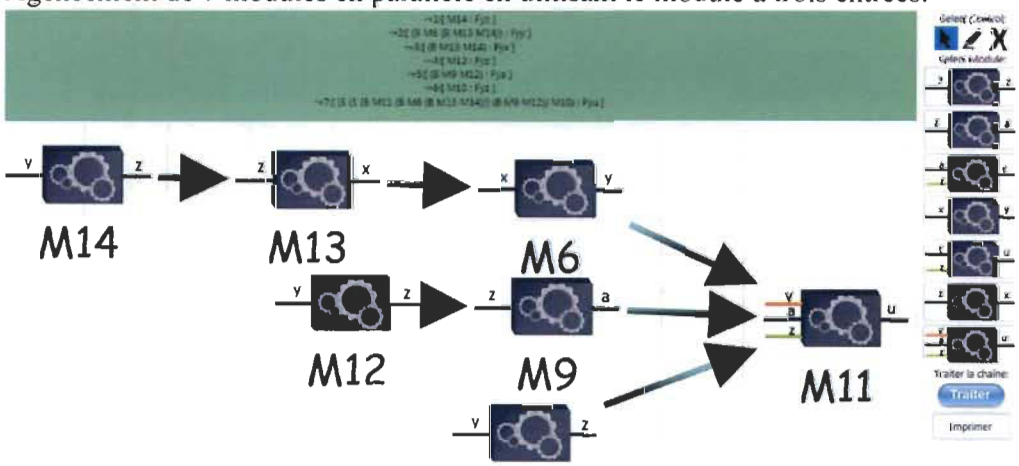
26. Agencement de 5 modules en parallèle en utilisant le module à trois entrées.



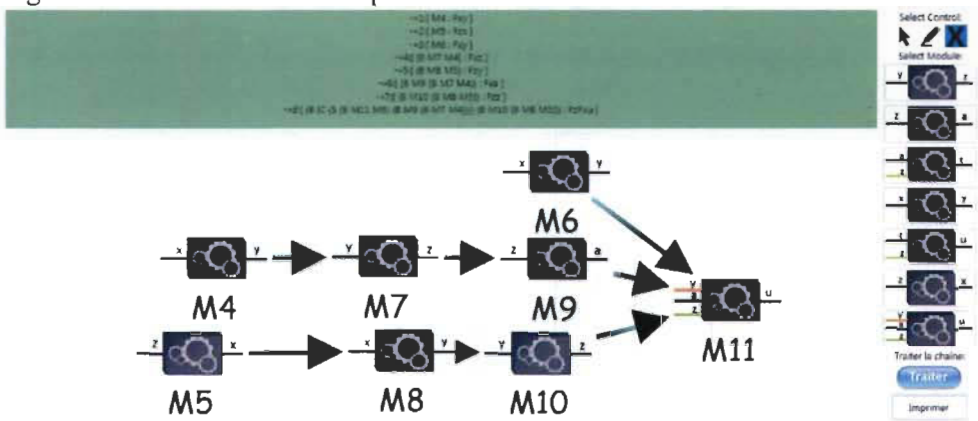
27. Agencement de 5 modules en parallèle en utilisant le module à trois entrées.



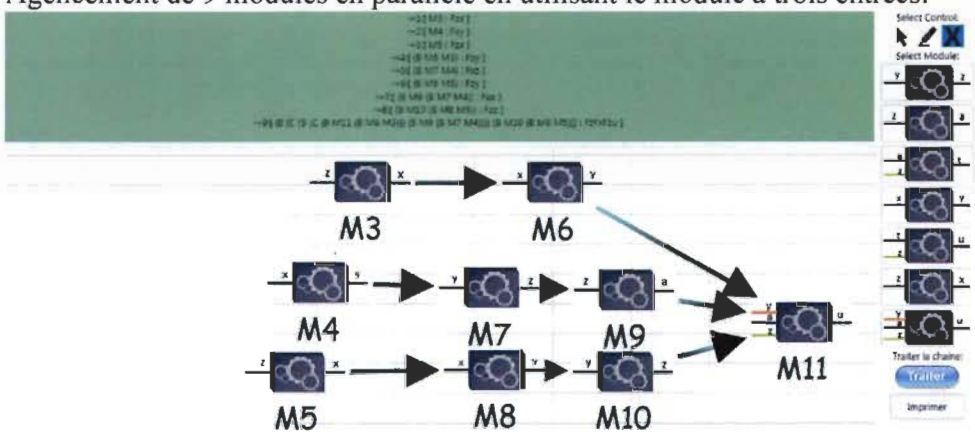
28. Agencement de 7 modules en parallèle en utilisant le module à trois entrées.



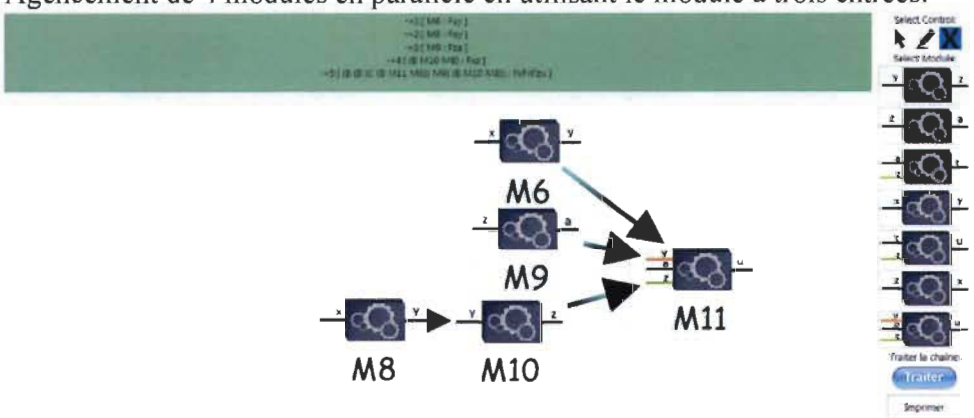
29. Agencement de 8 modules en parallèle en utilisant le module à trois entrées.



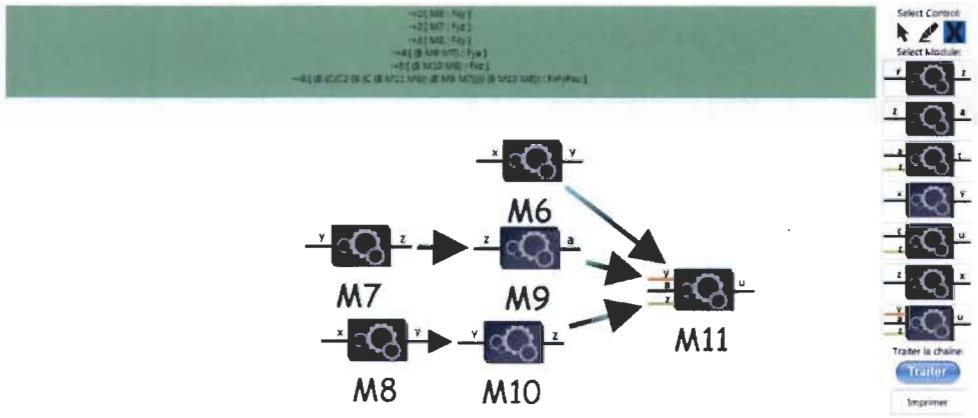
30. Agencement de 9 modules en parallèle en utilisant le module à trois entrées.



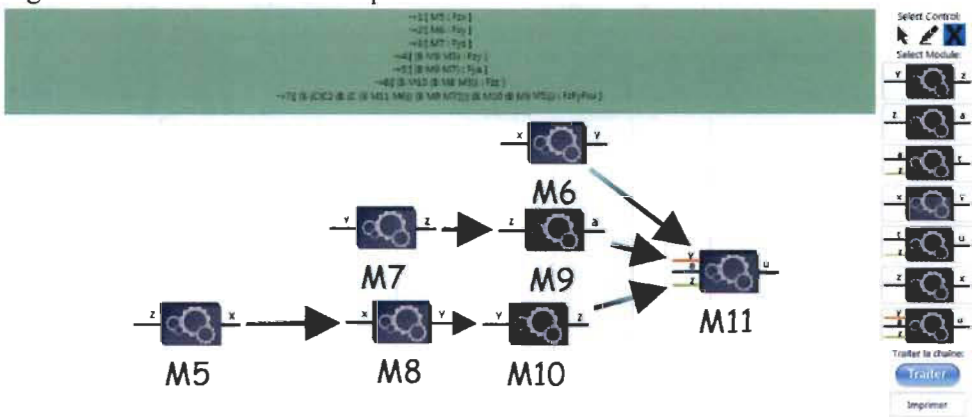
31. Agencement de 4 modules en parallèle en utilisant le module à trois entrées.



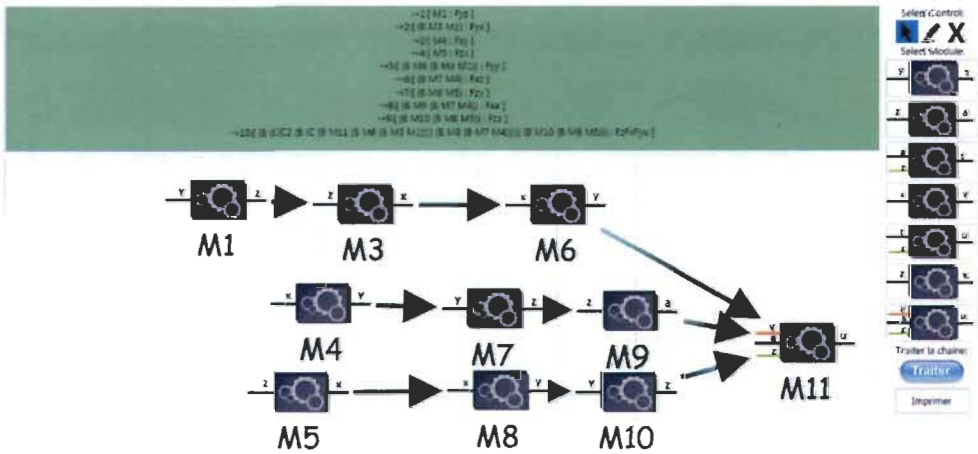
32. Agencement de 6 modules en parallèle en utilisant le module à trois entrées.



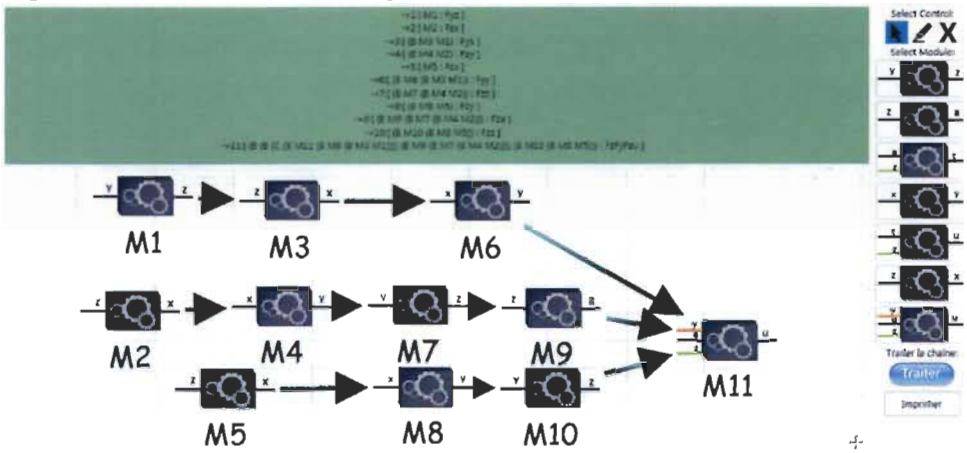
33. Agencement de 7 modules en parallèle en utilisant le module à trois entrées.



34. Agencement de 10 modules en parallèle en utilisant le module à trois entrées.



35. Agencement de 11 modules en parallèle en utilisant le module à trois entrées.



Les enregistrements des étapes pour les derniers tests avec le module à trois entrées :

(B M2 M1)[Y; X]
(B M3 (B M2 M1))[Y; Y]
(B M6 (B M3 (B M2 M1)))[Y; A; Z; U]
(C (B M6 (B M3 (B M2 M1))))[A; Y; Z; U]
(B (C (B M6 (B M3 (B M2 M1)))) M4)[Z; Y; Z; U]
(B (B (C (B M6 (B M3 (B M2 M1)))) M4) M5)[Y; Y; Z; U]
---------- Fin de l'enregistrement ----------
(B M3 M2)[Z; Y]
(B M6 (B M3 M2))[Z; A; Z; U]
(S (B M6 (B M3 M2)) M4)[Z; Z; U]
(B (S (B M6 (B M3 M2)) M4) M5)[Y; Z; U]
---------- Fin de l'enregistrement ----------
(B M6 M3)[X; A; Z; U]
(C (B M6 M3))[A; X; Z; U]
(B (C (B M6 M3)) M4)[Z; X; Z; U]
(B (B (C (B M6 M3)) M4) M5)[Y; X; Z; U]
---------- Fin de l'enregistrement ----------
(B M3 M1)[Y; X]
(B M6 (B M3 M1))[Y; Y]
(B M7 M4)[X; Z]
(B M9 (B M7 M4))[X; A]
(B M8 M5)[Z; Y]
(B M10 (B M8 M5))[Z; Z]
(B M11 (B M6 (B M3 M1)))[Y; A; Z; U]
(C (B M11 (B M6 (B M3 M1))))[A; Y; Z; U]

(B (C (B M11 (B M6 (B M3 M1)))) (B M9 (B M7 M4)))[X; Y; Z; U]
(C(C2 (B (C (B M11 (B M6 (B M3 M1)))) (B M9 (B M7 M4))))[Z; X; Y; U]
(B (C(C2 (B (C (B M11 (B M6 (B M3 M1)))) (B M9 (B M7 M4)))) (B M10 (B M8 M5)))[Z; X; Y; U]
---------- Fin de l'enregistrement ----------
(B M6 M3)[Z; Y]
(B M7 M4)[X; Z]
(B M9 (B M7 M4))[X; A]
(B M8 M5)[Z; Y]
(B M10 (B M8 M5))[Z; Z]
(B M11 (B M6 M3))[Z; A; Z; U]
(C (B M11 (B M6 M3)))[A; Z; Z; U]
(B (C (B M11 (B M6 M3))) (B M9 (B M7 M4)))[X; Z; Z; U]
(C (B (C (B M11 (B M6 M3))) (B M9 (B M7 M4))))[Z; X; Z; U]
(B (C (B (C (B M11 (B M6 M3))) (B M9 (B M7 M4)))) (B M10 (B M8 M5)))[Z; X; Z; U]
---------- Fin de l'enregistrement ----------
(B M7 M4)[X; Z]
(B M9 (B M7 M4))[X; A]
(B M8 M5)[Z; Y]
(B M10 (B M8 M5))[Z; Z]
(B M11 M6)[X; A; Z; U]
(S (B M11 M6) (B M9 (B M7 M4)))[X; Z; U]
(C (S (B M11 M6) (B M9 (B M7 M4))))[Z; X; U]
(B (C (S (B M11 M6) (B M9 (B M7 M4)))) (B M10 (B M8 M5)))[Z; X; U]
---------- Fin de l'enregistrement ----------
(B M8 M5)[Z; Y]
(B M10 (B M8 M5))[Z; Z]
(B M9 M7)[Y; A]
(B M11 M6)[X; A; Z; U]
(C (B M11 M6))[A; X; Z; U]
(B (C (B M11 M6)) (B M9 M7))[Y; X; Z; U]
(C(C2 (B (C (B M11 M6)) (B M9 M7))))[Z; Y; X; U]
(B (C(C2 (B (C (B M11 M6)) (B M9 M7)))) (B M10 (B M8 M5)))[Z; Y; X; U]
---------- Fin de l'enregistrement ----------
(B M9 M7)[Y; A]
(B M10 M8)[X; Z]
(B M11 M6)[X; A; Z; U]
(C (B M11 M6))[A; X; Z; U]
(B (C (B M11 M6)) (B M9 M7))[Y; X; Z; U]
(C(C2 (B (C (B M11 M6)) (B M9 M7))))[Z; Y; X; U]
(B (C(C2 (B (C (B M11 M6)) (B M9 M7)))) (B M10 M8))[X; Y; X; U]
---------- Fin de l'enregistrement ----------
(B M10 M8)[X; Z]
(B M11 M6)[X; A; Z; U]
(C (B M11 M6))[A; X; Z; U]

(B (C (B M11 M6)) M9)[Z; X; Z; U]
(B (B (C (B M11 M6)) M9) (B M10 M8))[X; X; Z; U]
---------- Fin de l'enregistrement ----------
(B M11 M6)[X; A; Z; U]
(C (B M11 M6))[A; X; Z; U]
(B (C (B M11 M6)) M9)[Z; X; Z; U]
(B (B (C (B M11 M6)) M9) M10)[Y; X; Z; U]
---------- Fin de l'enregistrement ----------
(B M13 M14)[Y; X]
(B M6 (B M13 M14))[Y; Y]
(B M9 M12)[Y; A]
(B M11 (B M6 (B M13 M14)))[Y; A; Z; U]
(S (B M11 (B M6 (B M13 M14))) (B M9 M12))[Y; Z; U]
(S (S (B M11 (B M6 (B M13 M14))) (B M9 M12)) M10)[Y; U]
---------- Fin de l'enregistrement ----------
(B M6 M13)[Z; Y]
(B M9 M12)[Y; A]
(B M11 (B M6 M13))[Z; A; Z; U]
(C (B M11 (B M6 M13)))[A; Z; Z; U]
(B (C (B M11 (B M6 M13))) (B M9 M12))[Y; Z; Z; U]
(S (B (C (B M11 (B M6 M13))) (B M9 M12)) M10)[Y; Z; U]
---------- Fin de l'enregistrement ----------

Annexe B : Guide d'utilisateur

SYSTÈME NAILI

Guide d'utilisateur

Dans ce guide

Important: créez un dossier C:\resultat

1- Résumé

NAILI (Nouvelle Architecture pour l'Ingénierie de la Langue et de l'Information) est un système flexible, modulaire, consistant et cohérent où chaque tâche est représentée par une fonction autonome et indépendante du reste des fonctions de l'architecture. Les processus d'analyse sont des assemblages de telles fonctions. Il permet la construction de multiples processus d'analyse de textes selon des objectifs déterminés. Il permet aussi de systématiser la vérification pour un processus d'analyse la cohérence de la séquence dans l'assemblage des fonctions de ce processus.

Ce guide présente l'essentiel pour utiliser le système NAILI

2- Prérequis

Il est préférable d'utiliser ce système sous les systèmes d'exploitation suivants : win server 2012, win 7 ou ultérieurs versions de windows.

Après la décompression du dossier du système dans votre répertoire préféré il est nécessaire de créer un dossier appeler resultat à la racine du disque local C:\

Ce dossier comportera un fichier texte resultat.txt qui va être créé par le système. Les résultats de traitement des chaînes seront enregistrés dans ce fichier

3- Interface d'accueil

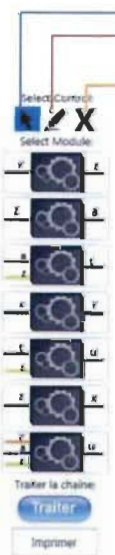
L'interface d'accueil présente le projet et deux lien pour commencer l'assemblage du projet aussi il permet d'accéder au guide de l'utilisateur voir la figure suivante:

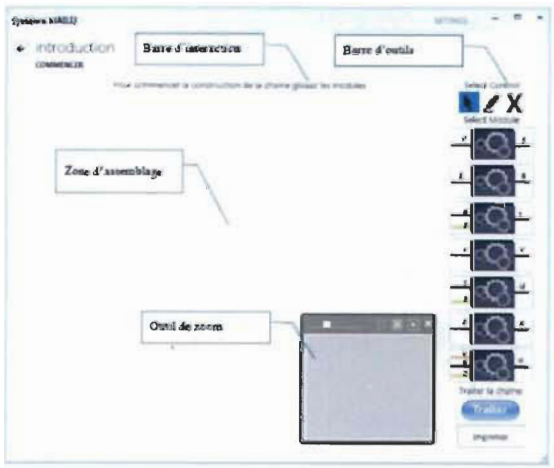
Barre d'interaction affiche les états des modules et autres messages.

4- Interface principal

La figure suivante est la page principale du système :
Il se compose de la zone de zoom, d'assemblage et la barre d'outils qui contient les modules et fonctionnalités suivante :

- Sélectionner : sélectionne un module pour le déplacer
- Connecteur de modules : Connecte un module avec son suivant souhaité
- Supprimer un lien ou un module : supprime le lien ou le module choisit





Barre d'outils

5- Modules disponibles

Module	Type
	Fzx
	Fxy
	Fyz
	Fza
	FtFzu
	FzFaFzu
	FaFzt

Modules avec leur types

Le système fournie 7 modules différents. Chaque module aura une expression donné par le système, ce qui permet la réutilisation du module dans la même chaîne. Seulement le type qui est prédéfinit.

M: Fzx accepte en entrée le type z, sa sotie est de type x

M: Fxy accepte en entrée le type x sa sotie est de type y

M: Fyz accepte en entrée les type y sa sotie est de type z

M: Fza accepte en entrée le type z sa sotie est de type a

M: FtFzu accepte en entrée les types t et z sa sotie est de type u

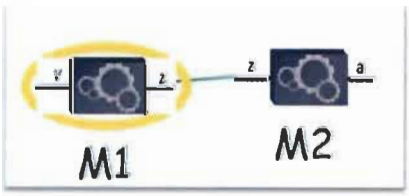
M: FzFaFzu accepte en entrée les types y, a et z sa sotie est de type u

M: FaFzt accepte en entrée les types a et z sa sotie est de type t

6- Assemblage: cas d'une seule entrée

Pour assembler des modules et créer une chaîne de traitement, procédez, procédez de la manière suivante :

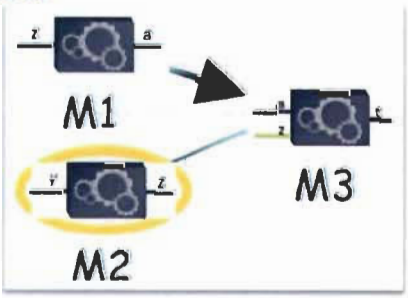
- Sélectionnez un module de la barre d'outils et glissez-le dans la zone d'assemblage, répéter cette action pour chaque module souhaité.
- Choisissez le connecteur de modules puis cliquez sur le premier module et ensuite cliquez sur le module avec lequel vous désirez connecter le premier.



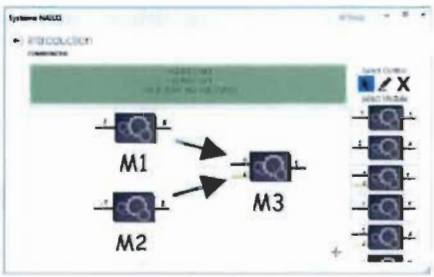
7- Assemblage: cas de plusieurs entrées

Pour assembler des modules et créer une chaîne de traitement avec un module à plusieurs entrées, procédez, procédez de la même manière d'assemblage précédente en respectant l'ordre suivant:

- Déposez les modules dans la zone d'assemblage.
- Choisissez le connecteur de modules qui a le même type en commençant par le haut vers le bas comme montré dans l'image à droite.



8- Traitement

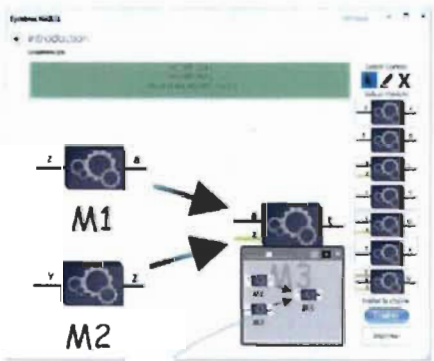


Après finir les assemblages de modules vous pouvez procéder au traitement, de la chaîne que vous avez créé, en cliquant sur le bouton:

Maintenant le fichier C:\resultat\resultat.txt contiendra le résultat du traitement, chaque ligne d'enregistrement est une étape comme suit :

```
B M3 M1)[2; 2; T]
B (B M3 M1) M2)[Y; 2; T]
------------ Fin de l'enregistrement ----------------
```

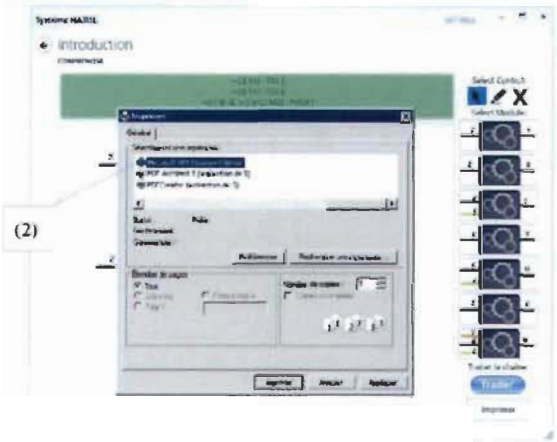
9- Impression



Pour faire l'impression de votre chaîne, centrez votre travail à l'aide de l'outil de zoom (1) et cliquez sur le bouton suivant:

Puis sélectionnez l'imprimante souhaité (2) de fichier d'impression préféré et modifiez les préférences si nécessaire et cliquez sur imprimer.

(1)



Annexe C : Language F#

Le langage de programmation F#

D'après le site Fsharp.org, F# est un langage mature, *open source*, multiplateforme et premier langage de programmation fonctionnelle. Il permet aux utilisateurs et aux organisations de résoudre des problèmes de calcul complexes avec un code simple, maintenable et robuste. Distribué par Microsoft pour sa plateforme .NET, il a été précédé de prototypes comme SML.NET, un compilateur Standard ML (SML) pour .NET. Ce langage combine plusieurs paradigmes de programmation, dont les paradigmes fonctionnels, objet et impératif.

Il est inspiré du langage OCaml, dont il partage un grand nombre de caractéristiques. Il s'en distingue toutefois sur plusieurs points, dont son mécanisme objet hérité de .NET.

Les caractéristiques de F# sont comme suit :

1. La caractéristique fonctionnelle et le typage dans F#

Un programme purement fonctionnel, F# est une suite de déclarations et d'expressions. Un environnement est un ensemble de liaisons d'identificateurs à leurs valeurs. Une expression est construite à partir d'applications de fonctions et est évaluée dans un environnement précisé par la sémantique.

F # est en premier lieu un langage fonctionnel statiquement fortement typé. Dans cette partie, nous présentons les caractéristiques fonctionnelles du langage F#, leurs mécanismes et les garanties qu'ils apportent [32].

La gestion des noms

En F#, la déclaration globale se fait par la construction *let* et sa portée couvre tout le code source suivant son introduction. La déclaration locale, quant à elle, se fait par la construction *let E = expr_E in expr* et crée la liaison de *E* à la valeur de *expr_E*, le temps de l'évaluation de l'expression *expr*.

La portée statique désigne un intervalle que la syntaxe du programme détermine pour chaque l'identificateur. Elle facilite l'analyse et la maintenance du texte source [32].

- **Les valeurs**

En F#, les valeurs sont des quantités qui ont un type spécifique; les valeurs peuvent être des nombres intégraux ou à virgule flottante, des caractères ou du texte, des listes, des séquences, des tableaux, des tuples, des unions discriminées, des enregistrements, des types de classe ou des valeurs de fonction.

Les valeurs immuables sont des valeurs qui ne peuvent pas être modifiées durant toute l'exécution d'un programme. Dans les langages fonctionnels purs, il n'y a pas de variables, et les fonctions se comportent strictement comme des fonctions mathématiques. Les valeurs mutables, quant à elles, peuvent être modifiées. En F#, les variables mutables doivent généralement avoir une portée limitée, soit en tant que champ d'un type, soit en tant que valeur locale[8].

- **La transparence référentielle**

La transparence référentielle permet une certaine aisance dans la factorisation permettant la réutilisation du code et en facilite principalement l'analyse, alors que le

[8] https://msdn.microsoft.com/fr-fr/library/dd233185.aspx

noyau fonctionnel pur de F# a pour résultat un code transparent pour lequel une expression peut être remplacée par sa valeur sans modifier le comportement du programme [32].

- **Les opérateurs**

L'utilisation d'un opérateur dans une expression contraint l'inférence de type sur cet opérateur. Certains des opérateurs sont définis à l'aide de paramètres de type résolu statiquement, donnant ainsi des définitions individuelles pour chaque type spécifique fonctionnant avec cet opérateur. Les types surchargés sont la catégorie qui englobe tous les opérateurs arithmétiques et de bits unaires[9] et binaires[10]. Les opérateurs de comparaison sont génériques et fonctionnent par conséquent avec n'importe quel type, pas seulement avec des types arithmétiques primitifs. Les types d'union discriminée et d'enregistrement ont leurs propres implémentations personnalisées qui sont générées par le compilateur F#. Les types de classe utilisent la méthode *Equals*.

2. Les caractéristiques impératives du langage F#

La programmation fonctionnelle ne se base pas sur des suites d'instructions et ne s'appuie pas sur l'idée d'état global [33, p. 67]. Puisque le langage F# est multiparadigmes, il est possible de bénéficier de certaines garanties du compilateur, alors que d'autres sont directement issus de l'interface avec la plateforme .NET afin de pouvoir programmer impérativement, aussi appelée la programmation à effet secondaire[11].

3. La caractéristique modulaire du langage F#

---

[9] Opérateurs arithmétiques unaires + (positif) ; et - (négatif)

[10] Opérateurs arithmétiques binaires : + (addition, plus) ; - (soustraction, moins) ; * (multiplication, fois) ;% (modulo, modulo) ; / (division, divisé par) ; et ** (élévation à la puissance, puissance).

[11] C'est la modification de valeurs du programme autres que celles explicitement spécifiées lors de l'appel d'une fonction ou un sous-programme

Pour l'organisation du code dans un programme orienté objet, une structure de données et les fonctions qui agissent sur elle seraient regroupées dans une classe. Cependant, dans un programme fonctionnel en F#, la structure de données et ses fonctions sont regroupées dans des modules.

Il existe trois modèles communs pour mélanger les types et les fonctions ensemble :

- le type déclaré et les fonctions sont dans le même module,
- le type déclaré est séparé des fonctions, mais dans le même fichier et
- le type déclaré est séparé des fonctions et dans un fichier différent, lequel contient généralement les définitions de type.

Dans la première approche, les types sont définis à l'intérieur du module, de même que leurs fonctions associées. S'il n'y a qu'un seul type primaire, il a souvent un nom simple comme « T » ou le nom du module [34].

4. La spécification du langage F#

F # est un langage extensible, succinct et de type sécurisé de type inféré, exécuté efficacement fonctionnel/impératif/de programmation orientée objet. Ainsi, c'est le premier langage de programmation fonctionnel typé pour la plateforme .NET et d'autres implémentations de la spécification ECMA 335 *Common Language Infrastructure* (CLI). F# a été en partie inspiré par le langage OCaml et partage certains constructeurs communs avec ce dernier [32].

5. L'interfaçage avec d'autres langages

On peut créer des interfaces par l'utilisation du langage F# avec les bibliothèques .NET, comme System.Net.WebRequest et System.Text.RegularExpressions; l'intégration se fait sans faille.

Pour des besoins plus complexes, F# supporte nativement les classes .NET, les interfaces et les structures avec une interopérabilité simple. Par exemple, nous pouvons écrire une interface ISomething en C # et, avec la mise en œuvre, la déclarer en F #.

F# peut être appelé non seulement dans un code .NET existant, mais il peut aussi exposer presque toutes les API .NET à d'autres langues. Par exemple, il est possible d'écrire des classes et des méthodes en F# et les exposer à C #, VB ou COM.

Bibliographie

[1] I. Biskri et A. Jibali, Traitement automatique des langues naturelles de l'analyse à l'application, LA VOISIER, 2011.

[2] S. Mondiales, «langues parlées dans le monde,» décembre 2015. [En ligne]. Available: http://statistiques-mondiales.com/langues.htm.

[3] I.Tellier, «Introduction au TALN et à l'ingénierie linguistique,» Université de Lille 3, lille, 2012.

[4] D. R. Sophie, «système-expert,» hypergeo, 2014. [En ligne]. Available: http://www.hypergeo.eu/spip.php?article84. [Accès le 2015].

[5] D. Mattison, «Music to Soothe the Savage Searcher: Classical Music Databases and Web Resources,» Vol. 14 No. 7 — Jul/Aug 2006, vol. Vol. 14, n° %1No. 7, Jul/Aug 2006.

[6] M.-A. Rochette, «Vers une ingénierie fléxible pour le traitement de l'information,» UQTR, Trois-Rivières, 2008.

[7] J. Searsmith, «Text mining with D2k/T2k,» 09 07 2004. [En ligne]. Available: http://www.library.illinois.edu/spx/slw-dtw/presentations/searsmith_files/v3_document.htm. [Accès le 23 11 2015].

[8] I. Biskri, M. Anastacio, A. Joly et B. A. Bensaber, «A typed applicative system for a language and text processing engineering,» Journal of Innovation in Digital Ecosystems, vol. Volume 1, n° %1 Issues 1–2, p. 26–37, December 2014.

[9] Gate, «A one-stop-shop for text analytics and semantics,» 01 12 2015. [En ligne]. Available: https://gate.ac.uk/biz/usps.html.

[10] B. Rana, «Vers un système d'aide à la décision pour la concéption en génie: Une approche basée sur les connaissances,» UQTR, Trois-Rivières, 2012.

[11] M. L. G. a. t. U. o. Waikato, «documentation,» Waikato, 2015. [En ligne]. Available: http://www.cs.waikato.ac.nz/ml/weka/documentation.html.

[12] V. Granville, «WEKA: Pluses and minuses,» 11 October 2013. [En ligne]. Available: http://www.datasciencecentral.com/forum/topics/weka-pluses-and-minuses. [Accès le 2015].

[13] T. J, «First Look – KNIME Analytics Workbench update,» 2012. [En ligne]. Available: http://jtonedm.com/2012/01/23/first-look-knime-analytics-workbench-update/. [Accès le 03 Novembre 2015].

[14] W. W. A., «Scientific workflow systems: pipeline pilot and KNIME.,» J. Comput. Aided Mol. Des. 26, 801–80410.1007/s10822-012-9577-7, 2012.

[15] Dassault Systèmes, «Pilpline Pilot Overview datasheet,» 2015. [En ligne]. Available: http://accelrys.com/products/datasheets/pipeline-pilot/pipeline-pilot-overview.pdf. [Accès le 01 Novembre 2015].

[16] M. Hassan1, R. D. B. et S. V. &. David, «Cheminformatics analysis and learning in a data pipelining environment,» Vols. %1 sur %2DOI: 10.1007/s11030-006-9041-5, p. 283–299 , 2006.

[17] S. H. Edgar D., « Pipeline Pilot Interface to FTrees Version 2.4.5.1 User Guide,» Ftrees, 2015.

[18] T. Garneau, «La Programmation Orientée-Agent DMAS Builder : Un Environnement de Développement De Systèmes Multi-Agents Totalement Distribués,» UQTR, Trois-Rivières, 2003.

[19] J.-C. Gelin, P. Paquier, L. Walterthum et T. &. F. Group, «Application de la POO pour la conception d'un logiciel de simulation par éléments finis en mise en forme des matériaux,» Revue Européenne des Éléments Finis, vol. Vol.7(5), pp. p.505-533, 1998.

[20] L. Fabresse, «Du découplage à l'assemblage non-anticipé de composants: Conception et mise en œuvre du langage à composants SCL,» Univesité Monpelier II, 2007.

[21] Wikipedia, «Programmation orientée aspect,» 17 09 2015. [En ligne]. Available: https://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_aspect#cite_note-1. [Accès le 12 11 2015].

[22] grafikart, «La programmation fonctionnelle,» 27 12 2014. [En ligne]. Available: http://www.grafikart.fr/blog/programmation-fonctionnelle. [Accès le 15 11 2015].

[23] Microsoft, «XAML Syntax In Detail,» 2015 . [En ligne]. Available: https://msdn.microsoft.com/fr-fr/library/ms788723(v=vs.110).aspx. [Accès le 14 11 2015].

[24] wikipedia, «Common Language Runtime,» 06 09 2015. [En ligne]. Available: https://fr.wikipedia.org/wiki/Common_Language_Runtime. [Accès le 14 11 2015].

[25] S. M. Sadok, «Applications linguistiques multilingues : apports des grammaires catégorielles et de la logique combinatoire,» UQTR, Trois-Rivières, 2010.

[26] B. C. Haskell et F. Robert, Combinatory Logic, Volume 1, North-Holland Publishing Company, 1958.

[27] J. P. Déclès, Langages applicatifs, langues naturelles et cognition, Paris: Hermes, 1990.

[28] I. Biskri, «La grammaire catégorielle combinatoire applicative dans le cadre de la grammaire applicative et cognitive,,» Thèse de Doctorat, Paris, 1995.

[29] I. Biskri et J. Déclès, «Du phénotype au génotype : la Grammaire Catégorielle Combinatoire Applicative,» chez Colloque TALN96, Marseille, 1996.

[30] I. Biskri, L. Emirkanian et A. Jebali, «Les marqueurs de sujet de l'arabe standard : analyse et formalisation,» chez Traitement automatique des langues naturelles : de l'analyse à l'application., Paris/Londres, Hermès Sciences. Série Cognition et traitement de l'information., 2011, pp. 179-218.

[31] Panthernet, «Project description,» 21 08 2015. [En ligne]. Available: https://graphx.codeplex.com/.

[32] Agence nationale de la sécurité des systèmes d'information, «Analyse des langages OCaml, F# et Scala,» 2011.

[33] S. Don, G. Adam et C. Antonio, Expert F#, 2007, p. 67.

[34] Scottw, «Organizing modules in a project,» 26 05 2013. [En ligne]. Available: http://fsharpforfunandprofit.com/posts/recipe-part3/. [Accès le 25 11 2015].

[35] wikipedia, «Linguistique informatique,» 2015. [En ligne]. Available: https://fr.wikipedia.org/wiki/Linguistique_informatique#cite_note-1.