50376
1988
245

USTL
FLANDRES ARTOIS

LIFL
U.A 369 du C.N.R.S

LABORATOIRE D'INFORMATIQUE FONDAMENTALE DE LILLE

N° d'ordre : 297

THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FALANDRES ARTOIS

pour obtenir le titre de

DOCTEUR en INFORMATIQUE

par

OUSSOUS Nour-éddine

ETUDE ET TRAITEMENT DES SERIES FORMELLES NON COMMUTATIVES, POUR LA REPRESENTATION MINIMALE DES SYSTEMES DYNAMIQUES NON LINEAIRES

Thèse soutenue le 14 decembre 1988 devant la commission d'Examen

Membres du jury

M. DAUCHET	Président
F. LAMNABHI-LAGARRIGUE	Rapporteur
C. REUTENAUER	Rapporteur
G. JACOB	Directeur de thèse
G.VIENNOT	Examinateur
F. ROTELLA	Examinateur

UNIVERSITE DES SCIENCES
ET TECHNIQUES DE LILLE
FLANDRES ARTOIS

DOYENS HONORAIRES DE L'ANCIENNE FACULTE DES SCIENCES

M. H. LEFEBVRE, M. PARREAU.

PROFESSEURS HONORAIRES DES ANCIENNES FACULTES DE DROIT ET SCIENCES ECONOMIQUES, DES SCIENCES ET DES LETTRES

MM. ARNOULT, BONTE, BROCHARD, CHAPPELON, CHAUDRON, CORDONNIER, DECUYPER, DEHEUVELS, DEHORS, DION, FAUVEL, FLEURY, GERMAIN, GLACET, GONTIER, KOURGANOFF, LAMOTTE, LASSERRE, LELONG, LHOMME, LIEBAERT, MARTINOT-LAGARDE, MAZET, MICHEL, PEREZ, ROIG, ROSEAU, ROUELLE, SCHILTZ, SAVARD, ZAMANSKI, Mes BEAUJEU, LELONG.

PROFESSEUR EMERITE

M. A. LEBRUN

ANCIENS PRESIDENTS DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

MM. M. PARREAU, J. LOMBARD, M. MIGEON, J. CORTOIS.

PRESIDENT DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS

M. A. DUBRULLE.

PROFESSEURS - CLASSE EXCEPTIONNELLE

M. CONSTANT Eugène	Electronique
M. FOURET René	Physique du solide
M. GABILLARD Robert	Electronique
M. MONTREUIL Jean	Biochimie
M. PARREAU Michel	Analyse
M. TRIDOT Gabriel	Chimie appliquée

PROFESSEURS - 1ère CLASSE

M. BACCHUS Pierre	Astronomie
M. BIAYS Pierre	Géographie
M. BILLARD Jean	Physique du solide
M. BOILLY Bénoni	Biologie
M. BONNELLE Jean Pierre	Chimie-Physique
M. BOSCQ Denis	Probabilités
M. BOUGHON Pierre	Algèbre
M. BOURIQUET Robert	Biologie végétale
M. BREZINSKI Claude	Analyse numérique
M. BRIDOUX Michel	Chimie-Physique
M. CARREZ Christian	Informatique
M. CELET Paul	Géologie générale
M. CHAMLEY Hervé	Géotechnique
M. COEURE Gérard	Analyse
M. CORDONNIER Vincent	Informatique
M. DEBOURSE Jean Pierre	Gestion des entreprises
M. DHAINAUT André	Biologie animale
M. DOUKHAN Jean Claude	Physique du solide
M. DYMENT Arthur	Mécanique
M. ESCAIG Bertrand	Physique du solide
M. FAURE Robert	Mécanique
M. FOCT Jacques	Métallurgie
M. FRONTIER Serge	Ecologie numérique
M. GRANELLE Jean jacques	Sciences Economiques
M. GRUSON Laurent	Algèbre
M. GUILLAUME Jean	Microbiologie
M. HECTOR Joseph	Géométrie
M. LABLACHE-COMBIER Alain	Chimie organique
M. LACOSTE Louis	Biologie végétale
M. LAVEINE Jean Pierre	Paléontologie
M. LEHMANN Daniel	Géométrie
Mme LENOBLE Jacqueline	Physique atomique et moléculaire
M. LEROY Jean Marie	Spectrochimie
M. LHOMME Jean	Chimie organique biologique
M. LOMBARD Jacques	Sociologie
M. LOUCHEUX Claude	Chimie physique
M. LUCQUIN Michel	Chimie physique
M. MACKE Bruno	Physique moléculaire et rayonnements atmosphériques
M. MIGEON Michel	E.U.D.I.L.
M. PAQUET Jacques	Géologie générale
M. PETIT Francis	Chimie organique
M. POUZET Pierre	Modélisation - Calcul scientifique
M. PROUVOST Jean	Minéralogie
M. RACZY Ladislas	Electronique
M. SALMER Georges	Electronique
M. SCHAMPS Joel	Spectroscopie moléculaire
M. SEGUIER Guy	Electrotechnique
M. SIMON Michel	Sociologie
Mlle SPIK Geneviève	Biochimie
M. STANKIEWICZ François	Sciences Economiques
M. TILLIEU Jacques	Physique théorique
M. TOULOTTE Jean Marc	Automatique
M. VIDAL Pierre	Automatique
M. ZEYTOUNIAN Radyadour	Mécanique

PROFESSEURS - 2ème CLASSE

M. ALLAMANDO Etienne	Composants électroniques
M. ANDRIES Jean Claude	Biologie des organismes
M. ANTOINE Philippe	Analyse
M. BART André	Biologie animale
M. BASSERY Louis	Génie des procédés et réactions chimiques
Mme BATTIAU Yvonne	Géographie
M. BEGUIN Paul	Mécanique
M. BELLET Jean	Physique atomique et moléculaire
M. BERTRAND Hugues	Sciences Economiques et Sociales
M. BERZIN Robert	Analyse
M. BKOUCHE Rudolphe	Algèbre
M. BODARD Marcel	Biologie végétale
M. BOIS Pierre	Mécanique
M. BOISSIER Daniel	Génie civil
M. BOIVIN Jean Claude	Spectrochimie
M. BOUQUELET Stéphane	Biologie appliquée aux enzymes
M. BOUQUIN Henri	Gestion
M. BRASSELET Jean Paul	Géométrie et topologie
M. BRUYELLE Pierre	Géographie
M. CAPURON Alfred	Biologie animale
M. CATTEAU Jean pierre	Chimie organique
M. CAYATTE Jean Louis	Sciences Economiques
M. CHAPOTON Alain	Electronique
M. CHARET Pierre	Biochimie structurale
M. CHIVE Maurice	Composants électroniques optiques
M. COMYN Gérard	Informatique théorique
M. COQUERY Jean Marie	Psychophysiologie
M. CORIAT Benjamin	Sciences Economiques et Sociales
Mme CORSIN Paule	Paléontologie
M. CORTOIS Jean	Physique nucléaire et corpusculaire
M. COUTURIER Daniel	Chimie organique
M. CRAMPON Norbert	Tectolique Géodynamique
M. CROSNIER Yves	Electronique
M. CURGY Jean jacques	Biologie
Mle DACHARRY Monique	Géographie
M. DAUCHET Max	Informatique
M. DEBRABANT Pierre	Géologie appliquée
M. DEGAUQUE Pierre	Electronique
M. DEJAEGER Roger	Electrochimie et Cinétique
M. DELORME Pierre	Physiologie animale
M. DELORME Robert	Sciences Economiques
M. DEMUNTER Paul	Sociologie
M. DENEL Jacques	Informatique
M. DE PARIS Jean Claude	Analyse
M. DEPREZ Gilbert	Physique du solide - Cristallographie
M. DERIEUX Jean Claude	Microbiologie
Mle DESSAUX Odile	Spectroscopie de la réactivité chimique
M. DEVRAINNE Pierre	Chimie minérale
Mme DHAINAUT Nicole	Biologie animale
M. DHAMELINCOURT Paul	Chimie physique
M. DORMARD Serge	Sciences Economiques
M. DUBOIS Henri	Spectroscopie hertzienne
M. DUBRULLE Alain	Spectroscopie hertzienne
M. DUBUS Jean Paul	Spectrométrie des solides

M. DUPONT Christophe	Vie de la firme (I.A.E.)
Mme EVRARD Micheline	Génie des procédés et réactions chimiques
M. FAKIR Sabah	Algèbre
M. FAUQUEMBERGUE Renaud	Composants électroniques
M. FONTAINE Hubert	Dynamique des cristaux
M. FOUQUART Yves	Optique atmosphérique
M. FOURNET Bernard	Biochimie structurale
M. GAMBLIN André	Géographie urbaine, industrielle et démographie
M. GLORIEUX Pierre	Physique moléculaire et rayonnements atmosphériques
M. GOBLOT Rémi	Algèbre
M. GOSSELIN Gabriel	Sociologie
M. GOUDMAND Pierre	Chimie physique
M. GOURIEROUX Christian	Probabilités et statistiques
M. GREGORY Pierre	I.A.E.
M. GREMY Jean Paul	Sociologie
M. GREVET Patrice	Sciences Economiques
M. GRIMBLOT Jean	Chimie organique
M. GUILBAULT Pierre	Physiologie animale
M. HENRY Jean Pierre	Génie mécanique
M. HERMAN Maurice	Physique spatiale
M. HOUDART René	Physique atomique
M. JACOB Gérard	Informatique
M. JACOB Pierre	Probabilités et statistiques
M. JEAN Raymond	Biologie des populations végétales
M. JOFFRE Patrick	Vie de la firme (I.A.E.)
M. JOURNEL Gérard	Spectroscopie hertzienne
M. KREMBEL Jean	Biochimie
M. LANGRAND Claude	Probabilités et statistiques
M. LATTEUX Michel	Informatique
Mme LECLERCQ Ginette	Catalyse
M. LEFEBVRE Jacques	Physique
M. LEFEVRE Christian	Pétrologie
Mle LEGRAND Denise	Algèbre
Mle LEGRAND Solange	Algèbre
M. LEGRAND Pierre	Chimie
Mme LEHMANN Josiane	Analyse
M. LEMAIRE Jean	Spectroscopie hertzienne
M. LE MAROIS Henri	Vie de la firme (I.A.E.)
M. LEROY Yves	Composants électroniques
M. LESENNE Jacques	Systèmes electroniques
M. LHENAFF René	Géographie
M. LOCQUENEUX Robert	Physique théorique
M. LOSFELD Joseph	Informatique
M. LOUAGE Francis	Electronique
M. MAHIEU Jean Marie	Optique - Physique atomique
M. MAIZIERES Christian	Automatique
M. MAURISSON Patrick	Sciences Economiques et Sociales
M. MESMACQUE Gérard	Génie Mécanique
M. MESSELYN Jean	Physique atomique et moléculaire
M. MONTEL Marc	Physique du solide
M. MORCELLET Michel	Chimie organique
M. MORTREUX André	Chimie organique
Mme MOUNIER Yvonne	Physiologie des structures contractiles
M. NICOLE Jacques	Spectrochimie
M. NOTELET Francis	Systèmes électroniques
M. PARSY Fernand	Mécanique
M. PECQUE Marcel	Chimie organique
M. PERROT Pierre	Chimie appliquée

M. PERTUZON Emile	Physiologie animale
M. PONSOLLE Louis	Chimie physique
M. PORCHET Maurice	Biologie animale
M. POSTAIRE Jack	Informatique industrielle
M. POVY Lucien	Automatique
M. RICHARD Alain	Biologie animale
M. RIETSCH François	Physique des polymères
M. ROBINET Jean CLaude	EUDIL
M. ROGALSKI Marc	Analyse
M. ROY Jean Claude	Psychophysiologie
Mme SCHWARZBACH Yvette	Géométrie
M. SLIWA Henri	Chimie organique
M. SOMME Jean	Géographie
M. STAROSWIECKI Marcel	Informatique
M. STERBOUL François	Informatique
M. TAILLIEZ Roger	Génie alimentaire
M. THERY Pierre	Systèmes électroniques
M. THIEBAULT François	Sciences de la terre
M. THUMERELLE Pierre	Démographie - Géographie Humaine
Mme TJOTTA Jacqueline	Mathématiques
M. TOURSEL Bernard	Informatique
M. TREANTON Jean René	Sociologie du Travail
M. TURREL Georges	Spectrochimie infrarouge et Raman
M. VANDORPE Bernard	Chimie minérale
M. VASSEUR Christian	Automatique
M. VAST Pierre	Chimie inorganique
M. VERBERT André	Biochimie
M. VERNET Philippe	Génétique
M. WACRENIER Jean Marie	Electronique
M. WALLART Francis	Spectrochimie infrarouge et Raman
M. WARTEL Michel	Chimie inorganique
M. WATERLOT Michel	Géologie générale
M. WEINSTEIN Olivier	Analyse économique de la recherche et développement
M. WERNER Georges	Informatique théorique
M. WOZNIAK Michel	Spectrochimie
Mme ZINN JUSTIN Nicole	Algèbre

A Samira

Remerciements

Je tiens à exprimer dans ces lignes toute ma reconnaissance à Monsieur le Professeur Max DAUCHET, directeur du Laboratoire d'Informatique Fondamentale de Lille, d'avoir bien voulu accepter de présider ce jury, et aussi pour l'intérêt qu'il a porté à ce travail.

Je tiens aussi à exprimer ma profonde gratitude à Monsieur le Professeur Gérard JACOB qui m'a proposé et dirigé ce travail, et m'a fructueusement orienté au cours de celui-ci. Il a su, avec une gentillesse inlassable, m'apporter l'aide de sa compétence au long des nombreuses discussions qui ont accompagnées la construction de cette thèse.

Je remercie Madame Françoise LAMNABHI-LAGARRIGUE, chargé de recherche au CNRS à l'Ecole Supérieure d'Electricité de Paris, d'avoir accepté de rapporter sur ce travail.

Je remercie également Monsieur Christophe REUTENAUER, chargé de recherche au CNRS et responsable du PRC Mathématiques-Informatique, pour l'intérêt qu'il a porté à ce travail en acceptant de rapporter sur celui-ci.

Je tiens à remercier Monsieur Frédéric ROTELLA, Docteur d'Etat, Maître de conférence à l'IDN, pour l'intérêt qu'il porte à ce travail en acceptant de participer au jury.

Je tiens aussi à remercier Monsieur Gérard VIENNOT, directeur de recherche au CNRS, de l'Université de Bordeaux I, pour l'honneur qu'il me fait en acceptant de faire partie du jury de cette thèse.

Je remercie enfin tous mes camarades du Laboratoire d'Informatique Fondamentale de Lille pour les échanges de vue que nous avons eu, et en particulier Monsieur Bernard SZELAG qui m'a aidé pour effectuer le transfert du logiciel du DPS 8 sur les stations SUN et Monsieur Henri GLANC qui a assuré le tirage de ce mémoire.

Je remercie enfin ma femme Samira pour le soutient moral nécessaire à l'élaboration de ce travail.

PRESENTATION

Dans ce travail nous présentons un ensemble de rapports sur les séries formelles, sur les mots et sur les systèmes dynamiques non linéaires.

En premier lieu, une introduction et une heuristique, présentent le cadre du travail et son évolution.

Dans le premier rapport, nous donnons, dans le chapitre I, quelques rappels sur les algèbres de Lie et le produit tensoriel. Ensuite, dans le chapitre II, nous rappelons des définitions et propriétés des séries formelles non commutatives. Enfin, dans le chapitre III, nous démontrons un ensemble de propriétés sur les séries de Lie et les exponnentielles de Lie d'après un résultat de REE.

Dans le deuxième rapport, nous présentons la première version du logiciel de calcul de la réalisation. Cette version utilise les mots caractéristiques pour le calcul des coordonnées locales. Nous donnons une brève explication des algorithmes puis les textes de ceux-ci en MACSYMA, et nous terminons par des exemples traités par ce logiciel.

Dans le troisième rapport, nous donnons quelques rappels sur les mots de Lyndon. Nous avons implémenté des algorithmes en MACSYMA sur ces mots et sur la base de Lyndon pour l'algèbre de Lie libre. Nous terminons par des exemples traités par ces algorithmes. Ce travail nous a permis d'envisager d'utiliser les mots de Lyndon dans le logiciel du calcul de la réalisation.

Dans le quatrième rapport, nous donnons des rappels sur la réalisation des systèmes dynamiques non linéaires, des rappels sur les mots de Lyndon et l'algorithme de calcul de la réalisation. Cet algorithme est implémenté en MACSYMA. Dans le complément de ce papier, nous donnons la méthode de calcul des coordonnées locales à partir des mots de Lyndon. Nous donnons aussi un exemple traité par cette méthode.

Dans le cinquième rapport, nous présentons un support théorique du problème de la réalisation.

Enfin, dans l'annexe, nous donnons les programmes en MACSYMA de la dernière version de notre logiciel et des exemples traités par celle-ci.

Pour chacun de ces rapports, nous donnons une table des matières et une bibliographie. Chaque papier possède ses propres notations, donc peut être étudié indépendamment des autres.

PLAN

I. INTRODUCTION ET HEURISTIQUE

INTRODUCTION

Les séries formelles non commutatives ont été introduites aux alentours des années soixante par M.P.SCHUTZENBERGER. Elles se rattachent à la théorie des langages (M.FLIESS[7]) puisque une série formelle n'est rien d'autre qu'une application du monoïde libre X^*, engendré par un alphabet X fini, dans un demi-anneau, un anneau ou un corps et qui associe à tout mot (élément du monoïde libre X^*) un coefficient (ou une multiplicité) qui peut être un entier naturel, un entier relatif, un booléen, un réel, ... Le support d'une série formelle (ensemble des mots dont le coefficient est non nul) est un langage (souvent infini). Les séries rationnelles possèdent des propriétés très intéressantes. On y retrouve des richesses comparables à celles des langages rationnels (A.SALOMAA& M.SOITTOLA[27] ou J.BERSTEL & C.REUTENAUER[1]).

Les séries formelles non commutatives se rattachent aussi à l'algèbre, voire à la géométrie. Elles jouent en effet, le même rôle, en algèbre non commutative ou dans les algèbres d'opérateurs, que celui joué par les séries formelles commutatives en algèbre commutative et en géométrie algébrique (J.BERSTEL & C.REUTENAUER[1], G.JACOB & N.OUSSOUS[17],ou C.REUTENAUER[25]).

L'étude des séries formelles non commutatives peut bénéficier des acquis de ces domaines mais, elle nécessite souvent des techniques spécifiques nouvelles. En retour, les techniques développées et les résultats obtenus jettent souvent des éclairages nouveaux sur chacun de ces domaines précités. Ainsi, les séries algébriques non commutatives ont permis d'étudier la complexité de certains algorithmes (Ph.FLAJOLET, B.SALVY& P.ZIMMERMANN[6]), de coder des graphes planaires ou des arbres (P.CORI[4]), ou encore de démontrer différentes structures combinatoires (G.VIENNOT[30]).

Depuis 1978, les séries formelles non commutatives ont permis à M.FLIESS de résoudre plusieurs problèmes ouverts en Automatique. En effet, les séries génératrices sont un codage symbolique du comportement Entrée/Sortie des systèmes dynamiques non linéaires et permettent des preuves plus simples et souvent algorithmiques. Ainsi, les systèmes dynamiques apparaissent comme un champs d'application priviligié des techniques de séries formelles.

Notre première tâche était une étude fondamentale des séries formelles non commutatives. Pour l'étude des séries de Lie, nous avons développé des techniques de manipulation des séries formelles : définitions algébriques des opérateurs et leur implantation, ainsi que la résolution d'équations en les séries formelles non commutatives (G.JACOB& N.OUSSOUS[15]). Les deux opérations essentielles sur les séries formelles sont le calcul des restes à gauche (ou à droite) d'une série par une lettre, par un polynôme ou par une autre série et le calcul du mélange de deux séries. Les définitions de ces deux opérations sont récursives et se prêtent bien à l'implémentation. Les nouvelles techniques développées par G.JACOB dans son papier[14] donnent un nouvel algorithme pour le calcul

du mélange des séries formelles non commutatives définies par leurs représentations matricielles.

Pour le calcul du rang de Lie et le rang de Lie-Hankel (M.FLIESS[11], G.JACOB& N.OUSSOUS[17], C.REUTENAUER[26]) d'une série, nous avons besoin de calculer la matrice de Hankel (M.FLIESS[9]) et la matrice de Lie-Hankel (C.REUTENAUER[26], G.JACOB& N.OUSSOUS[17]). Pour construire ces matrices nous avons besoin d'une base pour $\mathbb{R}<X>$ et d'une base pour $\mathcal{L}ie<X>$. Nous avons choisi la base canonique X^* pour $\mathbb{R}<X>$ (X^* étant le monoïde libre engendré par un alphabet fini X), ordonnée pour l'ordre lexicographique par longueur. Pour $\mathcal{L}ie<X>$, nous avons choisi la base de CHEN-FOX-LYNDON ordonnée aussi pour le même ordre (pour des raisons que nous verrons plus loin). Comme çà, nous pouvons avoir, pour une série donnée, les premières lignes et les premières colonnes des matrices citées auparavant. Et, en particulier, toutes les lignes et toutes les colonnes dans le cas des polynômes.

Pour construire les éléments de X^* jusqu'à un degré donné, nous nous sommes inspirés d'un algorithme utilisé par C.HESPEL& G.JACOB[12]. Par contre, pour la base de Lyndon, nous avons implémenté tout simplement l'algorithme de construction (N.BOURBAKI[2], G.VIENNOT[30]). Nous aurions pu utilisé l'algorithme de génération des mots de Lyndon donné par J.P.DUVAL[5] qui demande moins de comparaisons et moins de taille mémoire. Cependant, cet algorithme construit les mots de Lyndon ordonnés pour l'ordre lexicographique, ce qui ne nous intéresse pas directement. En outre, le temps gagné par l'algorithme de J.P.DUVAL serait perdu par l'algorithme de tri des mots pour l'ordre voulu et l'algorithme de passage d'un mot de Lyndon à un polynôme de Lie (G.MELANCON& C.REUTENAUER[21] ou N.OUSSOUS[23]).

Durant tout notre travail, nous nous sommes efforcés d'implémenter systématiquement tous les outils algébriques et syntaxiques de traitement ou de résolution. Pour cette implémentation, nous avons choisi le système de Calcul Formel MACSYMA dans un premier temps sur DPS 8 sous MULTICS, puis ensuite, sur les stations de travail SUN sous UNIX. C'est un outil bien adapté à ce genre de traitement. Nous avons la possibilité de manipuler des symboles sans avoir à construire la structure de données correspondante. En plus les structures de données et les opérations offertes par ce système permettent une bonne implémentation et aussi une bonne présentation des résultats qui sont directement lisibles par un mathématicien (ce qui n'est pas le cas pour LISP). Nous avons cependant le problème suivant : la version de MACSYMA dont nous disposons actuellement ne nous permet pas de compiler nos programmes d'où la lenteur de certaines exécutions.

L'intérêt de l'utilisation du système de calcul formel MACSYMA (ou toute autre système) est qu'elle nous a permis tout au long de notre travail la validation d'un certain nombre d'hypothèses. Le fait de disposer d'un ordinateur pour faire les calculs encourage et pousse à traiter des exemples de plus en plus importants. Ce qui permet souvent de remettre en cause certaines hypothèses ou conjectures qui paraissent au début évidentes.

Nous avons ensuite utilisé les algorithmes et outils déjà implantés en MACSYMA

pour résoudre un problème technique et difficile pour les systèmes dynamiques non linéaires : le problème de la réalisation.

Le problème de la réalisation des systèmes dynamiques a été étudié par plusieurs auteurs. Pour la réalisation globale des systèmes dynamiques non linéaires, nous citons entre autres B.JAKUBCZYK[18],H.J.SUSSMANN[29] Pour les systèmes bilinéaires, nous mentionnons les travaux de D'ALESSANDRO, A.ISODORI& A.RUBERTI[3], M.FLIESS[8], G.JACOB[13] et H.J.SUSSMANN[28]. La réalisation locale et minimale des systèmes dynamiques non linéaires a été étudiée notamment par M.FLIESS[11]. C.REUTENAUER[26] avait repris les travaux de M.FLIESS avec d'autres moyens rendant les démonstrations des théorèmes plus faciles.

Nous avons repris ces travaux et nous avons essayé de valider la conjecture que posait G.JACOB et selon laquelle "pour un système dynamique non linéaire de série génératrice finie, on peut toujours trouver la réalisation minimale sous forme polynômiale", c'est-à-dire qu'on peut trouver une fonction d'observation qui soit un polynôme commutatif et des champs de vecteurs dont les composantes soient toutes des polynômes commutatifs.

Nous savons que la réalisation d'un système dynamique non linéaire existe si, et seulement si, le *rang de Lie* de sa série génératrice est fini (M.FLIESS[11], C.REUTENAUER[26], G.JACOB& N.OUSSOUS[17]) et que cette réalisation est unique à un isomorphisme analytique près.

Pour vérifier la conjecture de G.JACOB, nous nous sommes mis à implanter un algorithme qui permettra de calculer la réalisation de séries formelles non commutatives de support fini.

Le problème délicat de cet algorithme est le calcul des coordonnées locales qui doivent être, pour nous, des polynômes pour le produit de mélange.

Dans un premier temps, nous avons essayé de calculer ces polynômes en utilisant ce que nous appelons des mots caractéristiques, suivant l'idée de C.REUTENAUER[26], et qui sont en fait des mots (éléments de X^*) qui indicent des colonnes linéairement indépendantes de la matrice de Lie-Hankel associée à la série à réaliser. Les techniques de calcul relatives à ce premier essai seront exposées plus loin.

Nous avons ensuite commencé à étudier les mots de Lyndon et comme pour les séries, nous avons implémenté systématiquement tous les algorithmes correspondant. En particulier, nous avons implémenté l'algorithme de décomposition d'un mot en produit décroissant de puissances mélanges de mots de Lyndon. L'algorithme de décomposition d'un mot dans la base de POINCARE-BIRKOFF-WITT associée à la base de Lyndon (G.MELANÇON& C.REUTENAUER[21]). Ayant l'algorithme qui génère la base de Lyndon et sachant que le coefficient du mot de Lyndon dans la décomposition, sur X^*, du polynôme de Lie correspondant est 1, nous avons alors essayé une nouvelle approche

pour calculer les coordonnées locales en utilisant les mots de Lyndon et leurs mélanges. Cette approche qui paraissait, au départ, évidente et facile nous a entrainé dans des calculs très fins d'algèbre linéaire et des algorithmes très compliqués. Une fois de plus le système de Calcul Formel nous a permis de remettre en cause nos hypothèses.

Dans la partie suivante, nous détaillons les moyens utilisés et les difficultés rencontrées tout au long de ce travail.

HEURISTIQUE

1. Le système MACSYMA.

Pour commencer, nous allons exposer les moyens offerts par le système de Calcul Formel MACSYMA[20]. Nous voulons travailler sur les mots et sur les séries formelles non commutatives, nous avons donc besoin d'une multiplication abstraite non commutative qui représentera le produit de concaténation. En MACSYMA, il existe une telle opération. Elle est représentée par le point. Les polynômes non commutatifs deviennent alors naturellement une somme de monômes; ces derniers sont le produit commutatif d'un scalaire (élément de $\mathbb{R}$) par un mot.

Les polynômes de Lie sont des sommes de monômes de Lie. Les monômes de Lie sont le produit d'un scalaire par un mot de Lie. Et les mots de Lie sont représentés par des listes à deux éléments (le First et le Last). Les éléments d'une liste sont mis entre deux crochets ce qui donne aux polynômes de Lie une représentation et une présentation naturelle et claire pour un mathématicien. Les écritures sont aussi proches que possible de la structure algébrique du problème.

En plus de ces deux points, le système MACSYMApermet comme tout langage de programmation moderne une programmation modulaire. Ainsi, nous avons la possibilité d'écrire des programmes indépendants et utilisables par tout programme écrit en macsyma (et éventuellement les programmes écrits en MAC-LISP). Ce style de programmation permet de mettre nos outils à la disposition de tous les utilisateurs de MACSYMAet de tous ceux qui travaillent ou qui veulent travailler sur les séries formelles et sur les systèmes dynamiques non linéaires.

2. Notations et définitions.

Avant d'aller plus loin, nous allons d'abord fixer certaines notations et définitions. On note X un *alphabet fini* et X^* le *monoïde libre* engendré par X. Un élément de X^* sera appelé *mot*. Le mot *vide* sera noté ε.

Une série formelle S en les variables non commutatives dans X sera une application de X^* dans $\mathbb{R}$. L'image d'un mot w sera notée $< S|w >$ et appelée le *coefficient* de w dans S. La série S sera notée sous forme d'une somme formelle :

$$S = \sum_{w \in X^*} < S|w > w.$$

L'ensemble de toutes ces séries possède la structure d'algèbre. Il sera noté $\mathbb{R} \ll X \gg$. Le *support* d'une série formelle est l'ensemble des mots de coefficients non nuls dans la série S. C'est un sous ensemble de X^* noté $supp(S)$.

Un polynôme non commutatif est une série formelle non commutative de support fini. La sous-algèbre de $\mathbb{R}\ll X\gg$ formée des polynômes sera notée $\mathbb{R}<X>$.

On notera $\mathcal{L}ie<X>$ la $\mathbb{R}$-algèbre de Lie libre sur X et où le *crochet de Lie* est défini par :

$$[x,y] = xy - yx, \qquad x,y \in \mathcal{L}ie<X>.$$

La base canonique de $\mathbb{R}<X>$ est le monoïde libre X^* ordonnée pour l'ordre lexicographique par longueur. La base de Lie choisie pour $\mathcal{L}ie<X>$ est la base de CHEN-FOX-LYNDON ordonnée pour le même ordre. Elle sera notée $(P_i)_{i\geq 1}$.

3. Les étapes délicates du travail.

Pour écrire le logiciel calculant la réalisation minimale des systèmes dynamiques non linéaires de série génératrice finie, nous disposions déjà de tous les outils que nous avions implémenté auparavant et indépendamment. Essentiellement des outils de manipulation des séries formelles non commutatives (calcul des restes à gauche et à droite, en simplifiant par des polynômes non commutatifs ou par des polynômes de Lie, le produit de mélange de deux polynômes). Nous avions aussi le moyen de construire une base de $\mathbb{R}<X>$ et une de $\mathcal{L}ie<X>$. Ces deux bases, nous les avons dans l'ordre voulu. Il restait donc à construire la *matrice de Lie-Hankel*, à calculer son rang, à construire les *coordonnées locales* et enfin à exprimer la série de départ et les composantes des *champs de vecteurs* comme polynômes commutatifs sur les coordonnées locales.

3.1. Matrice de Lie-Hankel.

Soit S le polynôme à réaliser. La matrice de Lie-Hankel associée est finie et chaque ligne est formée par les coefficients du reste à droite de S par un élément de la base de $\mathcal{L}ie<X>$. Bien sûr, puisqu'il s'agit de polynômes, nous n'avons pas à générer tous les éléments des bases de $\mathbb{R}<X>$ et de $\mathcal{L}ie<X>$. En fait, nous n'avons besoin que des éléments de degré inférieur ou égal au degré de S. Au dela de ce degré, le reste à droite du simplifié de S sera nul. Une fois que cette matrice est construite, son rang est obtenu en appelant la fonction MACSYMA*Rank*.

Cette matrice ne sert pas uniquement pour calculer le rang de Lie de S, elle contient toutes les informations dont nous aurons besoin pour la suite.

3.2. Coordonnées locales.

Pour la construction des coordonnées locales, nous sommes passés par plusieurs moyens. Ces différentes options ont été testées et soit approuvées soit rejetées. Pour montrer l'importance de l'utilisation d'un système de Calcul Formel, nous avons choisi d'exposer les différentes étapes par lesquelles nous sommes passés.

Soit S un polynôme en les variables non commutatives dans un alphabet X.

L'*Annulateur* de S est la sous-algèbre de $\mathcal{L}ie\langle X\rangle$ définie par :

$$\mathcal{A}(S) = \{\ P \in \mathcal{L}ie\langle X\rangle \mid S \triangleright P = 0\ \},$$

$\mathcal{A}(S)$.est engendrée par les polynômes de Lie qui annulent S. On pose :

$$\mathcal{V}(\mathcal{A}(S)) = \{\ Q \in \mathbb{R}\langle X\rangle \mid Q \triangleright \mathcal{A}(S) = 0\ \}.$$

Soit $(P_i)_{i\geq 1}$ une base de $\mathcal{L}ie\langle X\rangle$ telle que $P_1, ..., P_r$ (r étant le rang de Lie de S) soit une base de $\mathcal{L}ie\langle X\rangle$ modulo $\mathcal{A}(S)$ et $P_{r+1}, ..., P_n, ...$ soit une base de $\mathcal{A}(S)$. Les coordonnées locales seront des polynômes propres (i.e. sans coefficient constant)$Q_1, ..., Q_r$ vérifiant :

$$(3.2)\quad \begin{cases} < Q_j \triangleright P_i|\varepsilon > = \delta_{ij} \quad \text{pour } i \leq r, \\ Q_j \in \mathcal{V}(\mathcal{A}(S)), \end{cases}$$

où $< Q_j \triangleright P_i|\varepsilon >$ est le coefficient de ε dans le reste à droite de Q_j par le polynôme de Lie P_i.

(i) Mots caractéristiques.

Cette idée vient de C.REUTENAUER[26]. On utilise la matrice de Lie-Hankel et on obtient les mots caractéristiques en prenant les mots indiçant des colonnes linéairement indépendantes. Supposons que le rang de Lie soit r, il y aura alors r colonnes linéairement indépendantes. Soient donc $(w_i)_{1\leq i\leq r}$ les mots caractéristiques retenus. Nous calculons les coordonnées locales par la formule suivante :

$$Q_i = S \triangleright w_i - < S|w_i >, \quad 1 \leq i \leq r,$$

où $S \triangleright w_i$ est le reste à droite de S par le mot w_i. La formule donne des polynômes propres (c'est-à-dire sans terme constant).

Une vérification empérique, sur quelques exemples nous a conduit à implémenter l'algorithme. A notre grande déception, nous avons tout de suite trouvé des contre-exemples. Le problème avec cet algorithme est que nous n'avons pas un moyen sûr ni pour le choix des mots caractéristiques ni pour le choix des polynômes de la base de Lie. Si le nombre de colonnes non nulles est supérieur au rang de Lie, on aura plusieurs choix possibles. Il faut ensuite vérifier les deux conditions (3.2) pour les polynômes Q_i.

(ii) Mots de Lyndon.

Nous avons commencé l'étude des mots de Lyndon au moment où nous avons voulu implémenter l'algorithme construisant la base de CHEN-FOX-LYNDON (M.LOTHAIRE[19], G.VIENNOT[30]). Nous avons approfondi cette étude en implémentant d'autres algorithmes relatifs aux mots de Lyndon (G.MELANÇON & C.REUTENAUER[21], J.P.DUVAL[5]). Nous avons essayé d'avoir la base de C.-F.-L. en utilisant l'algorithme de J.P.DUVAL[5] mais, nous nous sommes aperçus que l'algorithme de décomposition et l'algorithme de tri lui font perdre tout son avantage.

Cette étude nous a permis de bien comprendre et manipuler ces mots et les propriétés qui leur sont rattachées. En plus, elle nous a donné l'idée d'utiliser ces mots pour calculer les coordonnées locales. La propriété qui nous intéresse plus particulièrement est la suivante : soit l un mot de Lyndon. Soit P le polynôme de la base de C.-F.-L. correspondant. Alors, la décomposition de P sur X^* s'écrit :

$$P = l + \sum_i \alpha_i l_i,$$

où $\alpha_i \in \mathbb{Z}$ et $l_i > l$ pour l'ordre lexicographique (G.MELANÇON & C.REUTENAUER[21], M.LOTHAIRE[19]). Ainsi, nous avons toujours :

$$< l|P >= 1, \qquad \text{coefficient du mot vide dans } l \triangleright P.$$

L'autre intérêt vient du fait que la base de C.-F.-L. et les mots de Lyndon sont tous deux construits par le même algorithme.

(iii) Mots de Lyndon et coordonnées locales.

◇ Dans un premier temps, et après avoir traité quelques exemples à la main, nous nous sommes dits qu'il suffirait de prendre comme coordonnées locales les mots de Lyndon correspondant aux éléments de la base de C.-F.-L. qui donnent des lignes non nulles dans la matrice de Lie-Hankel.

Nous avons implémenté l'algorithme en nous basant sur cette idée et sur le fait que les mots de Lyndon forment une *base de trenscendance* de $\mathbb{R}< X >$ pour le *produit de mélange*. Nous avons donc un logiciel et de nouveau, nous pouvons traiter plus d'exemples et surtout des exemples que nous n'aurions pas pu faire à la main.

Ce moyen nous a permis de remettre, une fois de plus, notre hypothèse en cause et donc de rechercher l'origine de l'erreur. Nous nous sommes tout de suite aperçus qu'on peut avoir dans la matrice de Lie-Hankel des lignes linéairement dépendantes. et donc se pose le problème du choix des lignes à retenir.

◇ Une première solution consistait à chercher les relations duales aux relations trouvées entre les lignes. Ces relations, on les cherche sur les mots de Lyndon et on sait qu'elles sont linéaires. Ensuite, pour le choix des polynômes de Lie à utiliser pour vérifier les conditions sur les coordonnées locales, nous gardons tout simplement les polynômes de la base de C.-F.-L. correspondant aux mots de Lyndon de plus haut degré apparaissant dans les expressions des coordonnées locales.

De nouveau, nous avons eu une nouvelle version du logiciel et donc le moyen de traiter des exemples plus intéressants. Une fois encore, nous trouvons des contre-exemples et nous devons donc revoir nos hypothèses.

Le problème vient cette fois-ci du fait qu'avec nos choix, il est possible que les expressions choisies pour les coordonnées locales ne vérifient pas la deuxième condition sur celles-ci.

◇ La dernière version du logiciel prend en compte tous ces cas de figure et permet d'avoir des coordonnées locales sous forme de combinaisons linéaires de mélanges de mots de Lyndon.

Ce qu'il faut souligner c'est que la taille du logiciel a pratiquement doublé. Pour la dernière version nous avons fait appel à des notions très fines d'algèbre linéaire. En particulier, nous avons implémenté un algorithme qui permet de définir une application linéaire à partir des relations de dépendance trouvées entre les lignes de la matrice de Lie-Hankel. Ensuite, nous avons implémenté un algorithme qui permet de construire une base canonique du noyau d'une telle application. Et enfin, nous avons implémenté un algorithme qui determine les expressions duales aux relations trouvées entre les lignes de la matrice de Lie-Hankel qui sont au fait des relations sur les éléments de la base de C.-F.-L. Il est évident que sans un système de Calcul Formel, il serait pratiquement impossible de développer ce travail.

3.3. Calcul de la fonction d'observation.

Calculer la fonction d'observation consiste à exprimer le polynôme à réaliser comme combinaison linéaire des mélanges des coordonnées locales trouvées précédemment.

La méthode utilisée au départ, dans toutes les versions, est la suivante : on dispose des coordonnées locales $(Q_i)_{1\leq i\leq r}$. On commence par déterminer les *coordonnées utiles* (c'est-à-dire les coordonnées dont le degré est inférieur ou égal au degré du polynôme de départ). Il est évident que nous n'aurons besoin que de celles-ci pour exprimer le polynôme en question. Ensuite, nous calculons tous les *mélanges utiles* (mélanges dont le degré reste inférieur ou égal au degré du polynôme de départ). Supposons que nous ayons trouvé $(ml_i)_{1\leq i\leq n}$:coordonnées et mélanges confondus. Nous posons alors :

$$P = \sum_{i=1}^{n} a_i ml_i,$$

et nous identifions le polynôme P avec le polynôme de départ (tout deux considérés comme polynômes sur X^*) pour déterminer les a_i.

Cette méthode, bien qu'elle permette d'avoir l'expression de la fonction d'observation a plusieurs inconvénients. D'une part, elle demande beaucoup de calculs et surtout, on calcule des mélanges qui ne sont pas tous utiles et on sait que le calcul des mélanges demande beaucoup de temps et beaucoup de place memoire. D'autre part, nous sommes obligés de garder tous les mélanges. Enfin, le calcul des a_i demande la résolution de systèmes d'équations qui prend beaucoup de temps et nécessite beaucoup de calculs intermédiares.

Le problème c'est que cette opération n'est pas utilisée uniquement pour le calcul de la fonction d'observation mais aussi pour le calcul des composantes des champs de vecteurs. Pour toutes ces raisons, il nous a fallu trouver une autre méthode.

3.4. Un algorithme petit mais qui fait le maximum.

L'algorithme que nous allons décrire dans ce qui suit est un algorithme qui peut être utilisé indépendamment du problème de la réalisation. Nous allons essayé d'en donner une description la plus indépendante possible des systèmes.

Soit S un polynôme en les variables non commutatives dans un alphabet X. L'*Annulateur* de S est la sous-algèbre de $\mathcal{L}ie<X>$ définie par :

$$\mathcal{A}(S) = \{ P \in \mathcal{L}ie<X> \mid S \triangleright P = 0 \},$$

$\mathcal{A}(S)$ est engendrée par les polynômes de Lie qui annulent S. On pose :

$$\mathcal{V}(\mathcal{A}(S)) = \{ Q \in \mathbb{R}<X> \mid Q \triangleright \mathcal{A}(S) = 0 \}.$$

Soit $(P_i)_{i\geq 1}$ une base de $\mathcal{L}ie<X>$ telle que $P_1, ..., P_r$ soit une base de $\mathcal{L}ie<X>$ modulo $\mathcal{A}(S)$ et $P_{r+1}, ..., P_n, ...$ soit une base de $\mathcal{A}(S)$. Soit $q_1, ..., q_r$ des polynômes non commutatifs de terme constant nul et vérifiant :

$$\begin{cases} < q_j \triangleright P_i | \varepsilon > = \delta_{ij} \quad \text{pour } i \leq r, \\ q_j \in \mathcal{V}(\mathcal{A}(S)), \end{cases}$$

où $< q_j \triangleright P_i | \varepsilon >$ est le coefficient de ε dans le reste à droite de q_j par le polynôme de Lie P_i.

L'algorithme que nous proposons est un algorithme qui permet d'exprimer le polynôme S en fonction des mélanges des polynômes $(q_j)_{1\leq i\leq r}$ lorsque c'est possible. Cet algorithme utilise peu de mémoire puisqu'on ne mémorise que la valeur de S et la valeur de son dernier reste. On construit l'expression de S au fur et à mesure du déroulement de l'algorithme.

Nous donnons ci-dessous le texte de l'algorithme en pseudo- MACSYMA:

- ◇ $T \leftarrow S, \quad H \leftarrow 0,$
- ◇ $j \leftarrow r,$
- ◇ $fact1 \leftarrow fact \leftarrow 1,$
- ◇ **TQ** ($j \neq 0$ et $S \neq 0$) **FAIRE**
  - • $n \leftarrow 0,$
  - • **TQ** $T \triangleright P_j \neq 0$ **FAIRE**
    - ○ $T \leftarrow T \triangleright P_j,$
    - ○ $n \leftarrow n+1,$
  - • **FTQ**
  - • cste $\leftarrow$ *terme constant de* $T,$
  - • **SI** *cste* $\neq 0$ **ALORS**
    - ○ $S \leftarrow S - \frac{cste}{n!} * (fact1 \sqcup q_j{}^{\sqcup n}),$
    - ○ $H \leftarrow H + \frac{cste}{n!} * (fact * q_j{}^n),$
    - ○ $fact1 \leftarrow fact \leftarrow 1,$
    - ○ $T \leftarrow S,$
    - ○ $j \leftarrow j+k, \quad k \leftarrow 0,$
  - • **SINON**
    - ○ $fact1 \leftarrow \frac{1}{n!} * (fact1 \sqcup q_j{}^{\sqcup n}),$
    - ○ $fact \leftarrow \frac{1}{n!} * (fact * q_j{}^n),$
    - ○ $j \leftarrow j-1, \quad k \leftarrow k+1,$
- ◇ **FTQ**

La boucle (2) s'arrête puisque les restes deviennent nuls au bout d'un certain nombre de simplifications. La boucle (1) se termine aussi puisque le nombre de polynômes P_j est fini ($1 \leq j \leq r$.)

Cet algorithme a comme avantage, en plus de sa simplicité et son efficacité, le fait qu'il permet de décider si un polynôme peut ou non être écrit comme combinaison linéaire des mélanges d'un système de polynômes donné. Au cas où c'est possible, il nous renvoie cette expression comme résultat.

4. Prolongements et ouvertures.

Nous comptons poursuivre l'étude fondamentale des séries formelles non commutatives, et aussi l'implémentation systématique des outils algébriques et syntaxiques de traitement ou de résolution. Ceci nous semble en effet être nécessité par la longueur et la complexité des calculs, dès qu'il s'agit de traiter des exemples non commutatifs. La nécessité de l'implémentation est prouvée par le travail présenté.

Nous comptons aussi prolonger l'algorithme de la réalisation dans deux voies : d'une part étudier la réalisation des séries rationnelles (qui sont des séries génératrices des systèmes bilinéaires) et d'autre part faire les implémentations sur le nouveau système de Calcul Formel SCRATCHPAD II. C.HESPEL& G.JACOB ont déjà implémenté un logiciel calculant, pour toute série génératrice d'un système dynamique analytique, un approximant à un ordre donné et de rang minimal. Nous souhaitons pouvoir inclure l'ensemble des logiciels dans une même bibliothèque en bénéficiant de l'accord avec IBM.

Nous organisons des journées nationales-séminaire à Lille sur "*les traitements algébriques et informatiques des séries formelles non commutatives*" et nous espérons que par cette initiative nous pourrons tisser des liens avec d'autres équipes dans d'autres laboratoires.

BIBLIOGRAPHIE

[1] J.BERSTEL & C.REUTENAUER. – Les séries rationnelles et leurs langages. – *Editions Masson*, 1984.

[2] N.BOURBAKI. – Groupes et Algèbres de Lie. Chapitre II et III. – *Diffusion C.C.L.S., Paris.*

[3] D'ALESSANDRO, A.ISODORI & A.RUBERTI. – Realisation and structure theory of bilinear dynamical systems . – *SIAM J. Control*, **12**, 1974, p. 517 – 535.

[4] P.CORI. – Un code pour les graphes planaires et ses applications. – *Astérisque*, 1975, p. 77-85.

[5] J.P.DUVAL. – Génération d'une section des classes de conjugaison et arbre des mots de Lyndon de longueur bornée. – *Publication du LITP Paris*, **88–20**, 1988.

[6] P.FLAJOLET, B.SALVY& P.ZIMMERMANN. – Lambda-Upsilon-Omega : An assistant algorithms analyzer. – *Submitted to AECC-6, Rome*, 1988.

[7] M.FLIESS. – Formal languages and formal power series. – *Séminaire IRIA Logique et Automates, Le Chesnay*, 1971, p. 77-85.

[8] M.FLIESS. – Sur la réalisation des systèmes bilinéaires . – *C. R. Acad. Sc. Paris*, **A-277**, 1973, p. 923 – 926.

[9] M.FLIESS. – Matrices de Hankel . – *J. Math. Pures APPL.*, **53**, 1974, p. 197 – 222.

[10] M.FLIESS. – Fonctionnelles causales non linéaires et indéterminées non commutatives . – *Bull. Soc. Math. France*, **109**, 1981, p. 3 – 40.

[11] M.FLIESS. – Réalisation locale des systèmes non linéaires, algèbres de Lie filtrées transitives et séries génératrices . – *Invent. Math.*, **71**, 1983, p. 521 – 537.

[12] C.HESPEL & G.JACOB. – Approximation of nonlinear systems by bilinear ones. – *Algebraic and Geometric Methods in Nonlinear Control Theory*, 1986, p. 511–520.

[13] G.JACOB. – Réalisation des systèmes réguliers (ou bilinéaires) et séries génératrices non commutatives. – *Séminaire d'Aussois, In Outils et Modèles mathématiques pour l'automatique, l'analyse des systèmes et le traitement du signal*, coord. I.D.LANDAU, éd. CNRS, t. **1 – N° 567**, 1980, p. 325-357.

[14] G.JACOB. – Sur le produit de mélange. – *Publication du LIFL Lille*, **IT–142**, 1988.

[15] G.JACOB & N.OUSSOUS. – Sur un résultat de REE: séries de Lie et algèbres de mélange. *Publication du LIFL Lille*, **IT–103**, 1987.

[16] G.JACOB & N.OUSSOUS. – Un logiciel calculant la réalisation des systèmes analytiques de série génératrice finie. – *Publication du LIFL Lille*, **IT–123**, 1988.

[17] G.JACOB & N.OUSSOUS. – Réalisation locale et minimale des systèmes dynamiques non linéaires et mots de Lyndon. – *Publication du LIFL Lille*, **IT–140**, 1988.

[18] B.JAKUBCZYK. – Existence and uniqueness of realisations of nonlinear systems . *SIAM J. Control Optimiz.*, **18**, 1977, p. 728 – 740.

[19] M.LOTHAIRE. – Combinatorics on words. – *Reading, Massachusetts*, 1983.

[20] MACSYMA. – Reference Manual, Version Ten. – *Massachusetts Institute of Technology and Symbolics*, 1983.

[21] G.MELANÇON & C.REUTENAUER. – Lyndon words, free algebras and shuffles. – *Publication du LITP Paris et UQAM Montreal, Quebec*, **87–63**, 1987.

[22] N.OUSSOUS. – .– , , 1988.

[23] N.OUSSOUS. – Mots de Lyndon et Macsyma. – *à paraitre*, 1988.

[24] D.E.RADFORD. – A natural ring basis for the shuffle algebra and an application to group schemes .– *Journal of Algebra*, **58**, 1979, p. 432 – 454.

[25] C.REUTENAUER. – Séries formelles et algèbres syntactiques .– *J. Algebra*, **66**, 1980, p. 448 – 483.

[26] C.REUTENAUER. – The local realisation of generating series of finite Lie rank. *Algebraic and Geometric Methods In Nonlinear Control Theory*, **eds.** M.FLIESS and M.HAZEWINKEL, 1986, p. 33–43.

[27] A.SALOMAA & M.SOITTOLA. – Automata-theoretic aspects of formal power series. *Springer-Verlag, New-York*, 1978.

[28] H.J.SUSSMANN. – Minimal realisations and canonical forms for bilinear systems .– *J. Franklin Inst.*, **301**, 1976, p. 593 – 604.

[29] H.J.SUSSMANN. – Existence and uniqueness of minimal realisations of nonlinear systems .– *Math. Systems Theory*, **10**, 1977, p. 263 – 284.

[30] G.VIENNOT. – Algèbres de Lie libres et monoïdes libres .– *Lecture Notes In Mathematics, Springer–Verlag*, 691, 1978.

II. SUR UN RESULTAT DE REE :

SERIES DE LIE
ET
ALGEBRES DE MELANGE

SUR UN RÉSULTAT DE REE : SÉRIES DE LIE ET ALGÈBRES DE MÉLANGE

G. Jacob[1] & N. Oussous[1]

IT – 103 Juin 1987

RESUME

Dans un article de 1958, R.REE a montré que les séries formelles non commutatives introduites par K.T.CHEN pour représenter les entrées d'un système dynamique sont des séries de Lie. Nous reprenons ici la démonstration de ce théorème, développant systématiquement les outils utilisés : algèbres de Lie libres, séries et polynômes de Lie, produit de mélange des séries et calcul de résiduels.

ABSTRACT

In his paper "Lie elements and an algebra associated with shuffles" (1958), R.REE has proved that the powers series used by K.T.CHEN to describe the commands of dynamical systems are Lie series. We present here a modern traitement of that theorem, and develop systematically the following tools : free Lie algebras, Lie polynoms and series, shuffle product of series and calculus of residuals.

[1] Laboratoire d'Informatique Fondamentale de Lille.– U.A. 369 du C.N.R.S.
Université de LILLE I – 59655 VILLENEUVE D'ASCQ CEDEX FRANCE.

TABLE DES MATIERES

INTRODUCTION

L'introduction par M. Fliess (*cf* [4]) des séries génératrices des systèmes dynamiques analytiques non linéaires de la forme:

$$\begin{cases} \dot{z} = A_0(z) + \sum_{i=1}^{n} u_i(t) A_i(z), \\ s(t) = h(z(t)). \end{cases}$$

permet d'obtenir des algorithmes effectifs et performants permettant, par exemple de développer pour ces systèmes un calcul symbolique (*cf* [5]) ou d'étudier les conditions de découplage par rapport aux perturbations (*cf* [3]) ou encore de calculer des approximants bilinéaires de rang minimal (*cf* [6]).

Le développement de ces algorithmes et leur compréhension nécessitent le développement d'outils algébriques pour la manipulation des séries formelles non commutatives, outils habituellement développés dans le cadre de la théorie des langages.

Le texte ici proposé vise à présenter les techniques de manipulation des séries formelles non commutatives nécessaires à la compréhension et à la démonstration complète des résultats prouvés par R. Ree dans [9].

Il s'agit essentiellement de l'étude des séries formelles de Lie, des exponentielles de Lie, ainsi que la preuve du théorème selon lequel la série formelle représentant les entrées d'un système (appelée aussi "série de CHEN") est l'exponentielle d'une série de Lie (voir aussi [2]).

Le texte présent est issu d'une part d'un cours de DEA-informatique du premier auteur, et d'autre part du mémoire de DEA du second auteur. Il sera suivi par une étude de la réalisation locale, et par un logiciel permettant, dans certains cas particuliers, le calcul de cette réalisation locale.

ALGEBRE DE LIE ET PRODUIT TENSORIEL

Introduction.

Vue l'utilité des *algèbres de Lie* et du *produit tensoriel* pour la suite de notre travail, nous avons jugé nécessaire de consacrer ce premier chapitre aux rappels de ces notions. Dans le premier paragraphe nous rappelons la notion d'*algèbre de Lie* et nous étudions en exemple les *algèbres de Lie linéaires* et les *algèbres de Lie de dérivations*. Nous poursuivons dans le même paragraphe par la notion d'*idéal* et nous terminons sur les notions d'*homomorphisme* et de *représentations*.

Dans le second paragraphe nous rappelons la définition du *produit tensoriel*. Nous commençons par le cas de deux **espaces vectoriels** puis de deux **applications linéaires**. Ensuite, nous étudions le cas général. Comme propriété du produit tensoriel nous étudions en détail l'**associativité**. Nous terminons ce paragraphe sur la notion d'*algèbre tensorielle*.

Pour ce premier chapitre, nous nous sommes référés à un certain nombre d'ouvrages mathématiques dont une liste figure à la fin de celui-ci.

1.1 ALGEBRE DE LIE

1.1.1 Notion d'algèbre de Lie.

DEFINITION 1.1.1.1. – *Soit $\mathcal{L}$ un espace vectoriel sur un corps K, muni de l'opération:*

$$\mathcal{L} \times \mathcal{L} \longrightarrow \mathcal{L},$$
$$(x, y) \longmapsto [x\ y], \qquad (\textit{crochet de } x \textit{ et de } y),$$

$\mathcal{L}$ est une algèbre de Lie sur K si les axiomes suivants sont satisfaits:

1. *Cette opération est bilinéaire,*
2. $\forall x \in \mathcal{L}, \qquad [x\ x] = 0,$
3. $\forall x, y, z \in \mathcal{L}, \qquad [x[y\ z]] + [y[z\ x]] + [z[x\ y]] = 0, \qquad (\textit{identité de } JACOBI).$

De (1) et (2) résulte, pour tout x et y de $\mathcal{L}$,

$$[x\ y] = -[y\ x]$$

ce qui fait que cette opération est anticommutative.

Si le corps K est de caractéristique différente de deux, alors l'anticommutativité implique l'axiome (2).

On dira que deux algèbres de Lie $\mathcal{L}$ et $\mathcal{L}'$ sur un corps K sont *isomorphes* si il existe un isomorphisme, ϕ, d'espaces vectoriels, $\phi : \mathcal{L} \longrightarrow \mathcal{L}'$, qui satisfait:

$$\phi([x\ y]) = [\phi(x)\ \phi(y)], \qquad \text{pour tout } x, y \text{ de } \mathcal{L},$$

ϕ est alors appelé un isomorphisme d'algèbres de Lie.

Un sous-espace vectoriel $\mathcal{L}'$ de $\mathcal{L}$ est une sous-algèbre de Lie de $\mathcal{L}$ si $\mathcal{L}'$ est stable pour les *crochets de Lie*, c'est-à-dire:

$$\forall x, y \in \mathcal{L}', \qquad [x\ y] \in \mathcal{L}'.$$

1.1.2 Algèbre de Lie linéaire.

Si V est un espace vectoriel de dimension finie, n, sur K, on note $\mathcal{E}nd(V)$ l'ensemble des applications linéaires de V dans lui-même. $\mathcal{E}nd(V)$ est un espace vectoriel de dimension n^2 sur K et est un anneau pour le produit usuel des applications.

On définit l'opérateur ‹ *crochet* › par:

$$[f, g] = fg - gf, \qquad \text{pour } f \text{ et } g \text{ dans } \mathcal{E}nd(V).$$

Avec cette opération $\mathcal{E}nd(V)$ devient une algèbre de Lie sur K. Cette nouvelle algèbre sera notée $\mathcal{G}l(V)$ et appelée *algèbre linéaire générale.*

Toute sous-algèbre de l'algèbre de Lie $\mathcal{G}l(V)$ est appelée *algèbre de Lie linéaire.*

1.1.3 Algèbre de Lie des dérivations.

Certaines algèbres de Lie des *transformations linéaires* se présentent plus naturellement comme des *algèbres de dérivations.*

Par K-algèbre (non nécessairement associative), on veut dire un espace vectoriel A sur le corps K muni d'une opération *bilinéaire*

$$A \times A \longrightarrow A,$$

notée en général par juxtaposition (sauf dans le cas où A est une algèbre de Lie auquel cas on utilise les crochets).

Par *dérivation de* A, on veut dire l'application linéaire: $\delta : A \longrightarrow A$, qui satisfait la relation:

$$\delta(ab) = a\delta(b) + \delta(a)b.$$

On peut montrer facilement que $\mathcal{D}er(A)$, l'ensemble des dérivations de A, est un sous-espace vectoriel de $\mathcal{E}nd(A)$ et qu'il est stable par ‹ *crochet* › . Pour cette dernière affirmation, on a en effet:

$$\forall \delta, \delta' \in \mathcal{D}er(A), \quad \forall a, b \in A, \qquad \begin{aligned} \delta(ab) &= a\delta(b) + \delta(a)b, \\ \delta'(ab) &= a\delta'(b) + \delta'(a)b. \end{aligned}$$

Alors

$$\begin{aligned} \delta\delta'(ab) &= \delta(a\delta'(b) + \delta'(a)b) \\ &= a\delta\delta'(b) + \delta(a)\delta'(b) + \delta'(a)\delta(b) + \delta\delta'(a)b, \end{aligned}$$

et

$$\begin{aligned} \delta'\delta(ab) &= \delta'(a\delta(b) + \delta(a)b) \\ &= a\delta'\delta(b) + \delta'(a)\delta(b) + \delta(a)\delta'(b) + \delta'\delta(a)b \end{aligned}$$

d'où

$$\begin{aligned}[\delta,\delta'](ab) &= (\delta\delta' - \delta'\delta)(ab)\\ &= \delta\delta'(ab) - \delta'\delta(ab)\\ &= a(\delta\delta' - \delta'\delta)(b) + (\delta\delta' - \delta'\delta)(a)b\\ &= a[\delta,\delta'](b) + [\delta,\delta'](a)b.\end{aligned}$$

Donc $[\delta,\delta'] \in \mathcal{D}er(A)$.

$\mathcal{D}er(A)$ est ainsi une sous-algèbre de Lie de $\mathcal{G}l(A)$.

Puisque l'algèbre de Lie $\mathcal{L}$ est une K-algèbre au sens défini ci-dessus alors $\mathcal{D}er(\mathcal{L})$ est définie. Certaines dérivations se présentent assez naturellement comme suit: si $x \in \mathcal{L}$, l'application: $y \longrightarrow [x\ y]$, est un endomorphisme de $\mathcal{L}$, que l'on note ad_x. En fait, $ad_x \in \mathcal{D}er(\mathcal{L})$, car, on peut réécrire l'identité de $JACOBI$ sous la forme suivante:

$$[x[y\ z]] = [[x\ y]z] + [y[x\ z]],$$

c'est-à-dire

$$ad_x([y\ z]) = [ad_x(y)\ z] + [y\ ad_x(z)].$$

Les dérivations de cette forme sont dites *internes*, les autres sont dites *externes*. Il est bien sûr parfaitement possible d'avoir $ad_x = 0$ même quand $x \neq 0$.

L'homomorphisme d'algèbres de Lie:

$$\begin{aligned}&\mathcal{L} \longrightarrow \mathcal{D}er(\mathcal{L}), \qquad \text{défini par :}\\ &x \longmapsto ad_x,\end{aligned}$$

tel que

$$\forall y \in \mathcal{L}, \quad ad_x(y) = [x\ y],$$

est appelé la *représentation adjointe* de $\mathcal{L}$.

1.1.4 Idéaux.

Soient I, J deux sous-espaces vectoriels de $\mathcal{L}$. On note $[I\ J]$ le sous-espace vectoriel de $\mathcal{L}$ engendré par les $[x\ y]$ pour $x \in I$ et $y \in J$.

Remarque. – Puisque $[x\ y] = -[y\ x]$, alors $[I\ J] = [J\ I]$.

DEFINITION 1.1.4.1. – ***Un sous-espace vectoriel J de l'algèbre de Lie $\mathcal{L}$ est un idéal de $\mathcal{L}$ si*** $[J\ \mathcal{L}] \subset J$. ***En d'autres termes si***

$$\forall (x,y) \in \mathcal{L} \times J, \qquad [x\ y] \in J.$$

Exemple:

1. Il est évident que $\{0\}$ et $\mathcal{L}$ sont des idéaux de $\mathcal{L}$. Si $\mathcal{L}$ n'a pas d'autre idéal que $\{0\}$ et $\mathcal{L}$, on dira que $\mathcal{L}$ est une algèbre de Lie *simple*.

2. On appelle *centre de* $\mathcal{L}$ l'ensemble :

$$Z(\mathcal{L}) = \{\ z \in \mathcal{L} \mid \forall x \in \mathcal{L},\ [x,z] = 0\ \} = \{\ z \in \mathcal{L} \mid ad_z = 0\ \}.$$

C'est un idéal de $\mathcal{L}$. En effet, soit $z_1 \in Z(\mathcal{L})$ et soit $z_2, x \in \mathcal{L}$. On a, d'après l'identité de *JACOBI*,

$$[x[z_1\ z_2]] = [z_1[x\ z_2]] - [z_2[x\ z_1]]$$

or

$$\begin{cases} \text{et} & z_1 \in Z(\mathcal{L}) & \Longrightarrow & [x\ z_1] = 0, \\ & z_2 \in \mathcal{L} & \Longrightarrow & [x\ z_2] \in \mathcal{L}, \end{cases}$$

donc

$$[x[z_1\ z_2]] = 0 \quad \Longrightarrow \quad [z_1\ z_2] \in Z(\mathcal{L}).$$

Il est facile de vérifier que $Z(\mathcal{L})$ est un sous-espace vectoriel de $\mathcal{L}$. Donc $Z(\mathcal{L})$ est un idéal de $\mathcal{L}$.

Remarques:

1. $\mathcal{L}$ est commutative si, et seulement si, $Z(\mathcal{L}) = \mathcal{L}$.

2. Si I et J sont deux idéaux de $\mathcal{L}$, alors $I + J$ et $[I\ J]$ sont des idéaux de $\mathcal{L}$, avec

$$\begin{cases} I + J = \{\ x + y \mid x \in I,\ y \in J\ \} \\ [I\ J] = \{\ \sum\limits_{finie} x_i y_i \mid x_i \in I,\ y_i \in J\ \}. \end{cases}$$

La construction de l'*algèbre quotient* $\mathcal{L}/J$, où J est un idéal de $\mathcal{L}$, est formellement la même que la construction de l'anneau quotient: L'espace vectoriel $\mathcal{L}/J$ est l'espace quotient et la multiplication de Lie est définie par:

$$[x + J, y + J] = [x\ y] + J.$$

A priori il n'y a pas d'ambiguité puisque, si $x + J = x' + J$ et $y + J = y' + J$, alors, on peut écrire:

$$\begin{gathered} x' = x + u \quad (u \in J) \qquad \text{et} \qquad y' = y + v \quad (v \in J) \\ \Longrightarrow \quad [x'\ y'] = [x + u, y + v] = [x\ y] + ([x\ v] + [u\ y] + [u\ v]), \end{gathered}$$

J étant un idéal de $\mathcal{L}$, donc ce qui est entre parenthèses est dans J. On en déduit que

$$[x'\ y'] + J = [x\ y] + J.$$

Le *normalisateur* d'une sous-algèbre A de $\mathcal{L}$ est défini par

$$N_{\mathcal{L}}(A) = \{x \in \mathcal{L} \mid [x\ A] \subset A\}.$$

$N_{\mathcal{L}}(A)$ est une sous-algèbre de Lie de $\mathcal{L}$. En effet, soit $x, y \in N_{\mathcal{L}}(A)$. On a:

$$\begin{aligned} [[x\ y]A] &\subseteq -[[y\ A]x] - [[A\ x]y] \qquad \text{(d'après l'identité de } JACOBI) \\ &\subseteq [x[y\ A]] - [y[x\ A]]. \end{aligned}$$

$$\left.\begin{array}{rcl} y \in N_{\mathcal{L}}(A) & \Longrightarrow & [y\ A] \subset A \\ x \in N_{\mathcal{L}}(A) & \Longrightarrow & [x\ A] \subset A \end{array}\right\} \quad \Longrightarrow \quad [[x\ y]A] \subset [x\ A] - [y\ A] \subset A.$$

Donc $\quad [x\ y] \in N_{\mathcal{L}}(A).$

Il est par ailleurs facile de voir que $N_{\mathcal{L}}(A)$ est un sous-espace vectoriel de $\mathcal{L}$.

Le *centraliseur* du sous-ensemble X de $\mathcal{L}$ est l'ensemble

$$C_{\mathcal{L}}(X) = \{\, x \in \mathcal{L} \mid [x\ X] = 0 \,\}.$$

Il est clair que $C_{\mathcal{L}}(X)$ est un sous-espace vectoriel de $\mathcal{L}$. En utilisant l'identité de JACOBI, on montre facilement que $C_{\mathcal{L}}(X)$ est une sous-algèbre de $\mathcal{L}$.

1.1.5 Homomorphisme et représentations.

DEFINITION 1.1.5.1. – *Soient $\mathcal{L}$ et $\mathcal{L}'$ deux algèbres de Lie sur un corps K. L'application $\phi : \mathcal{L} \longrightarrow \mathcal{L}'$, est appelée homomorphisme si:*

$$\forall x, y \in \mathcal{L}, \qquad \phi([x\ y]) = [\phi(x)\ \phi(y)].$$

On pose $\ker(\phi) = \phi^{-1}(\{0\})$ et $\mathrm{Im}(\phi) = \phi(\mathcal{L})$.
ϕ est un *monomorphisme* si $\ker(\phi) = \{0\}$.
ϕ est un *épimorphisme* si $\mathrm{Im}(\phi) = \mathcal{L}'$.

Tout épimorphisme qui est aussi un monomorphisme est un *isomorphisme*, c'est-à-dire admet un homomorphisme inverse.

Si ϕ est un homomorphisme de $\mathcal{L}$ dans $\mathcal{L}'$, alors $\ker(\phi)$ est un idéal de $\mathcal{L}$. En effet, soient $x \in \ker(\phi)$ et $y \in \mathcal{L}$. On a:

$$\phi([x\ y]) = [\phi(x)\ \phi(y)] = [0\ \phi(y)] = 0.$$

Donc $[x\ y] \in \ker(\phi)$. Il est aussi facile de montrer que $\ker(\phi)$ est un sous-espace vectoriel de $\mathcal{L}$.

$\mathrm{Im}(\phi)$ est une sous-algèbre de Lie de $\mathcal{L}'$. En effet, il est facile de voir que $\mathrm{Im}(\phi)$ est un sous-espace vectoriel de $\mathcal{L}'$ et ona:

$$\forall x', y' \in \mathrm{Im}(\phi), \quad \exists x, y \in \mathcal{L}, \quad \text{tels que,} \quad x' = \phi(x) \text{ et } y' = \phi(y).$$

Donc

$$[x'\ y'] = [\phi(x)\ \phi(y)] = \phi([x\ y]) \in \mathrm{Im}(\phi).$$

PROPOSITIONS:
Comme dans les autres théories d'algèbres, on a:

1. Si $\phi : \mathcal{L} \longrightarrow \mathcal{L}'$ est un homomorphisme d'algèbres de Lie, alors $\mathcal{L}/\ker(\phi)$ est isomorphe à $\mathrm{Im}(\phi)$. Si J est un idéal de $\mathcal{L}$ inclus dans $\ker(\phi)$, alors il existe un unique homomorphisme

$$\psi : \mathcal{L}/J \longrightarrow \mathcal{L}'$$

qui fait que le diagramme suivant soit commutatif

$$\begin{array}{ccc} \mathcal{L} & \xrightarrow{\ \phi\ } & \mathcal{L}' \\ {\scriptstyle \pi}\searrow & & \nearrow{\scriptstyle \psi} \\ & \mathcal{L}/J & \end{array}$$

(π =application canonique du quotient par J).

2. Si I et J sont deux idéaux de $\mathcal{L}$ tels que $I \subset J$, alors J/I est un idéal de $\mathcal{L}/I$ et $(\mathcal{L}/I)/(J/I)$ est isomorphe à $\mathcal{L}/J$.

3. Si I et J sont deux idéaux de $\mathcal{L}$, alors $(I+J)/J$ et $I/I \cap J$ sont isomorphes.

Une *représentation* d'une algèbre de Lie $\mathcal{L}$ est un homomorphisme

$$\varphi \ : \ \mathcal{L} \longrightarrow \mathcal{G}l(V),$$

où V est un espace vectoriel sur K. Quoiqu'on exige que $\mathcal{L}$ soit de dimension finie, V peut être de dimension quelconque vu que $\mathcal{G}l(V)$ aura toujours un sens.

On peut aussi associer canoniquement, à toute représentation d'une algèbre de Lie $\mathcal{L}$ dans $\mathcal{G}l(V)$, une *action de $\mathcal{L}$ sur l'espace vectoriel V*, en posant pour tout $x \in \mathcal{L}$ et $q \in V$

$$x \star q = \varphi(x)(q),$$

où l'on appelle action de $\mathcal{L}$ sur V toute opération

$$\star \ : \ \mathcal{L} \times V \longrightarrow V$$

vérifiant

$$\begin{cases} (ax+by)\star q = a(x\star q) + b(y\star q), \\ [x,y]\star q = x\star(y\star q) - y\star(x\star q), \end{cases} \qquad \text{pour tout} \quad x,y \in \mathcal{L} \quad \text{et} \quad a,b \in K.$$

Un exemple important de représentation est la *représentation adjointe* de l'algèbre de Lie $\mathcal{L}$,

$$\begin{aligned} ad \ : \mathcal{L} &\longrightarrow \mathcal{G}l(\mathcal{L}), \\ x &\longmapsto ad_x, \end{aligned}$$

où

$$ad_x(y) = [x\ y], \quad \text{pour tout} \quad y \in \mathcal{L}.$$

Il est clair que ad est une application linéaire et pour voir qu'elle préserve les crochets, on calcule, pour $z \in \mathcal{L}$:

$$\begin{aligned} [ad_x\ ad_y](z) &= ad_x ad_y(z) - ad_y ad_x(z) \\ &= ad_x([y\ z]) - ad_y([x\ z]) \\ &= [x[y\ z] - [y[x\ z] \\ &= [[x\ y]z] \qquad \text{(d'après l'identité de } JACOBI), \\ &= ad_{[x\ y]}(z). \end{aligned}$$

z étant quelconque, donc $ad_{[x\ y]} = [ad_x\ ad_y]$.

D'autre part, on a:

$$\begin{aligned} \ker(ad) &= \{\ x \in \mathcal{L} \mid ad_x = 0\ \} \\ &= \{\ x \in \mathcal{L} \mid \forall y \in \mathcal{L},\ ad_x(y) = 0\ \} \\ &= \{\ x \in \mathcal{L} \mid \forall y \in \mathcal{L},\ [x\ y] = 0\ \} \\ &= Z(\mathcal{L}) \qquad \text{(centre de } \mathcal{L}). \end{aligned}$$

Si $\mathcal{L}$ est *simple*, alors $Z(\mathcal{L}) = \{0\}$ et donc ad est un monomorphisme.

$\mathrm{Im}(ad)$ est une sous-algèbre de $\mathcal{G}l(\mathcal{L})$, c'est donc une algèbre de Lie *linéaire*. On en déduit que toute algèbre de Lie *simple* est isomorphe à une algèbre de Lie linéaire.

1.2 PRODUIT TENSORIEL

1.2.1 Définitions.

DEFINITION 1.2.1.1. – *Soient $U_1, U_2, ..., U_k, W$ des espaces vectoriels sur un corps K. L'application $f : U_1 \times U_2 ... \times U_k \longrightarrow W$ est dite k-linéaire si $f(x_1, x_2, ..., x_k)$ est linéaire par rapport à chacune des variables.*

1. Si $k = 1$, on dira que f est *linéaire*,
2. si $k = 2$, on dira que f est *bilinéaire*.

L'ensemble des applications *k-linéaires* forme un espace vectoriel sur K avec l'addition et le produit (par un scalaire) usuels pour les applications.

DEFINITION 1.2.1.2. – *Soient $U_1, U_2, ..., U_k$, k espaces vectoriels. On appelle structure k-linéaire sur les U_i, $(i = 1, 2, ..., k)$ tout couple (W, f) où*

- *W est un espace vectoriel sur K,*
- *$f : U_1 \times U_2 ... \times U_k \longrightarrow W$ est une application $k-$linéaire.*

DEFINITION. – *Avec les notations précédentes, on appelle produit tensoriel des U_i, $(i = 1, 2, ..., k)$ une structure k-linéaire (W_0, f_0) qui soit initiale, dans le sens suivant: pour toute structure k-linéaire (W, f) sur les U_i, il existe une et une seule application linéaire $\theta : W_0 \longrightarrow W$ qui rend commutatif le diagramme suivant:*

$$\begin{array}{ccc} U_1 \times U_2 ... \times U_k & \xrightarrow{f_0} & W_0 \\ & f \searrow \quad \swarrow \theta & \\ & W & \end{array}$$

il est clair qu'une telle structure initiale, si elle existe, est unique à isomorphisme près.

1.2.2 Produit tensoriel de deux espaces vectoriels.

Soient U et V deux espaces vectoriels sur K. Nous allons nous intéresser aux applications bilinéaires de $U \times V$ dans un espace vectoriel W sur K.

On considère l'ensemble E des éléments de la forme :

$$\sum_{i \in I} \alpha_i(x_i, y_i),$$

où $\alpha_i \in K$, $(x_i, y_i) \in U \times V$, pour tout $i \in I$ (I *fini*).

Un élément de E peut toujours se mettre sous la forme :

$$\sum_{1 \leq i \leq n} \alpha_i(x_i, y_i)$$

quitte à rajouter des éléments avec des coefficients nuls.

On munit E des deux opérations suivantes

- Addition : $a+b=\sum_{1\leq i\leq n}(\alpha_i+\beta_i)(x_i,y_i)$,
- Multiplication externe : $\lambda a=\sum_{1\leq i\leq n}\lambda(x_i,y_i)$,

avec

$$a=\sum_{1\leq i\leq n}\alpha_i(x_i,y_i),\quad b=\sum_{1\leq i\leq n}\beta_i(x_i,y_i)\quad\text{et}\quad\lambda\in K.$$

Ces deux opérations font de E un espace vectoriel sur K.

Soit N le sous-espace vectoriel de E engendré par les éléments de la forme:

- $(x+x',y)-(x,y)-(x',y)$,
- $(x,y+y')-(x,y)-(x,y')$,
- $(\lambda x,y)-\lambda(x,y)$,
- $(\lambda x,\lambda y)-\lambda(x,y)$.

Nous nous intéressons à l'espace vectorielquotient E/N. Soit le diagramme suivant:

$$\begin{array}{ccc} U\times V & \xrightarrow{\tau} & E/N \\ & {\scriptstyle i}\searrow \quad \nearrow{\scriptstyle \bar{\pi}} & \\ & E & \end{array}$$

où i est l'injection canonique de $U\times V$ dans E, et $\bar{\pi}$ est la projection canonique de E dans E/N.

THEOREME 1.2.2.1. – *$(E/N,\tau)$ est la structure bilinéaire initiale sur U,V.*

Preuve :

a- Le diagramme suivant est commutatif:

$$\begin{array}{ccc} U\times V & \xrightarrow{f} & W \\ & {\scriptstyle i}\searrow \quad \nearrow{\scriptstyle \bar{f}} & \\ & E & \end{array}$$

où (W,f) est une structure bilinéaire sur U,V. i étant l'injection canonique de $U\times V$ sur E. $\bar{f}$ est l'application linéaire unique de E dans W telle que

$$\bar{f}(\sum_{i\in I}\alpha_i(x_i,y_i))=\sum_{i\in I}\alpha_i f(x_i,y_i).$$

b- f étant une application bilinéaire, ceci signifie exactement que l'extension $\bar{f}$ de f à E annule le sous-espace N. D'où la factorisation unique de $\bar{f}$ à travers E/N :

$$\begin{array}{ccc} U\times V & \xrightarrow{f} & W \\ i\downarrow & \bar{f}\nearrow & \uparrow g \\ E & \xrightarrow{\pi} & E/N \end{array}$$

d'où finalement une application linéaire unique g vérifiant

$$f = g \circ \tau, \qquad (\tau = \bar{\pi} \circ i)$$

ce qui montre que $(E/N, \tau)$ est bien la structure bilinéaire initiale sur U, V. ∎

L'espace vectoriel E/N est usuellement appelé *produit tensoriel* de U et V, et noté $U \otimes V$.

Un élément de $U \otimes V$ sera noté $x \otimes y$ et sera appelé *le produit tensoriel de x par y*. Les éléments de $U \otimes V$ sont des sommes de la forme

$$\sum_{i \in I} x_i \otimes y_i, \quad x_i \in U, \ y_i \in V, \quad \text{pour tout } i \in I, \quad I \text{ fini.}$$

Nous avons, d'après la définition de N, les relations suivantes:

$\forall \alpha \in K, \forall x, x' \in U, \ \forall y, y' \in V$

- $(\alpha x) \otimes y = x \otimes (\alpha y) = \alpha(x \otimes y)$,
- $(x + x') \otimes y = x \otimes y + x' \otimes y$,
- $x \otimes (y + y') = x \otimes y + x \otimes y'$.

Si U est de dimension n et de base $(e_i)_{1 \leq i \leq n}$ et V est de dimension m et de base $(f_j)_{1 \leq j \leq m}$, alors $(e_i \otimes f_j)_{1 \leq i \leq n, 1 \leq j \leq m}$ est une base de $U \otimes V$ qui est donc de dimension $n.m$.

1.2.3 Produit tensoriel des applications.

THEOREME 1.2.3.1. – *Soient $f_i : U_i \longrightarrow V_i$ et $g_i : V_i \longrightarrow W_i$ $(i = 1, 2)$ quatre applications linéaires. $(U_i, V_i, W_i, \ i = 1, 2, \ $ sont des espaces vectoriels sur un corps $K)$.*
a- Il existe une unique application linéaire $f : U_1 \otimes U_2 \longrightarrow V_1 \otimes V_2$ et une unique application linéaire $g : V_1 \otimes V_2 \longrightarrow W_1 \otimes W_2$ qui soient "compatibles au produit tensoriel", c'est-à-dire, telles que, pour tout $x_1 \otimes x_2 \in U_1 \otimes U_2$ et $y_1 \otimes y_2 \in V_1 \otimes V_2$,

- $f(x_1 \otimes x_2) = f_1(x_1) \otimes f_2(x_2)$,
- $g(y_1 \otimes y_2) = g_1(y_1) \otimes g_2(y_2)$.

Pour cette raison, on les notera:

$$f = f_1 \otimes f_2 \quad et \quad g = g_1 \otimes g_2.$$

Nous avons donc :

- $(f_1 \otimes f_2)(x_1 \otimes x_2) = f_1(x_1) \otimes f_2(x_2)$,
- $(g_1 \otimes g_2)(y_1 \otimes y_2) = g_1(y_1) \otimes g_2(y_2)$.

b- Nous avons de plus :

$$(g_1 \otimes g_2) \circ (f_1 \otimes f_2) = (g_1 \circ f_1) \otimes (g_2 \otimes f_2).$$

Démonstration. – On part du diagramme suivant

$$\begin{array}{ccccc} U_1 \times U_2 & \xrightarrow{(f_1,f_2)} & V_1 \times V_2 & \xrightarrow{(g_1,g_2)} & W_1 \times W_2 \\ \tau_u \downarrow & & \tau_v \downarrow & & \tau_w \downarrow \\ U_1 \otimes U_2 & \xrightarrow{f} & V_1 \otimes V_2 & \xrightarrow{g} & W_1 \otimes W_2 \\ & & \xrightarrow[h]{} & & \end{array}$$

$(V_1 \otimes V_2, \tau_v \circ (f_1, f_2))$ est une structure bilinéaire sur U_1, U_2. $\tau_v \circ (f_1, f_2)$ se factorise donc à travers τ_u. De même $(W_1 \otimes W_2, \tau_w \circ (g_1, g_2))$ est une structure bilinéaire sur V_1, V_2. Donc $\tau_w \circ (g_1, g_2)$ se factorise à travers τ_v. Enfin, $\tau_w \circ (g_1, g_2) \circ (f_1, f_2)$ se factorise à travers τ_u, puisque $(W_1 \otimes W_2, \tau_w \circ (g_1, g_2) \circ (f_1, f_2))$ est une structure bilinéaire sur U_1, U_2.

D'où l'existence et l'unicité des trois fonctions f, g et h rendant tout le diagramme commutatif.

Nous avons donc, pour tout $x_1 \otimes x_2 \in U_1 \otimes U_2$,

$$\begin{aligned} f(x_1 \otimes x_2) &= f \circ \tau_u(x_1, x_2) \\ &= \tau_v \circ (f_1, f_2)(x_1, x_2) \\ &= \tau_v(f_1(x_1), f_2(x_2)) \\ &= f_1(x_1) \otimes f_2(x_2). \end{aligned}$$

De même, on obtient, pour tout $y_1 \otimes y_2 \in V_1 \otimes V_2$,

$$g(y_1 \otimes y_2) = g_1(y_1) \otimes g_2(y_2).$$

Et enfin, pour tout $x_1 \otimes x_2 \in U_1 \otimes U_2$,

$$h(x_1 \otimes x_2) = g_1 \circ f_1(x_1) \otimes g_2 \circ f_2(x_2).$$

En outre, par l'unicité de h, on a:

$$h = g \circ f.$$

∎

DEFINITION 1.2.3.1. – *L'application linéaire $f : U_1 \otimes U_2 \longrightarrow V_1 \otimes V_2$, du théorème précédent, est appelée produit tensoriel des applications linéaires f_1 et f_2.*

1.2.4 Produit tensoriel de plus de deux espaces vectoriels.

De la même façon que dans le cas de deux espaces vectoriels, on considère k espaces vectoriels $U_1, U_2, ..., U_k$ sur un corps $K (k > 2)$. Soit E l'ensemble des éléments de la forme:

$$\sum_{i \in I} \alpha_i (x_{1i}, x_{2i}, ..., x_{ki}),$$

où les $\alpha_i \in K$ et $x_{ji} \in U_j, \quad j = 1, 2, ..., k$.

E est un espace vectoriel sur K. Soit N le sous-espace vectoriel de E engendré par les éléments de la forme:

$$\left.\begin{array}{l} \bullet\ (x_1,...,x_j+y_j,...,x_k)-(x_1,...,x_j,...,x_k)-(x_1,...,y_j,...,x_k), \\ \bullet\ (x_1,...,\lambda x_j,...,x_k)-\lambda(x_1,...,x_j,...,x_k). \end{array}\right\} \quad j=1,2,...,k.$$

Comme dans le cas de deux espaces vectoriels, nous allons nous intéresser à l'espace vectoriel quotient E/N. Nous avons

$$\begin{array}{ccc} U_1\times U_2...\times U_k & \xrightarrow{\tau} & E/N \\ & i\searrow \quad \nearrow \bar{\pi} & \\ & E & \end{array}$$

et $(E/N,\tau)$ est une structure k-linéaire sur $U_1,U_2,...,U_k$. Nous avons un théorème équivalent au théorème 1.2.2.1.

THEOREME 1.2.4.1. – *$(E/N,\tau)$ est la structure k-linéaire sur U_i, $i=1,...,k$.*

E/N sera encore appelé *produit tensoriel* des espaces vectoriels $U_1,U_2,...,U_k$, et noté $U_1\otimes U_2...\otimes U_k$.

Un élément $x_1\otimes x_2...\otimes x_k$ de $U_1\otimes U_2...\otimes U_k$ sera appelé *produit tensoriel* de $x_1,x_2,...,x_k$.

Comme dans le paragpraphe précédent, si $f_i: U_i \longrightarrow V_i$ est une application linéaire de l'espace vectoriel U_i dans l'espace vectoriel V_i, $i=1,...,k$, alors, on définit

$$f_1\otimes f_2...\otimes f_k: U_1\otimes U_2...\otimes U_k \longrightarrow V_1\otimes V_2...\otimes V_k,$$

par :

$$\forall x_i\in U_i,\quad i=1,2,...,k,$$
$$(f_1\otimes f_2...\otimes f_k)(x_1\otimes x_2...\otimes x_k)=f_1(x_1)\otimes f_2(x_2)...\otimes f_k(x_k).$$

1.2.5 Associativité du produit tensoriel.

1.2.5.1 Cas de trois éléments.

Soient U,V,W et P des espaces vectoriels sur le corps K. Soient $(U\otimes V,\tau_{uv})$ la structure bilinéaire initiale sur U et V, $((U\otimes V)\otimes W,\tau_{uv,w})$ la structure bilinéaire initiale sur $U\otimes V$ et W. On pose alors:

$$\tau=\tau_{uv,w}\circ(\tau_{uv},Id_w),$$

où Id_w est l'application identique de W.

τ est une application trilinéaire de $U\times V\times W$ dans $(U\otimes V)\otimes W$. On montre le théorème suivant.

THEOREME 1.2.5.1.1. – *Avec les notations précédentes, $((U\otimes V)\otimes W,\tau)$ est la structure trilinéaire initiale sur U,V et W.*

Démonstration:

Soit $g: U\times V\times W \longrightarrow P$ une application trilinéaire. On va montrer qu'il existe une et une seule application linéaire $g': (U\otimes V)\otimes W \longrightarrow P$, telle que

$$g=g'\circ\tau.$$

Pour z fixé dans W, soit

$$g_z : U \times V \longrightarrow P,$$
$$(x, y) \longmapsto g(x, y, z).$$

g étant trilinéaire, alors g_z est bilinéaire. Par définition du produit tensoriel $U \otimes V$, $\exists! \theta_z$ telle que le diagramme suivant soit commutatif:

$$\begin{array}{ccc} U \times V & \xrightarrow{g_z} & P \\ & {\scriptstyle \tau_{uv}} \searrow \quad \nearrow {\scriptstyle \theta_z} & \\ & U \otimes V & \end{array}$$

et donc,

$$\forall (x, y) \in U \times V, \qquad g_z(x, y) = g(x, y, z) = \theta_z(x \otimes y).$$

On considère le diagramme suivant :

$$\begin{array}{ccc} U \times V \times W & \xrightarrow{\quad g \quad} & P \\ {\scriptstyle (\tau_{uv}, Id_w)} \downarrow & (i) \quad \exists! \theta \quad (ii) & \uparrow {\scriptstyle \exists! g'} \\ (U \otimes V) \times W & \longrightarrow & (U \otimes V) \otimes W \end{array}$$

et on va montrer qu'il existe une unique application bilinéaire θ qui rend le triangle (i) commutatif et une unique application linéaire g' qui rend le triangle (ii) commutatif.

1 – *Existence et unicité de θ.*

Unicité:

$\forall (x, y, z) \in U \times V \times W,$

$$\begin{array}{ccc} (x, y, z) & \xrightarrow{g} & g(x, y, z) \\ & \searrow \quad \nearrow {\scriptstyle \theta} & \\ & (x \otimes y, z) & \end{array}$$

donc, si θ existe, elle doit forcément vérifier,

$$\theta(x \otimes y, z) = g(x, y, z), \qquad \text{pour tout} \quad (x, y, z) \in U \times V \times W.$$

Donc, si θ existe, elle est unique.

Existence:

On pose, pour tout $(t, z) \in (U \otimes V) \times W$,

$$\theta(t, z) = \theta_z(t).$$

2 – *Bilinéarité de θ.*

◇ Pour z fixé,

$$\theta(t, z) = \theta_z(t), \qquad \text{pour tout} \quad t \in U \otimes V.$$

Or θ_z est une application linéaire de $U \otimes V$ dans P, alors θ est linéaire par rapport à la première variable.

◇ Pour t fixé,
si $t = x \otimes y \in U \otimes V$, alors

$$\theta(x \otimes y, z) = \theta_z(x \otimes y) = g(x, y, z), \qquad \text{pour tout} \quad z \in W.$$

g trilinéaire, donc elle est linéaire par rapport à z pour x et y fixés, donc θ est linéaire pour t fixé.

Si $t \neq x \otimes y$, on sait par ailleurs qu'on peut toujours l'écrire comme somme finie des $x_i \otimes y_i$:

$$t = \sum_{i \in I} x_i \otimes y_i, \qquad x_i \in U,\ y_i \in V,\ I\ fini.$$

Alors

$$\begin{aligned} \theta(t, z) &= \theta_z(\sum_{i \in I} x_i \otimes y_i) \\ &= \sum_{i \in I} \theta_z(x_i \otimes y_i) \\ &= \sum_{i \in I} g(x_i, y_i, z). \end{aligned}$$

Or g est trilinéaire ce qui implique θ linéaire par rapport à z.

θ est bilinéaire de $(U \otimes V) \times W$ dans P. Donc (P, θ) est une structure bilinéaire sur $U \otimes V$ et W. D'après la définition du produit tensoriel $(U \otimes V) \otimes W$, il existe une unique application linéaire $g' : (U \otimes V) \otimes W \longrightarrow P$ qui rend commutatif le triangle (ii).

∎

De manière symétrique, on considère $(V \otimes W, \tau_{vw})$ la structure bilinéaire initiale sur U et $V \otimes W$. On pose alors

$$\tau = \tau_{u,vw} \circ (Id_u, \tau_{vw}),$$

où Id_u est l'application identique de U. On a alors le corollaire suivant.

COROLLAIRE 1.2.5.1.2. – *$(U \otimes (V \otimes W), \tau)$ est la structure trilinéaire sur U, V et W.*

On sait par ailleurs que la structure trilinéaire initiale sur U, V et W est unique à isomorphisme près, donc les trois espaces vectorielsU $\otimes V \otimes W$, $(U \otimes V) \otimes W$ et $U \otimes (V \otimes W)$ sont isomorphes. D'où l'associativité du produit tensoriel.

1.2.5.2 Cas de plus de trois éléments.

Soient $U_1, U_2, ..., U_k$, $(k > 3)$ des espaces vectoriels sur un corps K. Soit r un entier tel que $1 < r < k$. On considère $(U_1 \otimes U_2 ... \otimes U_r, \tau_r)$ (resp. $(U_{r+1} \otimes U_{r+2} ... \otimes U_k, \tau_{k-r})$) la structure r-linéaire (resp. $(k-r)$-linéaire) initiale sur $U_1, U_2, ..., U_r$ (resp. $U_{r+1}, U_{r+2}, ..., U_k$). Soit enfin $((U_1 \otimes U_2 ... \otimes U_r) \otimes (U_{r+1} \otimes U_{r+2} ... \otimes U_k), \sigma)$ le structure bilinéaire initiale sur $U_1 \otimes U_2 ... \otimes U_r$ et $U_{r+1} \otimes U_{r+2} ... \otimes U_k$. On pose :

$$\tau = \sigma \circ (\tau_r, \tau_{k-r})$$

et on a le théorème suivant.

THEOREME 1.2.5.2.1. – *Avec les notations ci-dessus,*

$$((U_1 \otimes U_2 ... \otimes U_r) \otimes (U_{r+1} \otimes U_{r+2} ... \otimes U_k), \tau)$$

est la structure k-linéaire initiale sur $U_1, U_2, ..., U_k$.

Démonstration:
On considère le diagramme suivant :

$$\begin{array}{ccc} U_1, U_2, ..., U_k & \xrightarrow{\quad g \quad} & P \\ {\scriptstyle (\tau_r, \tau_{k-r})}\downarrow \quad (i) \quad \exists!\theta & & (ii) \quad \exists!g' \uparrow \\ (U_1 \otimes ... \otimes U_r) \times (U_{r+1} \otimes ... \otimes U_k) & \xrightarrow[\sigma]{} & (U_1 \otimes ... \otimes U_r) \otimes (U_{r+1} \otimes ... \otimes U_k) \end{array}$$

Il existe une unique application bilinéaire θ qui rend (i) commutatif et donc il existe une unique application linéaire g' qui rend (ii) commutatif.

Donc $((U_1 \otimes U_2 ... \otimes U_r) \otimes (U_{r+1} \otimes U_{r+2} ... \otimes U_k), \tau)$ est la structure k-linéaire initiale sur $U_1, U_2, ..., U_k$. ∎

On en déduit, d'après l'unicité de la structure initiale sur $U_1, U_2, ..., U_k$, que $(U_1 \otimes U_2 ... \otimes U_r) \otimes (U_{r+1} \otimes U_{r+2} ... \otimes U_k)$ est isomorphe à $U_1 \otimes U_2 ... \otimes U_k$.

D'une manière générale: Soit $(I_i)_{1 \leq i \leq p}$ une partition de l'intervalle $[1, k]$ de $\mathbb{N}$. A toute partie $I_i = \{i_1, ..., i_m\}$, on associe l'espace vectoriel $V_i = U_{i_1} \otimes U_{i_2} ... \otimes U_{i_m}$. On obtient donc p espaces vectoriels. Un élément de V_i sera de la forme $x_{i_1} \otimes x_{i_2} ... \otimes x_{i_m}$ où $x_{i_j} \in U_{i_j}$.

On considère (V_i, τ_i) la structure $|I_i|$-linéaire initiale sur U_{i_j}, $\quad i_j \in I_i$, et $(\bigotimes_{1 \leq i \leq p} V_i, \sigma)$ la structure p-linéaire initiale sur V_i, $\quad 1 \leq i \leq p$.

On pose $\tau = \sigma \circ (\tau_1, \tau_2, ..., \tau_i)$ et on a le théorème suivant.

THEOREME 1.2.5.2.2. – $(\bigotimes_{1 \leq i \leq p} V_i, \tau)$ *est la structure k-linéaire sur* $U_1, U_2, ..., U_k$.

De ce théorème et d'après l'unicité, à isomorphisme près, de la structure k-linéaire sur les U_j, $\quad j \in [1, k]$, on déduit que $\bigotimes_{1 \leq i \leq p} V_i$ est isomorphe à $U_1 \otimes U_2 ... \otimes U_k$.

Remarques:
1- Soit U un espace vectoriel sur K. Les espaces vectoriels $K \otimes U$ et $U \otimes K$ sont canoniquement isomorphes à U. En effet,

$$\begin{array}{l} u : K \otimes U \longrightarrow U \quad (\text{resp. } U \otimes K \longrightarrow U) \quad \text{définie par :} \\ \qquad \alpha \otimes x \longmapsto \alpha x \quad (\text{resp. } x \otimes \alpha \longmapsto \alpha x), \end{array}$$

est un isomorphisme canonique.

$$\begin{array}{ccc} K \times U & \xrightarrow{\quad f \quad} & U \\ (\alpha, x) & \longmapsto & \alpha x \\ & {\scriptstyle \tau}\searrow \quad \nearrow {\scriptstyle \exists!u} & \\ & K \otimes U & \end{array}$$

f bilinéaire, $(K \otimes U, \tau)$ est la structure bilinéaire initiale sur K et U. u rend le diagramme commutatif.

Il existe une application linéaire

$$v : U \longrightarrow K \otimes U,$$
$$y \longmapsto 1 \otimes y,$$

telle que

$$u \circ v = Id_U \quad \text{et} \quad v \circ u = Id_{K \otimes U}.$$

Même preuve pour $U \otimes K$ isomorphe à U.

2- Soient U^*, V^* et $(U \otimes V)^*$ les duaux respectifs des espaces vectoriels U, V et $U \otimes V$. Alors $U^* \otimes V^*$ est isomorphe à $(U \otimes V)^*$.

1.2.6 Algèbre tensorielle.

On appelle algèbre tensorielle d'un espace vectoriel U sur un corps K, l'algèbre T définie de la manière suivante:

1- T est l'espace vectoriel gradué :

$$T = T^0 \oplus T^1 ... \oplus T^n \oplus ...,$$

où $T^0 = K$, $T^1 = U$ et $T^n = U^{\otimes n}$ (puissance tensorielle $n^{\text{ième}}$ de U).

2- La multiplication dans T est définie par le produit tensoriel. Si u et v sont des éléments homogènes de degrés respectifs p et q, $u \otimes v$ est un élément homogène de degré $p+q$. Si u et v ne sont pas homogènes, ils sont de la forme:

$$u = \sum_i u_i, \quad \text{où} \quad u_i \in T^i$$
$$et$$
$$v = \sum_j v_j, \quad \text{où} \quad v_j \in T^j,$$

on pose alors

$$u \otimes v = \sum_{i,j} u_i \otimes v_j.$$

L'algèbre tensorielle $T(U)$ de U est une algèbre graduée sur K, associative, à élément unité (mais non commutative). Elle est engendrée par l'unité 1 de K et les éléments de U identifiés à T^1.

$(T(U))^*$, l'algèbre duale de l'algèbre tensorielle $T(U)$, est canoniquement isomorphe à $T(U^*)$, algèbre tensorielle du dual de U, par le produit scalaire:

$$T(U) \times T(U^*) \longrightarrow K$$

$$< f_1 \otimes f_2 ... \otimes f_p | x_1 \otimes x_2 ... \otimes x_p > = < f_1 | x_1 > < f_2 | x_2 > ... < f_p | x_p > .$$

RÉFÉRENCES

N.BOURBAKI. – *Algèbre, Chapitre III.* – Hermann. – 1948.

N.BOURBAKI. – *Groupes et Algèbres de Lie. Chapitre I.* – Hermann. – 1971.

J.E. HUMPHERYS. – *Introduction to Lie Algebras and Representation Theory.* – Springer-Verlag 1972.

G.D.MOSTOV J.H.SAMPSON J.P.MEYER. – *Fundamental structures of algebra.* – International student edition.

PHAM MAU QUAN. – *Introduction à la géométrie des variétés différentiables.* – Dunod Paris. – 1969.

RAPPELS SUR LES SERIES FORMELLES

Introduction.

Dans ce deuxième chapitre, nous avons tenu, dans un premier paragraphe, à rappeler la définition des *séries formelles,* ensuite à définir une topologie sur l'algèbre $K\ll X\gg$ des séries formelles, puis la propriété selon laquelle une série est inversible si, et seulement si, son terme constant l'est ([1]). Dans le second paragraphe, nous avons commencer par rappeler la définition du *produit de mélange* de deux mots puis de deux séries. Ensuite, nous avons présenté un certain nombre de calculs que l'on peut faire sur les séries formelles, en particulier le calcul des simplifiés d'une série, à droite et à gauche, par un polynôme.

2.1 SERIES FORMELLES

2.1.1 Définitions.

Soit X^* le *monoïde libre* engendré par un ensemble fini, non vide X, appelé *alphabet.* Un élément de X^* est appelé *mot,* c'est une suite finie de *lettres* éléments de X).

Le *produit* de deux mots est la concaténation de ceux-ci:

$$\text{si} \quad \left\{ \begin{array}{ll} et & u = x_1x_2...x_p \\ & v = y_1y_2...y_q, \end{array} \right. \quad \text{alors} \quad uv = x_1x_2...x_py_1y_2...y_q.$$

L'élément neutre, le *mot vide,* est noté ε ou 1.

La *longueur* d'un mot w, notée $|w|$, est le nombre de lettres qui le composent. Ainsi, la longueur du mot vide est nulle. Et, pour tout $x \in X$, $|w|_x$ désigne le nombre d'occurences de la lettre x dans le mot w.

Une *série formelle* S est une application de X^* dans un corps K qui à un mot w associe l'élément $S(w)$, noté $< S|w >$, dans K. $< S|w >$ est appelé coefficient du mot w dans la série S. Cette série sera notée

$$S = \sum_{w \in X^*} < S|w > w.$$

Le *support* d'une série S est le langage

$$\text{Supp}(S) = \{ w \in X^* \mid < S|w > \neq 0 \}.$$

On notera $K\ll X\gg$ l'ensemble des séries formelles sur X à coefficients dans le corps K.

L'*addition* est définie sur $K\ll X\gg$ par:

$$< S+T|w > = < S|w > + < T|w >, \quad \text{pour tout } S,T \in K\ll X\gg .$$

Le *produit de Cauchy* est lui défini par:

$$< ST|w > = \sum_{uv=w} < S|u >< T|v >, \quad \text{pour tout } S,T \in K\ll X\gg .$$

La somme ci-dessus est bien finie, puisque l'ensemble $\{(u,v) \mid uv = w\}$ comporte un nombre fini de termes.

Le produit de Cauchy est associatif.

Muni de ces deux opérations $K\ll X\gg$ est un anneau. On définit une opération externe de K sur $K\ll X\gg$ par:

$$(a,S) \in K \times K\ll X\gg \longmapsto aS \in K\ll X\gg$$

où aS est définie par:

$$< aS|w >= a < S|w >, \quad \text{pour tout } w \in X^*.$$

Avec ces trois opérations $K\ll X\gg$ est une *K-algèbre*.

Remarques:

1. X^* s'injecte naturellement dans $K\ll X\gg$, comme sous-monoïde multiplicatif. On identifiera X^* et son image dans $K\ll X\gg$.

2. K s'injecte aussi dans $K\ll X\gg$ comme sous-algèbre. On l'identifiera aussi avec son image.

Un *polynôme* est une série formelle de support fini. L'ensemble des polynômes est noté $K< X >$. C'est une sous-algèbre de $K\ll X\gg$.

Le *degré* d'un polynôme $P \in K< X >$ est défini par:

$$\deg(P) = \begin{cases} -\infty, \text{ si } P = 0, \\ \sup\{|w| \text{ avec } w \in \mathrm{Supp}(P)\}, \text{ si } P \neq 0. \end{cases}$$

2.1.2 Topologie sur $K\ll X\gg$.

DEFINITION 2.1.2.1. – *Soit $S \in K\ll X\gg$. On définit l'ordre de S, que l'on note $\omega(S)$, par:*

$$\omega(S) = \begin{cases} \inf\{\ |w| \mid w \in \mathrm{Supp}(S)\ \}, & \text{si } S \neq 0, \\ +\infty, & \text{si } S = 0. \end{cases}$$

PROPRIETE. – *Soient $S,T \in K\ll X\gg$.* $\omega(S+T) \geq \inf(\omega(S),\omega(T))$.

On définit une *norme* sur $K\ll X\gg$ en posant, pour $S \in K\ll X\gg$,

$$||S|| = 2^{-\omega(S)}.$$

En effet, on a:

1. $\forall S \in K\ll X\gg, \quad ||S|| \geq 0.$
2. $||S|| = 0 \iff 2^{-\omega(S)} = 0 \iff \omega(S) = +\infty \iff S = 0.$
3. $\forall S,T \in K\ll X\gg, \quad \omega(S+T) \geq \inf(\omega(S),\omega(T))$
$$\Longrightarrow \quad ||S+T|| = 2^{-\omega(S+T)} \geq \max(||S||, ||T||).$$

Donc, on a bien une *norme ultramétrique* sur $K\ll X\gg$.

On définit la *distance d* associée à cette norme en posant:

$$\forall S,T \in K\ll X\gg, \quad d(S,T) = ||S-T||.$$

On définit un système de voisinages de zéro en posant, pour tout $k \in \mathbb{N}$,

$$V_k = \{\ S \in K\ll X\gg \ |\ \omega(S) \geq k\ \},$$

soit encore:

$$V_k = \{\ S \in K\ll X\gg \ |\ ||S|| \leq 2^{-k}\ \}.$$

On remarque qu'on peut mettre ces voisinages sous la forme suivante:

$$V_0 = K\ll X\gg$$

et

$$V_k = X^k.K\ll X\gg, \quad \textit{pour } k \geq 1,$$

car

$$V_k = \{\ S \in K\ll X\gg \ |\ \mathrm{Supp}(S) \subseteq X^k.X^*\ \}.$$

Et on a:

$$\bigcap_{k\in\mathbb{N}} V_k = \{0\}.$$

On définit ainsi une *topologie séparée* sur $K\ll X\gg$.

PROPOSITION 2.1.2.1. – *Toute suite de Cauchy est localement fini.*

Démonstration. – Soit $(S_n)_{n\in\mathbb{N}}$ une suite d'éléments de $K\ll X\gg$. On suppose que (S_n) est une suite de Cauchy. Le critère de *Riemann* s'écrit, pour cette suite:

$$\forall \varepsilon > 0, \quad \exists N > 0, \quad \forall p,q \geq N, \quad ||S_p - S_q|| \leq \varepsilon.$$

En particulier, on peut prendre $\varepsilon = 2^{-k}$, $\exists N_\varepsilon$ tel que

$$\forall q \geq N_\varepsilon \quad ||S_{N_\varepsilon} - S_q|| \leq \varepsilon = 2^{-k} \quad \Longrightarrow \quad \omega(S_{N_\varepsilon} - S_q) \geq k,$$

et donc,

$$\forall w \in X^* \ \text{ tel que } \ |w| < k, \quad < S_{N_\varepsilon}|w > = < S_q|w > .$$

Donc à partir de N_ε, la suite (S_n) devient *stationnaire* pour $|w| < k$. Elle est donc localement finie. ∎

L'espace topologique $K\ll X\gg$ est donc *complet.*

La *limite topologique* d'une suite $(S_n)_{n\in\mathbb{N}}$ de $K\ll X\gg$ se calcule point par point. C'est-à-dire:

$$\text{si} \quad L = \lim_{n\to\infty} S_n, \quad \text{alors} \quad \forall w \in X^*, \quad < L|w > = \lim_{n\to\infty} < S_n|w > .$$

2.1.3 Inverse d'une série formelle.

PROPOSITION 2.1.3.1 ([1]). – *Une série $S \in K\ll X\gg$ est inversible si, et seulement si, son terme constant, $< S|\varepsilon >$, l'est dans K.*

Preuve.

Supposons que S soit inversible, alors il existe une série $S' \in K\ll X\gg$ telle que $SS' = S'S = \mathbb{1}$, où $\mathbb{1}$ désigne la série unité. C'est-à-dire, telle que,

$$\forall w \in X^*, \quad < \mathbb{1}|w > = \begin{cases} 1 & \text{si} \quad w = \varepsilon, \\ 0 & \text{si} \quad w \neq \varepsilon. \end{cases}$$

$$SS' = \mathbb{1} \implies \begin{cases} et & < SS'|\varepsilon > = < \mathbb{1}|\varepsilon > = 1 \\ & < SS'|\varepsilon > = < S|\varepsilon >< S'|\varepsilon > \end{cases}$$

de même

$$S'S = \mathbb{1} \implies \begin{cases} et & < S'S|\varepsilon > = < \mathbb{1}|\varepsilon > = 1 \\ & < S'S|\varepsilon > = < S'|\varepsilon >< S|\varepsilon > \end{cases}$$

$\implies$ $< S|\varepsilon >$ est inversible dans K et d'inverse $< S'|\varepsilon >$.

Réciproquement, si l'on suppose $< S|\varepsilon >$ inversible dans K, d'inverse a, on peut alors poser

$$S_1 = aS = 1 - S',$$

où S' est *propre*, c'est à dire $< S'|\varepsilon >= 0$.

La famille $(S'^n)_{n\geq 0}$ est donc localement finie. Elle est alors sommable de somme

$$S'^* = \sum_{n\geq 0} S'^n.$$

On désigne par S'^+ la série :

$$S'^+ = \sum_{n\geq 1} S'^n.$$

On a:

$$\left.\begin{array}{l} et \\ \end{array}\begin{array}{l} S'^* = 1 + S'^+ \\ S'^+ = S'S'^* = S'^*S' \end{array}\right\} \Rightarrow \begin{cases} et & S'^*(1-S') = S'^* - S'^*S' = S'^* - S'^+ = \mathbb{1} \\ & (1-S')S'^* = S'^* - S'S'^* = S'^* - S'^+ = \mathbb{1}. \end{cases}$$

Donc S'^* est l'inverse de $1-S'$ dans $K\ll X\gg$. Donc S_1 est inversible, d'inverse S'^*. Et finalement, S est inversible dans $K\ll X\gg$ d'inverse aS'^*.

∎

2.2 CALCULS SUR LES SERIES FORMELLES

2.2.1 Produit de mélange.

DEFINITION 2.2.1.1. – *On définit le produit de mélange – en anglais "shuffle product"– ([7]) comme étant une application*

$$\text{ш} : X^* \times X^* \longrightarrow K < X >,$$

définie récursivement comme suit :

1. $\forall w \in X^*, \quad \varepsilon \text{ш} w = w \text{ш} \varepsilon = w,$
2. $\forall u, v \in X^*, \quad \forall x, y \in X, \quad (xu)\text{ш}(yv) = x(u\text{ш}(yv)) + y((xu)\text{ш}v).$

C'est une application bilinéaire.

Ce produit est *commutatif* et *associatif.*

Si $w, w' \in X^*$, $w \sqcup w'$ est un polynôme homogène de degré $|w| + |w'|$.

Exemple:
Soit $X = \{x, y\}$, $w = xy \in X^*$, $w' = xy^2 \in X^*$. On veut calculer $w \sqcup w'$. On a

$$\begin{aligned} xy \sqcup xy^2 &= x(y \sqcup xy^2) + x(xy \sqcup y^2), \qquad \text{(d'après (2))} \\ &= xy(\varepsilon \sqcup xy^2) + x^2(y \sqcup y^2) + x^2(y \sqcup y^2) + xy(xy \sqcup y) \end{aligned}$$

$$\begin{aligned} y \sqcup y^2 &= y(\varepsilon \sqcup y^2) + y(y \sqcup y) \\ &= yy^2 + y^2(\varepsilon \sqcup y) + y^2(y \sqcup \varepsilon) \\ &= y^3 + y^2y + y^2y \\ &= 3y^3. \end{aligned}$$

$$\begin{aligned} xy \sqcup y &= x(y \sqcup y) + y(xy \sqcup \varepsilon) \\ &= xy(\varepsilon \sqcup y) + xy(y \sqcup \varepsilon) + yxy \\ &= xy^2 + xy^2 + yxy \\ &= 2xy^2 + yxy \end{aligned}$$

Donc

$$\begin{aligned} w \sqcup w' &= xyxy^2 + 3x^2y^3 + 3x^2y^3 + 2xyxy^2 + xy^2xy \\ &= 6x^2y^3 + 3xyxy^2 + xy^2xy. \end{aligned}$$

Autre définition

Soit $z \in X$. On définit l'opérateur ∂_z pour tout $w \in X^*$, par :

$$\partial_z w = \begin{cases} 0, & \textit{si } w \textit{ ne se termine pas par } z, \\ v, & \textit{si } w = vz. \end{cases}$$

Le produit de mélange est donc tel que, pour tous $u, v \in X^*$,

- $\partial_z(u \sqcup v) = (\partial_z u) \sqcup v + u \sqcup (\partial_z v)$, (*i.e.* ∂_z *est une dérivation pour le produit de mélange*)
- $< u \sqcup v | \varepsilon > = \begin{cases} 0, & \textit{si } uv \neq \varepsilon, \\ 1, & \textit{si } uv = \varepsilon. \end{cases}$

Et ces égalités peuvent aussi être considérées comme une définition du mélange, puisque par le *lemme de reconstruction,*(Voir plus loin) on a:

$$u \sqcup v = ((\partial_z u) \sqcup v)z + (u \sqcup (\partial_z v))z.$$

Remarques:
Soient $x, y \in X$. On peut composer les dérivations ∂_x et ∂_y, mais

1. $\partial_x \partial_y$ n'est pas une dérivation,
2. en général $\partial_x \partial_y \neq \partial_y \partial_x$,
3. le crochet de Lie $[\partial_x, \partial_y] = \partial_x \partial_y - \partial_y \partial_x$ est encore une dérivation comme nous le verrons plus loin.

Exemples:

Soient $x, y \in X$, $x \neq y$. Nous avons

1. $\partial_x\partial_y x = \partial_x\partial_y y = 0$, mais, $\partial_x\partial_y(x \sqcup y) = \partial_x((\partial_y x) \sqcup y + x \sqcup (\partial_y y)) = \partial_x x = 1$.
2. Pour $w = vxy \in X^*$, on a : $\begin{cases} \partial_x\partial_y w = \partial_x(\partial_y w) = \partial_x(vx) = v, \\ \partial_y\partial_x w = \partial_y(\partial_x w) = \partial_y(0) = 0. \end{cases}$ et

Le produit de mélange peut être étendu à $K\ll X\gg$ en posant, par définition, pour tous S et T de $K\ll X\gg$:

$$S \sqcup T = \sum_{u,v \in X^*} < S|u >< T|v > u \sqcup v.$$

En effet,

$$\{ w \in X^* \mid w \in \mathrm{Supp}(u \sqcup v) \}$$

est un ensemble fini, quels que soient $u, v \in X^*$. On rappelle que

$$\mathrm{Supp}(u \sqcup v) = \{ w \in X^* \mid < u \sqcup v | w > \neq 0 \}.$$

Soient $S \in K\ll X\gg$ et $z \in X$.

$$S = \sum_{w \in X^*} < S|w > w$$

et

$$\partial_z S = \sum_{w \in X^*} < S|w > \partial_z w.$$

On vérifie que ∂_z est une ***dérivation*** pour l'algèbre $K\ll X\gg$, muni du produit de mélange. (Voir preuve plus loin).

2.2.2 Calcul des simplifiés d'une série à gauche ou à droite par un polynôme.

DEFINITION 2.2.2.1. – *Soit $S \in K\ll X\gg$. On pose, pour tout $u \in X^*$, $u \sqcup S = \sum_{w \in X^*} < S|wu > w$ (resp. $S \circ u = \sum_{w \in X^*} < S|uw > w$); que l'on appelera le <u>simplifié</u> de S <u>à droite</u> (resp. <u>à gauche</u>) par u.*

En d'autres termes, on a, pour tout $w \in X^*$:

$$< u \sqcup S|w > = < S|uw >,$$
$$< S \circ u|w > = < S|wu > .$$

Remarques:

1. $\forall S \in K\ll X\gg$, $\varepsilon \sqcup S = S \circ \varepsilon = S$.

2. Par définition du support d'une série et d'après la définition 2.2.2.1, on peut voir facilement que :

$$\mathrm{Supp}(u \sqcup S) = \{ w \in X^* \mid wu \in \mathrm{Supp}(S) \}$$
$$\mathrm{Supp}(S \circ u) = \{ w \in X^* \mid uw \in \mathrm{Supp}(S) \}. \quad \textit{et}$$

3. Pour $z \in X$, on a $z \square S = \partial_z S$, mais, on notera que si $u \notin X$, l'application $S \longrightarrow u \square S$ n'est pas une dérivation de l'algèbre de mélange. (exemple $\partial_x \partial_y : w \longmapsto xy \square w$ n'est pas une dérivation).

PROPOSITION 2.2.2.1. – *Soient* $u, v \in X^*$, $S \in K\ll X \gg$. *On a :*

$$u \square (v \square S) = uv \square S \quad \text{et} \quad (S \circ u) \circ v = S \circ uv.$$

En d'autres termse, les simplifiés à droite ‹ $\square$ › définissent une action à gauche de X^* sur $K\ll X \gg$. Les simplifiés à gauche ‹ $\circ$ › définissent une action à droite de X^* sur $K\ll X \gg$.

Preuve:

Nous avons, pour tout mot $w \in X^*$,

$$\begin{aligned} < u \square (v \square S) | w > &= < v \square S | wu > \qquad \text{"définition2.2.2.1"} \\ &= < S | wuv > \qquad " \\ &= < uv \square S | w > \qquad " \end{aligned}$$

D'où

$$u \square (v \square S) = uv \square S.$$

De même,

$$\begin{aligned} < (S \circ u) \circ v | w > &= < S \circ u | vw > \\ &= < S | uvw > \\ &= < S \circ uv | w > . \end{aligned}$$

Donc

$$(S \circ u) \circ v = S \circ uv.$$

∎

PROPOSITION 2.2.2.2. – *Soient* $u, v \in X^*$, $S \in K\ll X \gg$. *On a :*

$$(u \square S) \circ v = u \square (S \circ v).$$

En d'autres termes les actions à gauche ‹ $\square$ › et à droite ‹ $\circ$ › de X^* sur $K\ll X \gg$ commutent.

Preuve:

Nous avons, pour tout mot $w \in X^*$, d'après la définition 2.2.2.1,

$$\begin{aligned} < (u \square S) \circ v | w > &= < u \square S | vw > \\ &= < S | vwu > \\ &= < S \circ v | wu > \\ &= < u \square (S \circ v) | w > . \end{aligned}$$

Donc

$$(u \square S) \circ v = u \square (S \circ v).$$

∎

La K-algèbre, $K\ll X \gg$, des séries formelles sur X à coefficients dans K, s'identifie canoniquement au *dual* de $K<X>$. En effet, soit $S \in K\ll X \gg$ une série formelle.Alors S est une application de X^*

dans K qui associe au mot $w \in X^*$ le coefficient $< S|w > \in K$. On peut prolonger cette application en une application de $K < X >$ dans K en posant, pour tout polynôme $Q = \sum_{w \in X^*} < Q|w > w$ de $K<X>$,

$$S(Q) = \sum_{w \in X^*} < Q|w >< S|w >,$$

que l'on note $< S|Q >$. S est donc un élément du dual de $K<X>$.

Réciproquement, soit f un élément du dual de $K<X>$. f est donc une application linéaire de $K<X>$ dans K :

$$Q = \sum_{w \in X^*} < Q|w > w \longmapsto f(Q) = \sum_{w \in X^*} < Q|w > f(w).$$

On a vu que X^* s'injecte canoniquement dans $K\ll X\gg$. On peut donc associer à f sa restriction à X^* qui est la série formelle

$$\sum_{w \in X^*} = f(w)w.$$

Si l'on prolonge cette application $f : X^* \longrightarrow K$ à $K<X>$ par le procédé ci-dessus, on retrouve bien la forme linéaire f.

∎

$K<X>$ agit canoniquement, à droite et à gauche, sur son dual $K\ll X\gg$, de la manière suivante.

$\forall Q \in K<X>$, on associe à toute série $S \in K\ll X\gg$ la série

$$Q \square S = \sum_{w \in X^*} < S|wQ > w$$

(resp. $S \circ Q = \sum_{w \in X^*} < S|Qw > w$). On dira qu'on a simplifié S à droite (resp. à gauche) par Q.

$$\forall w \in X^*, \quad < S|wQ > = \sum_{v \in X^*} < S|wv >< Q|v >$$
$$\text{et} \quad < S|Qw > = \sum_{u \in X^*} < S|uw >< Q|u > .$$

Cela s'écrit encore:

$$\forall w \in X^*, \quad < Q \square S|w > =< S|wQ >$$
$$\text{et} \quad < S \circ Q|w > =< S|Qw > .$$

PROPRIETES ([10]):

1. $\forall P, Q \in K<X>, \forall S \in K\ll X\gg$, on a $\begin{cases} (PQ) \square S = P \square (Q \square S) \\ \text{et} \quad S \circ (PQ) = (S \circ P) \circ Q. \end{cases}$
2. $\forall P, Q \in K<X>, \forall S \in K\ll X\gg$, on a $(P \square S) \circ Q = P \square (S \circ Q)$.

La propriété (1) signifie que pour simplifier à droite par le produit PQ, on simplifie à droite par Q puis on simplifie le résultat à droite par P. Et symétriquement pour la simplification à gauche.

La propriété (2) signifie que pour simplifier à droite par P puis simplifier le résultat à gauche par Q est équivalent à simplifier d'abord à gauche par Q puis de simplifier le résultat à droite par P.

Ces deux propriétés sont de simples extensions par linéarité des propositions 2.2.2.1 et 2.2.2.2.

Dorénavant, nous nous limiterons à l'étude des propriétés et résultats de l'opération « ▫ » (simplification à droite), puisque nous savons que pour l'opération « ∘ » (simplification à gauche) les résultats sont symétriques.

L'application $S \longrightarrow z \square S$, $z \in X$ et $S \in K\ll X \gg$, est une dérivation pour le produit de mélange. On peut donc, pour $z \in X$ (i.e. une lettre) et $S \in K\ll X \gg$, poser

$$\partial_z S = z \square S,$$

ce qui est justifié par le lemme suivant.

LEMME 2.2.2.1. – *Soit $z \in X$. ∂_z est une dérivation dans $K\ll X \gg$ pour le produit de mélange.*

Preuve:
Soient $S, T \in K\ll X \gg$.

$$\begin{aligned}
\partial_z(S \sqcup T) &= \sum_{u,v \in X^*} < S|u >< T|v > \partial_z(u \sqcup v) \\
&= \sum_{u,v \in X^*} < S|u >< T|v > [(\partial_z u) \sqcup v + u \sqcup (\partial_z v)] \\
&= \sum_{u,v \in X^*} < S|u >< T|v > (\partial_z u) \sqcup v + \sum_{u,v \in X^*} < S|u >< T|v > u \sqcup (\partial_z v) \\
&= (\sum_{u \in X^*} < S|u > (\partial_z u)) \sqcup (\sum_{v \in X^*} < T|v > v) \\
&\qquad + (\sum_{u \in X^*} < S|u > u) \sqcup (\sum_{v \in X^*} < T|v > (\partial_z v)) \\
&= (\partial_z S) \sqcup T + S \sqcup (\partial_z T).
\end{aligned}$$

∎

Ayant les simplifiés de S à droite par les lettres $z \in X$ et le terme constant $< S|\varepsilon >$ de S, on peut reconstruire S. On a le lemme suivant dit *lemme de reconstruction.*

LEMME 2.2.2.2 ([8]). – *Soit $S \in K\ll X \gg$ une série formelle. $< S|\varepsilon >$ le terme constant de S. Alors*

$$S = < S|\varepsilon > \varepsilon + \sum_{z \in X} (\partial_z S) z.$$

LEMME 2.2.2.3. – *Soient D_1 et D_2 deux dérivations pour un produit, noté « . », dans un ensemble E. Alors le crochet de Lie $[D_1, D_2] = D_1 D_2 - D_2 D_1$ est encore une dérivation pour ce produit.*

Preuve:
Soient $x, y \in E$. On a:

$$\begin{aligned}
D_1 D_2(x.y) &= D_1(D_2(x.y)) \\
&= D_1((D_2 x).y + x.(D_2 y)) \\
&= (D_1 D_2 x).y + (D_2 x).(D_1 y) + (D_1 x).(D_2 y) + x.(D_1 D_2 y),
\end{aligned}$$

et, par symétrie,

$$\begin{aligned} D_2D_1(x.y) &= D_2(D_1(x.y)) \\ &= D_2((D_1x).y + x.(D_1y)) \\ &= (D_2D_1x).y + (D_1x).(D_2y) + (D_2x).(D_1y) + x.(D_2D_1y), \end{aligned}$$

donc

$$\begin{aligned} & D_1D_2(x.y) - D_2D_1(x.y) = (D_1D_2x).y + x.(D_1D_2y) - (D_2D_1x).y - x.(D_2D_1y) \\ \Longrightarrow \quad & [D_1, D_2](x.y) = ([D_1, D_2]x).y + x.([D_1, D_2]y). \end{aligned}$$

∎

On sait que, pour $x, y \in X$, ∂_x et ∂_y sont des dérivations dans $K\ll X\gg$ pour le produit de mélange. Donc, d'après le lemme 2.2.2.3, $[\partial_x, \partial_y]$ est une dérivation pour le produit de mélange.

On posera:

$$[\partial_x, \partial_y] = \partial_{[x,y]}.$$

D'une manière plus générale, puisque un polynôme de Lie est obtenu par des crochets de Lie à partir de lettres, si P est un polynôme de Lie, ∂_P est obtenu par des crochets de Lie des ∂_x, pour $x \in X$. Donc ∂_P est une dérivation pour le produit de mélange ce qui justifie cette notation.

PROPOSITION 2.2.2.3. – *Soit $P \in \mathcal{L}ie < X >$, un polynôme de Lie. Soient $S, T \in K\ll X\gg$, deux séries formelles. Alors*

$$P \circ (S \sqcup T) = (P \circ S) \sqcup T + S \sqcup (P \circ T).$$

Preuve:

Il suffit de remarquer que, pour $P \in \mathcal{L}ie < X >$, $P \circ S = \partial_P(S)$.

∎

ELEMENTS DE LIE ET ALGEBRE DE MELANGE D'APRES REE

Introduction.

Le but essentiel de ce chapitre est la démonstration du théorème de REE sur les séries de Lie. On sait que REE avait montré ([9]) qu'une série de CHEN est une *exponentielle de Lie.* Dans ([4]), on a attaché aux entrées (il s'agit des entrées d'un système dynamique non linéaire) continues par morceaux, $u_i : [0,t] \longrightarrow I\!R$, $(i = 1,2,...,n)$, le chemin de $I\!R^{n+1}$:

$$\begin{cases} \xi_0(\tau) = \tau, \\ \xi_i(\tau) = \int_0^\tau u_i(\sigma)d\sigma, \qquad (0 \leq \tau \leq t, \quad i = 1,2,...,n). \end{cases}$$

La série de CHEN de ce chemin est:

$$C_u = \sum_{w \in X^*} (\int_0^t \delta_u w) w$$

où

- $X = \{x_0, x_1, ..., x_n\}$, « alphabet »
- $\int_0^t \delta_u \varepsilon = 1$,
- $\int_0^t \delta_u w = \int_0^t u_{i_1}(\tau)(\int_0^\tau d\xi_{i_2}...d\xi_{i_p})d\tau$, si $w = x_{i_1}x_{i_2}...x_{i_p}$.

Elle sera dite *série de* CHEN des entrées pendant l'intervalle de temps $[0,t]$.

$< C_u|w >= \int_0^t \delta_u w$, est le coefficient du mot w dans la série C_u. Sur ces coefficients nous avons le résultat suivant:

$$\forall w, w' \in X^*, \qquad < C_u|w >< C_u|w' >=< C_u|w \sqcup\!\sqcup w' > .$$

D'autre part, si l'on définit la concaténation de deux entrées u et v comme suit:

$$u \natural v : [0, t_u + t_v] \longrightarrow I\!R^{n+1},$$

telle que,

$$(u \natural v)(t) = \begin{cases} u(t), & \text{si } 0 \leq t \leq t_u, \\ v(t - t_u), & \text{si } t_u \leq t \leq t_u + t_v. \end{cases}$$

alors

$$C_{u \natural v} = C_u C_v.$$

Pour la démonstration du théorème de REE, nous avons introduit un certain nombre d'outils comme les séries formelles sur $M \otimes P$, où M et P sont des *monoïdes à subdivisions finies* avec *longueur.* Dans le dernier paragraphe nous étudions essentiellement les *séries formelles de Lie,* les *représentations adjointes et coadjointes,* le *groupe de Magnus,* l'*exponentielle* et le *logarithme* d'une série formelle, l'*exponentielle d'une série de Lie,* les *séries de Hausdorff* et enfin le calcul de l'*inverse* d'une exponentielle de Lie.

Nous développons systématiquement des techniques qui rejoignent celles déjà introduites par C. Reutenauer[11] pour l'étude du même papier de R. Ree[9].

3.1 SERIES FORMELLES SUR UN PRODUIT.

3.1.1 Définitions et Notations.

DEFINITION 3.1.1.1. – *Un monoïde M est dit à subdivisions finies si, pour tout $m \in M$, l'ensemble $\{ (m_1, m_2) \mid m_1 m_2 = m \}$ est fini.*

On ne considère ici que les monoïdes M à subdivisions finies avec longueurs, c'est-à-dire pour lesquels on peut associer à chaque élément m sa longueur, notée $|m|_M$, qui est un entier et qui vérifie, pour $m, m_1, m_2 \in M$,

1. $m \neq 1 \implies |m|_M \neq 0 \quad (i.e.\ |m|_M = 0 \Rightarrow m = 1)$,
2. $|m_1 m_2|_M = |m_1|_M + |m_2|_M$,
3. $\forall k \in \mathbb{N}, \quad \{m \in M \mid |m|_M = k\}$ *est fini.*

Un monoïde muni d'une longueur vérifiant ces axiomes est à subdivisions finies.

Soient M et P deux monoïdes à subdivisions finies avec longueurs. On notera $|m|_M$ et $|p|_P$ les longueurs de m et p respectivement dans M et P. Le monoïde $M \times P = \{(m,p) \mid m \in M,\ p \in P\}$ est aussi à subdivisions finies avec longueur. On pose, par définition,

$$|(m,p)|_{M\times P} = |m|_M + |p|_P.$$

On considère $K<M>, K<P>$ et $K<M \times P>$ les algèbres de polynômes sur M, P et $M \times P$ respectivement à coefficients dans le corps K. Soit enfin $K<M> \otimes K<P>$ le produit tensoriel des K-espaces vectoriels $K<M>$ et $K<P>$.

Les éléments $m \in M$ (resp. $p \in P$) forment une base de $K<M>$ (resp. $K<P>$).

Soit f l'application bilinéaire de $K<M> \times K<P>$ dans $K<M \times P>$ qui à tout élément $(m,p) \in M \times P \subset K<M> \times K<P>$ associe le monôme $(m,p) \in K<M \times P>$. Si $(K<M> \otimes K<P>, \tau)$ est la *structure bilinéaire initiale* (*cf* CHAPITRE 1) sur $K<M>$ et $K<P>$, alors il existe une unique application linéaire η de $K<M> \otimes K<P>$ dans $K<M \times P>$ telle que le diagramme suivant soit commutatif :

$$\begin{array}{ccc} K<M> \times K<P> & \xrightarrow{\ f\ } & K<M \times P> \\ & {\scriptstyle\tau}\searrow \quad \nearrow{\scriptstyle \exists!\eta} & \\ & K<M> \otimes K<P> & \end{array}$$

Les éléments $(m,p) \in M \times P$ forment une base de $K<M \times P>$. Les éléments $m \otimes p$, $m \in M$ et $p \in P$, forment une base de $K<M> \otimes K<P>$. η envoie la base $(m \otimes p)_{m\in M, p\in P}$ de $K<M> \otimes K<P>$ dans la base $((m,p))_{m\in M, p\in P}$ de $K<M \times P>$, c'est donc un isomorphisme.

En identifiant (m,p) et $m \otimes p$, on pourra noter $K<M \times P> = K<M \otimes P>$, où $M \otimes P = \tau(M \times P)$. Par l'isomorphisme η on a encore :

$$K<M \otimes P> \cong K<M> \otimes K<P>.$$

Un élément $Q \in K<M \otimes P>$ s'écrit :

$$Q = \sum_{m\in M, p\in P} <Q|m \otimes p>\, m \otimes p,$$

avec

$$\text{Supp}(Q) = \{\ m \otimes p \mid\ < Q | m \otimes p > \neq 0\ \}$$

est fini.

On notera $K\ll M \otimes P\gg$ l'algèbre des séries formelles sur $M \otimes P$ à coefficients dans K. Un élément $S \in K\ll M \otimes P\gg$ s'écrira :

$$S = \sum_{m \in M, p \in P} < S | m \otimes p > m \otimes p.$$

On définit l'*addition* et la *multiplication* dans $K\ll M \otimes P\gg$ comme suit:

$$\forall S, T \in K\ll M \otimes P\gg, \qquad \forall m \otimes p \in M \otimes P,$$
$$< S + T | m \otimes p > = < S | m \otimes p > + < T | m \otimes p >,$$
$$< S.T | m \otimes p > = \sum_{\substack{m_1 m_2 = m \\ p_1 p_2 = p}} < S | m_1 \otimes p_1 >< T | m_2 \otimes p_2 > .$$

Cette dernière somme est finie puisque M et P sont à subdivisions finies.

3.1.2 Topologie sur $K\ll M \otimes P\gg$.

- $\forall (m, p) \in M \times P$, on définit $|m \otimes p|_{M \otimes P} = |m|_M + |p|_P$.

- Soit $Q \in K< M \otimes P >$ un polynôme,

$$Q = \sum_{m \in M, p \in P} < Q | m \otimes p > m \otimes p.$$

On définit le *degré* de Q par:

$$\deg(Q) = \begin{cases} -\infty, & si\ \ Q = 0, \\ \max\{\ |m \otimes p|_{M \otimes P} \mid m \otimes p \in \text{Supp}(Q)\ \}, & si\ \ Q \neq 0. \end{cases}$$

- Soit $S \in K\ll M \otimes P\gg$ une série formelle. On définit l'*ordre* de S par:

$$\omega(S) = \begin{cases} +\infty, & si\ \ S = 0, \\ \inf\{\ |m \otimes p|_{M \otimes P} \mid m \otimes p \in \text{Supp}(S)\ \}, & si\ \ S \neq 0. \end{cases}$$

- On définit, pour tout entier $k \geq 1$, l'*idéal* J_k de $K\ll M \otimes P\gg$ en posant :

$$J_k = \{\ T \in K\ll M \otimes P\gg \mid \omega(T) \geq k\ \}.$$

$\forall k, k' \geq 1$, on a :

$$\begin{cases} J_k . J_{k'} \subset J_{k+k'}, \\ J_k + J_{k'} \subset J_{\inf(k,k')}. \end{cases}$$

On a aussi :

$$\bigcap_{k \geq 1} J_k = \{0\}.$$

La famille $(J_k)_{k\geq 1}$ forme alors un système de voisinages de zéro pour une topologie séparée. Cette topologie peut aussi être définie par la norme:

$$||S|| = 2^{-\omega(S)}, \qquad \textit{pour tout } S \in K\ll M \otimes P\gg,$$

ou encore par la métrique associée :

$$d(S,T) = 2^{-\omega(S-T)}, \qquad \textit{pour tout } S,T \in K\ll M \otimes P\gg.$$

Une suite de séries $(S_n)_{n\in\mathbb{N}}$ est de *Cauchy* si, et seulement si, pour tout $m \otimes p \in M \otimes P$, la suite $(< S_n|m \otimes p >)_{n\in\mathbb{N}}$ est stationnaire. Et l'on a :

$$< \lim_{n\to\infty} S_n|m \otimes p >= \lim_{n\to\infty} < S_n|m \otimes p >,$$

pour tout $m \otimes p \in M \otimes P$.

$K\ll M \otimes P\gg$ est donc le *complété séparé* de $K< M \otimes P>$.

Remarque :
Soit j l'injection canonique de $K\ll M\gg \otimes K\ll P\gg$ dans $K\ll M \otimes P\gg$. $\mathrm{Im}(j) \subset K\ll M \otimes P\gg$. En effet, dans $K\ll M\gg \otimes K\ll P\gg$ on n'a que les sommes finies :

$$\sum_{finie} S_i \otimes T_i,$$

et si $m \neq 1$ et $p \neq 1$, alors

$$\sum_i m^i \otimes p^i \in K\ll M \otimes P\gg,$$

mais pas à $K\ll M\gg \otimes K\ll P\gg$.

Par contre, on a :

$$[K\ll M\gg]\ll P\gg \cong K\ll M \otimes P\gg \cong [K\ll P\gg]\ll M\gg .$$

Maintenant, on va prendre $P = X^*$, le monoïde libre sur un alphabet X. On considère donc l'algèbre $K\ll M \otimes X^*\gg$ des séries formelles sur $M \otimes X^*$ à coefficients dans K. ([8])

On définit un opérateur D_z $(z \in X)$ agissant sur les éléments de $M \otimes X^*$ de la façon suivante :

$$\forall m \otimes u \in M \otimes X^*, \qquad D_z(m \otimes u) = m \otimes (\partial_z u),$$

où ∂_z est l'opérateur défini précédemment.

Cet opérateur peut être étendu linéairement à $K\ll M\otimes X^*\gg$ et alors, pour

$$H = \sum_{m\in M, u\in X^*} < H|m\otimes u > m\otimes u,$$

on aura :

$$D_z H = \sum_{m\in M, u\in X^*} < H|m\otimes u > m\otimes(\partial_z u)$$

que l'on peut écrire :

$$D_z H = \sum_{m\in M, v\in X^*} < H|m\otimes vz > m\otimes v.$$

On définit également un produit de mélange sur $M\otimes X^*$, que l'on note $\underline{\sqcup\!\sqcup}$, comme suit :

$$\forall m\otimes u,\ n\otimes v \in M\otimes X^*, \qquad (m\otimes u)\underline{\sqcup\!\sqcup}(n\otimes v) = mn\otimes(u\sqcup\!\sqcup v),$$

où $\sqcup\!\sqcup$ est le produit de mélange définit sur X^* et mn est le produit, dans M, des éléments m et n.

On prolonge ce produit aux éléments de $K\ll M\otimes X^*\gg$ en posant, pour

$$S = \sum_{m\in M, u\in X^*} < S|m\otimes u > m\otimes u \quad et \quad T = \sum_{n\in M, v\in X^*} < T|n\otimes v > n\otimes v,$$

$$S\underline{\sqcup\!\sqcup}T = \sum_{\substack{m,n\in M\\ u,v\in X^*}} < S|m\otimes u >< T|n\otimes v > (mn)\otimes(u\sqcup\!\sqcup v).$$

3.1.3 Spécialisation en un mot u.

On pourra poser, pour $u \in X^*$,

$$H_{|_u} = \sum_{m\in M} < H|m\otimes u > m.$$

H devient ainsi une application de X^* dans $K\ll M\gg$:

$$\begin{array}{ccc} X^* & \xrightarrow{H} & K\ll M\gg \\ u & \longmapsto & H_{|_u} \end{array}$$

$K\ll M\gg$ est l'algèbre des séries formelles sur M à coefficients dans K. Donc H est une série formelle sur X^* à coefficients dans $K\ll M\gg$, c'est-à-dire :

$$H \in [K\ll M\gg]\ll X\gg .$$

En fait, on a un isomorphisme entre $K\ll M\otimes X^*\gg$ et $[K\ll M\gg]\ll X\gg$ et donc

$$H = \sum_{m\in M, u\in X^*} < H|m\otimes u > m\otimes u$$

sera identifiée à

$$H = \sum_{u\in X^*} H_{|_u} u,$$

où

$$H_{|_u} = \sum_{m\in M} < H|m\otimes u > m.$$

Ainsi notre opérateur D_z défini sur $K\ll M\otimes X^*\gg$ n'est rien d'autre que l'opérateur ∂_z défini sur $[K\ll M\gg]\ll X\gg$. Et le produit de mélange $\underline{\sqcup\!\sqcup}$ défini sur $K\ll M\otimes X^*\gg$ n'est rien d'autre que le produit de mélange $\sqcup\!\sqcup$ défini sur $[K\ll M\gg]\ll X\gg$. Enfin la spécialisation de H en $u \in X^*$ n'est rien d'autre que le coefficient du mot u dans la série $H \in [K\ll M\gg]\ll X\gg$.

LEMME 3.1.3.1 ([8]). – *L'opérateur D_z est une dérivation pour le produit $\sqcup\!\sqcup$:*

$$\forall S,T \in K\ll M\otimes X^*\gg, \qquad D_z(S \sqcup\!\sqcup T) = (D_zS)\sqcup\!\sqcup T + S\sqcup\!\sqcup(D_zT).$$

Preuve :
Immédiate d'après les constatations ci-dessus et la définition de $\sqcup\!\sqcup$.

3.1.4 Reconstruction d'une série formelle sur $K\ll M\otimes X^*\gg$.

Soit H un élément de $K\ll M\otimes X^*\gg$. On peut l'écrire :

$$H = \sum_{\substack{m\in M\\ u\in X^*}} < H|m\otimes u > m\otimes u.$$

Le lemme 2.2.2.2, dit lemme de reconstruction, a donc comme traduction immédiate:

$$H = H_{|_\varepsilon}\otimes\varepsilon + \sum_{z\in X} D_zH(1\otimes z).$$

Il est aussi utile de noter que l'on a :

$$\begin{aligned} D_zH &= \sum_{\substack{m\in M\\ u\in X^*}} < H|m\otimes u > m\otimes\partial_z u \\ &= \sum_{\substack{m\in M\\ u\in X^*}} < H|m\otimes uz > m\otimes u. \end{aligned}$$

3.2 POLYNOMES DE LIE ET SERIES DE LIE

3.2.1 Mélange, produit tensoriel et éléments primitifs.

On généralise le cas traité dans ([8]) en prenant

$$M = X^{*\otimes p}, \qquad \text{« } \textit{puissance tensorielle } p^{\text{ième}} \textit{ de } X^* \text{ » .}$$

On définit l'application :

$$\Gamma_n : K<X> \longrightarrow K<X>^{\otimes n},$$

($K<X>^{\otimes n}$ est la puissance tensorielle $n^{\text{ième}}$ de $K<X>$), pour tout $x\in X$, par :

$$\Gamma_n(x) = \sum_{i+j+1=n} 1^{\otimes i}\otimes x\otimes 1^{\otimes j}$$

et telle que Γ_n soit un morphisme, c'est-à-dire,

$$\begin{aligned} &\forall u,v\in X^*, \qquad \Gamma_n(uv) = \Gamma_n(u)\Gamma_n(v) \\ \Longrightarrow\ &\forall u,v\in X^*, \qquad \Gamma_n(uv)\otimes uv = (\Gamma_n(u)\otimes u)(\Gamma_n(v)\otimes v). \qquad (\alpha) \end{aligned}$$

LEMME 3.2.1.1 ([7]). –

$$\sum_{u_1,u_2,\ldots,u_p\in X^*} u_1\otimes u_2\ldots\otimes u_p\otimes(u_1 \sqcup\!\sqcup u_2\ldots \sqcup\!\sqcup u_p) = \sum_{w\in X^*}\Gamma_p(w)\otimes w.$$

Preuve :

Soit R un élément de $K\ll X^{*\otimes p}\otimes X^*\gg$, tel que,

$$R = \sum_{u_1,u_2,\dots,u_p\in X^*} u_1\otimes u_2\dots\otimes u_p\otimes(u_1 \sqcup u_2\dots \sqcup u_p).$$

Alors, pour $z \in X$, on a :

$$\begin{aligned} D_z R &= \sum_{u_1,u_2,\dots,u_p\in X^*} u_1\otimes u_2\dots\otimes u_p\otimes\partial_z(u_1 \sqcup u_2\dots \sqcup u_p) \\ &= \sum_{u_1,u_2,\dots,u_p\in X^*} (u_1 z\otimes u_2\dots\otimes u_p + u_1\otimes u_2 z\dots\otimes u_p \\ &\qquad + \dots + u_1\otimes u_2\dots\otimes u_p z)\otimes(u_1 \sqcup u_2\dots \sqcup u_p) \\ &= R.[z\otimes 1^{\otimes p} + 1\otimes z\otimes 1^{\otimes p} + \dots + 1^{\otimes(p-1)}\otimes z\otimes 1] \\ &= R.(\Gamma_p(z)\otimes z). \end{aligned}$$

D'autre part $R_{|_\varepsilon} = 1^{\otimes p}$. Donc, d'après le lemme de reconstruction,

$$R = 1^{\otimes p}\otimes 1 + R.\sum_{z\in X}(\Gamma_p(z)\otimes 1)(1^{\otimes p}\otimes z),$$

$$R = 1^{\otimes p}\otimes 1 + R.\sum_{z\in X}\Gamma_p(z)\otimes z.$$

On en déduit :

$$R = \left[\sum_{z\in X}\Gamma_p(z)\otimes z\right]^*$$

et, d'après la relation (α),

$$R = \sum_{w\in X^*}\Gamma_p(w)\otimes w.$$

∎

De ce lemme on déduit un certain nombre de résultats :

COROLLAIRE 3.2.1.1 ([9]). – *Pour tout $S \in K\ll X\gg$, on a :*

$$\Gamma_p(S) = \sum_{u_1,u_2,\dots,u_p\in X^*} u_1\otimes u_2\dots\otimes u_p < S|u_1 \sqcup u_2\dots \sqcup u_p > .$$

Preuve :

Pour tout $S \in K\ll X\gg$, on considère $1^{\otimes p}\otimes S$ que l'on applique aux deux membres de l'égalité du lemme 3.2.1.1. On obtient :

$$\sum_{u_1,u_2,\dots,u_p\in X^*} u_1\otimes u_2\dots\otimes u_p < S|u_1 \sqcup u_2\dots \sqcup u_p >= \sum_{w\in X^*}\Gamma_p(w) < S|w > .$$

Comme Γ_p est un morphisme, on a :

$$\Gamma_p(S) = \sum_{w\in X^*} < S|w > \Gamma_p(w).$$

D'où le corollaire.

∎

En particulier, pour $P \in K<X>$, on a :

$$\Gamma_p(P) = \sum_{u_1,u_2,\ldots,u_p \in X^*} u_1 \otimes u_2 \ldots \otimes u_p < P | u_1 \sqcup u_2 \ldots \sqcup u_p > .$$

COROLLAIRE 3.2.1.2 ([10]). – *Soient* $S_1, S_2, \ldots, S_p \in K\ll X \gg$ *des séries et* $P \in K<X>$, *un polynôme. Alors*

$$< S_1 \sqcup S_2 \ldots \sqcup S_p | P > = < S_1 \otimes S_2 \ldots \otimes S_p | \Gamma_p(P) > .$$

Preuve:

$$\begin{aligned} < S_1 \otimes S_2 \ldots \otimes S_p | \Gamma_p(P) > &= \sum_{u_i \in X^*} < P | u_1 \sqcup u_2 \ldots \sqcup u_p > < S_1 | u_1 > < S_2 | u_2 > \ldots < S_p | u_p > \\ &= \sum_{u_i \in X^*} < P | u_1 \sqcup u_2 \ldots \sqcup u_p > < S_1 \sqcup S_2 \ldots \sqcup S_p | u_1 \sqcup u_2 \ldots \sqcup u_p > \\ &= < S_1 \sqcup S_2 \ldots \sqcup S_p | P > . \end{aligned}$$

Le fait que P soit un polynôme assure la convergence de la somme $< S_1 \otimes S_2 \ldots \otimes S_p | \Gamma_p(P) > .$

∎

COROLLAIRE 3.2.1.3 ([9]). – *Soient* $u_1, u_2, \ldots, u_p \in X^*$. *Alors* $\forall w \in X^*$,

$$< u_1 \otimes u_2 \ldots \otimes u_p | \Gamma_p(w) > = < u_1 \sqcup u_2 \ldots \sqcup u_p | w > .$$

Preuve:

On peut poser, pour $u_i, v_i \in X^*, \quad i = 1, 2, \ldots, p,$

$$< u_1 \otimes u_2 \ldots \otimes u_p | v_1 \otimes v_2 \ldots \otimes v_p > = < u_1 | v_1 > < u_2 | v_2 > \ldots < u_p | v_p > .$$

En utilisant les deux expressions de la série R du lemme 3.2.1.1 et en calculant

$$C = < u_1 \otimes u_2 \ldots \otimes u_p \otimes w | R >, \qquad w \in X^*,$$

on obtient d'une part

$$C = < u_1 \otimes u_2 \ldots \otimes u_p | \Gamma_p(w) > < w | w >$$

et d'autre part:

$$C = < u_1 | u_1 > < u_2 | u_2 > \ldots < u_p | u_p > < w | u_1 \sqcup u_2 \ldots \sqcup u_p > .$$

Sachant que

$$< u | v > = \begin{cases} 1, & si\ u = v, \\ 0, & si\ u \neq v, \end{cases}$$

d'où le corollaire.

∎

De ce corollaire on déduit, pour tout $w \in X^*$,

$$\begin{aligned} \Gamma_p(w) &= \sum_{u_i \in X^*} < \Gamma_p(w) | u_1 \otimes u_2 \ldots \otimes u_p > u_1 \otimes u_2 \ldots \otimes u_p \\ &= \sum_{u_i \in X^*} < u_1 \sqcup u_2 \ldots \sqcup u_p | w > u_1 \otimes u_2 \ldots \otimes u_p . \end{aligned}$$

D'où le corollaire suivant.

COROLLAIRE 3.2.1.4 ([7]). –

$$\forall w \in X^*, \qquad \Gamma_p(w) = \sum_{u_i \in X^*} < u_1 \sqcup u_2 ... \sqcup u_p | w > u_1 \otimes u_2 ... \otimes u_p.$$

DEFINITION 3.2.1.1. –

1. *Soit* $m \in M = X^{*\otimes p}$. *m est dit un* <u>*terme linéaire*</u> *s'il est de la forme*
$$m = 1^{\otimes(i-1)} \otimes u \otimes 1^{\otimes(p-i)}, \text{ avec } u \neq \varepsilon.$$
2. *Si* $S \in K\ll M \gg$, *elle est dite* <u>*linéaire*</u> *si*
$$< S | u_1 \otimes u_2 ... \otimes u_p > \neq 0 \implies (\exists! i \in [1,p] \text{ tel que } u_i \neq \varepsilon).$$
Ou encore si Supp(S) *ne contient que les termes linéaires.*
3. $S \in K\ll X \gg$ *est dite* <u>$p-$*primitif*</u> *si* $\Gamma_p(S) = \sum_i 1^{\otimes(i-1)} \otimes S \otimes 1^{\otimes(p-i)}$.

COROLLAIRE 3.2.1.5 (REE). – *Soit* $S \in K\ll X \gg$. *Les trois propriétés suivantes sont équivalentes*

1. S *est* $p-$*primitif.*
2. $\Gamma_p(S)$ *est linéaire.*
3. $< S | u_1 \sqcup u_2 ... \sqcup u_p > \neq 0 \implies (\exists! i \in [1,p] \text{ tel que } u_i \neq \varepsilon).$

Preuve:
(1) $\Rightarrow$ (2) :

$$S \text{ est } p\text{–primitif} \Leftrightarrow \Gamma_p(S) = \sum_i 1^{\otimes(i-1)} \otimes S \otimes 1^{\otimes(p-i)}$$
$$\Rightarrow (< \Gamma_p(S) | u_1 \otimes u_2 ... \otimes u_p > \neq 0 \Rightarrow (\exists! i \in [1,p] \text{ tel que } u_i \neq 0))$$
$$\Leftrightarrow \Gamma_p(S) \text{ est linéaire.}$$

(2) $\Rightarrow$ (3) :

$$\text{le corollaire 3.2.1.1} \Rightarrow \Gamma_p(S) = \sum_{u_i \in X^*} u_1 \otimes u_2 ... \otimes u_p < S | u_1 \sqcup u_2 ... \sqcup u_p >$$

$$\Gamma_p(S) \text{ linéaire} \Leftrightarrow (< \Gamma_p(S) | u_1 \otimes u_2 ... \otimes u_p > \neq 0 \Rightarrow (\exists! \ i \in [1,p] \text{ tel que } u_i \neq \varepsilon))$$
$$\Rightarrow (< S | u_1 \sqcup u_2 ... \sqcup u_p > \neq 0 \Rightarrow (\exists! \ i \in [1,p] \text{ tel que } u_i \neq \varepsilon)).$$

(2) $\Rightarrow$ (1) : le corollaire 3.2.1.1 $\Longrightarrow < \Gamma_p(S) | 1^{\otimes(i-1)} \otimes u \otimes 1^{\otimes(p-i)} > = < S | u >$.

$$\Gamma_p(S) \text{ linéaire} \Leftrightarrow (< \Gamma_p(S) | u_1 \otimes u_2 ... \otimes u_p > \neq 0 \Longrightarrow (\exists! \ i \in [1,p] \text{ tel que } u_i \neq \varepsilon))$$
$$\Leftrightarrow (< \Gamma_p(S) | 1^{\otimes(i-1)} \otimes u \otimes 1^{\otimes(p-i)} > \neq 0, \quad 1 \leq i \leq p, \quad u \in X^*)$$
$$\Rightarrow \Gamma_p(S) = \sum_i \sum_{u \in X^*} < \Gamma_p(S) | 1^{\otimes(i-1)} \otimes u \otimes 1^{\otimes(p-i)} > 1^{\otimes(i-1)} \otimes u \otimes 1^{\otimes(p-i)}$$
$$\Leftrightarrow \Gamma_p(S) = \sum_i \sum_{u \in X^*} < S | u > 1^{\otimes(i-1)} \otimes u \otimes 1^{\otimes(p-i)}$$
$$\Leftrightarrow \Gamma_p(S) = \sum_i 1^{\otimes(i-1)} \otimes u \otimes 1^{\otimes(p-i)}$$
$$\Leftrightarrow S \quad \textit{est } p\text{–primitif.}$$

∎

LEMME 3.2.1.2 ([7]). – *Soit $P \in K<X>$. P est un polynôme de Lie si, et seulement si, P est un polynôme p-primitif. C'est-à-dire :*

$$\Gamma_p(P) = \sum_i 1^{\otimes(i-1)} \otimes P \otimes 1^{\otimes(p-i)}.$$

Preuve :

On pose :

$$E = \{ P \in K<X> \mid \Gamma_p(P) = \sum_{1 \leq n \leq p} 1^{\otimes(n-1)} \otimes P \otimes 1^{\otimes(p-n)} \}.$$

Par définition de Γ_p, $\forall x \in X$, $x \in E$. Donc $E \neq \emptyset$.

Soient $P, Q \in E$. En remarquant que

$$\begin{cases} \Gamma_p([P,Q]) = \Gamma_p(PQ) - \Gamma_p(QP) \\ \text{et} \quad \Gamma_p(PQ) = \Gamma_p(P)\Gamma_p(Q), \end{cases}$$

on montre facilement que :

$$\Gamma_p([P,Q]) = \sum_{1 \leq n \leq p} 1^{\otimes(n-1)} \otimes [P,Q] \otimes 1^{\otimes(p-n)},$$

c'est-à-dire que $[P,Q] \in E$ et par conséquent E est *stable par crochets de Lie.*

Finalement, $\forall P, Q \in E$, $\forall \alpha, \beta \in K$, on a :

$$\begin{aligned} \Gamma_p(\alpha P + \beta Q) &= \alpha\Gamma_p(P) + \beta\Gamma_p(Q), \qquad \text{par linéarité de } \Gamma_p \\ &= \alpha\Big(\sum_{1 \leq n \leq p} 1^{\otimes(n-1)} \otimes P \otimes 1^{\otimes(p-n)}\Big) + \beta\Big(\sum_{1 \leq n \leq p} 1^{\otimes(n-1)} \otimes Q \otimes 1^{\otimes(p-n)}\Big) \\ &= \sum_{1 \leq n \leq p} 1^{\otimes(n-1)} \otimes (\alpha P + \beta Q) \otimes 1^{\otimes(p-n)}, \qquad \text{par linéarité de } \otimes. \end{aligned}$$

Donc $\alpha P + \beta Q \in E$, c'est-à-dire que E est *stable par combinaisons linéaires.* Donc $E \subseteq \mathcal{L}ie\ <X>$, ensemble des polynômes de Lie.

La réciproque est évidente, puisque tout polynôme de Lie peut être obtenu par des crochets de Lie à partir des éléments de X qui sont dans E qui est stable par crochets.

∎

LEMME 3.2.1.3 ([7]). – *Soient $m > n$, $S_1, S_2, \ldots, S_m \in K\ll X \gg$ des séries propres et $P_1, P_2, \ldots, P_n \in \mathcal{L}ie\ <X>$ des polynômes de Lie. Alors*

$$< S_1 \sqcup S_2 \ldots \sqcup S_m | P_1 P_2 \ldots P_n >= 0.$$

Preuve :

Le corollaire 3.2.1.2 permet d'écrire :

$$< S_1 \sqcup S_2 \ldots \sqcup S_m | P_1 P_2 \ldots P_n > = < S_1 \otimes S_2 \ldots \otimes S_m | \Gamma_m(P_1 P_2 \ldots P_n) > .$$

Puisque Γ_m est un morphisme

$$\Gamma_m(P_1 P_2 \ldots P_n) = \prod_{i=1}^{n} \Gamma_m(P_i).$$

Or $m > n$, alors, d'après le lemme 3.2.1.2, on aura dans tous les termes de

$$\prod_{i=1}^{n} \Gamma_m(P_i)$$

un facteur 1 ‹ ε ›. D'autre part les S_i sont propres, donc tous les termes de

$$< S_1 \otimes S_2 ... \otimes S_m | \prod_{i=1}^{n} \Gamma_m(P_i) >$$

seront nuls.

Exemple :

On prend $m = 2$ et $n = 1$. S_1, S_2 propres et $P_1 \in \mathcal{L}ie < X >$.

$$\begin{aligned} < S_1 \sqcup S_2 | P_1 > &=< S_1 \otimes S_2 | \Gamma_2(P_1) > \\ &=< S_1 \otimes S_2 | P_1 \otimes 1 + 1 \otimes P_1 > \\ &=< S_1 | P_1 >< S_2 | 1 > + < S_1 | 1 >< S_2 | P_1 > \\ &= 0 \end{aligned}$$

car $< S_1 | 1 >= 0$ et $< S_2 | 1 >= 0$.

LEMME 3.2.1.4 ([10]). – *Soient* $S_1, S_2, ..., S_n \in K\ll X \gg$, *propres et soient* $P_1, P_2, ..., P_n \in \mathcal{L}ie < X >$. *Alors*

$$< S_1 \sqcup S_2 ... \sqcup S_n | P_1 P_2 ... P_n >= \sum_{\tau \in \sigma_n} < S_1 | P_{\tau(1)} >< S_2 | P_{\tau(2)} > ... < S_n | P_{\tau(n)} > .$$

Preuve :

D'après le corollaire 3.2.1.2, on a :

$$< S_1 \sqcup S_2 ... \sqcup S_n | P_1 P_2 ... P_n >=< S_1 \otimes S_2 ... \otimes S_n | \prod_{i=1}^{n} \Gamma_n(P_i) > .$$

Les $P_i \in \mathcal{L}ie < X >$, $1 \leq i \leq n$. Alors le lemme 3.2.1.2 permet d'écrire :

$$\begin{aligned} \prod_{i=1}^{n} \Gamma_n(P_i) = &(P_1 \otimes 1 ... \otimes 1 + 1 \otimes P_1 \otimes 1 + ... + 1 \otimes 1 ... \otimes P_1). \\ &(P_2 \otimes 1 ... \otimes 1 + 1 \otimes P_2 \otimes 1 + ... + 1 \otimes 1 ... \otimes P_2). \\ &- - - - - - - - - - - - - - - \\ &(P_n \otimes 1 ... \otimes 1 + 1 \otimes P_n \otimes 1 + ... + 1 \otimes 1 ... \otimes P_n) \\ = &\sum_{\tau \in \sigma_n} P_{\tau(1)} \otimes P_{\tau(2)} ... \otimes P_{\tau(n)} + Q, \end{aligned}$$

où Q est une somme de produits tensoriels dont les termes contiennent au moins un facteur égal à 1. Comme les S_i, $1 \leq i \leq n$, sont, par hypothèse, propres, alors

$$\begin{aligned} &< S_1 \otimes S_2 ... \otimes S_n | Q > = 0 \\ \Longrightarrow \quad &< S_1 \sqcup S_2 ... \sqcup S_n | P_1 P_2 ... P_n > =< S_1 \otimes S_2 ... \otimes S_n | \sum_{\tau \in \sigma_n} P_{\tau(1)} \otimes P_{\tau(2)} ... \otimes P_{\tau(n)} > \\ &= \sum_{\tau \in \sigma_n} < S_1 \otimes S_2 ... \otimes S_n | P_{\tau(1)} \otimes P_{\tau(2)} ... \otimes P_{\tau(n)} > . \end{aligned}$$

Ce qui donne finalement

$$< S_1 \omega S_2 ... \omega S_n | P_1 P_2 ... P_n >= \sum_{\tau \in \sigma_n} < S_1 | P_{\tau(1)} >< S_2 | P_{\tau(2)} > ... < S_n | P_{\tau(n)} > .$$

∎

Exemple : On prend $n = 2$, $S_1, S_2 \in K\ll X \gg$, propres, $P_1, P_2 \in \mathcal{L}ie < X >$.

$$\begin{aligned}\Gamma_2(P_1P_2) &= \Gamma_2(P_1)\Gamma_2(P_2)\\ &= (P_1 \otimes 1 + 1 \otimes P_1)(P_2 \otimes 1 + 1 \otimes P_2)\\ &= P_1P_2 \otimes 1 + P_1 \otimes P_2 + P_2 \otimes P_1 + 1 \otimes P_1P_2.\end{aligned}$$

$$\begin{aligned}< S_1 \omega S_2 | P_1P_2 > &=< S_1 \otimes S_2 | \Gamma_2(P_1P_2) >\\ &=< S_1 \otimes S_2 | P_1P_2 \otimes 1 + P_1 \otimes P_2 + P_2 \otimes P_1 + 1 \otimes P_1P_2 >\\ &=< S_1 \otimes S_2 | P_1P_2 \otimes 1 > + < S_1 \otimes S_2 | P_1 \otimes P_2 >\\ &\qquad + < S_1 \otimes S_2 | P_2 \otimes P_1 > + < S_1 \otimes S_2 | 1 \otimes P_1P_2 >\\ &=< S_1 | P_1P_2 >< S_2 | 1 > + < S_1 | P_1 >< S_2 | P_2 >\\ &\qquad + < S_1 | P_2 >< S_2 | P_1 > + < S_1 | 1 >< S_2 | P_1P_2 >\\ &=< S_1 | P_1 >< S_2 | P_2 > + < S_1 | P_2 >< S_2 | P_1 >\\ &= \sum_{\tau \in \sigma_2} < S_1 | P_{\tau(1)} >< S_2 | P_{\tau(2)} > .\end{aligned}$$

Remarque :

On a vu que $K\ll X \gg$ est isomorphe au dual de $K< X >$. Cette dualité peut être étendue en une dualité entre $K\ll X \gg^{\otimes n}$ et $K< X >^{\otimes n}$ qui s'exprime par :

$$< S_1 \otimes S_2 ... \otimes S_n | P_1 \otimes P_2 ... \otimes P_n >=< S_1 | P_1 >< S_2 | P_2 > ... < S_n | P_n > .$$

Cette dualité a été utilisée dans les démonstrations des lemmes 3.2.1.3 et 3.2.1.4.

3.2.2 Séries formelles de Lie.

$\mathcal{L}ie < X >$ désigne la sous-algèbre de $K< X >$ des polynômes de Lie.

DEFINITION 3.2.2.1. – *On appelle <u>série formelle de Lie</u> sur X à coefficients dans K, tout élément de l'algèbre $\mathcal{L}ie\ll X \gg$. Un tel élément S s'écrit de manière unique :*

$$S = \sum_{k \geq 1} P_k,$$

où, pour tout $k \geq 1$, P_k est un polynôme de Lie homogène de degré k.

3.2.2.1 Représentations adjointes et coadjointes.

Soient $\mathcal{L}, \mathcal{G} : X^* \longrightarrow K< X >$, deux applications définies récursivement de la manière suivante :

$$\forall z \in X, \quad \text{et} \quad \forall v \in X^*, \quad \begin{cases} \mathcal{L}(z) = z \quad \text{et} \quad \mathcal{G}(z) = z, \\ \mathcal{L}(vz) = \mathcal{L}(v)z - z\mathcal{L}(v) \quad \text{et} \quad \mathcal{G}(vz) = z\mathcal{G}(v) - \mathcal{G}(v)z. \end{cases}$$

Exemple :

$v = xy \in X^*$,

$$\mathcal{L}(v) = \mathcal{L}(xy) = \mathcal{L}(x)y - y\mathcal{L}(x) = xy - yx,$$
$$\mathcal{G}(v) = \mathcal{G}(xy) = x\mathcal{G}(y) - \mathcal{G}(y)x = xy - yx.$$

Remarque :

D'après la définition de $\mathcal{G}$, on peut écrire, pour tout $z \in X$ et $v \in X^*$,

$$\mathcal{G}(zv) = z\mathcal{G}(v) - \mathcal{G}(v)z = ad_z(\mathcal{G}(v)),$$

où *ad* a été définie dans le premier chapitre.

Cette remarque nous permet d'écrire, pour $w = z_1 z_2 ... z_k \in X^*$,

$$\begin{aligned}\mathcal{G}(w) &= ad_{z_1}(\mathcal{G}(z_2 ... z_k)) \\ &= ad_{z_1} ad_{z_2} ... ad_{z_{k-1}}(z_k).\end{aligned}$$

Si ϱ est une application qui associe à tout mot $v = z_1 z_2 ... z_{k-1}$, $\varrho(v) = (ad_{z_1})(ad_{z_2})...(ad_{z_{k-1}})$, alors

$$\mathcal{G}(vz_k) = (\varrho v)(z_k).$$

DEFINITION 3.2.2.1.1. – *La fonction coad est définie, pour tout $z \in X$, par $z \longmapsto coad_z$ telle que,*

$$\forall w \in X^*, \qquad w(coad_z) = wz - zw = -(ad_z)w,$$

où ad est définie dans le chapitre précédent.

Conséquence.

$$\begin{aligned}\mathcal{L}(z_1 z_2 ... z_k) &= \mathcal{L}(z_1 ... z_{k-1}) z_k - z_k \mathcal{L}(z_1 ... z_{k-1}) \\ &= \mathcal{L}(z_1 ... z_{k-1})(coad_{z_k}) \\ &= - \ - \ - \ - \ - \ - \ - \\ &= z_1 (coad_{z_2}) ... (coad_{z_k}) \\ &= (-1)^{k-1} (ad_{z_k}) ... (ad_{z_2}) z_1.\end{aligned}$$

3.2.2.2 Série de Lie comme éléments primitifs ‹ d'après Ree ›.

On considère les deux séries formelles :

$$F = \sum_{u \in X^*} u \otimes u \quad \text{et} \quad G = \sum_{v \in X^*} \mathcal{L}(v) \otimes v,$$

éléments de $K\ll X^* \otimes X^* \gg$.

Dans $K\ll X^* \otimes X^* \gg$ on définit les deux produits suivants :

$\bullet = . \otimes .$, où . est le produit de concaténation défini sur X^*.

$\underline{ш} = . \otimes ш$, où ш est le produit de mélange défini sur X^*.

Par hypothèse $G_{|_e} = 0 \implies (F \underline{ш} G)_{|_e} = F_{|_e} . G_{|_e} = 0.$

Pour tout $z \in X$, on a :

$$D_z = Id \otimes \partial_z, \qquad \text{où } \partial_z \text{ a été défini précédamment. Ainsi}$$

$$D_z F = \sum_{u \in X^*} u \otimes (\partial_z u) = \sum_{u \in X^*} (uz) \otimes u = \sum_{u \in X^*} (u \otimes u)(z \otimes 1),$$

d'où $D_z F = F \bullet (z \otimes 1)$.

$$\begin{aligned} D_z G &= \sum_{v \in X^*} \mathcal{L}(v) \otimes (\partial_z v) \\ &= \sum_{v \in X^*} \mathcal{L}(vz) \otimes v \\ &= \sum_{v \in X^*} (\mathcal{L}(v)z) \otimes v - \sum_{v \in X^*} (z\mathcal{L}(v)) \otimes v + z \otimes 1 \\ &= G \bullet (z \otimes 1) - (1 \otimes z) \bullet G + z \otimes 1. \end{aligned}$$

On en déduit :

$$\begin{aligned} D_z(F \sqcup\!\sqcup G) &= (D_z F) \sqcup\!\sqcup G + F \sqcup\!\sqcup (D_z G) \\ &= [F \bullet (z \otimes 1)] \sqcup\!\sqcup G + F \sqcup\!\sqcup [G \bullet (z \otimes 1)] - F \sqcup\!\sqcup [(z \otimes 1) \bullet G] + F \sqcup\!\sqcup (z \otimes 1) \\ &= F \sqcup\!\sqcup (z \otimes 1) + F \sqcup\!\sqcup G \bullet (z \otimes 1). \end{aligned}$$

On va reconstruire $F \sqcup\!\sqcup G$ à partir de $F \sqcup\!\sqcup G_{|_\varepsilon}$ et $D_z(F \sqcup\!\sqcup G)$:

$$\begin{aligned} F \sqcup\!\sqcup G &= (F \sqcup\!\sqcup G)_{|_\varepsilon} + \sum_{z \in X} (F \sqcup\!\sqcup (z \otimes 1) + F \sqcup\!\sqcup G \bullet (1 \otimes z)) \bullet (1 \otimes z) \\ &= \sum_{z \in X} F \bullet (z \otimes z) + \sum_{z \in X} (F \sqcup\!\sqcup G) \bullet (z \otimes z). \end{aligned}$$

On pose $H = F \sqcup\!\sqcup G$ et il vient :

$$H = F \bullet \sum_{z \in X} z \otimes z + H \bullet \sum_{z \in X} z \otimes z,$$

soit, en posant $M = \sum_{z \in X} z \otimes z$,

$$H = F \bullet M + H \bullet M.$$

En itérant, on obtient :

$$H = F \bullet M + F \bullet M^{\bullet 2} + F \bullet M^{\bullet 3} + \ldots + F \bullet M^{\bullet k} + \ldots$$

Etant donné que

$$\left.\begin{array}{r} F = \sum_{u \in X^*} u \otimes u \\ \text{et} \\ M = \sum_{z \in X} z \otimes z \end{array}\right\} \implies F \bullet M = M^{\bullet +}, \quad \text{‹ } \textit{étoile propre pour } \bullet \text{ } \text{›}.$$

Par substitution, on obtient :

$$H = M^{\bullet +} + M^{\bullet +} \bullet M + M^{\bullet +} \bullet M \bullet M + \ldots$$

Donc

$$H = \sum_{n \in \mathbb{N}} n M^{\bullet n},$$

et donc $\forall w \in X^*$,

$$< H | w \otimes w >= |w| \quad \Longrightarrow \quad H = \sum_{w \in X^*} |w| w \otimes w \qquad (I).$$

D'autre part

$$H = F \sqcup\!\sqcup G \quad \Longrightarrow \quad H = \sum_{u,v \in X^*} (u.\mathcal{L}(v)) \otimes (u \sqcup\!\sqcup v) \qquad (II).$$

Soit $S \in K\ll X \gg$ une série formelle. En utilisant successivement les deux expressions de H, on obtient :

$$(Id_{X^*} \otimes < S|. >)H = \sum_{w \in X^*} |w| < S|w > w, \qquad \textit{d'après } (I),$$

$$(Id_{X^*} \otimes < S|. >)H = \sum_{u,v \in X^*} u.\mathcal{L}(v) < S|u \sqcup v >, \qquad \textit{d'après } (II).$$

Ceci démontre le lemme de Ree,([9]), suivant.

LEMME 3.2.2.2.1. –

$$\sum_{\substack{u \in X^* \\ v \in X^+}} u \bullet \mathcal{L}(v) \otimes (u \sqcup v) = \sum_{w \in X^*} |w| w \otimes w.$$

THEOREME 3.2.2.2.1 (REE). – *S est un élément primitif si, et seulement si, S est une somme « infinie» de crochets de Lie.*

Preuve :

Supposons que S soit un élément primitif, ce qui est équivalent à

$$\Gamma_2(S) = S \otimes 1 + 1 \otimes S \iff (S \text{ propre et orthogonale aux mélanges propres}).$$

Alors

$$\sum_{u,v \in X^*} u.\mathcal{L}(v) < S|u \sqcup v > = \sum_{v \in X^*} \mathcal{L}(v) < S|v > = \sum_{w \in X^*} |w| < S|w > w.$$

Pour avoir l'expression exacte de S, soit $k > 0$ fixé. Projetons sur les mots de degré égal à k.

$$\sum_{|v|=k} \mathcal{L}(v) < S|v > = \sum_{|w|=k} k < S|w > w.$$

Supposons que $|w|$ soit inversible dans K, on obtient alors :

$$\sum_{|v|=k} \frac{< S|v >}{|v|} \mathcal{L}(v) = \sum_{|w|=k} < S|w > w,$$

d'où

$$S = \sum_{v \in X^*} \frac{< S|v >}{|v|} \mathcal{L}(v)$$

avec v tel que $|v|^{-1} \in K$. Donc S est une somme infinie de crochets de Lie.

La réciproque est immédiate en étendant le lemme 3.2.1.2 aux sommes infinies.

∎

3.3 EXPONENTIELLE DE LIE

3.3.1 Groupe de MAGNUS.

Soit $K_+\ll X\gg = \{T \in K\ll X\gg \mid < T|\varepsilon >= 0\}$. C'est un groupe pour l'addition. On le munit de la topologie induite par celle de $K\ll X\gg$, et $K_+\ll X\gg$ est donc un groupe topologique.

Posons

$$M(X) = \{ S \in K\ll X\gg \mid < S|\varepsilon >= 1 \},$$

on le munit aussi de la topologie induite par celle de $K\ll X\gg$.

LEMME 3.3.1.1. – *$M(X)$ est un groupe topologique pour le produit de Cauchy.*

Preuve :

1. Soit $S = 1 - R$ une série de $M(X)$. On peut calculer son inverse, c'est :

$$S^{-1} = 1 + R + R^2 + ... + R^n + ...$$

(ce qui a un sens car R est propre, donc $\omega(R^n) \geq n$). Donc $M(X)$ est un groupe. On peut alors prendre comme système fondamental de voisinages de 1 les ensembles :

$$V_k = \{ S \in K\ll X\gg \mid \omega(S-1) \geq k \}.$$

2. Montrons que le produit de Cauchy :

$$\begin{array}{ccc} M(X) \times M(X) & \longrightarrow & M(X) \\ (S, S') & \longmapsto & S.S' \end{array}$$

est continu. Soient donc $(S_i)_{i\geq 1}$ une suite de séries de $M(X)$ tendant vers S, et de même $(S'_j)_{j\geq 1}$ une suite de séries tendant vers S' dans $M(X)$.

$\forall k \in \mathbb{N}$, il existe des entiers $i(k)$ et $j(k)$ tels que

$$\begin{array}{rcl} i \geq i(k) & \Longrightarrow & (S - S_i) \in V_k \\ j \geq j(k) & \Longrightarrow & (S' - S'_j) \in V_k. \end{array}$$

Montrons que $SS' - S_iS'_j$ appartient aussi à V_k :

$$\begin{aligned} SS' - S_iS'_j &= SS' - SS'_j + SS'_j - S_iS'_j \\ &= S(S' - S'_j) + (S - S_i)S'_j. \end{aligned}$$

$$\begin{aligned} \omega(SS' - S_iS'_j) &\geq \inf\{ \omega(S(S' - S'_j)), \omega((S - S_i)S'_j) \} \\ &\geq \inf\{ \omega(S' - S'_j), \omega(S - S_i) \}, \end{aligned}$$

car, on a clairement :

$$\omega(AB) \geq \omega(A) \quad \text{et} \quad \omega(AB) \geq \omega(B),$$

et finalement

$$\omega(SS' - S_iS'_j) \geq k,$$

ce qui est le résultat cherché.

3. Reste à montrer que l'application $S \longmapsto S^{-1}$ de $M(X)$ dans lui-même est aussi une application continue.

Soit donc $S = 1 - R \in M(X)$. Alors

$$S^{-1} = 1 + R + R^2 + ... + R^n + ...$$

Soit $(S_i)_{i\geq 1}$ une suite tendant vers S dans $M(X)$. Posons $S_i = 1 - R_i$ et donc

$$S_i{}^{-1} = 1 + R_i + R_i{}^2 + ... + R_i{}^n + ...$$

Pour tout k, soit $i(k)$ un entier tel que,

$$i \geq i(k) \implies S - S_i \in V_k.$$

$$S^{-1} - S_i{}^{-1} = R - R_i + R^2 - R_i{}^2 + ... + R^n - R_i{}^n + ...$$

Montrons le résultat intermédiaire suivant :

$$\begin{aligned} R^{n+1} - R_i{}^{n+1} &= (R - R_i)R^n + R_i(R - R_i)R^{n-1} + ... + R_i{}^n(R - R_i) \\ &= \sum_{a+b=n} R_i{}^a(R - R_i)R^b. \end{aligned}$$

Ceci se prouve par récurrence sur n, en effet :

$$\begin{aligned} R^{n+2} - R_i{}^{n+2} &= RR^{n+1} - R_iR^{n+1} + R_iR^{n+1} - R_iR_i{}^{n+1} \\ &= (R - R_i)R^{n+1} + R_i(R^{n+1} - R_i{}^{n+1}). \end{aligned}$$

On a par hypothèse $\omega(R - R_i) \geq k$, d'où $\omega(R_i{}^a(R - R_i)R^b) \geq k$, d'où encore $\omega(R^{n+1} - R_i{}^{n+1}) \geq k$, et enfin, par sommation,

$$\omega(S^{-1} - S_i{}^{-1}) \geq k$$

La suite $(S_i)_{i\geq 1}$ converge donc vers S^{-1}.

∎

3.3.2 Exponentielle et Logarithme d'une série formelle.

Soient $S = 1 + R \in M(X)$ et $T \in K_+ \ll X \gg$. On pose

$$\log(1 + R) = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{R^n}{n},$$

$$\exp(T) = \sum_{k=0}^{\infty} \frac{T^k}{k!}.$$

Ces deux expressions ont clairement un sens pour la topologie de $K \ll X \gg$, et donc aussi dans $K_+ \ll X \gg$, puisque l'on a :

$$\omega(R^n) \geq n \quad \text{et} \quad \omega(T^k) \geq k.$$

Les calculs classiques montrent alors que

$$\text{et} \quad \begin{aligned} &\log : M(X) \longrightarrow K_+ \ll X \gg, \\ &\exp : K_+ \ll X \gg \longrightarrow M(X) \end{aligned}$$

sont des bijections inverses l'une de l'autre.

Notons W_k les voisinages de zéro dans $K_+ \ll X \gg$

$$W_k = \{ T \in K_+ \ll X \gg \mid \omega(T) \geq k \}.$$

On vérifie aisément que pour tout k, on a :

$$\log(V_k) \subset W_k \quad \text{et} \quad \exp(W_k) \subset V_k.$$

Ce qui s'écrit encore :

$$\exp^{-1}(V_k) \subset W_k \quad \text{et} \quad \log^{-1}(W_k) \subset V_k, \qquad \text{pour tout } k.$$

Ainsi on a démontré la proposition suivante.

PROPOSITION 3.3.2.1 ([2]). – *Les applications*

$$\log\ M(X) \longrightarrow K_+ \ll X \gg, \qquad \text{et} \qquad \exp\ K_+ \ll X \gg \longrightarrow M(X)$$

sont deux homeomorphismes inverses l'un de l'autre.

Remarque :
Soient S et T deux séries de $K_+ \ll X \gg$. Si S et T commutent, (*i.e.* si $ST = TS$) alors les calculs classiques montrent que

$$\exp(S+T) = \exp(S).\exp(T) = \exp(T).\exp(S).$$

Si S et T ne commutent pas, cette égalité est fausse.

Donc, en général,

$$\log(\exp(S).\exp(T)) \neq S+T.$$

En d'autres termes, le produit de Cauchy dans $M(X)$ ne correspond pas en général à la somme dans $K_+ \ll X \gg$.

Mais, en utilisant log et exp, on peut ‹ transporter › dans $K_+ \ll X \gg$ une nouvelle opération que l'on appelle *produit de Hausdorff* et que l'on note ‹ $_H$ › :

$$S \,{}_H T = \log(\exp(S).\exp(T))$$

comme le montre le diagramme commutatif suivant :

$$\begin{array}{ccc}
M(X) \times M(X) & \longrightarrow & M(X) \\
(\exp(S), \exp(T)) & \longmapsto & \exp(S).\exp(T) \\
{\scriptstyle (\exp,\exp)}\uparrow & & \downarrow{\scriptstyle \log} \\
(S,T) & \longmapsto & \log(\exp(S).\exp(T)) \\
K_+ \ll X \gg \times K_+ \ll X \gg & \xrightarrow[H]{} & K_+ \ll X \gg
\end{array}$$

3.3.3 Exponentielle d'une série de Lie.

DEFINITION 3.3.3.1. – *Soit $S \in Lie \ll X \gg \subset K_+ \ll X \gg$ une série de Lie. La série $T = \exp(S)$ est appelée une exponentielle de Lie.*

Le théorème qui suit est l'équivalent du théorème de REE donné au paragraphe 3.2.1 pour les séries de Lie.

THEOREME 3.3.3.1 (REE). – *Soit $T \in K\ll X\gg$ une série formelle. Les trois énoncés suivants sont équivalents :*

1. $< T|. >$ *est un morphisme pour le mélange.*
2. $\Gamma_n(T) = T^{\otimes n}$, *(voir définition de Γ_n au paragraphe 3.2.1).*
3. T *est une exponentielle de Lie.*

Preuve :

(1) $\Longleftrightarrow$ (2) : D'après le lemme 3.2.1.1, on a :

$$\Gamma_n(T) = \sum_{u_1,u_2,\ldots,u_n \in X^*} u_1 \otimes u_2 \ldots \otimes u_n < T|u_1 \sqcup u_2 \ldots \sqcup u_n >,$$

et on a, d'autre part :

$$T^{\otimes n} = \sum_{u_1,u_2,\ldots,u_n \in X^*} < T|u_1 >< T|u_2 > \ldots < T|u_n > u_1 \otimes u_2 \ldots \otimes u_n.$$

Donc

$$\begin{aligned} \Gamma_n(T) = T^{\otimes n} &\Leftrightarrow (\forall u_1, u_2, \ldots, u_n \in X^*, \\ &\qquad < \Gamma_n(T)|u_1 \otimes u_2 \ldots \otimes u_n > = < T^{\otimes n}|u_1 \otimes u_2 \ldots \otimes u_n >) \\ &\Leftrightarrow (\forall u_1, u_2, \ldots, u_n \in X^*, \\ &\qquad < T|u_1 \sqcup u_2 \ldots \sqcup u_n > = < T|u_1 >< T|u_2 > \ldots < T|u_n >) \\ &\Leftrightarrow < T|. > \text{ est un morphisme pour le produit de mélange.} \end{aligned}$$

(2) $\Longrightarrow$ (3) : Soit $T = \exp(S)$ une exponentielle de Lie (*i.e.* $S \in \mathcal{L}ie\ \ll X \gg$). Alors

$$\begin{aligned} \Gamma_n(T) &= \Gamma_n(\exp(S)) \\ &= \exp(\Gamma_n(S)) && \text{(car } \Gamma_n \text{ est un morphisme)} \\ &= \exp\Big(\sum_{i+j+1=n} 1^{\otimes i} \otimes S \otimes 1^{\otimes j}\Big) && \text{(d'après le corollaire 3.2.1.5)} \\ &= \prod_{i+j+1=n} \exp(1^{\otimes i} \otimes S \otimes 1^{\otimes j}) && \text{(car les } 1^{\otimes i} \otimes S \otimes 1^{\otimes j} \text{ commutent)} \\ &= \prod_{i+j+1=n} 1^{\otimes i} \otimes \exp(S) \otimes 1^{\otimes j} \\ &= \prod_{i+j+1=n} 1^{\otimes i} \otimes T \otimes 1^{\otimes j} \\ &= T^{\otimes n}. \end{aligned}$$

(3) $\Longrightarrow$ (2) : On suppose que $\Gamma_n(T) = T^{\otimes n}$ et que $\omega(T^{\otimes n} - 1^{\otimes n}) \geq 1$, alors

$$\Gamma_n(T) = T^{\otimes n} = \prod_{i+j+1=n} 1^{\otimes i} \otimes T \otimes 1^{\otimes j},$$

on applique le logarithme et on obtient :

$$\log(\Gamma_n(T)) = \sum_{i+j+1=n} \log(1^{\otimes i} \otimes T \otimes 1^{\otimes j}) = \sum_{i+j+1=n} 1^{\otimes i} \otimes \log(T) \otimes 1^{\otimes j}.$$

Γ_n étant un morphisme, on peut encore écrire :

$$\Gamma_n(\log(T)) = \sum_{i+j+1=n} 1^{\otimes i} \otimes \log(T) \otimes 1^{\otimes j},$$

donc $\log(T) \in \mathcal{L}ie\ \ll X \gg$ (corollaire 3.2.1.5) et donc T est une exponentielle de Lie.

∎

THEOREME 3.3.3.2 ([2]). – *La restriction de l'application exponentielle de $K_+ \ll X \gg$ à $\mathcal{L}ie \ll X \gg$ est une bijection de $\mathcal{L}ie \ll X \gg$ sur un sous-groupe fermé du groupe $M(X)$.*

Preuve :

On pose :

$$D(X) = \{ T \in M(X) \mid \Gamma_2(T) = T \otimes T \}.$$

Soit $S \in K_+ \ll X \gg$ et $T = \exp(S)$. D'après le théorème 3.3.3.1, $S \in \mathcal{L}ie \ll X \gg$ (*i.e.* T est une exponentielle de Lie) ce qui est encore équivalent à T élément de $D(X)$. Donc l'application exponentielle est une bijection de $\mathcal{L}ie \ll X \gg$ sur $D(X)$.

Montrons que $D(X)$ est un sous-groupe de $M(X)$ et qu'il est fermé.

1. $D(X)$ est stable pour le produit de Cauchy car, Γ_2 est un morphisme et donc, pour tout $T_1, T_2 \in D(X)$,

$$\Gamma_2(T_1.T_2) = \Gamma_2(T_1).\Gamma_2(T_2) = (T_1 \otimes T_1)(T_2 \otimes T_2) = T_1T_2 \otimes T_1T_2,$$

donc $T_1.T_2 \in D(X)$.

2. Soit $T \in D(X) \iff \exists!\ S \in \mathcal{L}ie \ll X \gg$ telle que $T = \exp(S)$

$$\Longrightarrow \quad T^{-1} = (\exp(S))^{-1} = \exp(-S) \qquad (\text{car } S \text{ et } -S \text{ commutent})$$

$$S \in \mathcal{L}ie \ll X \gg \implies -S \in \mathcal{L}ie \ll X \gg \implies T^{-1} \in D(X).$$

Donc $D(X)$ est un sous-groupe de $M(X)$.

3. Soit $(T_i)_{i \geq 1}$ une suite de séries de $D(X)$ convergeant vers $T \in M(X)$. Puisque l'application logarithme est continue, la suite $(\log(T_i))_{i \geq 1}$ converge vers $\log(T)$.

Posons, pour tout $i \geq 1$,

$$S_i = \log(T_i) \qquad \text{et} \qquad S = \log(T).$$

$$\forall k, \quad \exists i(k) \in \mathbb{N}, \text{tel que} \quad i \geq i(k) \implies \omega(S - S_i) \geq k.$$

Notons $\Pi_{\leq k}$ la projection de $K \ll X \gg$ qui à toute série $S \in K \ll X \gg$ associe sa partie polynômiale de degré k. On a donc :

$$i \geq i(k) \implies \Pi_{\leq k}(S) = \Pi_{\leq k}(S_i).$$

Puisque S_i est une série de Lie, on en déduit que $\Pi_{\leq k}(S_i)$, et donc aussi $\Pi_{\leq k}(S)$, est un polynôme de Lie.

Donc quel que soit k, $\Pi_{\leq k}(S)$ est un polynôme de Lie. On en déduit que S est une série de Lie et donc que T est une exponentielle de Lie c'est-à-dire $T \in D(X)$.

3.3.4 Série de Hausdorff.

On a vu au paragraphe 3.3.2 que le *produit de Hausdorff* est défini pour tout $S, T \in \mathcal{L}ie \ll X \gg$ par :

$$S_H T = \log(\exp(S).\exp(T)).$$

Comme $(D(X),.)$ est un groupe (Théorème 3.3.3.2), alors $(\mathcal{L}ie \ll X \gg, {}_H)$ est aussi un groupe appelé *groupe de Hausdorff* (déduit de X relativement à K).

L'élément $H = S_H T$ est appelé *série de Hausdorff* en les indéterminées S et T. On a :

$$\exp(S) = 1 + S + \frac{S^2}{2} + \frac{S^3}{6} + \ldots$$
$$\exp(T) = 1 + T + \frac{T^2}{2} + \frac{T^3}{6} + \ldots$$

$$\exp(S).\exp(T) = 1 + (S+T) + \frac{1}{2}(S^2 + T^2 + 2ST) + \frac{1}{6}(S^3 + T^3 + 3S^2T + 3ST^2) + \ldots$$
$$= 1 + u$$

$$\log(\exp(S).\exp(T)) = \log(1+u) = u - \frac{u^2}{2} + \frac{u^3}{3} - \frac{u^4}{4} + \ldots$$

Après des calculs, on trouve à l'ordre trois (*cf* [2]) :

$$S_H T = S + T + \frac{1}{2}[S,T] + \frac{1}{12}[S,[S,T]] + \frac{1}{12}[T,[T,S]] - \frac{1}{24}[S,[T,[S,T]]] + \ldots$$

3.3.5 Calcul de l'inverse d'une exponentielle de Lie.

Soit T une exponentielle de Lie. On se propose dans ce paragraphe de calculer l'inverse de T.

Soit l'application $\varrho : X^* \longrightarrow X^*$ qui associe à tout mot $n \in X^*$ le mot $(-1)^{|n|}\tilde{n} \in X^*$. Il est évident que $\varrho^2 = Id_{X^*}$ et donc ϱ est une *involution*. On peut prolonger ϱ à $K\ll X \gg$ en posant, pour toute série $S \in K\ll X \gg$,

$$\varrho(S) = \sum_{w \in X^*} < S|w > \varrho(w).$$

PROPOSITION 3.3.5.1. – *La restriction de ϱ à $M(X)$ est l'application passage à l'inverse.*

Preuve :

Soient les séries formelles :

$$E = \sum_{n \in X^*} (-1)^{|n|}\tilde{n} \otimes n \qquad \text{et} \qquad F = \sum_{m \in X^*} m \otimes m$$

éléments de $K\ll X^* \otimes X^* \gg$. Soit $z \in X$. On a alors :

$$D_z E = \sum_{n \in X^*} (-1)^{|n|}\tilde{n} \otimes (\partial_z n) = \sum_{n \in X^*} (-1)^{|nz|}(z \otimes 1)(\tilde{n} \otimes n) = -(z \otimes 1)E.$$
$$D_z F = \sum_{m \in X^*} m \otimes (\partial_z m) = \sum_{m \in X^*} (m \otimes m)(z \otimes 1) = F(z \otimes 1).$$

$$\left.\begin{array}{l} E_{|_\epsilon} = 1 \otimes 1 \\ F_{|_\epsilon} = 1 \otimes 1 \end{array}\right\} \text{ et } \Longrightarrow \quad (F \text{ш} G)_{|_\epsilon} = 1 \otimes 1.$$

$$\begin{aligned} D_z(F \sqcup\!\sqcup E) &= (D_z F) \sqcup\!\sqcup E + F \sqcup\!\sqcup (D_z E) \\ &= F(z\otimes 1) \sqcup\!\sqcup E - F \sqcup\!\sqcup (z\otimes 1)E \\ &= 0. \end{aligned}$$

$\Longrightarrow$ $F \sqcup\!\sqcup E = 1\otimes 1$. (d'après le lemme de reconstruction). Donc

$$1\otimes 1 = \sum_{m,n\in X^*} m(-1)^{|n|}\tilde{n}\otimes(m \sqcup\!\sqcup n).$$

Id_{X^*} étant l'application identique de X^*, alors

$$< Id_{X^*}\otimes T|1\otimes 1 > = < Id_{X^*}\otimes T| \sum_{m,n\in X^*} m(-1)^{|n|}\tilde{n}\otimes(m \sqcup\!\sqcup n) >$$

$$\Longleftrightarrow \qquad 1 = \sum_{m,n\in X^*} m(-1)^{|n|}\tilde{n} < T|m \sqcup\!\sqcup n >$$

$$\Longleftrightarrow \qquad 1 = \sum_{m,n\in X^*} (-1)^{|n|} m\tilde{n} < T|m >< T|n >$$

$$\Longleftrightarrow \qquad 1 = T.\varrho(T) \qquad (\text{avec } T = \sum_{m\in X^*} < T|m > m)$$

et puisque $\varrho(1) = 1$ et ϱ est une involution, alors

$$\varrho(T\varrho(T)) = \varrho(1) \quad \Longleftrightarrow \quad \varrho(T)\varrho^2(T) = 1 \quad \Longleftrightarrow \quad \varrho(T)T = 1.$$

Donc $\varrho(T)$ est l'inverse de T. ∎

ϱ donne une expression de l'inverse de la série T qui est par hypothèse une exponentielle de Lie.

BIBLIOGRAPHIE

[1] J.BERSTEL & C.REUTENAUER. – Les séries rationnelles et leurs langages. – ***Editions Masson***, 1984.

[2] N.BOURBAKI. – *Groupes et Algèbres de Lie. Chapitre II et III.* – Diffusion C.C.L.S. Paris.

[3] D.CLAUDE. – Découplage des systèmes non linéaires, Séries génératrices non commutatives et Algèbres de Lie . – *SIAM J.Control and Optimisation*, vol. **24**, N. 3, 1981.

[4] M.FLIESS. – Fonctionnelles causales non linéaires et indéterminées non commutatives . – *Bull. Soc. Math. France*, **109**, 1981, p. 3 – 40.

[5] M.FLIESS, M.LAMNABHI and F.LAMNABHI-LAGARRIGUE. – An algebraic approch to nonlinear functional expansions . – *IEEE Trans. Circuits And Systems*, **30**, 1983, p. 554 – 570.

[6] C.HESPEL & G.JACOB. – Approximation of nonlinear systems by bilinear ones. – *Algebraic and Geometric Methods in Nonlinear Control Theory*, 1986, p. 511–520.

[7] G. JACOB. – Cours de Calcul formel et séries formelles. DEA-Informatique. – *Université de Lille I*, 1985–1986.

[8] G. JACOB. – Cours de Calcul formel et séries formelles. DEA-Informatique. – *Université de Lille I*, 1986–1987.

[9] R.REE. – Lie elements and an algebra associated with shuffle . – *Ann. of Math*, **68**, 1958, p. 210 – 220.

[10] C.REUTENAUER. – Sur la réalisation locale des séries génératrices de rang de Lie fini. – *Algebraic and Geometric Methods In Nonlinear Control Theory*, 1986, p. 33–43.

[11] C.REUTENAUER. – Theorem of Poincare-Birkhoff-Witt Logarithm and representations of the symetric group whose orders are the stirling Numbers. – *Publication du LITP Paris, Decembre* 1985.

III. UN LOGICIEL CALCULANT LA REALISATION MINIMALE DES SYSTEMES ANALYTIQUES DE SERIE GENERATRICE FINIE

UN LOGICIEL CALCULANT LA REALISATION MINIMALE DES SYSTEMES ANALYTIQUES DE SERIE GENERATRICE FINIE

G. Jacob(†) *& N. Oussous*(†)

IT — 123 Janvier 1988

RESUME

En nous basant essentiellement sur des travaux de M.Fliess et C.Reutenauer portant sur la réalisation des systèmes dynamiques non linéaires, nous avons implémenté un algorithme qui permet de calculer la réalisation locale et minimale des systèmes dynamiques non linéaires de série génératrice finie. Pour implémenter cet algorithme nous avons utilisé comme langage de programmation le système de calcul symbolique MACSYMA sur MULTICS.

ABSTRACT

On the base of works of M.Fliess and C.Reutenauer, concerning local realization of non linear dynamical systems, we present an algorithm allowing to compute the local and minimal realization of finite generating power series. We describe that algorithm in the algebraic computation language MACSYMA.

(†) Laboratoire d'Informatique Fondamentale de Lille.– U.A. 369 du C.N.R.S. – Université de LILLE I. – 59655 VILLENEUVE D'ASCQ CEDEX.

SOMMAIRE

INTRODUCTION

Le but de notre travail est de programmer le calcul de la réalisation locale des systèmes dynamiques non linéaires donnés par leurs *séries génératrices* dans le cas où celles-ci sont *polynômiales* (i.e. de support fini).

Théoriquement, nous savons qu'un système dynamique non linéaire est réalisable si et seulement si son *rang de Lie* est fini (M.Fliess[2] et C.Reutenauer[1]). On montre facilement que le rang de Lie d'une série est égal au rang de sa matrice de *Lie-Hankel.*

Nous supposons donc que notre système est donné par sa série génératrice, que nous supposons polynômiale, et nous cherchons les *champs de vecteurs* et *l'observation* correspondants.

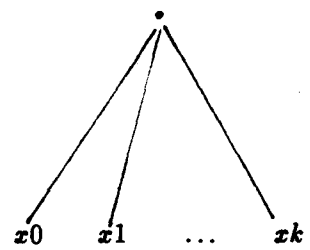
Nous donnons un ensemble de fonctions permettant la manipulation de mots et de polynômes non commutatifs (calcul de résiduels, de mélange et des bases) qui vont nous servir dans notre problème mais qui peuvent aussi être intégrés dans n'importe quel programme (écrit en macsyma). Dans la deuxième partie nous donnons les fonctions qui, en utilisant celles de la première partie, permettent de calculer la réalisation. Dans la troisième partie nous donnons un programme qui permet de vérifier si la réalisation que nous avons calculée par la deuxième partie est correcte ou non.

Dans les annexes A,B,C,D et E nous donnons les sources des programmes écrits en MACSYMA et dans l'annexe F nous donnons quelques exemples.

Nous avons utilisé comme langage de programmation le langage MACSYMA (Macsyma[6]) qui est un langage symbolique écrit en *maclisp* et a été développé au **M.I.T.** Comme tout langage, *macsyma* a ses contraintes et nous sommes alors obligés de les respecter. Nous avons alors choisi certaines représentations qui nous permettent de garder nos notations tout en respectant les conditions du langage, et en profitant de ses possibilités.

→ Pour représenter une lettre de l'alphabet (*son cardinal est choisi par l'utilisateur*), nous utilisons la fonction *macsyma* **<Concat>** et alors $xi = Concat(x, i)$ et pour l'interpréteur xi est un atome (*c'est ce que nous voulons*).

→ Pour la concaténation des mots nous avons utilisé la multiplication non commutative notée " . " . Ainsi le mot $w = x_0 x_1 \cdots x_k$ est entré sous la forme $x0.x1 \cdots xk$. La représentation interne de ce mot est la suivante:



→ L'opérateur principal dans une expression est toujours la racine de l'arbre. Il existe deux fonctions *macsyma* qui permettent d'accéder à chaque terme de l'expression: la première **<Part>** travaille sur la représentation *externe* de l'expression (telle qu'on la voit sur l'écran). La seconde **<Inpart>** travaille sur la représentation *interne* de l'expression. Ces deux fonctions reviennent souvent dans les codes des fonctions que nous avons écrites. Nous donnons ci-dessus deux exemples montrant comment elles fonctionnent.

Exemples:

(*i*) $exp1 = \frac{y}{z^2} + x$

$$Part(exp1, 0) = +,$$
$$Part(exp1, 1) = \frac{y}{z^2},$$
$$Part(exp1, 2) = x,$$
$$Part(exp1, 1, 0) = /,$$
$$Part(exp1, 1, 2, 2) = 2,$$
$$\ldots$$

(*ii*) $exp2 = x + y + w.z$

$$Inpart(exp2, 0) = +,$$
$$Inpart(exp2, 3) = w.z,$$
$$Inpart(exp2, 3, 2) = z,$$
$$Part(exp2, 1, 2) = z,$$

En tenant compte des conditions imposées par *macsyma*, la grammaire qui génère les polynômes non commutatifs a été modifiée en la suivante:

$$< polynome > ::= < monome > \mid < monome > + < polynome >,$$
$$< monome > ::= < mot > \mid < scalaire > * < mot >,$$
$$< mot > ::= 1 \mid < lettre > \mid < lettre > . < mot > \mid < mot >^{<<entier>>},$$
$$< lettre > ::= x0 \mid x1 \mid \ldots \mid x(m-1),$$
$$< scalaire > ::= element\ de\ R,$$
$$< entier > ::= element\ de\ N.$$

Pour représenter un *mot de Lie* nous avons utilisé les listes: ceci lui donne une représentation naturelle à l'affichage, et d'autre part la manipulation des listes est très aisée sous *macsyma*.

APERCU THEORIQUE SUR LA REALISATION

On considère un système de la forme

$$(I) \qquad \begin{cases} \dot{q}(t) = \sum_{i=0}^{m} u_i(t) A_i(q) & \textit{avec } u_0(t) \equiv 1 \textit{ pour tout } t, \\ y(t) = h(q(t)). \end{cases}$$

où

$q \in Q$, Q une variété **R**-analytique connexe de dimension n,
A_i, $0 \leq i \leq m$, sont des champs de vecteurs analytiques,
h, fonction de sortie, une fonction analytique réelle,
Les A_i, et h sont définis dans un voisinage de $q(0)$, où $q(0)$ est donné,
u_i, $0 \leq i \leq m$, sont des fonctions réelles continues par morceaux. On posera par la suite $u = (u_0, u_1, ..., u_k)$.

On considère l'homomorphisme Y de X^* dans l'ensemble des opérateurs différentiels qui à tout mot $w \in X^*$, ($X = \{x_0, x_1, ..., x_{m-1}\}$) associe l'opérateur différentiel $Y(w)$ noté Y_w.
$\forall x_i \in X$, Y_{x_i} est un champ de vecteurs formel (convergent).
Pour des entrées et le temps suffisamment petits, la sortie est donnée par la formule de **Peano-Baker**(M.Fliess[1]):

$$y(t) = \sum_{w \in X^*} (Y_w \circ h)|_{q(0)} \int_0^t \delta_u \tilde{w}$$

où $|_{q(0)}$ signifie l'évaluation en $q(0)$ et $\int_0^t \delta_u \tilde{w}$ est l'intégrale itérée relative à l'entrée u pendant l'intervalle de temps $[0, t]$ (M.Fliess[1]).

D'après M. Fliess[2], la série génératrice associée à y est une série formelle en les variables non commutatives $x_0, x_1, ..., x_n$:

$$g = \sum_{w \in X^*} (Y_w \circ h)|_{q(0)} w$$

Le comportement Entrée/Sortie du système (I) est entièrement défini par sa série génératrice g.

$< g|w >$ étant le coefficient du mot w dans la série g, $y(t)$ peut s'écrire:

$$y(t) = \sum_{w \in X^*} < g|w > \int_0^t \delta_u \tilde{w}$$

g est une série formelle sur les variables non commutatives $x_0, x_1, ..., x_n$

$$g = \sum_{w \in X^*} < g|w > w$$

Soit $\mathcal{L}ie< X >$ l'algèbre de Lie libre engendrée par X et dans laquelle le crochet de Lie est défini par

$$[x_i, x_j] = x_i x_j - x_j x_i$$

On définit le rang de Lie d'une série g (M.Fliess[2] et C.Reutenauer[7]) par

$$RL(g) = \dim \mathcal{L}ie < X > \circ g = \dim\{ P \circ g \mid P \in \mathcal{L}ie < X > \}$$

Un système de type (I) est localement réalisable si et seulement si le rang de Lie de sa série génératrice, $RL(g)$, est fini. La réalisation locale est unique à isomorphisme près.

On montre facilement que $RL(g) = rang(MatLieHankel)$, où $MatLieHankel$ est la matrice de Lie-Hankel (*) associée à g.

Après le calcul du rang de Lie de la série génératrice du système donné, et après s'être assuré que celui-ci est fini et donc que le système est localement réalisable, on calcule les coordonnées locales puis les champs de vecteurs et la fonction d'observation.

Les coordonnées locales $(Z_i)_{1 \leq i \leq d}$ (d étant le rang de Lie de g) sont définies par C.Reutenauer (C.Reutenauer[7]) comme étant des séries vérifiant

$$\begin{cases} < Z_i | P_j > = \delta_{ij} \\ < Z_i | P^\alpha > = 0 \ si \ |\alpha| \geq 2 \ ou \ |\alpha| = 0 \end{cases}$$

où si $\alpha = (j_1, j_2, ..., j_k)$, alors

$$P^\alpha = P_1^{j_1} P_2^{j_2} ... P_k^{j_k},$$

$$et \quad |\alpha| = \sum_{i=1}^{k} j_i$$

$(P_j)_{j \geq 1}$ est une base totalement ordonnée de l'algèbre des polynômes de Lie.

Ayant les coordonnées locales, on calcule les champs de vecteurs comme suit

Soit $X = \{x_0, x_1, ..., x_{m-1}\}$ l'alphabet

$$\forall i, \ 0 \leq i \leq m-1, \ x_i \longrightarrow X_i = \sum_{j=1}^{d} Trad(x_i \circ Z_j) \frac{\partial}{\partial z_j}$$

où $Trad(x_i \circ Z_j)$ consiste à exprimer le simplifié $x_i \circ Z_j$ en fonction des mélanges des $(Z_i)_{1 \leq i \leq d}$ puis à transformer le produit de mélange en le produit commutatif usuel et enfin à remplacer les Z_j par les z_j.

Pour la fonction d'observation, on prend $h = g$, où la série g est exprimée en fonction des mélanges des $(Z_i)_{1 \leq i \leq d}$.

(*) Pour la définition voir dans la partie pratique.

OUTILS INDISPENSABLES

1. RESIDUELS A GAUCHE ET A DROITE.

Tous les codes en *macsyma* des fonctions qui seront décrites dans ce paragraphe seront donnés dans l'ANNEXE B.

1.1. Simplification à gauche d'un polynôme par une lettre.

On définit le simplifié (ou résiduel) d'un mot $w \in X^*$ par une lettre $x \in X$ comme suit:

$$(1) \qquad w \circ x = (yv) \circ x = \begin{cases} 0 & si \quad y \neq x, \\ v & si \quad y = x. \end{cases} \qquad y \in X,$$
$$1 \circ x = 0 \; et \; x \circ 1 = x.$$

Nous avons alors la propriété suivante:

Soient $x, y \in X$ et $w \in X^*$. Alors $(w \circ x) \circ y = w \circ (xy)$. Ce qui permettra de calculer le simplifié d'un mot (resp. un polynôme) par un autre mot (resp. par un mot ou par un polynôme).

$$S = \sum_{w \in X^*} < S|w > w, \qquad x \in X, \qquad S \circ x = \sum_{w \in X^*} < S|w > w \circ x.$$

On distingue les cas particuliers suivants:

- → Si S est un nombre alors $S \circ x = 0$.
- → Si S est une lettre alors

$$S \circ x = \begin{cases} 0 & si \quad S \neq x, \\ 1 & si \quad sinon. \end{cases}$$

- → Si $S = w$ est un mot non vide, alors (1).
- → Si $S = a * w$ est un monôme, alors $S \circ x = a * (w \circ x)$.
- → Si S est une puissance $S = w^{<n>}$, et $w \neq 1$ alors $S \circ x = (w \circ x) w^{<n-1>}$.

La fonction qui calcule le simplifié d'une série par une lettre, appelée **<SimGauLet>**, est une fonction récursive.

1.2. Simplification à gauche d'un polynôme par un autre polynôme.

Soit à calculer le simplifié du polynôme S par le polynôme P. Alors il y a plusieurs cas particuliers à traiter avant de voir le cas général.

→ Si P est un scalaire, alors $S \circ P = P * S$
→ Si P est la lettre x, alors $S \circ P = S \circ x$ (voir ci-dessus).
→ Si $P = yv$ avec y une lettre, alors $S \circ P = (S \circ y) \circ v$.
→ Si $P = a * w$ est un monôme, alors $S \circ P = a * (S \circ w)$.
→ Si $P = w^{<n>}$ est une puissance, alors $S \circ P = (S \circ w) \circ w^{<n-1>}$.
→ Si

$$P = \sum_{w \in X^*} < P|w > w, \qquad S \circ P = \sum_{w \in X^*} S \circ (< P|w > w) = \sum_{w \in X^*} < P|w > S \circ w.$$

Nous avons écrit une fonction **<SimGauPol>** qui utilise la fonction **<SimGauLet>** et permet de calculer le simplifié d'un polynôme par un autre.

1.3. Simplification à gauche d'un polynôme par un polynôme de Lie.

Un mot de Lie est soit une lettre de l'alphabet soit un crochet de Lie de la forme : $[M, N]$ où M et N sont des mots de Lie. Sachant que

$$[M, N] = MN - NM$$

et vu la définition récursive du mot de Lie, nous définissons une fonction **<SimGauML>** récursive qui permet de calculer le simplifié d'un polynôme par un mot en utilisant la fonction **<SimGauLet>** qui simplifie un polynôme par une lettre.

Un polynôme de Lie est une combinaison linéaire de monômes de Lie; c'est donc un élément du langage généré par la grammaire suivante:

$< polynome\ de\ Lie > ::= < monome\ de\ Lie > | < monome\ de\ Lie > + < polynome\ de\ Lie >,$
$< monome\ de\ Lie > ::= < mot\ de\ Lie > | < coefficient > * < mot\ de\ Lie >,$
$< mot\ de\ Lie > ::= < lettre > | [< mot\ de\ Lie >, < mot\ de\ Lie >],$
$< lettre > ::= element\ de\ l'alphabet,$
$< coefficient > ::= element\ du\ corps\ R.$

La fonction **<SimGauLie>** qui permet de simplifier un polynôme par un polynôme de Lie utilise les deux fonctions **<SimGauLet>** et **<SimGauML>** précédentes.

1.4. Simplification à droite d'un polynôme par une lettre.

De manière tout à fait symétrique à la simplification à gauche, nous avons :

Soient $v, w \in X^*$, $x, y \in X$: si $w = vy$, alors

$$x \square w = x \square (vy) = \begin{cases} v & si \quad y = x, \\ 0 & si \quad y \neq x \end{cases},$$

$x \square 1 = 0$ *et* $1 \square x = x$.

Soit $S \in R \ll X \gg$, une série, $S = \sum_{w \in X^*} < S|w > w$,

$$x \circ S = \sum_{w \in X^*} < S|w > x \circ w$$

$$x \circ (y \circ w) = (xy) \circ w, \quad et \quad x \circ (y \circ S) = (xy) \circ S$$

La fonction **<SimDroLet>** qui permet de calculer le simplifié d'un polynôme à droite par une lettre est une fonction récursive symétrique de **<SimGauLet>**.

1.5. Simplification à droite d'un polynôme par un autre polynôme.

Soient S, P deux polynômes, $P = \sum_{w \in X^*} < P|w > w$, alors

$$P \circ S = \sum_{w \in X^*} < P|w > w \circ S.$$

Nous utilisons la fonction **<SimDroLet>** pour construire la fonction **<SimDroPol>** qui permet de calculer le simplifié d'un polynôme par un autre.

2. PRODUIT DE MELANGE.

Tous les codes en *macsyma* des fonctions qui seront données dans ce paragraphe se trouvent dans l'ANNEXE C.

Un polynôme est un élément de $R < X >$. En tenant compte du langage *macsyma*, un polynôme est un élément du langage engendré par la grammaire donnée dans l'introduction.

Avant de calculer le mélange de deux mots (resp. monômes, polynômes) nous testons si ces deux mots (resp. monômes, polynômes) respectent ladite grammaire. Pour cela nous avons écrit les trois fonctions **<EstMot>**, **<EstMon>** et **<EstPol>**.

2.1. Produit de mélange de deux mots.

Le produit de mélange de deux mots est défini récursivement (M.Fliess[1]) comme suit:

$$\begin{cases} \forall w \in X^*, & w \sqcup 1 = 1 \sqcup w = w, \\ \forall u, v \in X^*, \quad \forall x, y \in X, & (xu) \sqcup (yv) = x\,(u \sqcup (yv)) + y\,((xu) \sqcup v). \end{cases}$$

1 est le mot vide.

Donc, pour calculer le mélange de deux mots *non vides* nous aurons besoin de les décomposer chacun en une *lettre* concaténée avec le reste du mot d'où l'intérêt de la fonction **<Decomp>**. La fonction qui permet de calculer le mélange de deux mots est appelée **<MelMot>**.

2.2. Produit de mélange de deux monômes.

On peut étendre le produit de mélange aux monômes comme suit:

$$\forall\, u,v \in X^*, \qquad \forall\, a,b \in R \qquad (a * u) \sqcup (b * v) = ab * (u \sqcup v).$$

La fonction **<MelMon>** qui permet de calculer le mélange de deux monômes utilise les fonctions **<EstMot>**, **<EstMon>** et **<MelMot>**.

2.3. Produit de mélange de deux polynômes.

Le produit de mélange est étendu aux polynômes, et aux séries comme suit:

$\forall\, S,T \in R \ll X \gg$ des séries.

$$S = \sum_{u \in X^*} < S|u > u, \qquad T = \sum_{v \in X^*} < T|v > v.$$

$$S \sqcup T = \sum_{u,v \in X^*} < S|u >< T|v > u \sqcup v.$$

La fonction **<MelPol>** qui calcule le mélange de deux polynômes utilise les fonctions **<EstPol>**, **<EstMon>** et **<MelMon>**.

3. CONSTRUCTION DES BASES.

Les codes en *macsyma* des fonctions qui seront définies dans ce paragraphe seront donnés dans l'ANNEXE C.

3.1. Base de $R < X >$.

$R < X >$ est l'algèbre des polynômes non commutatifs sur l'alphabet $X = \{x_0, x_1, \ldots, x_{m-1}\}$ et à coefficients dans le corps R. X^*, monoïde libre sur X, ensemble des mots sur X, est la base canonique de $R < X >$. Pour construire cette base nous nous sommes inspirés d'un codage utilisé par G. JACOB et C. HESPEL dans [4]. Nous construisons les éléments de ladite base jusqu'à l'ordre voulu et nous les mettons en même temps dans un tableau **<b>** et dans une liste **<lis>**, pour usages ultérieurs.

La fonction **<BaseDeKX>** qui construit la base de $R < X >$ prend en argument le cardinal de l'alphabet X et le degré jusqu'auquel on veut aller. Nous traitons le cas particulier où X est réduit à une seule lettre en premier et le cas général ensuite.

3.2. Base de $Lie < X >$.

$Lie < X >$ est l'algèbre des polynômes de Lie sur l'alphabet X totalement ordonné. On note " $<$ " l'ordre lexicographique obtenu sur X^* Pour construire la matrice de **Lie-Hankel** (voir plus loin), nous avons besoin d'une base de $Lie < X >$. Nous avons alors choisi la base de **Chen-Fox-Lyndon** (G.Viennot[8],M.Lothaire[5]). Nous rappelons que cette base est obtenue à partir de l'ensemble des mots de **Lyndon** qui est donné par:

$$L = \{w \in X^+ | \ \forall u, v \in X^+, w = uv \Rightarrow w < vu\}.$$

Nous pouvons aussi caractériser les mots de Lyndon par la propriété suivante:

$$(w \in L) \iff (w \in X \ ou \ w = lm \ avec \ l, m \in L \ et \ l < m)$$

et c'est cette dernière propriété que nous utiliserons dans notre algorithme.

Soit $\mu : L \longrightarrow Lie < X >$ définie par:

$$\begin{cases} \mu(x) = x \ si \ x \in X, \\ \mu(w) = [\mu(u), \mu(v)] \ si \ w \in L \backslash X \ et \ \sigma(w) = (u, v). \end{cases}$$

$\sigma(w)$ est la *factorisation standard* de w, (*voir Lothaire* [l1]), alors $\mu(L)$ est une base de $Lie < X >$ appelée base de **C.-F.-L.** et nous avons:

$$\dim Lie_n < X >= card(L \cap X^n)$$

$Lie_n < X >$ est la sous-algèbre de $Lie < X >$ des polynômes de Lie de degré égal à n.

Pour notre algorithme, nous commençons par définir un ordre sur l'alphabet X. Ensuite, nous construisons tous les éléments de degré 1. Pour pouvoir tester si un mot est de Lyndon ou non, nous avons besoin de garder les mots correspondants aux *mots de Lie* dans un tableau **<mli>** sous-jacent au tableau **<lli>** contenant les éléments de la base déjà construits. Les éléments de degré d sont construits à partir des éléments de degré j et des éléments de degré $d-j$ pour j allant de 1 à $\frac{d-1}{2}$ si d est impair, et jusqu'à $\frac{d}{2}-1$ si d est pair, tout en nous assurant que nous avons bien des mots de Lyndon et que ceux-ci ne se répètent pas. Nous avons utilisé une fonction **<EstInf>** qui nous permet de tester si un mot de Lie est inférieur ou non à un autre.

La fonction **<BaseDeLie>** qui permet de construire la base de Lie prend en argument le cardinal de l'alphabet X et le degré k jusqu'auquel on veut aller. Elle utilise les fonctions **<Trait1>**, **<Trait2>**, **<Const1>**, **<Const2>**, **<Const3>** et enfin **<EstInf>**.

CALCUL DE LA REALISATION

Pour le calcul de la réalisation nous aurons besoin de tous les outils que nous avons décrits dans la partie précédente mais aussi d'autres que nous allons définir dans ce qui suit et dont les codes en *macsyma* seront donnés dans l'ANNEXE D.

1. DEGRE TOTAL ET TERME CONSTANT D'UN POLYNOME.

Dans tous les calculs qui vont suivre, nous avons besoin de connaître le degré total de la série génératrice plynômiale du système à étudier. En particulier, pour la construction de la base de $R < X >$ et de celle de $Lie < X >$, nous avons écrit une fonction **<DegTo>** qui prend en argument un polynôme et renvoie son degré total.

Par les fonctions *macsyma* prédéfinies nous ne pouvons pas avoir directement le terme constant d'un polynôme donné. Nous avons alors écrit une fonction qui, étant donné un polynôme S, renvoie le coefficient, $< S|1 >$, du mot vide dans celui-ci. Nous avons appelée cette fonction **<Constant>**.

2. MATRICE DE LIE-HANKEL ET RANG DE LIE.

On définit la matrice de Lie-Hankel comme étant un tableau infini dont les lignes sont indicées par les éléments d'une base de $Lie < X >$ et dont les colonnes sont indicées par une base de $R < X >$. On montre facilement que si S est une série formelle non commutative sur un alphabet X alors son *rang de Lie*, $RL(S)$ est égal au rang de la matrice de Lie-Hankel.

Etant donné un polynôme S, nous calculons son degré total et nous construisons nos bases jusqu'à cet ordre. Pour avoir la ligne indicée par le polynôme de Lie P de la matrice de Lie-Hankel, nous simplifions S à gauche par P et nous exprimons le résultat dans la base de $R < X >$. Nous obtenons une liste de coefficients que nous transformons en une matrice ligne. Nous appliquons le même procédé à tous les éléments de la base de $Lie < X >$ et nous obtenons ainsi la matrice de Lie-Hankel.

Ayant la matrice de Lie-Hankel, nous utilisons la fonction *macsyma* **<Rank>** pour calculer son rang qui est le rang de lie de S.

La fonction **<LieHankel>**, qui permet de construire la matrice de Lie-Hankel et de calculer le rang de Lie de S, utilise les fonctions **<SimGauML>**, **<DegTo>**, **<BaseDeKX>** et enfin **<BaseDeLie>**. Elle prend en argument un polynôme S.

3. MOTS CARACTERISTIQUES.

Nous appelons *mots caractéristiques* les mots indiçant les colonnes linéairement indépendantes de la matrice de Lie-Hankel. Leur nombre est égal au rang de Lie de la série étudiée. Algorithmiquement, nous déterminons les mots caractéristiques en prenant la colonne non nulle la plus à droite de la matrice de Lie-Hankel et nous parcourons la matrice de droite à gauche en cherchant les

colonnes linéairement indépendantes de celle-ci. Nous mémorisons les indices de ces colonnes dans un tableau <**MotCar**>. La fonction <**MotsCar**> qui calcule les mots caractéristiques prend en argument une matrice, qui sera à l'appel la matrice <**MatLieHan**> de Lie-Hankel, et nous renvoie un tableau <**MotCar**> à une seule dimension qui contiendra les mots caractéristiques.

Remarque:

On peut mettre tous les indices des colonnes non nulles dans un tableau puis, choisir à chaque fois r mots ($r = Rang\ de\ Lie(S)$) indiçant r colonnes linéairement indépendantes. Ce qui nous permettra plusieurs choix de mots caractéristiques.

4. COORDONNEES LOCALES.

4.1. Calcul des coordonnées locales.

Nous calculons les *coordonnées locales* par la formule

$$Z_i = w_i \circ S- < S|w_i >, \qquad 1 \leq i \leq RL(S),$$

où S est une série formelle et où les w_i sont les mots caractéristiques.

Le symbole $< \circ >$ désigne la simplification à droite de S par le mot w_i. Nous utilisons alors la fonction <**SimDroPol**>. La fonction qui calcule les coordonnées locales est appelée <**CordLoc**>. Elle prend en argument un polynôme, qui sera à l'appel la série génératrice du système à étudier, et renvoie un tableau $< Z >$.

Le coefficient $< S|w_i >$ est calculé en utilisant la fonction <**Constant**> puisque

$$< S|w_i >=< w_i \circ S|1 > .$$

4.2. Actions sur les coordonnées locales.

Pour calculer les *actions* $x_i \circ Z_j$, nous utilisons la fonction <**SimDroLet**> déjà définie qui permet de simplifier un polynôme S à droite par une lettre x . Nous écrivons donc une fonction <**Actionx**> qui utilise le tableau $< Z >$ et l'entier $< RL(S) >$ et qui nous renvoie un tableau <**matx**> à deux dimensions tel que $matx[i,j] = x_j \circ Z_i$ et un tableau <**degre**> aussi à deux dimensions tel que $degre[i,j] = DegTo(matx[i,j])$.

5. FONCTION D'OBSERVATION ET CHAMPS DE VECTEURS.

5.1. Calcul des champs de vecteurs.

Les étapes 4.1 et 4.2 fournissent la réalisation, mais représentée dans l'algèbre des séries non commutatives. Pour obtenir la réalisation, il faut les traduire en termes de variables commutatives usuelles. Cela peut s'exprimer comme suit.

A chaque lettre x_i de l'alphabet $X = \{x_0, x_1, \ldots, x_{m-1}\}$ est associé un champ de vecteurs

X_i. Le champ de vecteurs X_i, $0 \leq i \leq m-1$ est donné par la formule:

$$X_i = \sum_{j=1}^{RL(S)} Trad(x_i \,\square\, Z_j)\frac{\partial}{\partial z_j},$$

où $(Z_j)_{1\leq j\leq RL(S)}$ sont les coordonnées locales associées à S, la transformation ***Trad*** signifie:

- → exprimer $x_i \,\square\, Z_j$ en fonction des mélanges des Z_k ,
- → remplacer les Z_k par les y_k ,
- → remplacer le produit de mélange $< \sqcup >$ par le produit commutatif usuel.

Remarque:

Pour le langage *macsyma* il n'y a pas de distinction entre les lettres majuscules et les lettres minuscules. Donc, pour qu'il n'y ait pas de confusion nous utilisons y_i au lieu de z_i .

5.2. Mélanges utiles.

5.2.1. Comparaison de deux listes.

Vu que le système *macsyma* n'offre pas une fonction faisant la comparaison de deux listes, nous avons écrit une fonction **<Comparelist>** qui prend en argument deux listes et qui renvoie comme résultat **<True>** si les deux listes sont égales et **<False>** sinon.

5.2.2. Recherche d'une liste dans un tableau.

Nous avons écrit une fonction **<recherche>** qui prend en argument une liste $< l >$, un tableau $< t >$ et deux entiers $< k_1 >$ et $< k_2 >$. Cette fonction effectue une recherche de la liste $< l >$ dans le tableau $< t >$ entre les indices $< k_1 >$ et $< k_2 >$. Elle renvoie **<True>** si la recherche aboutit et **<False>** sinon.

5.2.3. Recherche des coordonnées locales utiles.

Pour chaque couple (i,j) il faut exprimer le polynôme $x_j \,\square\, Z_i$ en fonction des mélanges des $(Z_k)_{1\leq k\leq RL(S)}$. On remarque qu'on ne se servira effectivement que des coordonnées dont le degré total reste inférieur au degré total de $x_j \,\square\, Z_i$. Nous avons écrit une fonction **<CoordUtiles>** qui prend en argument un polynôme, qui sera à l'appel $x_j \,\square\, Z_i$, et utilise le tableau **<Z>** des coordonnées locales et nous renvoie un tableau **<tableau>** de listes d'indices des coordonnées locales susceptibles d'être utiles, et un tableau **<somdeg>** contenant les sommes des degrés totaux des coordonnées dont les indices apparaissent dans les listes.

Exemple:
Si $Z_{k_1} \sqcup Z_{k_2} \ldots \sqcup Z_{k_l}$ est un mélange utile alors dans *tableau*[*i*] on mémorise la liste $[k_1, k_2, \ldots, k_l]$ et dans *somdeg*[*i*] , on mémorise $\mathrm{DegTo}(Z_{k_1}) + \mathrm{DegTo}(Z_{k_2}) + \ldots + \mathrm{DegTo}(Z_{k_l})$.

5.2.4. Mélange de coordonnées utiles.

Par la fonction <**CoordUtiles**>, nous avons obtenu un tableau <**tableau**> de listes d'indices. Nous avons écrit une fonction <**MelListe**> qui prend en argument une liste d'entiers et qui renvoie le mélange des coordonnées locales dont les indices sont contenus dans cette liste. Ensuite, nous avons écrit une fonction <**MelUtile**> qui, à partir du tableau <**tableau**> et en utilisant la fonction <**MelListe**> précédente, nous construit un tableau <**TableMel**> qui contient les mélanges des éléments de <**tableau**>.

5.2.5. Produit commutatif.

Par la suite nous aurons à transformer le produit de mélange $<\omega>$ par le produit usuel $< * >$, nous avons alors écrit une fonction <**Produit**> qui prend en argument une liste d'indices et qui renvoie le produit commutatif de variables indicées par les éléments de cette liste.

Exemple: $Produit([i_1, i_2, i_3]) = yi_1 * yi_2 * yi_3$.

5.3. Expression d'un polynôme en fonction des coordonnées utiles.

Nous avons écrit une fonction <**Resolution**> qui prend en argument un polynôme $< P >$. Cette fonction commence par construire une matrice <**mat**> dont les lignes sont formées par les coefficients des éléments du tableau <**TableMel**> dans la base de $R < X >$.

Chaque ligne $< i >$ de la matrice <**mat**> est multipliée par un coefficient $< a[i] >$. On fait ensuite la somme de toutes les lignes obtenues et on obtient une liste <**SystEq**> d'équations en les variables $< a[i] >$. D'autre part, on exprime le polynôme $< P >$ dans la base de $R < X >$ ce qui nous donne une liste <**liste**> de coefficients et on résout le système:

$$< SystEq >=< liste >,$$

suivant les variables $< a[i] >$. On obtient une liste de valeurs qui sont en fait les valeurs des $a[i]$.

5.4. Réalisation.

Nous avons maintenant tous les outils nécessaires pour pouvoir calculer la réalisation d'un système analytique donné par sa série génératrice $< S >$. La fonction que nous allons écrire maintenant prendra comme arguments une série $< S >$ et un entier $< m >$; ce dernier sera le cardinal de l'alphabet sur lequel est écrite la série $< S >$. Nous traitons en premier le cas particulier où la série $< S >$ est une constante, puis le cas général. Nous calculons la matrice de Lie-Hankel et son rang par la fonction <**LieHankel**> puis, par la fonction <**MotsCar**> nous calculons les mots caractéristiques, puis par <**CoordLoc**> nous calculons les coordonnées locales et enfin par <**Actionx**> nous calculons les actions des lettres sur les coordonnées locales. Nous utilisons ensuite les fonctions <**CoordUtiles**>, <**MelUtil**> et <**Resolution**> pour calculer les champs de vecteurs et la fonction d'observation.

VERIFICATION DE LA REALISATION

Les codes en *macsyma* des fonctions qui seront données dans ce paragraphe seront dans l'ANNEXE E.

Pour vérifier la réalisation calculée par le logiciel décrit dans la partie précédente, nous proposons dans cette partie un programme qui permet de partir des champs de vecteurs et de la fonction d'observation trouvés pour retrouver la série génératrice du départ.

1. ACTION D'UN CHAMP DE VECTEURS SUR UN POLYNOME.

Nous aurons besoin d'une fonction qui permettra de calculer l'action d'un champ de vecteurs sur un polynôme.

Soit $X_j = \sum_{i=1}^{RL(S)} c_{ij} \frac{\partial}{\partial y_i}$ un champ de vecteurs et P un polynôme. Alors on a:

$$X_j \circ P = \sum_{i=1}^{RL(S)} c_{ij} \frac{\partial P}{\partial y_i}.$$

Pour le calcul de la dérivée partielle, $\frac{\partial P}{\partial y_i}$, du polynôme P par rapport à la variable y_i , nous utilisons la fonction *macsyma* **<Diff>**.

Nous avons écrit une fonction **<Composition>** qui prend comme arguments un polynôme P et un entier j , elle utilise le rang de Lie de S : $RL(S)$ et un tableau **<ChVec>** à deux dimensions qui est tel que $ChVec[i,j] = C_{ij}$ et nous renvoie la valeur de $X_j \circ P$.

2. VERIFICATION.

Nous avons écrit une fonction **<Calculer>** qui utilise la fonction **<Composition>** précédente, la fonction **<Constant>** déjà vue, mais aussi les fonctions: **<MotAss>** qui permet d'associer à une liste d'entiers le mot obtenu en faisant le produit non commutatif des x indicés par les éléments de la liste, **<Empiler>** qui permet d'empiler une liste de trois éléments dont le premier est un polynôme, le deuxième est une liste d'entiers et le troisième est un entier, et enfin **<Depiler>** qui permet de dépiler une information empilée par la fonction précédente.

La fonction **<Calculer>** nous renvoie une série qui est la série de départ si notre réalisation est correcte.

BIBLIOGRAPHIE

[1] M.FLIESS. – Fonctionnelles causales non linéaires et indéterminées non commutatives . – *Bull. Soc. Math. France*, **109**, 1981, p. 3 – 40.

[2] M.FLIESS. – Réalisation locale des systèmes non linéaires, algèbres de Lie filtrées transitives et séries génératrices . – *Invent. Math.*, **71**, 1983, p. 521 – 537.

[3] G.JACOB. – Réalisation des systèmes réguliers (ou bilinéaires) et séries génératrices non commutatives . – *Séminaire d'Aussois, In Outils et Modèles mathématiques pour l'automatique, l'analyse des systèmes et le traitement du signal,* CNRS LANDAU, **567**, 1980, p. 325 – 357.

[4] C.HESPEL & G.JACOB. – Approximation of nonlinear systems by bilinear ones. – *Algebraic and Geometric Methods in Nonlinear Control Theory*, 1986, p. 511–520.

[5] M.LOTHAIRE. – Combinatorics on words. – *Reading, Massachusetts*, 1983.

[6] MACSYMA. – Reference Manual, Version Ten. – *Massachusetts Institute of Technology and Symbolics*, 1983.

[7] C.REUTENAUER. – Sur la réalisation locale des séries génératrices de rang de Lie fini. – *Algebraic and Geometric Methods In Nonlinear Control Theory*, 1986, p. 33–43.

[8] G.VIENNOT. – Algèbres de Lie libres et monoïdes libres. – *Lecture Notes In Mathematics, Springer-Verlag, 691*, 1978.

ANNEXE A

<SimGauLet>

```
/*   Cette fonction renvoie  le simplifie a gauche  du polynome  <P>  par la */
/* variable <x>.                                                              */

SimGauLet(p,x) := if numberp(p) then 0
                  else
                if atom(p) then
                          if p=x then 1
                       else 0
                     /* cas d'une puissance*/
                else
                  if inpart(p,0) = "^^" then
                        SimGauLet(inpart(p,1),x).(inpart(p,1)^^(inpart(p,2)-1))
                      /* cas d'une multiplication par un scalaire*/
                      else
                    if inpart(p,0) = "*" then
                          if numberp(inpart(p,1)) then
                            inpart(p,1)*SimGauLet(inpart(p,allbut(1)),x)
                          else print("SimGauLet : ERREUR POLYNOME MAL ECRIT")
                        else
                      if inpart(p,0) = "." then
                            SimGauLet(inpart(p,1),x).inpart(p,allbut(1))
                          else
                            if inpart(p,0)="+" then
                              SimGauLet(inpart(p,1),x)
                                + SimGauLet(inpart(p,allbut(1)),x)
                            else
                              print("SimGauLet --> ERREUR POLYNOME MAL ECRIT")$
```

<SimGauPol>

```
/*  Cette fonction renvoie le simplifie a gauche du polynome <S> par le poly-*/
/* nome <P>, tous les deux fournis en arguments.                              */


SimGauPol(S,P):= block(
                if not(member(simgaulet,listfonction)) then
                (loadfile("simgaulet.ldf"),
                 listfonction:append(listfonction,[simgaulet])),

                if numberp(P) then P*S
                  else
                if atom(P) then SimGauLet(S,P)
                    else
                  if inpart(P,0) = "^^" then
                        SimGauPol(SimGauPol(S,inpart(P,1)),
                                        inpart(P,1)^^(inpart(P,2)-1))
                     else
                    if inpart(P,0) ="." then
                          SimGauPol(SimGauPol(S,inpart(P,1)),
                                inpart(P,allbut(1)))
                       else
                      if inpart(P,0)= "*" then
                            inpart(P,1)*SimGauPol(S,inpart(P,allbut(1)))
                         else
                        if inpart(P,0) = "+" then
                        SimGauPol(S,inpart(P,1))
                                        + SimGauPol(S,inpart(P,allbut(1)))
                           else
                         print("SimGauPol --> ERREUR POLYNOME MAL ECRIT"))$
```

<SimGauLie>

```
/*   Cette fonction renvoie  le simplifie du polynome <P>  par le mot de Lie */
/* <ML>, tous les deux fournies en arguments.                                 */

SimGauML(p,ML):= block(
                if not(member(simgaulet,listfonction)) then
                (loadfile("simgaulet.ldf"),
                 listfonction:append(listfonction,[simgaulet])),

                if atom(ML) then SimGauLet(p,ML)
                  else SimGauML(SimGauML(p,first(ML)),last(ML))
                         - SimGauML(SimGauML(p,last(ML)),first(ML)))$


/*   Cette fonction renvoie  le simplifie  a gauche  du polynome <S>  par le */
/* polynome de Lie <PL>.                                                      */

SimGauPL(S,PL) := if atom(PL) then SimGauLet(S,PL)
                  else
                    if inpart(PL,0) = "[" then SimGauML(S,PL)
                    else
                  if inpart(PL,0) = "*" then
                       inpart(PL,1)*SimGauPL(S,inpart(PL,allbut(1)))
                     else
                    if inpart(PL,0) = "+" then
                         SimGauPL(S,inpart(PL,1))
                             + SimGauPL(S,inpart(PL,allbut(1)))
                        else print("SimGauPL --> ERREUR DANS POLY. DE LIE")$
```

<SimDroLet>

```
/*   Fonction qui renvoie le simplifie a droite du polynome <P> par la lettre*/
/* <x>, <P>  et <x> sont fournis en arguments.                                 */

SimDroLet(P,x) := if numberp(P) then 0
                  else
                    if atom(P) then if P=x then 1 else 0
                    else
                      if inpart(P,0) = "^^" then
                        (inpart(P,1)^^(inpart(P,2)-1)).SimDroLet(inpart(P,1),x)
                      else
                        if inpart(P,0) = "*" then
                          if numberp(inpart(P,1)) then
                            inpart(P,1)*SimDroLet(inpart(P,allbut(1)),x)
                          else print("SimDroLet --> ERREUR DANS POLYNOME")
                        else
                          if inpart(P,0) = "." then
                            inpart(P,1).SimDroLet(inpart(P,allbut(1)),x)
                          else
                            if inpart(P,0)="+" then
                              SimDroLet(inpart(P,1),x)
                                + SimDroLet(inpart(P,allbut(1)),x)
                            else print("SimDroLet --> ERREUR DANS POLYNOME")$
```

<SimDroPol>

```
/*   Fonction qui renvoie le simplifie a droite du polynome <S> par le poly- */
/* nome <P>, <P> et <S> sont fournis en arguments.                             */


SimDroPol(S,P):= block(
                if numberp(P) then P*S
                  else
                if atom(P) then SimDroLet(S,P)
                    else
                  if inpart(P,0) = "+" then
                        SimDroPol(S,inpart(P,1))
                       + SimDroPol(S,inpart(P,allbut(1)))
                      else
                    if inpart(P,0) ="." then
                          SimDroPol(SimDroPol(S,inpart(P,allbut(1))),
                                                              inpart(P,1))
                      else
                     if inpart(P,0)= "*" then
                           if numberp(inpart(P,1)) then
                             inpart(P,1)*SimDroPol(S,inpart(P,allbut(1)))
                           else print("SimDroPol --> ERREUR DANS POLYNOME")
                       else
                      if inpart(P,0) = "^^" then
                           SimDroPol(SimDroPol(S,inpart(P,1)),
                                          inpart(P,1)^^(inpart(P,2)-1))
                       else print("SimDroPol --> ERREUR DANS POLYNOME"))$
```

ANNEXE B

<BaseDeKX>

```
/*   Cette fonction nous permet, etant donnes le cardinal m de l'alphabet et */
/* le degre k jusqu'auquel on veut aller, de construire la base canonique de */
/* l'algebre des polynomes non commutatifs. Cette base est ordonnee par */
/* longueur et par l'ordre lexicographique pour chaque longueur.          */


BaseDeKX(m,k):=block([j,l,s,n,i],
                  lis:[],
                      if m=1 then
                        (for i from 1 thru k+1 do
                       (b[i]:(concat(x,0))^^(i-1),
                        lis:endcons(b[i],lis)
                           ),
                     dimtab:k+1
                    )
                      else
                        (b[1]:1,
                         for l from 0 thru k-1 do
                           for j from m^l thru 2*m^l-1 do
                             for s from 0 thru m-1 do
                               (i:m*j+s-(m-2)*(m^(l+1)-1)/(m-1),
                                n:j-(m-2)*(m^l-1)/(m-1),
                                b[i]:b[n].concat(x,s),
                                lis:endcons(b[i],lis)
                               ),
                     dimtab:i
                        )
                     )$
```

<EstInf>

```
/*  Cette fonction permet de tester  si le monome de Lie <ll>  est inferieur */
/* au monome de Lie <mm> ou non.                                              */


estinf(ll,mm):= if atom(ll) then
                  if atom(mm) then is(ll<=mm)
                  else estinf(ll,first(mm))
                else if atom(mm) then estinf(first(ll),mm)
                     else if is(first(ll)=first(mm)) then
                            estinf(last(ll),last(mm))
                          else estinf(first(ll),first(mm))$
```

<BaseDeLie>

```
/*   La fonction  <BaseDeLie>  permet de construire une base de Lie.  On lui */
/* passe comme parametres : m,  qui est le nombre de lettres de l'alphabet et*/
/* n, qui sera  le degre  jusqu'auquel on veut  aller. Dans notre cas n sera */
/* le degre  total de  la serie a etudier.                                    */


BaseDeLie(m,n):=block([j,ns,ur,tt,i],
                local(mli,aa,bb),
                  tt:[],
                  for j:0 thru m-2 do assume(concat(x,j)<concat(x,j+1)),
                  for i:0 thru m-1 do
                    (lli[1,i]:concat(x,i),
                     mli[1,i]:concat(x,i)
                    ),
                  la[1]:m,
                  if m>1 then
                    (ns:2,
                     while ns <= n do
                       (i:0,
                        if oddp(ns) then trait1(ns) else trait2(ns),
                    la[ns]:i,
                        ns:ns+1
                       )
                    )
                 )$
```

<Trait1>

```
/* Le parametre ns que l'on passe a la fonction <trait1> represente le numero*/
/* de la ligne que l'on veut construire; c'est aussi  le degre des <monomes> */
/* de Lie que cette ligne contient.                                           */

trait1(ks):=block([j],
                  for j:1 thru (ks-1)/2 do const1(j,ks-j),
                  for j:1 thru (ks-1)/2 do const2(ks-j,j)
                 )$
```

<Trait2>

```
/* <trait2>  comme <trait1>  prend  pour  parametre  le numero de la ligne a */
/* construire, mais dans le cas ou ce numero est pair.                        */


trait2(ks):=block([j],
                  for j:1 thru ks/2-1 do const1(j,ks-j),
                  const3(ks/2),
                  for j:1 thru ks/2-1 do const2(ks-j,j)
                 )$
```

<Const1>

```
/* A  la fonction <const1>  on passe  deux  parametres  qui representent les */
/* numeros  des deux lignes   a partir  desquelles on va construire la ligne */
/* courante.  Vous remarquerez  qu'en meme temps on construit une liste <tt> */
/* qui va  nous permettre  d'eviter  de construire des monomes dont les mots */
/* associes sont egaux.                                                       */

const1(a1,b1):=block([r:0,k,l],
                     tt:[],
                     for k:0 thru la[a1]-1 do
                       for l:0 thru la[b1]-1 do
                         (if estinf(lli[a1,k],lli[b1,l]) then
                            (lli[ns,i]:[lli[a1,k],lli[b1,l]],
                             mli[ns,i]:mli[a1,k].mli[b1,l],
                             tt:endcons(mli[ns,i],tt),
                             i:i+1
                            )
                          else
                            (aa[a1,r]:k,bb[b1,r]:l,
                             r:r+1
                            )),
                     ur:r
                    )$
```

<Const2>

```
/* Cette fonction  permet de construire les monomes qui n'ont pas ete obtenus*/
/* par <const1>.  On lui  passe comme  parametres  les numeros  des lignes a */
/* partir desquelles se fait la construction.                                 */


const2(z,t):=block([a,r],
                   for r:0 thru ur-1 do
                     (a:mli[z,bb[z,r]].mli[t,aa[t,r]],
                      if not(member(a,tt)) then
                        (mli[ns,i]:a,
                         lli[ns,i]:[lli[z,bb[z,r]],lli[t,aa[t,r]]],
                         i:i+1,
                         tt:endcons(a,tt)
                        )
                     )
                  )$
```

<Const3>

```
/*  Cette fonction est utilisee  uniquement dans le cas ou ns est pair. Elle */
/* permet donc  de construire  des monomes  a partir de la ligne ns/2.       */


const3(ka):=block([a,j,k],
                  for j:0 thru la[ka]-1 do
                  for k:0 thru la[ka]-1 do
                    if not(j=k) and estinf(lli[ka,j],lli[ka,k]) then
                      (a:mli[ka,j].mli[ka,k],
                       if not(member(a,tt)) then
                         (mli[ns,i]:a,
                          lli[ns,i]:[lli[ka,j],lli[ka,k]],
                          i:i+1,
                          tt:endcons(a,tt)
                         )
                      )
                 )$
```

ANNEXE C

<Decomp>

```
decomp(u):=block(
           if u=1 or atom(u) then
             (debut:u, suite:1)
           else
             if inpart(u,0) =”.” or inpart(u,0) = ”^^” then
               (decomp(inpart(u,1)), suite:SimGauLet(u,debut)))$
```

<EstMot>

```
EstMot(u) := if u=1 or atom(u) then true
             else
               if inpart(u,0) = ”^^” then EstMot(inpart(u,1))
               else
                 if inpart(u,0) = ”.” then
                   EstMot(inpart(u,1)) and EstMot(inpart(u,allbut(1)))
                 else false $
```

<EstMon>

```
EstMon(u) := if EstMot(u) then true
             else
               if inpart(u,0) = ”*” then
                 numberp(inpart(u,1)) and EstMot(inpart(u,allbut(1)))
               else false$
```

<EstPol>

```
EstPol(p) := if EstMon(p) then true
             else
               if inpart(p,0) =”+” then
                 EstMon(inpart(p,1)) and EstPol(inpart(p,allbut(1)))
               else false $
```

<MelMot>

```
MelMot(u,v) := block([debutu,suiteu,debutv,suitev],
               if EstMot(u) and EstMot(v) then
                 (if u=1 or v=1 then u*v
                  else
                    (decomp(u) , debutu :debut , suiteu:suite,
                     decomp(v),  debutv:debut,  suitev:suite,
                     debutu.MelMot(suiteu,v) + debutv.MelMot(u,suitev)
                    )
                 )
               else print("MelMot --> LES MOTS SONT MAL ECRITS")
               )$
```

<MelMon>

```
MelMon(m1,m2) := if EstMon(m1) and EstMon(m2) then
                   if EstMot(m1) then
                     if EstMot(m2) then MelMot(m1,m2)
                     else inpart(m2,1)*MelMot(m1,inpart(m2,allbut(1)))
                   else inpart(m1,1)*MelMon(inpart(m1,allbut(1)),m2)
                 else print("MelMon --> ERREUR DANS LES MONOMES")$
```

<MelPol>

```
MelPol(p1,p2) := if EstPol(p1) and EstPol(p2) then
                   if EstMon(p1) then
                     if EstMon(p2) then MelMon(p1,p2)
                     else MelMon(p1,inpart(p2,1))
                            + MelPol(p1,inpart(p2,allbut(1)))
                   else MelPol(inpart(p1,1),p2)
                              + MelPol(inpart(p1,allbut(1)),p2)
                      else print("MelPol --> ERREUR DANS LES POLYNOMES")$
```

ANNEXE D

<DegTo>

```
/*  Cette fonction  prend comme argument  une serie <S>  et nous renvoie son */
/* degre total.                                                               */

DegTo(S) := if numberp(S) then 0
            else if atom(S) then 1
                 else if inpart(S,0) = "^^" then
                        inpart(S,2)*DegTo(inpart(S,1))
                      else if inpart(S,0) = "." then
                             DegTo(inpart(S,1)) + DegTo(inpart(S,allbut(1)))
                           else if inpart(S,0) = "*" then
                                  DegTo(inpart(S,allbut(1)))
                                else if inpart(S,0) = "+" then
                                       max(DegTo(inpart(S,1)),
                                             DegTo(inpart(S,allbut(1))))
                                     else print("DegTo --> ERREUR DANS SERIE")$
```

<Constant>

```
/*  Cette  fonction permet,  etant donnee  une serie <S>,  d'isoler le terme */
/* constant de celle-ci.                                                     */

Constant(S) := if numberp(S) then S
               else
                 if atom(S) then 0
                 else
                   if inpart(S,0)="+" then
                     if numberp(inpart(S,1)) then inpart(S,1)
                     else 0
                   else 0 $
```

<LieHankel>

```
LieHankel(S,m) := block([i,j,k,p,tampon,l,liss],local(la),

                  k:DegTo(S),
                  print("     Calcul de la base de R<X>!"),
                  BaseDeKX(m,k),
                  if m=1 then
                    (liss:rest(lis),
                     tampon:SimGauLet(S,x0),
                     if tampon#0 then
                       (l:coefmatrix([tampon],liss),
                        l:cons(constant(tampon),l[1]),
                        MatLieHan:matrix(l),
                        nbligne:1)
                    )
                  else
                    (print(" "),
                     print("     Calcul de la base de Lie<X>!"),
                     BaseDeLie(m,k),
                   nbligne:0,
                     r:0,
                     for j:1 thru k do
                       for p from 0 thru la[j]-1 do
                         (tampon:SimGauML(S,lli[j,p]),
                          if tampon#0 then
                            (l: coefmatrix([tampon],lis),
                             l:cons(constant(tampon),l[1]),
                             l:matrix(l),
                             if r = 0 then (MatLieHan :l , r : 1)
                             else MatLieHan:addrow(MatLieHan,l),
                             nbligne:nbligne+1
                            )
                         )),

                    print(" "),
                    print("     La matrice de Lie-Hankel est:"),
                    print("     ____________________________"),
                    print(" "),
                    print("   MatLieHan = ",MatLieHan),
                    r : rank(MatLieHan),
                    print(" "),
                          print("     Le rang de Lie de S est: ",r)
                  )$
```

<MotsCar>

```
/*   Cette fonction  prend en argument  une matrice qui,  a l'appel sera la  */
/* matrice <MatLieHan> de Lie-Hankel construite par la fonction <LieHankel>. */


Motscar(matrice) := block([i,r,rl,mataux],
                    print(" "),
                    print("   Les mots caracteristiques sont :"),
                    print(" "),
                    mataux : col(matrice,dimtab),
                    /* ind sera le nombre de mots caracteristiques */
                    ind:1,
                    if rank(mataux)#0 then
                      (r:1,
                       MotCar[ind]:mataux,
                       print("   MotCar[",ind,"] = ",MotCar[ind]),
                       ind:ind+1)
                    else r:0,
                    for i from dimtab-1  thru 1 step -1 do
                      (if rank(col(matrice,i))#0 then
                         (mataux: addcol(mataux,col(matrice,i)),
                          rl: rank(mataux),
                          if rl # r then
                             (MotCar[ind] : b[i],
                              print("   MotCar[",ind,"] = ",MotCar[ind]),
                              r:rl,
                              ind:ind+1
                             )
                      )
                       ),
                    ind : ind-1
                    )$
```

<CoordLoc>

```
/*   Fonction qui calcule les coordonnees locales <Zi> du polynome <P>. Cette*/
/* fonction utilise les fonctions <SimDroPol> et <Constant> et la tableau qui*/
/* contient les mots caracteristiques construits par la fonction <MotsCar>.  */

CoordLoc(S):=block([j,poly],
             print("   Les coordonnees locales sont :"),
             print(" "),
             for j from 1 thru ind do
               (poly:SimDroPol(S,MotCar[j]),
                z[j]: poly - Constant(poly),
                print(concat(Z,j)," = ",z[j])
               )
             )$
```

<Actionx>

```
/*  Cette fonction qui calcule les actions a droite des lettres <xj> sur les */
/* coordonnees locales <Zi>.                                                  */

actionx(t):=block([i,y,j],
             for j from 0 thru (m-1) do
             for i from 1 thru ind do
               (y:concat(x,j),
                matx[i,j]:simdrolet(t[i],y),
                degre[i,j]:degto(matx[i,j])
               )
             )$
```

<CompareList>

```
/*   Cette fonction renvoie <true>  si les deux listes fournies en arguments */
/* sont egales, et <false> sinon.                                             */

comparelist(l1,l2):=block([bool],
          bool:false,
          if l1=[] and l2=[] then bool:true
          else
            if length(l1)=length(l2) then
              if member(first(l1),l2) then
                (l2:delete(first(l1),l2,1),
                 comparelist(rest(l1),l2)
                )
          )$
```

<Recherche>

```
/*   Cette fonction renvoie  <true> si la liste  <l>  fournie en argument se */
/* trouve dans le tableau <t> , et <false> sinon.                            */

recherche(l,t,k1,k2):=block([i],
        i:k1,
          while i#(k2+1) and not(comparelist(l,t[i])) do i:i+1,
          if i=k2+1 then false  else true
          )$
```

<CoordUtiles>

```
CoordUtiles(P,DP):=block([k:0,mm,n,fin:1,inddeb:1,indut,deg,list],

        for n from 1 thru ind do
            (deg:degto(z[n]),
             if deg#0 and deg<= DP then
               (fin:0,k:k+1,
                tableau[k]:[n],
                somdeg[k]:deg
               )
            ),

          indfin:indut:k,

          while  fin=0 do
            (fin:1,
             for n from inddeb thru indfin do
               (for mm from 1 thru indut do
                (deg:somdeg[mm]+somdeg[n],
                   if deg#0 and deg<=DP then
                     (list:endcons(first(tableau[mm]),tableau[n]),
                      if not(recherche(list,tableau,indfin+1,k)) then
                        (fin:0,k:k+1,
                         tableau[k]:list,
                         somdeg[k]:deg
                        )
                     )
                )
            ),
        inddeb:indfin+1,
          indfin:k
          )
          )$
```

<MelListe>

```
/*   Cette fonction calcule le melange des <Zi> dont les indices se trouvent */
/* dans la liste <l> fournie en argument.                                     */

MelListe(l):=block(
               if l=[] then l
               else MelPol(z[first(l)],MelListe(rest(l)))
             ) $
```

<MelUtil>

```
/*  Cette fonction calcule, etant donne un tableau de listes d'indices, les  */
/* melanges pour tous les elements du tableau.                                */

MelUtil(T):= block([n],
               for n from 1 thru indfin do tablemel[n]:MelListe(T[n]))$
```

<Produit>

```
/*  Cette fonction renvoie le mot forme par le produit commutatif de tous les*/
/* elements <yi> pour <i> dans la liste <l>. Ex : produit([i,j,k])= yi*yj*yk.*/

produit(l):=if l=[] then 1
            else concat(y,first(l))*produit(rest(l))$
```

<Resolution>

```
/*   Cette fonction effectue la resolution  d'un systeme d'equations dans le */
/* but d'exprimer un polynome <P>  en fonction  des melanges des coordonnees */
/* locales <Zi>.                                                             */


resolution(P):=block([i,r,listeq,listvar,liste,mat,systeq],
          liste:0, r:0,
          for i from 1 thru indfin do
            (liste:coefmatrix([tablemel[i]],lis),
             if r=0 then (mat:liste,r:1)
             else mat:addrow(mat,liste)
            ),

          systeq:0,
          listvar:[],
          for i from 1 thru indfin do
            (systeq:systeq+(a[i]*row(mat,i)),
             listvar:cons(a[i],listvar)
            ),

          liste:coefmatrix([P],lis),
          systeq:systeq-liste,
          listeq:[],

          for i:1 thru dimtab-1 do listeq:endcons(systeq[1,i],listeq),
          linsolvewarn:false,
          solvenullwarn:false,
          solve_inconsistent_error:false,
          globalsolve:true,
          test:solve(listeq,listvar)
          )$
```

<Calculer>

```
calculer(s,m):=block([i,j,mm,lis,indfin,dimtab,bool,nbligne,kk,MatLieHan],
                     local(matx,degre,a,z,tableau,motcar,somdeg,tablemel),
          if numberp(s) then
            (print("                Rang de Lie = 0"),
             print("     Il n'y a pas de champs de vecteurs ")
            )
          else
            (print(" "),
             LieHankel(S,m),
             MotsCar(MatLieHan),
             CoordLoc(S),
             Actionx(Z),
             bool:false,
             for i:1 thru ind
               while not(bool) do (bool:is(motcar[i]=1),kk:i),
             if bool then h:concat(z,kk)
             else
               (coordutiles(S,k),
                meluti(tableau),
                resolution(S),
                h:constant(S),
                for i:1 thru indfin do
                  h:h+a[i]*produit(tableau[i]),
                kill(a)
               ),
             print(" "),
             print("     La fonction d'observation est:"),
             print("     ____________________________________"),
             print(" "),
             print("     H = ",h),
             print(" "),
             print("                Calcul des champs de vecteurs!"),
             print(" "),

             for j from 0 thru (m-1) do
               for i from 1 thru ind do
                 if not(numberp(matx[i,j])) then
                   (coordutiles(matx[i,j],degre[i,j]),
                    melutil(tableau),
                    resolution(matx[i,j]),
                    chvec[i,j]:constant(matx[i,j]),
                    for k from 1 thru indfin do
                      chvec[i,j]:chvec[i,j]+(a[k]*produit(tableau[k])),
                    kill(a)
                   )
                 else chvec[i,j]:matx[i,j],
```

```
        print ("     Les champs de vecteurs sont:"),
        print ("     ____________________________"),
        print (" "),

    for i from 0 thru m-1 do
      (mm:0,
       for j from 1 thru ind do
         if chvec[j,i]#0 then mm:mm+(chvec[j,i].(d/dz[j])),
       print("     ",concat(X,i)," = ",mm)
      )
 ),
 read("   Tapez <c> puis ; puis <RETURN> pour continuer:")
)$
```

ANNEXE E

<Composition> <MotAss> <Empiler> <Depiler>

```
Composition(P,j):=block([i],
        q:0,
        for i:1 thru n do
          q:q+tab[i,j]*diff(P,concat(z,i))
        )$


Motass(l):= if l=[] then l
        else concat(x,first(l)).Motass(rest(l))$


Empiler(t):=block(
        if pilevide then pilevide:false,
        posc:posc+1,
        pile[posc]:t
        )$


Depiler():=block(
        if posc=0 then pilevide:true
        else
          (l:pile[posc],
           posc:posc-1,
           if posc=0 then pilevide:true
          )
        )$
```

<Verification>

```
Verification(tab,n):=block([i,posc:0,pilevide:true,constante,sa,q,t,l],
                           local(pile),
          constante:sa:constant(S),
          t:[h,[],0],
          Empiler(t),
          while not(pilevide) do
            (Depiler(),
             if (l[3]<m) and (not(numberp(l[1]))) then
               (t:[l[1],l[2],l[3]+1],
                Empiler(t),
                Composition(l[1],l[3]),
                constante:constant(q),
                sa:sa+constante*motass(cons(l[3],l[2])),
                t:[q,cons(l[3],l[2]),0],
                Empiler(t)
               )),
          print(sa),
          read("   Tapez <c> puis ; puis <return> pour continuer:"))$
```

ANNEXE F

PROGRAMME CALCULANT LA REALISATION D'UN SYSTEME LORSQU' ELLE EXISTE

menu principal

1- calculs independants.

2- calculer la realisation.

3- verifier la realisation.

4- fin.

tapez le numero de votre choix suivi de ";" :

2;

entrez le cardinal de votre alphabet termine par ";" :

2;

votre alphabet sera desormais le suivant:

[x0, x1]

entrez votre serie terminee par ";" :

(x1^^2).x0+x1.(x0^^2).x1+x1^^2-(x0^^3).x1+x0.x1.x0^^2+x0^^4+x0;

la matrice de lie-hankel est:

matliehan =

```
[ 1  0 0 0  0 0 0 1 -1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[                                                               ]
[ 0  0 1 0  0 1 0 0  1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[                                                               ]
[ 0  0 0 1 -1 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[                                                               ]
[ 0 -2 1 0  0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[                                                               ]
[ 1  0 0 0  0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[                                                               ]
[ 2  0 0 0  0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

le rang de lie de s est: 5

les mots caracteristiques sont:

$$motcar[1] = x1 \,.^{\langle 2\rangle}\, x0$$

$$motcar[2] = x0 \,.^{\langle 2\rangle}\, x1$$

$$motcar[3] = x0 \,.\, x1$$

$$motcar[4] = x1$$

$$motcar[5] = 1$$

les coordonnees locales sont:

$$z1 = x0$$

$$z2 = x1 - x0$$

$$z3 = x1 \,.^{\langle 2\rangle}\, x0 - x0$$

$$z4 = x1 \,.^{\langle 2\rangle}\, x0 + x1^{\langle 3\rangle} - x0$$

$$z5 = x1^{\langle 2\rangle} \,.\, x0 + x1^{\langle 2\rangle} \,.\, x0 \,.\, x1 + x1^{\langle 2\rangle} - x0^{\langle 3\rangle} \,.\, x1 + x0 \,.\, x1 \,.\, x0^{\langle 2\rangle} + x0^{\langle 4\rangle} + x0$$

la fonction d'observation est:

$$h = z5$$

les champs de vecteurs:

$$X0 = \left(-2\,z4 + z1\,z3 + \frac{z2^2}{2} + z1\,z2 + 2\,z2 + \frac{z1^3}{3} + \frac{z1^2}{2} + 2\,z1 + 1\right) . \frac{d}{dz_5} + z3 \,.\, \frac{d}{dz_4} + z2 \,.\, \frac{d}{dz_3} - \frac{d}{dz_2} + \frac{d}{dz_1}$$

$$X1 = z4 \,.\, \frac{d}{dz_5} + \frac{d}{dz_4} + \frac{d}{dz_2}$$

la serie correspondant a votre realisation est:

$$x1^{\langle 2\rangle} \cdot x0 + x1 \cdot x0^{\langle 2\rangle} \cdot x1 + x1^{\langle 2\rangle} - x0^{\langle 3\rangle} \cdot x1 + x0 \cdot x1 \cdot x0^{\langle 2\rangle} + x0^{\langle 4\rangle} + x0$$

PROGRAMME CALCULANT LA REALISATION D'UN SYSTEME LORSQU' ELLE EXISTE

menu principal

1- calculs independants.

2- calculer la realisation.

3- verifier la realisation.

4- fin.

tapez le numero de votre choix suivi de ";" :

2;

entrez le cardinal de votre alphabet termine par ";" :

2;

votre alphabet sera desormais le suivant:

[x0, x1]

entrez votre serie terminee par ";" :

```
x0.x1^^3-(x0^^3).x1-x0.x1^^2+x0.x1.x0^^2+x0^^4+x0;
```

la matrice de lie-hankel est:

```
           [ 1  0  0 0 0 0 -1 1 -1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
           [                                                                  ]
           [ 0  0 -1 1 0 0  1 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
           [                                                                  ]
           [ 0 -2  0 0 0 0  0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
matliehan= [                                                                  ]
           [-1  0  1 0 0 0  0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
           [                                                                  ]
           [ 2  0  0 0 0 0  0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
           [                                                                  ]
           [ 1  0  0 0 0 0  0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

le rang de lie de s est: 5

les mots caracteristiques sont:

$$motcar[1] = x1^{\langle 3 \rangle}$$
$$motcar[2] = x1^{\langle 2 \rangle}$$
$$motcar[3] = x1$$
$$motcar[4] = x0$$
$$motcar[5] = 1$$

les coordonnees locales sont:

$$z1 = x0$$
$$z2 = x0 \cdot x1 - x0$$
$$z3 = x0 \cdot x1^{\langle 2 \rangle} - x0 \cdot x1^{\langle 3 \rangle} - x0$$
$$z4 = x0 \cdot x1^{\langle 3 \rangle} \cdot x0 + x0$$
$$z5 = -x0^{\langle 3 \rangle} \cdot x1 + x0 \cdot x1^{\langle 3 \rangle} - x0 \cdot x1^{\langle 2 \rangle} + x0 \cdot x1^{\langle 2 \rangle} \cdot x0 + x0^{\langle 4 \rangle} + x0$$

la fonction d'observation est:

$$h = z5$$

les champs de vecteurs:

$$X0 = (z4 + 1) \cdot \frac{d}{dz_5} + \left(z2 + \frac{z1^2}{2} + z1\right) \cdot \frac{d}{dz_4} - \frac{z1^2}{2} \cdot \frac{d}{dz_3} - \frac{d}{dz_2} + \frac{d}{dz_1}$$

$$X1 = z3 \cdot \frac{d}{dz_5} + z2 \cdot \frac{d}{dz_3} + z1 \cdot \frac{d}{dz_2}$$

la serie correspondant a votre realisation est:

$$-x0^{\langle 3 \rangle} \cdot x1 + x0 \cdot x1^{\langle 3 \rangle} - x0 \cdot x1^{\langle 2 \rangle} + x0 \cdot x1^{\langle 2 \rangle} \cdot x0 + x0^{\langle 4 \rangle} + x0$$

IV. MOTS DE LYNDON ET MACSYMA

MOTS DE LYNDON ET MACSYMA

N. Oussous (*)

RESUME :

Dans ce papier nous rappelons la définition et quelques propriétés des mots de Lyndon. Ensuite, nous implémentons des algorithmes relatifs à ces mots d'après un papier de G.MELANÇON& C.REUTENAUER. Ces algorithmes, implémentés en MACSYMA, nous permettent, entre autres, de construire la base de CHEN-FOX-LYNDON, la base de POINCARE-BIRKOFF-WITT associée, la décomposition d'un mot ou un polynôme dans cette dernière base. Ce travail nous a conduit à utiliser les mots de Lyndon dans le problème de la réalisation.

ABSTRACT :

We recall some definition and properties of Lyndon words. We implement in MACSYMA some algorithms allowing to "manipulate" Lyndon words. In fact, we give two algorithms : the first one gives eatch word w as a decreasing product of Lyndon words and gives also the element of the P.B.W. basis associated to w. The second algorithm, permits to associate for eatch word w (or polynomial) its expression in the P.B.W.L. basis. With the first algorithm, we can also produce the Lyndon basis and the P.B.W.L. basis. This work has conduct us to use the Lyndon words in the problem of realization.

SOMMAIRE

(*) Laboratoire d'Informatique Fondamentale de Lille.– U.A. 369 du C.N.R.S.–
Université de LILLE I.– 59655 VILLENEUVE D'ASCQ CEDEX. **FRANCE.**

MOTS DE LYNDON & MACSYMA

Introduction.

En me basant sur un papier de G.MELANÇON& C.REUTENAUER[5], j'ai implémenté en MACSYMA deux algorithmes : le premier permet d'écrire tout mot $w \in X^*$ comme produit décroissant de mots de Lyndon (Théorème de Lyndon M.LOTHAIRE[4], G.MELANÇON& C.REUTENAUER[5]). Mais, en plus, il permet de construire l'ensemble des mots de Lyndon, la base de C.F.L. et la base de P.B.W.L. correspondante et ce en traitant les éléments de X^* un après l'autre. L'avantage de cet algorithme est qu'il permet d'obtenir ces trois ensembles ordonnés pour l'ordre lexicographique par longueur. Ce qui peut être intéressant, pour nous, en le comparant à l'algorithme donné par J.P.DUVAL[1]. Du point de vue vitesse de construction, il est clair que les bases de Hall sont les meilleures (P.V.KOSELEFF[3]) mais, pour notre application G.JACOB& N.OUSSOUS[2] nous préférons la base de C.F.L.. L'utilisation de MACSYMA rend la lecture des résultats plus aisée en comparaison avec les résultats donnés par LISP (P.V.KOSELEFF[3]). Le second algorithme permet de donner la décomposition d'un mot $w \in X^*$, puis d'un polynôme dans la base de P.B.W.L.

1. Ordre lexicographique et classes de conjugaison.

Soit X un alphabet totalement ordonné, et soit X^* le monoïde libre engendré par X dont les éléments sont appelés des mots. On définit un ordre total sur X^* comme suit:

$$\forall\, u,v \in X^*, \quad u < v \quad \text{ssi} \quad \begin{cases} \text{ou} & (i)\ \exists\, w \neq \varepsilon \text{ tel que } uw = v, \\ & (ii)\ \exists x,y,z \in X^* \text{ et } a,b \in X \text{ tels que } u = xay,\ v = xbz \text{ et } a < b. \end{cases}$$

On a ainsi l'ordre lexicographique usuel sur les mots de X^*. Pour cet ordre, on a (*cf* Lothaire[4]) les deux propriétés suivantes:

$$\begin{cases} (1) & \forall w \in X^*, & u < v \iff wu < wv. \\ (2) & \text{Si } v \notin uX^*,\ \forall w,z \in X^*, & u < v \implies uw < vz. \end{cases}$$

On dira qu'un mot u est un *facteur* du mot v s'il existe $x,y \in X^*$ tels que $v = xuy$.

Si $x = \varepsilon$ (resp. $y = \varepsilon$), on dira que u est le *facteur gauche* (resp. *droit*) de v, *propre* si $y \neq \varepsilon$ (resp. $x \neq \varepsilon$).

Deux mots u et v de X^* sont dits *conjugués* si il existe $x,y \in X^*$ *tels que* $u = xy$ *et* $v = yx$.

2. Mots de Lyndon.

a. Définition.

Pour plus de détails sur les mots de Lyndon, on peut consulter les références [1], [4] ou [5].

DEFINITION 2.1. – *Un mot* $w \in X^*$ *est un mot de Lyndon ssi:*

$$\begin{cases} & (i) & \textit{il est strictement plus petit que tous ses conjugués,} \\ \textit{ou} & & \\ & (ii) & \textit{il est strictement plus petit que ses facteurs droits propres.} \end{cases}$$

On notera L *l'ensemble des mots de Lyndon sur* X.

b. Propriétés :

(1) Soit $w \in L\backslash X$ et soit m son plus long facteur droit propre dans L. Si $w = lm$, alors $l \in L$ et $l < lm < m$. Le couple $\sigma(w) = (l,m)$ est appelé *la factorisation standard* de w.

(2) $w \in L$ si, et seulement si, $\begin{cases} \text{ou} & w \in X, \\ & w = lm \text{ avec } l,m \in L \text{ et } l < m. \end{cases}$

Ce qui donne un moyen *algorithmique* pour la construction des mots de Lyndon.

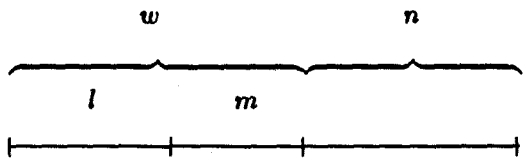
LEMME 2.2. – *Soient* $w \in L\backslash X$ *et* $\sigma(w) = (l, m)$ *sa factorisation standard, et soit* n *un mot de Lyndon vérifiant* $w < n$. *Alors le couple* (w, n) *est la factorisation standard de* wn *si, et seulement si,* $n \leq m$.

Preuve:

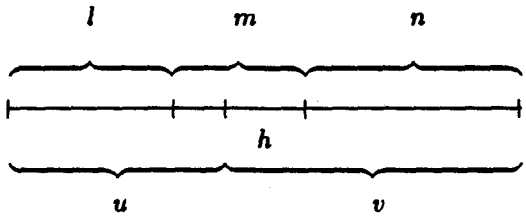
(a) Montrons que $\sigma(wn) = (w, n) \implies n \leq m$.

Supposons que $m < n$, alors d'après la propriété (2) $mn \in L$ et donc n n'est pas le plus long facteur droit propre de wn ce qui est en contradiction avec l'hypothèse: $(w, n) = \sigma(wn)$.

(b) On suppose $w < n \leq m$ et que $\sigma(wn) = (u, v)$ avec $v \neq n$.



On a alors nécessairement $|v| > |n|$. Donc $v = hn$ où h est un suffixe propre de w.



Soit alors k le plus petit suffixe, différent de ε, propre ou non de h. On a alors $k < m$. En effet:

$$k \leq h < hn = v < n \leq m.$$

Or $m \leq k$, puisque k est un suffixe propre de w, et que $\sigma(w) = (l, m)$. D'où contradiction.

□

3. Base de Lyndon.

A partir des mots de Lyndon, on peut construire une base (dite base de LYNDON ou de CHEN-FOX-LYNDON (*cf* LOTHAIRE[4] ou VIENNOT[6])) de l'algèbre $\mathcal{L}ie\langle X\rangle$ des polynômes de Lie. Cette base est obtenue récursivement comme suit:

$$\begin{cases} c(x) = x \quad \textit{pour} \quad x \in X, \\ c(w) = [c(l), c(m)], \quad \textit{pour } w \in L\backslash X, \textit{ tel que } \sigma(w) = (l, m). \end{cases}$$

en utilisant les crochets de Lie. Cette définition nous donne un algorithme pour construire cette base (*cf* VIENNOT[6]).

4. Listes standard et théorème de Lyndon.

DEFINITION [5]. – *Soit* $l = [u_1, u_2, \ldots, u_n]$ *une liste d'éléments de* X^+. *On dira que* l *est standard si, et seulement si, elle vérifie la propriété suivante:*

$$(S) \quad \begin{cases} \textit{ou} & u_i \text{ est une lettre,} \\ & \text{si } u_i = [c(x), c(y)] \text{ tel que } \sigma(u_i) = (x, y) \text{ alors } y \geq u_j,\ j \geq i. \end{cases}$$

(1) Si tous les facteurs u_i sont des lettres, l vérifie (S),

(2) Si l est décroissante (ie $u_1 \geq u_2 \ldots \geq u_n$), alors l vérifie encore (S).

Une *inversion* est un couple (u_i, u_{i+1}) tel que $u_i < u_{i+1}$.

THEOREME [4]. – *Chaque mot* $w \in X^*$ *peut être écrit de manière unique comme suit:*

$$w = l_1 l_2 \dots l_n \qquad (\alpha)$$

où chaque $l_i \in L$ *et* $l_1 \geq l_2 \geq \dots \geq l_n$.

Pour la preuve de ce Théorème voir LOTHAIRE[4] ou G.MELANÇON & C.REUTENAUER[5].

Dans L, on peut définir différentes relations d'ordre total, en particulier, on peut définir l'ordre lexicographique et l'ordre lexicographique par longueur. Vu la définition de la base de Lyndon, toute relation d'ordre total sur L induit naturellement une relation d'ordre total sur la base de Lyndon de $\mathcal{L}ie < X >$.

5. Théorème de P.B.W.

Soit $(P_i)_{i \geq 1}$ une base totalement ordonnée de l'algèbre de Lie $\mathcal{L}$. Alors $\{P_1^{i_1} P_2^{i_2} \dots P_n^{i_n} \mid i_1, i_2, \dots, i_n \geq 0,\ n \geq 0$ *et* $P_1 > P_2 > \dots > P_n\}$ est une base de l'algèbre enveloppante de $\mathcal{L}$ appelée base de Poincaré-Birkhoff-Witt.

Soit $\mathcal{L}ie < X >$ l'algèbre de Lie libre sur l'alphabet X. On considère dans cette algèbre la base de CHEN-FOX-LYNDON (C.F.L.) définie précédemment.

La base de P.B.W.L. est la base de l'algèbre enveloppante de $\mathcal{L}ie < X >$ obtenue par le théorème de P.B.W. à partir de la base de C.F.L.[5].

6. Algorithme de factorisation d'un mot en produit décroissant de mots de Lyndon.

Nous proposons dans ce qui suit un algorithme simple qui permet d'écrire tout mot $w \in X^*$ sous la forme (α).

Soit $w = x_1 x_2 \dots x_n \in X^*$ où tous les x_i sont des lettres.

---

(0) ◇ *Donnée :* $w = x_1 x_2 \dots x_n$,

(1) ◇ $ll \leftarrow [x_1, x_2, \dots, x_n]$, -liste des lettres qui forment w.

(2) ◇ *Chercher dans ll la première inversion depuis la droite,*

◇ **SI** (x_i, x_{i+1}) *est une inversion trouvée* **ALORS**

- $ll_1 \leftarrow [x_1, x_2, \dots, x_{i-1}, x_i x_{i+1}, x_{i+2}, \dots, x_n]$
- *reprendre en (2) avec* ll_1,

◇ **SINON**

- *Faire le produit des éléments de* ll_1 *pour avoir la factorisation de* w *en produit décroissant de mots de Lyndon,*

On s'arrête lorsque la liste est décroissante. Nous sommes sûrs que l'algorithme s'arrête puisque la liste ll_1 compte à chaque pas de l'algorithme une inversion en moins que ll.

A la sortie de l'algorithme nous avons dans ll_1 une liste décroissante de mots de Lyndon, il suffit donc d'en faire le produit pour obtenir le mot w sous la forme (α).

Pour implémenter cet algorithme en MACSYMA, nous avons utilisé la structure de liste pour représenter les listes ll, ll_1 et ll_2, le produit *non commutatif* "." pour représenter le produit de concaténation et enfin les crochets pour séparer les différents mots de Lyndon dans l'expression de w.

Paralèllement à la liste ll_1 nous aurions pu construire une autre liste ll_2 obtenue à partir de ll en crochetant les éléments de l'inversion au lieu de les concaténer. A la sortie, nous aurions dans ll_2 les monômes de Lie correspondant aux mots de Lyndon de la liste ll_1. En faisant le produit de ces monômes, nous obtenons l'élément de la base de P.B.W.L. correspondant au mot w.

Exemple:
Soit $X = \{a, b\}$, $a < b$ et soit $w = a.b.b.a$,

$$w \rightarrow [a,b,b,a] \begin{cases} \nearrow [[a,b],b,a] \rightarrow [[[a,b],b],a] = ll_2 \Rightarrow [[a,b],b].a \\ \searrow [a.b,b,a] \rightarrow [a.b.b,a] = ll_1 \Rightarrow (a.b.b).a \end{cases}$$

La seconde ligne nous donne l'expression du mot w comme produit décroissant de mots de Lyndon:

$$w = l_1.l_2 \quad avec \quad l_1 = a.b.b \quad et \quad l_2 = a,$$

avec $l_1, l_2 \in L$ *et* $l_1 > l_2$.

La première ligne donne l'élément de la base de PBWL (base obtenue par le théorème de Poincare-Birkhoff-Witt à partir de la base de Lyndon[5]) correspondant à w:

$$pbwl(w) = P_1.P_2 \quad avec \quad P_1 = [[a,b],b] \quad et \quad P_2 = a, \quad P_1 > P_2.$$

Par cet algorithme nous sommes capables d'associer à tout mot $w \in X^*$ l'élément correspondant de la base de P.B.W.L.. D'autre part, nous savons que X^* est la base canonique de $\mathbb{Z} < X >$. Donc pour avoir la base de P.B.W.L. de l'algèbre $\mathbb{Z} < X >$ en tant qu'algèbre enveloppante de l'algèbre de Lie $Lie_{\mathbb{Z}} < X >$, nous n'avons qu'à calculer les *transformés* $pbwl(w)$ pour chaque $w \in X^*$.

Remarque :

Les mots $w \in X^*$ pour lesquels on trouve une liste ll_1 qui ne contient qu'un seul élément sont des mots de Lyndon. Donc, on peut retenir de tels mots comme mot de Lyndon et l'élément trouvé dans ll_2 comme élément de la base de C.F.L. correspondant. On a donc un moyen de produire en même temps l'ensemble des mots de Lyndon, la base de C.F.L. et la base de P.B.W.L. en traitant les éléments de X^* l'un après l'autre.

7. Expression d'un mot puis d'un polynôme dans la base de PBWL.

a. Description de l'algorithme.

Soit à décomposer un mot $w = u_1 u_2 \ldots u_n$ où tous les u_i sont des lettres(G.MELANÇON& C.REUTENAUER[5]).

---

(0) ◇ *Donnée :* $w = u_1 u_2 \dots u_n$,

(1) ◇ $ll \leftarrow [u_1, u_2, \dots, u_n]$, -liste des lettres qui forment w.

(2) ◇ *Chercher dans ll la première inversion depuis la droite,*

◇ **SI** (u_i, u_{i+1}) *est une inversion trouvée* **ALORS**

- $ll_1 \leftarrow [u_1, u_2, \dots, u_{i-1}, [u_i, u_{i+1}], u_{i+2}, \dots, u_n]$
- $ll_2 \leftarrow [u_1, u_2, \dots, u_{i-1}, u_{i+1}, u_i, u_{i+2}, \dots, u_n]$
- *reprendre en (2) avec* ll_1,
- *reprendre en (2) avec* ll_2,

◇ **SINON**

- *Faire le produit des éléments de chaque liste,*
- *Faire la somme des éléments obtenus.*

L'algorithme s'arrête puisque après chaque passage la liste ll_1 est plus courte que ll et la liste ll_2 compte une inversion de moins que ll. En sortant de l'algorithme, toutes les listes obtenues sont décroissantes et formées de polynômes de Lie.

Pour implémenter l'algorithme en MACSYMA nous avons utilisé les listes pour représenter les listes standard. Pour les crochets de Lie, nous avons utilisé les listes à deux éléments. Pour le produit de concaténation, nous avons utilisé le produit non commutatif représenté par ".".

Exemple:

$X = \{a, b, c\}$ $\quad a < b < c \quad$ *et* $\quad w = a.b.c.$

$$
w \rightarrow [a,b,c] \begin{cases} [a,[b,c]] \begin{cases} [[a,[b,c]]] \Rightarrow [a,[b,c]] \\ [[b,c],a] \Rightarrow [b,c].a \end{cases} \\ [a,c,b] \begin{cases} [[a,c],b] \begin{cases} [[[a,c],b]] \Rightarrow [[a,c],b] \\ [b,[a,c]] \Rightarrow b.[a,c] \end{cases} \\ [c,a,b] \begin{cases} [c,[a,b]] \Rightarrow c.[a,b] \\ [c,b,a] \Rightarrow c.b.a \end{cases} \end{cases} \end{cases}
$$

Donc

$$w = a.b.c = [a,[b,c]] + [b,c].a + [[a,c],b] + b.[a,c] + c.[a,b] + c.b.a.$$

b. Expression d'un polynôme dans la base de PBWL.

Soit ϕ l'application de X^* dans l'algèbre des polynômes non commutatifs sur X à coefficients dans $\mathbb{Z}$ ($\mathbb{Z} < X >$) qui associe à tout mot $w \in X^*$ son expression dans la base de PBWL. On peut étendre par linéarité l'application ϕ aux polynômes de $\mathbb{R} < X >$ et donc on peut exprimer chaque polynôme dans la base de PBWL.

Soit $P = \sum_{w \in X^*} < P|w > w$ un polynôme. Nous posons

$$\phi(P) = \sum_{w \in X^*} < P|w > \phi(w).$$

En utilisant l'algorithme précédant nous pouvons écrire tout polynôme dans la base de P.B.W.L.

BIBLIOGRAPHIE

[1] J.P.DUVAL. – Génération d'une section des classes de conjugaison et arbre des mots de Lyndon de longueur bornée. – *Publication du LITP Paris*, **88–20**, 1988.

[2] G.JACOB & N.OUSSOUS. – Un logiciel calculant la réalisation des systèmes analytiques de série génératrice finie. – *Publication du LIFL Lille*, **IT–123**, 1988.

[3] P.V.KOSELEFF. – Quelques bases et formules dans les algèbres de Lie libres. – *Publication du Centre Scientifique I.B.M., Paris*, 1988.

[4] M.LOTHAIRE. – Combinatorics on words. – *Reading, Massachusetts*, 1983.

[5] G.MELANÇON & C.REUTENAUER. – Lyndon words, free algebras and shuffles. – *Publication du LITP Paris et UQAM Montreal, Quebec*, **87–63**, 1987.

[6] G.VIENNOT. – Algèbres de Lie libres et monoïdes libres . – *Lecture Notes In Mathematics, Springer-Verlag*, 691, 1978.

ANNEXE : Programmes et Exemples

Programmes

```
vertical(n):=block([i],for i:1 thru n do print(" "))$

Lyndon():= block([w,w0:0,w1,P,P0:0,m,k,dimtab,bool,rep,rep1,l,pbw],local(b),

        matchfix("{","}"),
        listarith:false,
        dotexptsimp:false,

        setup_autoload("pbw.ldf",infer,inflyn,motlist,listmot,inversion,
                    conslist,decomp,decompmo,decompol,listassoc,decassoc,donnee),

        bool:false,
        boucle,
        clear(),
        vertical(3),
        print("                              MENU PRINCIPAL"),
        print("                              --------------"),
        vertical(2),
        print("           1- DECOMPOSITION D'UN MOT EN UN PRODUIT DECROISSANT"),
        print("              DE MOTS DE LYNDON."),
        print(" "),
        print("           2- CONSTRUCTION DE LA BASE DE PBWL."),
        print(" "),
        print("           3- EXPRESSION D'UN MOT DANS LA BASE DE PBWL."),
        print(" "),
        print("           4- EXPRESSION D'UN POLYNOME DANS LA BASE DE PBWL."),
        print(" "),
        print("           5- FIN."),
        vertical(2),
        rep:read("            Entrez votre choix termine par un <\;> :"),
        clear(),
        if not(bool) and rep#5  then donnee(),
        clear(),
        if rep=1 then
          (vertical(3),
           w:read("            Entrez votre mot suivi d'un <\;>: "),
           vertical(2),
           print("           Votre mot se decompose en: "),
           vertical(2),
           l:motlist(w),
           print(decassoc(l,2)),
           w1:1,
           dotexptsimp:false,
           vertical(2),
           read("           Tapez <c\;> puis <Return> pour continuer: "),
           go(boucle)
          )
        else
          if rep=2 then
            (setup_autoload("basedekx.ldf",basedekx),
             vertical(3),
             k:read("            Entrez le degre voulu suivi d'un <\;>: "),
             basedekx(m,k),
             pbw:[],
             for i:1 thru dimtab do
               (l:motlist(b[i]),pbw:endcons(decassoc(l,1),pbw),
                dotexptsimp:false,w1:1),
             vertical(2),
```

```
     print("          La base desiree est dans la liste <pbw>"),
     rep1:read("          Voulez-vous l'afficher? <O/N> : "),
     if rep1=o then print("pbw = ",pbw),
     vertical(2),
     read("          Tapez <c\;> puis <Return> pour continuer:"),
     go(boucle))
else
  if rep=3 then
    (vertical(3),
     w:read("          Entrez un mot suivi d'un <\;>: "),
     vertical(2),
     print("          Votre mot se decompose en :"),
     vertical(2),
     decompmo(w),
     print(w0),
     w0:0,
     vertical(2),
     read("          Tapez <c\;> puis <Return> pour continuer:"),
     go(boucle))
else
  if rep=4 then
    (vertical(3),
     P:read("          Entrez votre polynome termine par un <\;>: "),
     vertical(2),
     print("          Votre polynome se decompose en :"),
     vertical(2),
     P0:decompol(P),
     print(P0),
     P0:0,
     vertical(2),
     read("          Tapez <c\;> puis <Return> pour continuer:"),
     go(boucle))
else
  if rep=5 then go(fin)
else go(boucle),
fin,
clear()
)$
```

```
/* Cette fonction compare deux monomes de Lie et renvoie soit True soit False */

infer(u,v):=if atom(u) and atom(v) then is(u<=v)
        else if atom(u) then infer(u,listmot(first(v)))
             else if atom(v) then if first(u)=v then false
                                  else infer(listmot(first(u)),v)
                  else if listmot(first(u))=listmot(first(v)) then
                         infer(last(u),last(v))
                       else infer(listmot(first(u)),listmot(first(v)))$


/* Cette fonction permet de definir l'ordre lexicographique.                  */

inflyn(m,n):=if atom(m) and atom(n) then is(m<=n)
        else
          if atom(m) then if m=first(n) then true
                          else inflyn(m,first(n))
          else
            if atom(n) then if first(m)=n then false
                            else inflyn(first(m),n)
            else
              if first(m)=first(n) then inflyn(rest(m),rest(n))
              else inflyn(first(m),first(n))$


/* Cette fonction transforme un mot en une liste de lettres.                  */

motlist(w):=if atom(w) then [w]
            else cons(first(w),motlist(rest(w)))$


/* Cette fonction transforme une liste de lettres en un mot.                  */

listmot(l):= if atom(l) then l
        else if l=[] then l else first(l).listmot(rest(l))$


/* Cette fonction parcourt la liste <l> de la droite vers la gauche et cherche*/
/*la premiere inversion. S'il y en a une, elle renvoie sa position <k>.       */
inversion(l,n):=block([i],
        i:length(l)-1,
        while (i>=1) and (if n=1 then infer(l[i+1],l[i])
                          else infLyn(l[i+1],l[i]))      do i:i-1,
        if i>=1 then k:i else k:0)$


/* A partir d'une liste contenant une inversion elle construit deux listes.   */
/* Dans la premiere, on met les crochets a l'inversion et dans la seconde on  */
/* on concatene                                                               */

conslist(l):=block([i],
        for i:1 thru (k-1) do (l1:endcons(l[i],l1),l2:endcons(l[i],l2)),
        l1:endcons([l[k],l[k+1]],l1),
        l2:endcons(l[k+1],l2), l2:endcons(l[k],l2),
        for i:k+1 thru length(l)-1 do
          (l1:endcons(l[i+1],l1),l2:endcons(l[i+1],l2)))$
```

```
/* Cette  fonction  repete  la  precedante  jusqu'a  ce qu'il n y ait  plus   */
/* d'inversion                                                                */

decomp(l):=block([l1,l2],
        inversion(l,1),
        if k#0 then
          (l1:l2:[],
           conslist(l),
           decomp(l1),
           decomp(l2))
        else w0:w0+listmot(l))$


/* Cette fonction permet de decomposer un mot dans la base de PBWL            */

decompmo(w):=block([l], l:motlist(w), decomp(l))$


/* Cette fonction permet de decomposer un polynome dans la base de PBWL       */

decompol(P):=if atom(P) or numberp(P) then P
        else
          if inpart(P,0)="." then (w0:0,decompmo(P))
          else
            if inpart(P,0)="*" then
              (w0:0,inpart(P,1)*decompmo(inpart(P,2)))
            else
              if inpart(P,0)="+" then
                decompol(inpart(P,1))
                  +decompol(inpart(P,allbut(1)))
              else print("  ERREUR --> decompol")$


/* Cette fonction permet, suivant la valeur de n, d'associer a la liste <l>   */
/*une nouvelle liste, soit en crochetant l'inversion soit en la concatenant.  */

listassoc(l,n):=block([i],
        for i:1 thru (k-1) do l1:endcons(l[i],l1),
        l1:endcons(if n=1 then [l[k],l[k+1]]
                   else l[k].l[k+1],l1),
        for i:k+1 thru length(l)-1 do l1:endcons(l[i+1],l1))$


/* Cette fonction applique a la liste <l> les fonctions inversion et listassoc*/
/*tant qu'il y a des inversions dans celle-ci. Ceci permet d'obtenir, suivant */
/*la valeur de <n>, soit la decomposition d'un mot en produit decroissant de  */
/*mots de Lyndon, soit le polynome de PBWL correspondant a ce mot.            */

decassoc(l,n):=block([l1],
        inversion(l,n),
        if k#0 then (l1:[], listassoc(l,n), decassoc(l1,n))
        else
          if n=1 then
            (dotexptsimp:true,
             w1:listmot(l))
          else
            (l1:[],
             for i:1 thru length(l) do
```

```
        ll:endcons( if atom(l[i]) then l[i]
                    else [l[i]],ll),
      wl:l,
      dotexptsimp:true,
      for i:1 thru length(ll) do wl:wl.ll[i]),
    wl)$


Donnee():=block([i,liste],
    vertical(3),
    m:read("                        Entrez le cardinal de votre alphabet:"),
    vertical(3),
    for i:0 thru m-2 do assume(concat(x,i)<concat(x,i+1)),
    liste:makelist(concat(x,i),i,0,m-1),
    print("                        Votre alphabet est desormais :"),
    print(" "),
    print("                           ",liste),
    bool:true)$
```

Exemples

```
(c1) lyndon();
```

MENU PRINCIPAL

1- DECOMPOSITION D'UN MOT EN UN PRODUIT DECROISSANT
   DE MOTS DE LYNDON.

2- CONSTRUCTION DE LA BASE DE PBWL.

3- EXPRESSION D'UN MOT DANS LA BASE DE PBWL.

4- EXPRESSION D'UN POLYNOME DANS LA BASE DE PBWL.

5- FIN.

Entrez votre choix termine par un <;> :

```
1;
```

Entrez le cardinal de votre alphabet:

```
2;
```

Votre alphabet est desormais :

[x0, x1]

```
          Entrez votre mot suivi d'un <;>:
x0.x1.x0.x1.x1;


          Votre mot se decompose en:

{x0 . x1 . x0 . x1 . x1}


          Tapez <c;> puis <Return> pour continuer:
c;



          Entrez votre mot suivi d'un <;>:
x1.x0.x0.x1.x0.x1;


          Votre mot se decompose en:

x1 . {x0 . x0 . x1 . x0 . x1}


          Tapez <c;> puis <Return> pour continuer:
c;



          Entrez votre mot suivi d'un <;>:
x1.x1.x0.x0.x1.x1.x1;


          Votre mot se decompose en:

  <2>
x1    . {x0 . x0 . x1 . x1 . x1}


          Tapez <c;> puis <Return> pour continuer:
c;



          Entrez votre mot suivi d'un <;>:
x0.x0.x1.x1.x1.x0.x1;


          Votre mot se decompose en:

{x0 . x0 . x1 . x1 . x1 . x0 . x1}


          Tapez <c;> puis <Return> pour continuer:
c;
```

MENU PRINCIPAL

1- DECOMPOSITION D'UN MOT EN UN PRODUIT DECROISSANT
   DE MOTS DE LYNDON.

2- CONSTRUCTION DE LA BASE DE PBWL.

3- EXPRESSION D'UN MOT DANS LA BASE DE PBWL.

4- EXPRESSION D'UN POLYNOME DANS LA BASE DE PBWL.

5- FIN.

Entrez votre choix termine par un <;> :
2;

Entrez le degre voulu suivi d'un <;>:
3;

Calcul de la base canonique de R<X> pour m= 2 et k= 3

La base desiree est dans la liste <pbw>. Voulez-vous l'afficher?<O/N>:
o;

$$pbw = [1, x0, x1, x0^{\langle 2\rangle}, [x0, x1], x1 \,.\, x0, x1^{\langle 2\rangle}, x0^{\langle 3\rangle}, [x0, [x0, x1]],$$

$$[x0, x1] \,.\, x0, [[x0, x1], x1], x1 \,.\, x0^{\langle 2\rangle}, x1 \,.\, [x0, x1], x1^{\langle 2\rangle} \,.\, x0, x1^{\langle 3\rangle}]$$

Tapez <c;> puis <Return> pour continuer:
c;

MENU PRINCIPAL

1- DECOMPOSITION D'UN MOT EN UN PRODUIT DECROISSANT DE MOTS DE LYNDON.

2- CONSTRUCTION DE LA BASE DE PBWL.

3- EXPRESSION D'UN MOT DANS LA BASE DE PBWL.

4- EXPRESSION D'UN POLYNOME DANS LA BASE DE PBWL.

5- FIN.

Entrez votre choix termine par un <;> :

```
3;
```

Entrez un mot suivi d'un <;>:

```
x0.x0.x1.x1.x0;
```

Votre mot se decompose en :

```
x1 . x1 . x0 . x0 . x0 + 4 (x1 . [x0, x1] . x0 . x0)
 + 2 (x1 . [x0, [x0, x1]] . x0) + 2 ([[x0, x1], x1] . x0 . x0)
 + 2 ([x0, x1] . [x0, x1] . x0) + [x0, [[x0, x1], x1]] . x0
```

Tapez <c;> puis <Return> pour continuer:

```
c;
```

MENU PRINCIPAL

1- DECOMPOSITION D'UN MOT EN UN PRODUIT DECROISSANT
DE MOTS DE LYNDON.

2- CONSTRUCTION DE LA BASE DE PBWL.

3- EXPRESSION D'UN MOT DANS LA BASE DE PBWL.

4- EXPRESSION D'UN POLYNOME DANS LA BASE DE PBWL.

5- FIN.

Entrez votre choix termine par un <;> :
4;

Entrez votre polynome termine par un <;>:
2*x0.x1.x0-x0.x1.x1;

Votre polynome se decompose en :

- x1 . x1 . x0 - 2 (x1 . [x0, x1]) + 2 (x1 . x0 . x0 + [x0, x1] . x0)

- [[x0, x1], x1]

Tapez <c;> puis <Return> pour continuer:
c;

MENU PRINCIPAL

1- DECOMPOSITION D'UN MOT EN UN PRODUIT DECROISSANT DE MOTS DE LYNDON.

2- CONSTRUCTION DE LA BASE DE PBWL.

3- EXPRESSION D'UN MOT DANS LA BASE DE PBWL.

4- EXPRESSION D'UN POLYNOME DANS LA BASE DE PBWL.

5- FIN.

Entrez votre choix termine par un <;> :

1;

Entrez le cardinal de votre alphabet:

2;

Votre alphabet est desormais :

[x0, x1]

```
          Entrez votre mot suivi d'un <;>:
x1.x1.x0.x1.x0.x0;


          Votre mot se decompose en:

  <2>                 <2>
x1    . {x0 . x1} . x0


          Tapez <c;> puis <Return> pour continuer:
c;


          Entrez votre mot suivi d'un <;>:
x1.x1.x0.x0.x1.x1.x1.x0;


          Votre mot se decompose en:

  <2>
x1    . {x0 . x0 . x1 . x1 . x1} . x0


          Tapez <c;> puis <Return> pour continuer:
c;


          Entrez votre mot suivi d'un <;>:
x0.x1.x0.x1.x0.x1;


          Votre mot se decompose en:

         <3>
{x0 . x1}


          Tapez <c;> puis <Return> pour continuer:
c;


          Entrez votre mot suivi d'un <;>:
x1.x0.x1.x0.x0.x1.x0;


          Votre mot se decompose en:

x1 . {x0 . x1} . {x0 . x0 . x1} . x0


          Tapez <c;> puis <Return> pour continuer:
c;
```

MENU PRINCIPAL

1- DECOMPOSITION D'UN MOT EN UN PRODUIT DECROISSANT
DE MOTS DE LYNDON.

2- CONSTRUCTION DE LA BASE DE PBWL.

3- EXPRESSION D'UN MOT DANS LA BASE DE PBWL.

4- EXPRESSION D'UN POLYNOME DANS LA BASE DE PBWL.

5- FIN.

Entrez votre choix termine par un <;> :

```
2;
```

Entrez le degre voulu suivi d'un <;>:

```
3;
```

Calcul de la base canonique de R<X> pour m= 2 et k= 3

La base desiree est dans la liste <pbw>. Voulez-vous l'afficher?<O/N>:

```
o;
```

```
                        <2>                          <2>    <3>
pbw =  [1, x0, x1, x0    , [x0, x1], x1 . x0, x1    , x0    , [x0, [x0, x1]],

                                        <2>                      <2>         <3>
  [x0, x1] . x0, [[x0, x1], x1], x1 . x0    , x1 . [x0, x1], x1     . x0, x1    ]
```

Tapez <c;> puis <Return> pour continuer:

```
c;
```

```
                    MENU PRINCIPAL

                    --------------


          1- DECOMPOSITION D'UN MOT EN UN PRODUIT DECROISSANT

             DE MOTS DE LYNDON.


          2- CONSTRUCTION DE LA BASE DE PBWL.


          3- EXPRESSION D'UN MOT DANS LA BASE DE PBWL.


          4- EXPRESSION D'UN POLYNOME DANS LA BASE DE PBWL.


          5- FIN.


          Entrez votre choix termine par un <;> :
3;


          Entrez un mot suivi d'un <;>:
x0.x1.x0.x1.x0.x1;


          Votre mot se decompose en :

x1 . x1 . x1 . x0 . x0 . x0 + 6 (x1 . x1 . [x0, x1] . x0 . x0)
 + 4 (x1 . x1 . [x0, [x0, x1]] . x0) + x1 . x1 . [x0, [x0, [x0, x1]]]
 + 4 (x1 . [[x0, x1], x1] . x0 . x0) + x1 . [[x0, [x0, x1]], [x0, x1]]
 + 7 (x1 . [x0, x1] . [x0, x1] . x0) + 4 (x1 . [x0, x1] . [x0, [x0, x1]])
 + x1 . [x0, [[x0, x1], x1]] . x0 + [[[x0, x1], x1], x1] . x0 . x0
 + 4 ([[x0, x1], x1] . [x0, x1] . x0) + [[x0, x1], x1] . [x0, [x0, x1]]
 + [[x0, x1], [[x0, x1], x1]] . x0 + [x0, x1] . [x0, x1] . [x0, x1]


          Tapez <c;> puis <Return> pour continuer:
c;
```

MENU PRINCIPAL

1- DECOMPOSITION D'UN MOT EN UN PRODUIT DECROISSANT
DE MOTS DE LYNDON.

2- CONSTRUCTION DE LA BASE DE PBWL.

3- EXPRESSION D'UN MOT DANS LA BASE DE PBWL.

4- EXPRESSION D'UN POLYNOME DANS LA BASE DE PBWL.

5- FIN.

Entrez votre choix termine par un <;> :
4;

Entrez votre polynome termine par un <;>:
2*x0.x1.x0+5*x1.x0.x1-3*x1.x0;

Votre polynome se decompose en :

5 (x1 . x1 . x0 + x1 . [x0, x1]) + 2 (x1 . x0 . x0 + [x0, x1] . x0)

- 3 (x1 . x0)

Tapez <c;> puis <Return> pour continuer:
c;

V. MACSYMA COMPUTATION OF LOCAL AND MINIMAL REALIZATION OF DYNAMICAL SYSTEMS OF WHICH GENERATING POWER SERIES ARE FINITE (A.C.M.A.R.)

MACSYMA COMPUTATION OF LOCAL MINIMAL REALIZATION OF DYNAMICAL SYSTEMS OF WHICH GENERATING POWER SERIES ARE FINITE ($\mathcal{ACMAR}$)

N. Oussous(*)

ABSTRACT

We present here a package of MACSYMA programs, allowing to "manipulate" words, and noncommutative power series over some finite alphabet.

On the base of works of M.Fliess and C.Reutenauer, concerning local realization of non linear dynamical systems, we present an algorithm allowing to compute the local and minimal realization of finite generating power series. We describe that algorithm in the algebraic computation language MACSYMA.

The programs in MACSYMA and some examples are given in annex.

SUMMARY

(*) Laboratoire d'Informatique Fondamentale de Lille.— U.A. 369 du C.N.R.S. – Université de LILLE I. — 59655 VILLENEUVE D'ASCQ CEDEX. **FRANCE.**

MACSYMA COMPUTATION OF LOCAL MINIMAL REALIZATION OF DYNAMICAL SYSTEMS OF WHICH GENERATING POWER SERIES ARE FINITE

0. INTRODUCTION.

We present here a package of MACSYMA programs, allowing to "manipulate" words, and noncommutative power series over some finite alphabet X.

This package contains, in particular, implementation of *shuffle product*, *left* and *right remainder*, production (up to some fixed degree k) of the *Lyndon basis* of the *free Lie algebra* $\mathcal{L}ie\langle X\rangle$, and the *canonical* "Poincaré-Birkhoff-Witt" basis of noncommutative polynomials over X.

As a development, we present in the some package an algorithm that computes the local and minimal realization of dynamical systems with finite generating power series. Lyndon words are used to compute the local coordinates.

I. DEFINITIONS AND NOTATIONS.

Let X be a finite and totally ordered set. The elements of X are called *letters* and the elements of the *free monoid* X^* generated by X are called *words*. The empty word is denoted ε.

We define a noncommutative power series S over $X = \{x_0, ..., x_{m-1}\}$ with coefficients in $\mathbb{R}$ as a map from X^* into $\mathbb{R}$. We denote $\langle S|w\rangle$ the image of w by S. The formal power series is denoted by a formal sum as follows :

$$S = \sum_{w \in X^*} \langle S|w\rangle w,$$

$\langle S|w\rangle$ is also called the *coefficient* of w in S. The set of noncommutative power series is an algebra denoted $\mathbb{R}\ll X \gg$.

The subset of X^* defined as below :

$$supp(S) = \{w \in X^* | \langle S|w\rangle \neq 0\}$$

is called the *support* of the power series S.

The power series with a *finite support* are called a *polynomials*. The set of all polynomials is an sub-algebra of $\mathbb{R}\ll X \gg$ denoted $\mathbb{R}\langle X\rangle$.

I.1. Left an right remainder of a noncommutative power series.

Let $S \in \mathbb{R}\ll X \gg$ be a noncommutative power series and let $u \in X^*$ be a word. We define and denote $S \triangleright u$ (resp. $u \triangleleft S$) (*cf* G.JACOB& N.OUSSOUS[4]) the *right* (resp. *left*) *remainder of the power series* S by the word u as follow :

$$S \triangleright u = \sum_{w \in X^*} \langle S|w\rangle w \triangleright u, \qquad (\text{resp. } u \triangleleft S = \sum_{w \in X^*} \langle S|w\rangle u \triangleleft w),$$

where

$$w \triangleright u = \begin{cases} v, & if \quad w = uv, \\ 0, & elsewhere, \end{cases} \qquad \left(\text{resp. } u \triangleleft w = \begin{cases} v, & if \quad w = vu, \\ 0, & elsewhere. \end{cases}\right)$$

I.2. Shuffle product.

Let $u, v, u', v' \in X^*$ be words, $x, y \in X$ be letters. We define (see, for example G.JACOB& N.OUSSOUS[4]) the *shuffle product* of u and v recursively as follows :

$$\begin{cases} u \sqcup \varepsilon = \varepsilon \sqcup u = u, \\ u \sqcup v = x(u' \sqcup v) + y(u \sqcup v'), \quad \textit{if} \quad u = xu' \textit{ and } v = yv'. \end{cases}$$

II. DYNAMICAL SYSTEMS AND GENERATING POWER SERIES.

We consider a system of the following form:

$$(\Sigma) \qquad \begin{cases} \dot{q}(t) = \sum_{i=0}^{m-1} u_i(t) Y_i(q) \quad \text{with} \quad u_0(t) \equiv 1 \text{ for all } t, \\ y(t) = h(q(t)). \end{cases}$$

where

$q \in Q, Q$ is an *$\mathbb{R}$-analytic connected variety* of dimension n,
$Y_i, 0 \leq i \leq m-1$, are *analytic vector fields*,
h, is a *$\mathbb{R}$-analytic function* called *observation*,
Y_i and h are defined in a neighbourhood of the given *initial state* $q(0)$,
u_i, $1 \leq i \leq m-1$, are *real and piecewise continuous functions*.

For small enough time and inputs, the output y is given by the PEANO-BAKER formula (*cf* M.FLIESS[3], C.REUTENAUER[10] or G.JACOB& N.OUSSOUS[4]) –the M. Fliess fundamental formula–:

$$y(t) = \sum_{w \in X^*} (Y_w \circ h)_{|_{q(0)}} \int_0^t \delta_u w$$

where $|_{q(0)}$ means evaluation in $q(0)$, $\int_0^t \delta_u w$ is the *iterated integral* defined recursively as follow:

$$\begin{cases} \text{if } w = \varepsilon, \text{ then } \int_0^t \delta_u \varepsilon = 1, \\ \text{if } w = x_i \in X, \text{ then } \int_0^t \delta_u x_i = \int_0^t u_i(\tau) d\tau, \quad 0 \leq i \leq n, \\ \text{if } w = v x_i, \text{ then } \int_0^t \delta_u w = \int_0^t \left(\int_0^\tau \delta_u v \right) u_i(\tau) d\tau. \end{cases}$$

(this definition is the symetric one of the definition given by M. FLIESS[2]).

In particular : $\int_0^t \delta_u x_0 = \int_0^t u_0(\tau) d\tau = t$, since $u_0(\tau) \equiv 1$. Y_w is the *differential operator* associated to the word w and defined (G.JACOB& N.OUSSOUS[4]) by :

$$\begin{cases} Y_w = Y_{i_1} \circ Y_{i_2} \ldots \circ Y_{i_k} \quad \text{for} \quad w = x_{i_1} x_{i_2} \ldots x_{i_k}, \\ Y_\varepsilon = \textit{Identity}. \end{cases}$$

The Input/Output behaviour of system (Σ) is completely defined by its *generating power series* in the *noncommutative variables* $x_0, \ldots, x_{m-1}$, given by the formula (M.FLIESS[3]) :

$$(II) \qquad g = \sum_{w \in X^*} (Y_w \circ h)_{|_0} w.$$

Let $\mathcal{L}ie<X>$ the *free-Lie-algebra* generated by X ($X=\{x_0,x_1,...,x_{m-1}\}$) in which the *Lie-bracket* is defined by :

$$[x_i,x_j]=x_ix_j-x_jx_i.$$

Lie polynomials are the linear combinaisons of *Lie words*. Lie words are either elements of X or brackets :

$$[P,Q]=PQ-QP,$$

where P and Q are Lie words.

DEFINITION. – *The Lie-Rank of a formal power series $S\in\mathbb{R}\ll X\gg$ is defined by :*

$$\begin{aligned}\mathcal{RL}(S)&=\dim S\triangleright\mathcal{L}ie<X>\\&=\dim[span\{S\triangleright P|P\in\mathcal{L}ie<X>\}],\end{aligned}$$

where "$S\triangleright P$" means the right remainder of S by the Lie polynomial P.

DEFINITION. – *Let $S\in\mathbb{R}\ll X\gg$. We define the Lie-Hankel matrix associated to S as the infinite array, denoted $\mathcal{LH}_S$, of which the lines are indexed by some totally ordered basis of $\mathcal{L}ie<X>$ and the columns are indexed by X^* (sorted for lexicographic by length order) such as :*

$$\mathcal{LH}_S(P_i,w)\ =<S|P_iw>=<S\triangleright P_i|w>.$$

We show easly that :

$$\mathcal{RL}(S)=Rank(\mathcal{LH}_S).$$

III. LYNDON WORDS AND LIE BASIS.

III.1. Order over words and conjugation classes.

Let X be a finite and totally ordered set. We totally order X^* as follows :

$\forall\, u,v\in X^*$,

$$u<v\ \text{ iff }\ \begin{cases}\text{ou}\begin{cases}(i)\ \exists\, w\neq\varepsilon\ \text{ such that }\ uw=v,\\(ii)\ \exists\, x,y,z\in X^*\ \text{ and }\ a,b\in X\ \text{ such that }\quad u=xay,\ \ v=xbz\ \text{ and }\ a<b.\end{cases}\end{cases}$$

So we have the *usual lexicographical order* on X^* (M.LOTHAIRE[6]). With this order, we have the following properties:

$$\begin{cases}(1)\ \ \forall w\in X^*,\quad u<v\iff wu<wv.\\(2)\ \ \text{Si }\ v\notin uX^*,\ \ \forall w,z\in X^*,\ u<v\implies uw<vz.\end{cases}$$

A word u is a *factor* of a word v if

$$\exists\, x,y\in X^*\ \text{ such that }\ v=xuy.$$

If $x=\varepsilon$ (resp. $y=\varepsilon$), we say that u is a *left* (resp. *right*) *factor* of v, *proper* if $y\neq\varepsilon$ (resp. $x\neq\varepsilon$).

Two words u and v are said to be *conjugate* if

$$\exists\, x,y\in X^*\ \text{ such that }\ u=xy\ \ and\ \ v=yx.$$

III.2. Lyndon words.

The definition and properties of Lyndon words can be found in M.LOTHAIRE[6], G.MELANÇON& C.REUTENAUER[7] or J.P.DUVAL[1].

DEFINITION. – *A word $w \in X^*$ is a Lyndon word if, and only if :*

$$\begin{cases} \text{(i) it is strictly smaller than any of its conjugates,} \\ ou \\ \text{(ii) it is strictly smaller than any of its proper right factors.} \end{cases}$$

Let L denote the set of Lyndon words over X.

PROPERTIES :

(1) Let $w \in L\backslash X$ and m its *longest proper right factor* in L. If $w = lm$, then $l \in L$ and $l < lm < m$. The couple $\sigma(w) = (l, m)$ is called the *standard factorisation* of w.

(2) $w \in L$ if and only if $\begin{cases} w \in X, \\ \text{or} \\ w = lm \text{ with } l, m \in L \text{ and } l < m. \end{cases}$

The last property gives us an algorithm to construct the Lyndon words up to a given degree. Some implementations of that algorithm were given in MACSYMA (N.OUSSOUS[8]).

III.3. Lyndon basis.

For more precisions, see G.VIENNOT[11]. We consider on $\mathcal{L}ie\langle X\rangle$ the *Lyndon basis* which is recursivly defined as follow :

$$\begin{cases} c(x) = x \quad for \quad x \in X, \\ c(w) = [c(l), c(m)], \quad for\ w \in L\backslash X,\ such\ that\ \sigma(w) = (l, m). \end{cases}$$

where the brackets are the Lie ones. This definition gives us an algorithm to construct the CHEN-FOX-LYNDON basis of the free Lie algebra.

IV. REALIZATION OF DYNAMICAL SYSTEMS.

Locally, the problem of realization can be expressed (*cf* M.Fliess[3]) as follow : Let an Input/Output behaviour given by its generating power series, is there a differential system like (Σ) which has the same generating power series? In positive case, describe it.

IV.1. Power series produced differentially.

DEFINITION[3]. – *The formal power series $g \in \mathbb{R}\ll X\gg$ is produced differentially if, and only if, there exists :*

(i) *$r \in \mathbb{N}$, and an homomorphism $\mathcal{Y}$ from X^* into the differential operator algebra over $\mathbb{R}[q_1, ..., q_r]$, such that : for all $x_i \in X$, $Y_i = \mathcal{Y}(x_i)$ is a formal vector field,*

(ii) *a commutative formal power series $h \in \mathbb{R}[q_1, ..., q_r]$ such that :*

$$\forall w \in X^* \quad < g|w >= Y_w \circ h_{|_0}$$

where $|_0$ means evaluation in $q_1 = ... = q_r = 0$.

We call the couple $(\mathcal{Y}, h)$ a *differential representation* of g, of dimension r.

From (II) it is obvious that:

g is the generating power series of a system like (Σ) if, and only if, g is produced differentially.

Thus:

The study of local realizations
is equivalent to
the study of differential representations

IV.2. Fliess theorem.

THEOREM[3]. – *The power series $g \in \mathbb{R}\ll X \gg$ is produced differentially if, and only if, its Lie-Rank, r, is finite. In this case, r is equal to the smallest dimension of all its differential representations. If $(\mathcal{Y}, h)$ and $(\mathcal{Y}', h')$ are two differential representation of dimension r of g, then there exists a continuous automorphism φ of $\mathbb{R}[q_1 ..., q_r]$ such that:*

$$\forall w \in X^*,\ \forall k \in \mathbb{R}[q_1, ..., q_r],\quad h' = \varphi(h) \text{ and } \varphi(Y_w \circ k) = Y'_w \circ \varphi(k).$$

(Y_w (resp. Y'_w) means the image of w by $\mathcal{Y}$ (resp. $\mathcal{Y}'$)).

The realization $(\mathcal{Y}, h)$, unique up to isomorphism, is said *minimal* or *reduced*. For us, the realization problem consists to compute the Lie-rank of the generating power series g and find some polynomials $q_1, ..., q_r$ such that g (in first noncommutative polynomial) can be expressed as a linear combinaison of the shuffles of the polynomials $q_1, ..., q_r$. We denote this new expression h. To define the homomorphism $\mathcal{Y}$, we compute the vector fields $Y_0, ..., Y_{m-1}$, which are the respective images of the letters $x_0, ..., x_{m-1}$ and are given by the following formula (M.FLIESS[3], G.JACOB& N.OUSSOUS[5] or C.REUTENAUER[10]) :

$$Y_i = \sum_{j=1}^{r} \theta_i^j(q_1, ..., q_r) \frac{\partial}{\partial q_j},\quad \theta_i^j(q_1, ..., q_r) \in \mathbb{R}[q_1, ..., q_r].$$

Elsewhere, if $k \in \mathbb{R}[q_1, ..., q_r]$ and if we denote $\eta(k)$ the coresponding generating power series, then:

$$\begin{aligned} \forall w \in X^*,\quad \forall v \in X^*,\quad < \eta(Y_w \circ k) | v > &= [Y_v \circ (Y_w \circ k)]_{|_0} \\ &= (Y_{vw} \circ k)_{|_0} \\ &= < \eta(k) | vw > \\ &= < w \triangleleft \eta(k) | v >, \end{aligned}$$

$$\text{Thus}\quad \eta(Y_w \circ k) = w \triangleleft \eta(k).$$

We show that the action of the differential operator Y_w on the (commutative) power series k is equivalent to the left remainder of the generating power series $\eta(k)$ by the word w.

In particular, $\eta(Y_i \circ q_j) = x_i \triangleleft \eta(q_j)$. Since $Y_i \circ q_j = \theta_i^j(q_1, ..., q_r)$, then we have a mean to compute $\theta_i^j(q_1, ..., q_r)$.

IV.3. Local coordinates.

We consider the sub-Lie-algebra of $\mathcal{L}ie < X >$, of codimension r, defined by (G.JACOB& N.OUSSOUS[4], C.REUTENAUER[10]) :

$$\mathcal{A}(g) = \{\ P \in \mathcal{L}ie < X > \ |\ g \triangleright P = 0\ \}$$

$\mathcal{A}(g)$ is generated by the Lie polynomials which annul the power series g.

We set:

$$\mathcal{V}(\mathcal{A}(g)) = \{\, Q \in \mathbb{R}\langle\langle X\rangle\rangle \mid Q \triangleright \mathcal{A}(g) = 0 \,\}.$$

Let $(P_i)_{i\geq 1}$ be Lie polynomials such that $P_1, ..., P_r$ is a basis of $\mathcal{L}ie\langle X\rangle$ modulo $\mathcal{A}(g)$, and $P_{r+1}, ..., P_n, ...$ is a basis of $\mathcal{A}(g)$. We define the polynomials $q_1, ..., q_r$ without constant term such that:

$$\text{(IV.3)} \quad \begin{cases} < q_j \triangleright P_i | \varepsilon > = \delta_{ij} & \text{for } i \leq r, \\ q_j \in \mathcal{V}(\mathcal{A}(g)). \end{cases}$$

where $< q_j \triangleright P_i | \varepsilon >$ is the coefficient of ε in the right remainder of q_j by the Lie polynomial P_i.

We know, with according to MELANÇON & REUTENAUER[7] or D.E.RADFORD[9], that the Lyndon words are a transcendance basis of the suffle algebra $\mathbb{R}\langle X\rangle$. Elsewhere, we have the following importante relation:

$$< l_j \triangleright P_i | \varepsilon >= \delta_{ij}$$

where l_j is a Lyndon word which corresponding to element P_i of Lie basis.

V. THE PROGRAM

Our program consists of two parts : the first part compute the local realization and the second part allows us to verify that this realization is correct.

The first part consists of one principal procedure which has two arguments and call four procedures. It is written in MACSYMA on a workstation SUN 3.

---

(1) CALYN(S, m) : the principal procedure. It has two arguments:

- S : a noncommutative polynomial (as a generating power series of a dynamical analytic system).
- m : a cardinal of the alphabet in which S is written.
- Output :
  - the system of *vector fields*, $\{Y_0, Y_1, ..., Y_{m-1}\}$,
  - the *observation* H.

This procedure calls four procedures which are discribed below.

(1.1) MATLYN(S, m) : has the same two arguments as CALYN.

- Output :
  - MatLieHan : the Lie-Hankel matrix associated to S.
  - r : the Lie-Rank of S.

(1.2) LYNCOORD(Mat) : has one argument :

- Mat : is the matrix constructed by the above procedure.
- Output :
  - a system of r local coordinates, $\{Q_1, ..., Q_r\}$, which are the polynomials

constructed over the Lyndon words.

(1.3) EXPRIME($Pol, Tab, ztab$) : has three arguments.

- Pol : is a noncommutative polynomial.
- Tab : is an array of Lie polynomials.
- $stab$: is an array of noncommutative polynomials $(Q_1, ..., Q_r)$.
- Output :
  - Pol : as a commutative polynomial over $Q_1, ..., Q_r$, with shuffle product.

This procedure is used to compute the observation and the components of vector fields.

(1.4) ACTIONX(Z) : has one argument.

- Z : is an array of local coordinates.
- Output :
  - A two-dimensional array of components (noncommutative polynomials) of vector fields.

This procedure computes, for each local coordinate, the left remainder by each letter.

(1.5) CHVECT($Matx$) : has one argument.

- $Matx$: is a two-dimensional array of components of vector fields produced by the above procedure.
- Output :
  - The system of vector fields $\{Y_0, ..., Y_{m-1}\}$.

The second part consists of one principal procedure which has three arguments.

(2) VERIFIER(Tab, H, n) : has two arguments :

- Tab : a two-dimensional array wich is produced by CHVECT and enclosing the components of vector fields,
- H : a commutative polynomial (the observation),
- n : is the Lie-Rank of S.
- Output :
  - The generating power series associated to this system of vector fields and the observation H.

BIBLIOGRAPHIE

[1] J.P.DUVAL. – Génération d'une section des classes de conjugaison et arbre des mots de Lyndon de longueur bornée. – *Publication du LITP Paris*, **88–20**, 1988.

[2] M.FLIESS. – Fonctionnelles causales non linéaires et indéterminées non commutatives . – *Bull. Soc. Math. France*, **109**, 1981, p. 3 – 40.

[3] M.FLIESS. – Réalisation locale des systèmes non linéaires, algèbres de Lie filtrées transitives et séries génératrices . – *Invent. Math.*, **71**, 1983, p. 521 – 537.

[4] G.JACOB & N.OUSSOUS. – Sur un résultat de REE: séries de Lie et algèbres de mélange. – *Publication du LIFL Lille*, **IT–103**, 1987.

[5] G.JACOB & N.OUSSOUS. – Réalisation locale et minimale des systèmes dynamiques non linéaires et mots de Lyndon. – *Publication du LIFL Lille*, **IT–140**, 1988.

[6] M.LOTHAIRE. – Combinatorics on words. – *Reading, Massachusetts*, 1983.

[7] G.MELANÇON & C.REUTENAUER. – Lyndon words, free algebras and shuffles. – *Publication du LITP Paris et UQAM Montreal, Quebec*, **87–63**, 1987.

[8] N.OUSSOUS. – Mots de Lyndon et Macsyma. – *à paraitre*, 1988.

[9] D.E.RADFORD. – A natural ring basis for the shuffle algebra and an application to group schemes . – *Journal of Algebra*, **58**, 1979, p. 432 – 454.

[10] C.REUTENAUER. – The local realisation of generating series of finite Lie rank. – *Algebraic and Geometric Methods In Nonlinear Control Theory*, **eds.** M.FLIESS and M.HAZEWINKEL, 1986, p. 33–43.

[11] G.VIENNOT. – Algèbres de Lie libres et monoïdes libres . – *Lecture Notes In Mathematics, Springer-Verlag*, 691, 1978.

VI. COMPLEMENT A.C.M.A.R.

COORDONNES LOCALES ET MOTS DE LYNDON

1. Introduction.

Dans ce complément nous présentons la méthode de calcul des coordonnées locales utilisant les mots de Lyndon. Ensuite, nous donnons un algorithme qui permet de calculer l'expression du polynôme à réaliser (quand elle existe) comme combinaison linéaire des mélanges des coordonnées locales. A la fin, nous donnons un exemple simple mais, qui présente toutes les difficultés que nous avons résolues.

2. Calcul des coordonnées locales.

Nous disposons de la matrice de Lie-hankel. Le calcul du rang nous permet d'avoir le nombre de lignes linéairement indépendantes de cette matrice.

A la série S nous associons la sous-algèbre de Lie $\mathcal{A}(S)$ définie par:

$$\mathcal{A}(S) = \{P \in \mathcal{L}ie\langle X \rangle \mid S \triangleright P = 0\}.$$

On considère la sous-algèbre de $\mathbb{R}\ll X \gg$ défini par:

$$\mathcal{V}(\mathcal{A}(S)) = \{ Q \in \mathbb{R}\langle X \rangle \mid Q \triangleright \mathcal{A}(S) = 0 \},$$

et on se propose de chercher une base de cette sous-algèbre.

Pour des séries polynomiales, $\mathcal{A}(S)$ est engendrée par des polynômes de Lie P_i vérifiant $S \triangleright P_i = 0$.

On suppose que l'on a n lignes non nulles ($n \geq r$). Il existe donc des relations de dépendance entre les lignes de la matrice de Lie-Hankel. On aura donc $n-r$ relations de dépendance : $F_1, ..., F_{n-r}$. Ces relations vérifient : $S \triangleright F_j = 0$, pour $1 \leq j \leq n-r$.

On doit trouver des polynômes $Q_1, ..., Q_r$ qui soient des combinaisons linéaires de mélanges de $l_1, ..., l_p$, où les l_i représentent les mots de Lyndon de degré inférieur ou égal au degré de S.

Il faut que les polynômes (Q_j), $1 \leq j \leq r$, soient dans $\mathcal{V}(\mathcal{A}(S))$ et donc, il faut qu'ils soient annulés par $\mathcal{A}(S)$. Soit PL_i le polynôme de Lie correspondant au mot de Lyndon de plus haut degré dans l'expression de Q_i. Alors les Q_j doivent vérifier :

$$< PL_i | Q_j >= \delta_{ij}, \qquad 1 \leq i, j \leq r.$$

où $< PL_i | Q_j >$ signifie le *coefficient du mot vide* dans l'expression du simplifié à gauche de Q_j par le polynôme PL_i. Pour avoir un système de générateurs de $\mathcal{A}(S)$, on prend les éléments suivants :

- Les éléments de la base de Lie qui donnent un simplifié à gauche de S nul,
- les relations $(F_i)_{1\leq i\leq n-r}$ ci-dessus.

Pour calculer les polynômes Q_j on procède de la manière suivante :

1) Si $n = r$, et donc il n'y a pas de relations de dépendance entre les lignes de la matrice de Lie-Hankel, alors on prend les polynômes (m_j), $1 \leq j \leq r$, qui ne sont rien d'autre que les mots de Lyndon correspondant aux polynômes de Lie indiçant les lignes de la matrice.

2) Sinon, c'est à dire, on a des relations de dépendance : $F_1, ..., F_{n-r}$, alors on cherche les relations duales sur les mots de Lyndon en déterminant une base du noyau de l'application linéaire F définie par les relations $F_1, F_2, ..., F_{n-r}$ comme suit :

$$F = \begin{pmatrix} < F_1|. > \\ < F_2|. > \\ \vdots \\ < F_{n-r}|. > \end{pmatrix} : \mathbb{R}^p \longrightarrow \mathbb{R}^{p-r},$$

$$x = (x_1, x_2, ..., x_p) \longmapsto F(x).$$

où p est le nombre de mots de Lyndon de degré plus petit ou égal au degré du plus grand polynôme de Lie n'annulant pas S.

On sait que $\dim(Ker(F)) \geq r$. En plus, si les F_i sont "normalisées", c'est-à-dire, si le coefficient du polynôme de plus haut degré est égal à 1, alors F sera surjective car sa matrice :

$$MF = \begin{pmatrix} \times & \times & \times & 1 & 0 & \dots & 0 \\ \times & \times & \times & \times & 1 & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ \times & \times & \times & \times & \times & \dots & 1 \end{pmatrix}$$

est de rang r et donc $\dim(Ker(F)) = r$.

On obtient r vecteurs linéairement indépendants $v_1, ..., v_r$. On calcule alors les polynômes (m_j) en effectuant les produits scalaires :

$$m_j = (l_1, \ ..., \ l_n).v_j, \qquad 1 \leq j \leq r.$$

Une fois qu'on a les m_j, par le (1) ou par le (2), il faut vérifier s'ils appartiennent à la sous-algèbre $\mathcal{V}(\mathcal{A}(S))$. Sinon, c'est-à-dire si $m_i \triangleright P = t \neq 0$, où P est un élément de $\mathcal{A}(S)$, alors on cherche un polynôme T qui soit combinaison linéaire des mélanges de mots de Lyndon et qui vérifie :

$$(T + m_i) \circ P = 0 \quad \Longleftrightarrow \quad T \circ P = -t.$$

Pour cela, on considère tous les mots de Lyndon de degré inférieur ou égal à $d = \deg(t) + \deg(P)$. Ensuite, on calcule tous les mélanges des mots retenus et

dont le degré reste inférieur ou égal à d. Supposons qu'on ait obtenu s polynômes $ml_1, ml_2, ..., ml_s$, on pose alors :

$$T = \sum_{i=1}^{s} a_i * ml_i,$$

puis, on calcule le simplifié de T, à gauche par le polynôme P, et on résout l'équation :

$$T \triangleright P = -t.$$

On peut avoir plusieurs solutions, on garde une parmi celles dont la somme avec m_i est non nulle, appartiennent à $\mathcal{V}(\mathcal{A}(S))$, et vérifient la condition :

$$< P_i | (m_j + T) >= \delta_{ij}.$$

Les polynômes (Q_j), $1 \leq j \leq r$, sont alors définis par :

$$Q_j = m_j + T, \qquad 1 \leq j \leq r,$$

et seront appelés *coordonnées locales.*

On sait que les mots de Lyndon forment une base de transcendance de l'algèbre de mélange (G.MELANÇON& C.REUTENAUER OU RADFORD). On peut donc exprimer S comme combinaison linéaire des mélanges des mots de Lyndon $l_1, ..., l_n$. Chacun des polynômes Q_j peut s'écrire, à un changement de notations près, sous la forme $l_j + Q'_j$ où l_j est le mot de Lyndon de plus haut degré dans l'expression de Q_j. On retient donc les polynômes de Lie P_j correspondant aux mots l_j.

Pour avoir l'expression de la *fonction d'observation*, on exprime la série S en fonction des mélanges des Q_j et pour cela, on utilise l'algorithme suivant :

- $T \leftarrow S$, $H \leftarrow 0$,
- $j \leftarrow r$,
- $fact1 \leftarrow fact \leftarrow 1$,
- **TQ** ($j \neq 0$ et $S \neq 0$) **FAIRE**
  - $n \leftarrow 0$,
  - **TQ** $T \triangleright P_j \neq 0$ **FAIRE**
    - $T \leftarrow T \triangleright P_j$,
    - $n \leftarrow n+1$,
  - **FTQ**
  - cste $\leftarrow$ *terme constant de* T,
  - **SI** *cste* $\neq 0$ **ALORS**
    - $S \leftarrow S - \frac{cste}{n!} * (fact1 \sqcup q_j{}^{\sqcup n})$,

- $H \leftarrow H + \frac{cste}{n!} * (fact * {q_j}^n)$,
- $fact1 \leftarrow fact \leftarrow 1$,
- $T \leftarrow S$,
- $j \leftarrow j+k, \quad k \leftarrow 0$,

• **SINON**

- $fact1 \leftarrow \frac{1}{n!} * (fact1 \sqcup {q_j}^{\sqcup n})$,
- $fact \leftarrow \frac{1}{n!} * (fact * {q_j}^n)$,
- $j \leftarrow j-1, \quad k \leftarrow k+1$,

◇ **FTQ**

La petite boucle se termine puisqu'au bout d'un certain nombre de simplifications le reste devient nul. La grande boucle s'arrête aussi puisque le nombre de polynômes P_j est fini.

Dans ce qui suit, je donne un exemple qui illustre ce complément.

3. Exemple.

$$X = \{x_0, x_1\}, \qquad S = x_1.x_0.x_1 + x_0.x_1.x_0.x_1$$

On produit les éléments de X^* jusqu'au degré 4, en respectant l'ordre lexicographique par longueur. Ensuite, on produit la base de Lie jusqu'à l'ordre 4 :

$$BLie = \{x_0, x_1, [x_0,x_1], [x_0,[x_0,x_1]], [[x_0,x_1],x_1], [x_0,[x_0,[x_0,x_1]]],$$

$$[x_0,[[x_0,x_1],x_1]], [[[x_0,x_1],x_1],x_1]\}.$$

Les polynômes de la base de Lie correspondants aux lignes non nulles sont :

$$\begin{aligned}
P_1 &= x_0 \\
P_2 &= x_1 \\
P_3 &= [x_0,x_1] &&= x_0.x_1 - x_1.x_0 \\
P_4 &= [x_0,[x_0,x_1]] &&= x_0^{<2>}.x_1 - 2x_0.x_1.x_0 + x_1.x_0^{<2>} \\
P_5 &= [[x_0,x_1],x_1] &&= x_0.x_1^{<2>} - 2x_1.x_0.x_1 + x_1^{<2>}.x_0 \\
P_6 &= [x_0,[[x_0,x_1],x_1]] &&= x_0^{<2>}.x_1^{<2>} - 2x_0.x_1.x_0.x_1 + 2x_1.x_0.x_1.x_0 + x_1^{<2>}.x_0^{<2>}
\end{aligned}$$

Le rang de Lie de cette série est : 4.

Les relations de dépendance sont :

$$\begin{cases} F_1 = P_6 - P_5, \\ F_2 = \frac{1}{2}P_4 - P_3 + P_2. \end{cases}$$

Les polynômes de Lie qui annulent S sont :

$$\begin{cases} F_1,\ F_2, \\ [x_0,[x_0,[x_0,x_1]]],\ [[[x_0,x_1],x_1],x_1], \\ \textit{et tous les polynomes de Lie de degré supérieur à 4.} \end{cases}$$

Aux deux relations F_1 et F_2 ci-dessus, on associe l'application linéaire F définie comme suit :

$$F = \begin{pmatrix} < F_1|.> \\ < F_2|.> \end{pmatrix} \ : \ \mathbb{R}^6 \longrightarrow \mathbb{R}^{6-4}$$

La matrice associée à cette application linéaire est :

$$MF = \begin{pmatrix} 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 1 & -1 & \frac{1}{2} & 0 & 0 \end{pmatrix}$$

Le noyau de F est de dimension 4 donc on peut trouver 4 vecteurs linéairement indépendants qui engendrent $Ker(F)$. Ces vecteurs sont les suivants :

$$v_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad v_2 = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad v_3 = \begin{pmatrix} 0 \\ -\frac{1}{2} \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad v_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}.$$

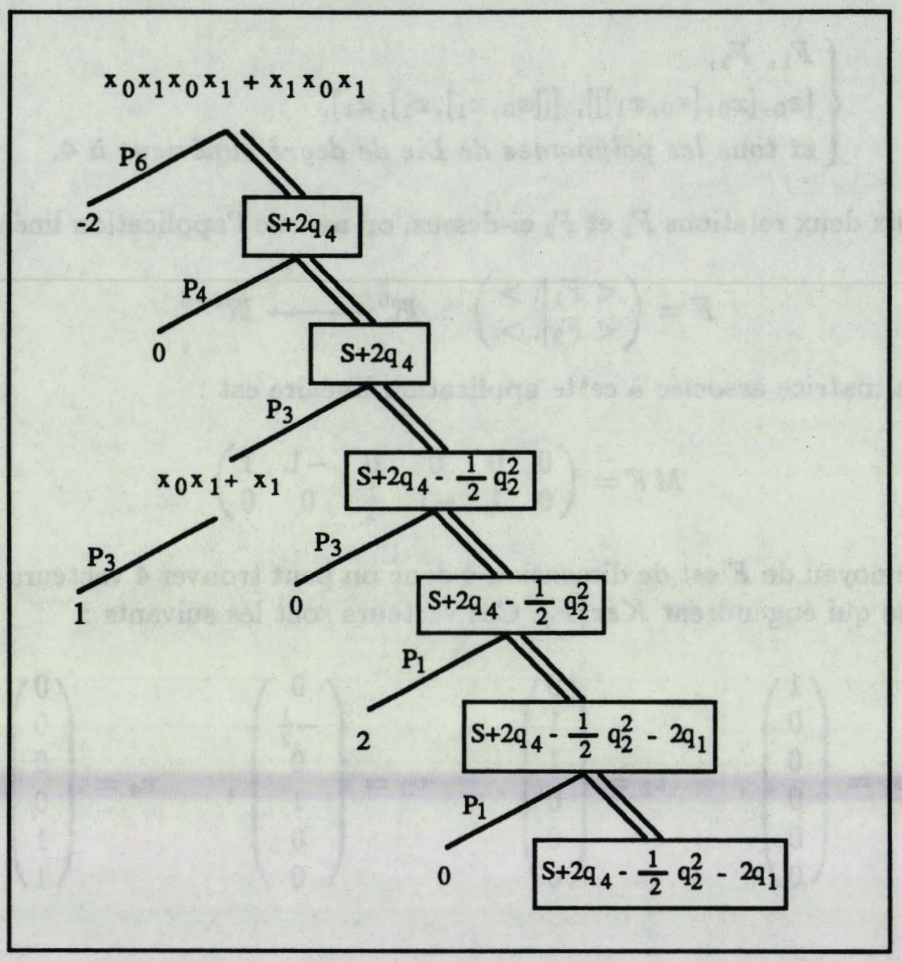
d'où :

$$\begin{aligned} m_1 &= (l_1\ l_2\ l_3\ l_4\ l_5\ l_6).v_1 = x_0, \\ m_2 &= (l_1\ l_2\ l_3\ l_4\ l_5\ l_6).v_2 = x_0.x_1 + x_1 \\ m_3 &= (l_1\ l_2\ l_3\ l_4\ l_5\ l_6).v_3 = x_0^{<2>}.x_1 - \frac{1}{2}x_1 \\ m_4 &= (l_1\ l_2\ l_3\ l_4\ l_5\ l_6).v_4 = x_0^{<2>}.x_1^{<2>} + x_0.x_1^{<2>} \end{aligned}$$

ces polynômes doivent être annulées par les annulateurs de S. Ce qui n'est pas le cas pour m_4 puisque $m_4 \triangleright F_2 = -\frac{1}{2}x_1$. Il faut donc trouver un polynôme T qui soit combinaison linéaire des mélanges des mots de Lyndon et dont le simplifié par F_2 soit égal à $\frac{1}{2}x_1$. Il en existe plusieurs, mais on en garde le polynôme $T = \frac{1}{4}l_2\omega l_2 + l_1 = \frac{1}{2}x_1^2 + x_0$. On complète m_4 par T. On aura alors :

$$\begin{aligned} q_i &= m_i, \quad 1 \leq i \leq 3, \\ q_4 &= x_0^{<2>}.x_1^{<2>} + x_0.x_1^{<2>} + \frac{1}{2}x_1^{<2>} + x_0 \end{aligned}$$

Expression de S en fonction des mélanges des polynômes $(q_i)_{1\leq i\leq 4}$. On retient les polynômes de Lie correspondant aux mots de Lyndon de plus haut degré apparaissant dans les $(q_i)_{1\leq i\leq 4}$. Dans notre exemple, on retiendra : P_1, P_3, P_4 et P_6. On utilise l'algorithme précédent et on obtient :



On obtient alors :

$$H = -2q_4 + \frac{1}{2}q_2^{\omega 2} + 2q_1.$$

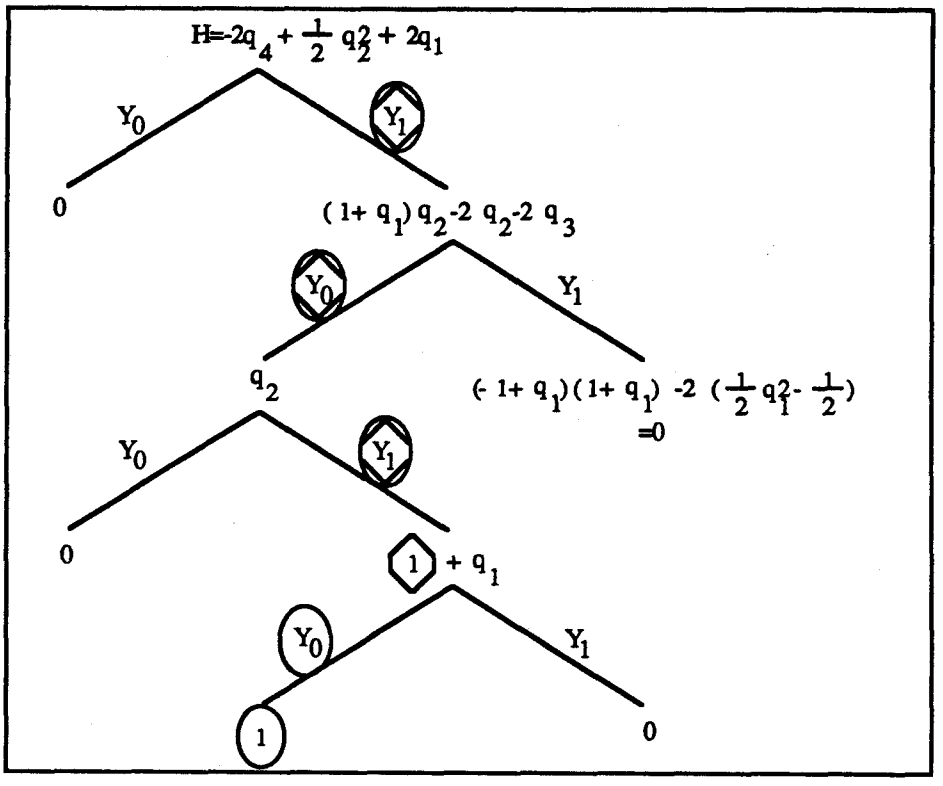
Calcul des champs de vecteurs :

	$x_0\triangleleft$	$x_1\triangleleft$	
$q_1 = x_0$	1	0	
$q_2 = x_0.x_1 + x_1$	0	$1 + x_0$	$= 1 + q_1$
$q_3 = x_0^{<2>}.x_1 - \frac{1}{2}x_1$	0	$1 - 2x_0^{<2>}$	$= -\frac{1}{2} + \frac{1}{2}q_1^{\omega 2}$
$q_4 = x_0^{<2>}.x_1^{<2>} + x_0.x_1^{<2>} + \frac{1}{2}x_1^{<2>} + x_0$	1	$x_0^{<2>}.x_1 + x_0.x_1 + \frac{1}{2}x_1 = q_2 + q_3$	

Les champs de vecteurs sont donc :

$$\begin{cases} Y_0 = \dfrac{\partial}{\partial q_1} + \dfrac{\partial}{\partial q_4}, \\ Y_1 = (1+q_1)\dfrac{\partial}{\partial q_2} + (-\dfrac{1}{2} + \dfrac{1}{2}q_1^2)\dfrac{\partial}{\partial q_3} + (q_2+q_3)\dfrac{\partial}{\partial q_4}. \end{cases}$$

En partant de la fonction d'observation H et des champs de vecteurs Y_0 et Y_1, nous allons chercher la série génératrice correspondant à cette réalisation (en principe, si la réalisation que nous avons calculée est correcte, nous devons retrouver la série de départ). L'algorithme de calcul est résumé par l'arbre suivant :



On trouve alors :

$$(Y_0 \circ Y_1 \circ Y_0 \circ Y_1 \circ H)_{|_0} = 1 \implies < S|x_0.x_1.x_0.x_1 >= 1,$$
$$(Y_1 \circ Y_0 \circ Y_1 \circ H)_{|_0} = 1 \implies < S|x_1.x_0.x_1 >= 1.$$

D'où :

$$S = x_1.x_0.x_1 + x_0.x_1.x_0.x_1.$$

VII. REALISATION LOCALE ET MINIMALE DES SYSTEMES DYNAMIQUES NON LINEAIRES ET MOTS DE LYNDON

TABLE DES MATIERES

RÉALISATION LOCALE ET MINIMALE DES SYSTÈMES DYNAMIQUES NON LINÉAIRES ET MOTS DE LYNDON

G. Jacob(*) *& N. Oussous* (*)

ABSTRACT :

This paper attempts to present an algorithmic computation of the minimal realization of analytical dynamic systems. For that, we use Lyndon words as transcendance basis of the formal power series algebra (for the shuffle product).

First, we recall the classical notions of Hankel and Lie-Hankel matrix of generating series and its Lie rank. We explain the natural correspondance between the differential geometry of the observations and the formal power series algebra. Then, we show that all geometrical computations in the observation space can be transposed in the formal power series algebra. Finally, we show that each dynamical system of finite generating series admits a polynomial minimal realization. We use a fact that any finite generating series can be written as a polynomial on Lyndon words for the shuffle product.

KEYWORDS :

Nonlinear systems, minimal realization, generating series, symbolic calculus, differential geometry.

0. INTRODUCTION.

Le problème de la réalisation a fait l'objet de plusieurs travaux dont nous citons R.E.Kalman[16] pour les systèmes linéaires, P.D'Alessandro, A.Isidori and A.Ruberti[3], M.Fliess[5], G.Jacob[12] et J.H.Sussmann[23] pour les systèmes non linéaires particuliers dits bilinéaires. Et enfin B.Jakubczyk[16], H.J.Sussmann[24] pour les systèmes non linéaires généraux avec solutions régulières quel que soit le temps et quel que soit l'entrée.

Chacun des auteurs cités a utilisé des moyens plus au moins sophistiqués pour montrer l'existence et l'unicité de la réalisation. Nous retenons les travaux de M.Fliess[9], repris par C.Reutenauer[22] dans lesquels tous les deux ont utilisé les séries formelles comme outil principal pour montrer l'existence et l'unicité de la réalisation locale minimale des systèmes dynamiques non linéaires.

Dans ce papier, nous commençons par la présentation d'un certain nombre d'outils de manipulation des séries formelles non commutatives (ces outils déjà étudiés dans [13] et implémentés dans [14])). Ensuite, nous exposons la correspondance entre l'aspect géométrique des systèmes dynamiques non linéaires et l'algèbre des séries formelles. Nous rappelons dans le paragraphe IV des définitions et certaines propriétés des mots de Lyndon dont une étude plus approfondie est faite par J.P.Duval[4], M.Lothaire[18] et G.Melançon & C.Reutenauer[20]. Dans le dernier paragraphe nous rappelons les résultats essentiels sur la réalisation locale des systèmes dynamiques non linéaires et nous montrons qu'en utlisant comme base de $\mathcal{L}ie\langle X\rangle$ la base de CHEN-FOX-LYNDON et le fait que les mots de Lyndon forment une base de transcendance de l'algèbre des séries formelles munie du mélange (G.Melançon & C.Reutenauer[20] ou D.E.Radford[21]) nous permettent de réaliser les systèmes dynamiques non linéaires de série génératrice finie. Cette réalisation a été implémentée sous le système de calcul formel MACSYMA[19] dans [14] et [15].

Dans ce cas particulier, nous calculons complètement la réalisation.

(*) Laboratoire d'Informatique Fondamentale de Lille.– U.A. 369 du C.N.R.S.– Université de LILLE I.– **59655 VILLENEUVE D'ASCQ CEDEX. FRANCE.** Tel. 20 43 47 18 ou 20 43 42 57.

I. POLYNOMES, SERIES FORMELLES ET AUTOMATES SYNTAXIQUES.

Soit $X = \{x_0, x_1, ..., x_{m-1}\}$, un ensemble de *lettres* appelé *alphabet*. Soit X^* le *monoïde libre* engendré par X. Un élément de X^* est appelé *mot*. Le mot *vide*, noté ε, est le mot qui ne contient aucune lettre. La *longueur* d'un mot w, notée $|w|$, est l'entier représentant le nombre de lettres qui le constituent (ainsi $|\varepsilon| = 0$).

I.1. Polynômes et séries formelles.

Une *série formelle* S est une application de X^* dans $\mathbb{R}$, qui à tout mot w associe $S(w)$, noté $< S|w >$, et appelé *coefficient* du mot w dans la série S; elle sera notée comme une somme infinie $\sum_{w \in X^*} < S|w > w$.

L'ensemble des séries formelles forme une algèbre qui sera notée dans la suite $\mathbb{R}\ll X \gg$.

L'*ordre* d'une série formelle S, noté $\omega(S)$, est défini (*cf* J.Berstel & C.Reutenauer[2] ou G.Jacob & N.Oussous[13]) par:

$$\omega(S) = \begin{cases} \inf\{\ |w|\ |\ < S|w > \neq\ 0\ \}, & \text{si } S \neq 0, \\ +\infty, & \text{si } S = 0. \end{cases}$$

Le *support* d'une série S est le langage:

$$\mathrm{Supp}(S) = \{\ w \in X^*\ |\ < S|w > \neq 0\ \}.$$

Un *polynôme* est une série formelle de support fini. On notera l'ensemble des polynômes $\mathbb{R}<X>$, c'est une sous-algèbre de $\mathbb{R}\ll X \gg$.

Le *degré* d'un polynôme $P \in \mathbb{R}<X>$ est défini par:

$$\deg(P) = \begin{cases} -\infty, & \text{si } P = 0, \\ \sup\{|w| \text{ avec } w \in \mathrm{Supp}(P)\}, & \text{si } P \neq 0. \end{cases}$$

On notera $\mathcal{L}ie\ < X >$, la $\mathbb{R}$-algèbre de Lie libre engendrée par X, dont $\mathbb{R} < X >$ est la $\mathbb{R}$-algèbre enveloppante universelle. Un élément de $\mathcal{L}ie < X >$ sera appelé polynôme de Lie, c'est une combinaison linéaire de mots de Lie et un mot de Lie est défini par:

$$[P, Q] = PQ - QP,$$

où P et Q sont des mots de Lie. Le produit PQ représente le produit de concaténation (produit associatif).

I.2. Produit de mélange.

Le produit de mélange de deux mots (*cf* G.Jacob & N.Oussous[13]) est défini récursivement par :

$$\begin{cases} \forall w \in X^*, & w \sqcup \varepsilon = \varepsilon \sqcup w = w, \qquad (\varepsilon \text{ étant le mot vide}). \\ \forall u, v \in X^*,\ \forall x, y \in X, & (xu) \sqcup (yv) = x\,(u \sqcup (yv)) + y\,((xu) \sqcup v). \end{cases}$$

On prolonge ce produit aux séries en posant, pour $S, T \in \mathbb{R}\ll X \gg$:

$$S \sqcup T = \sum_{u,v \in X^*} < S|u >< T|v > u \sqcup v.$$

Ce produit est commutatif et associatif.

I.3. Calcul de résiduels. (*cf* G.Jacob & N.Oussous[13]).

DEFINITION I.3.1. – *Soit $S \in \mathbb{R}\ll X \gg$. On pose, pour tout $u \in X^*$,*

$$u \triangleleft S = \sum_{w \in X^*} < S|wu > w \quad (\textit{resp. } S \triangleright u = \sum_{w \in X^*} < S|uw > w);$$

que l'on appellera le simplifié de S à droite (resp. à gauche) par u.

En d'autres termes, on a, pour tout $w \in X^*$:

$$< u \triangleleft S|w > = < S|wu >, \quad \text{et} \quad < S \triangleright u|w > = < S|uw > .$$

Remarques:

1. $\forall S \in \mathbb{R}\ll X\gg, \quad \varepsilon \triangleleft S = S \triangleright \varepsilon = S.$

2. Par définition du support d'une série et d'après la définition I.3.1, on peut voir facilement que:

$$\mathrm{Supp}(u \triangleleft S) = \{\ w \in X^* \mid wu \in \mathrm{Supp}(S)\ \} \quad \text{et} \quad \mathrm{Supp}(S \triangleright u) = \{\ w \in X^* \mid uw \in \mathrm{Supp}(S)\ \}.$$

I.4. $\mathbb{R}$-automates et matrice de Hankel.

DEFINITION I.4.1. – *On appelle $\mathbb{R}$-automate, le quadruplet $(\mathcal{E}, q_0, \mathcal{F}, *)$ tel que:*

$$\begin{cases} \mathcal{E} & \textit{est l'espace vectoriel des états,} \\ q_0 & \textit{est le vecteur état initial,} \\ \mathcal{F} & \textit{est une forme linéaire (dite d'observation) définie sur } \mathcal{E}, \\ * & \textit{est une action de } X^* \textit{ sur } \mathcal{E}, \textit{telle que l'application } q \longmapsto w * q, \\ & \textit{est une application linéaire de } \mathcal{E} \textit{ dans } \mathcal{E}. \end{cases}$$

DEFINITION I.4.2 [12]. – *Soit $S \in \mathbb{R} \ll X \gg$, une série formelle. On appelle automate syntaxique (à gauche) de S, le $\mathbb{R}$-automate $(\mathcal{E}, S, \mathcal{F}, \triangleleft)$ tel que:*

$$\begin{cases} \mathcal{E} = \mathrm{span}\{\ w \triangleleft S \mid w \in X^*\ \}, \\ S \quad \textit{est l'état initial,} \\ \mathcal{F} \quad \textit{est définie par} \quad \mathcal{F}(S) = \langle S|\varepsilon\rangle, \\ \triangleleft\ :\ X^* \times \mathcal{E} \longrightarrow \mathcal{E}, \\ \qquad (w, T) \longmapsto w \triangleleft T. \end{cases}$$

C'est l'automate des résiduels. Il est complètement accessible et complètement observable au sens des automates. C'est donc le $\mathbb{R}$-automate minimal reconnaissant S. Cet automate est en général de rang infini.

DEFINITION I.4.3 [5]. – *Soit $S \in \mathbb{R}\ll X\gg$ une série formelle. La matrice de Hankel associée à la série S est un tableau infini, noté $\mathcal{H}_S$, dont les lignes et les colonnes sont indicées par les éléments du monoïde libre X^* et tel que le terme se trouvant à la ligne u et à la colonne v est le coefficient $\langle S|uv\rangle$ du mot uv dans S.*

DÉFINITION I.4.4 [5]. – *On appelle rang de Hankel de la série S, et on note $\mathcal{RH}_S$, le rang de sa matrice de Hankel $\mathcal{H}_S$, (si ce rang est fini) et $+\infty$ sinon.*

Le rang de Hankel de S est en général infini. Si il est fini, alors S est *rationnelle* (*cf* M.Fliess[5] ou G.Jacob[12]).

Les lignes (resp. colonnes) de $\mathcal{H}_S$ engendrent un $\mathbb{R}$-espace vectoriel de dimension égale au rang de $\mathcal{H}_S$. On note $\mathcal{L}_u$ la ligne de $\mathcal{H}_S$ indicée par le mot u. On a (*cf* G.Jacob & N.Oussous[13]):

$$\forall w \in X^*, \qquad \mathcal{L}_u(w) = \langle S|uw\rangle = \langle S \triangleright u|w\rangle,$$

donc $\qquad \mathcal{L}_u = S \triangleright u.$

On en déduit que la ligne $\mathcal{L}_u$ de $\mathcal{H}_S$ contient exactement les coefficients de la série résiduelle $S \triangleright u$, et représente un état de l'automate minimal. On a donc

$$\mathcal{RH}_S = \dim\ span\{\ \mathcal{L}_u \mid u \in X^*\ \}.$$

Soit $P \in \mathbb{R}\langle X\rangle$. On définit la ligne $\mathcal{L}_P$ de $\mathcal{H}_S$ par:

$$\mathcal{L}_P = (\langle S|Pu\rangle)_{u\in X^*} = (\langle S \triangleright P|u\rangle)_{u\in X^*}.$$

Cette ligne contient donc les coefficients de la série résiduelle $S \triangleright P$. Et on a:

$$\mathcal{L}_P = \sum_{v\in X^*} \langle P|v\rangle\, \mathcal{L}_v.$$

Remarque. – Tout ce qu'on vient d'énoncer à propos des lignes de $\mathcal{H}_S$ peut être répété pour ses colonnes et les simplifiés de S à droite.

DÉFINITION I.4.5. – *Soit* $S \in \mathbb{R}\ll X \gg$. *On définit la matrice de Lie-Hankel associée à* S, *comme étant un tableau infini, noté* $\mathcal{LH}_S$, *dont les lignes sont indicées par une base totalement ordonnée de* $\mathcal{L}ie < X >$ *et dont les colonnes sont indicées par* X^*, *ordonné par l'ordre lexicographique par longueur, et défini par:*

$$\mathcal{LH}_S(P_i, w) = < S|P_i w > = < S \triangleright P_i|w > .$$

DÉFINITION I.4.6. – *On appelle rang de Lie de* S, *et on note* $\mathcal{RL}_S$, *le rang de sa matrice de Lie-Hankel, si celui-ci est fini et* $+\infty$ *sinon.*

On a donc (*cf* M.Fliess[9]) $\mathcal{RL}_S = \dim \text{span}\{\mathcal{L}_P \mid P \in \mathcal{L}ie < X >\}$.

II. CHAMPS DE VECTEURS ET SYSTEMES DYNAMIQUES.

On considère un système analytique de la forme:

$$(\Sigma) \quad \begin{cases} \dot{q}(t) = \sum_{i=0}^{m-1} u_i(t) Y_i(q) & \text{avec } u_0(t) \equiv 1 \text{ pour tout } t, \\ y(t) = h(q(t)). \end{cases}$$

où

$q \in Q$, Q une variété $\mathbb{R}$-analytique connexe de dimension N,
Y_i, $0 \leq i \leq m-1$, sont des champs de vecteurs analytiques,
h, fonction d'observation, est une fonction analytique réelle,
Les Y_i, et h sont définis dans un voisinage de $q(0)$, où $q(0)$ est donné,
u_i, $1 \leq i \leq m-1$, est une fonction réelle continue par morceaux. On posera $u = (u_0, u_1, ..., u_{m-1})$ qui sera appelée fonction d'entrée.

II.1. Formule fondamentale.

A chaque entrée u_i, on associe une *lettre* x_i, $(0 \leq i \leq m-1)$. L'ensemble de toutes les lettres $X = \{x_0, x_1, ..., x_{m-1}\}$ est appelé *alphabet de commandes.* Soit X^* le monoïde libre engendré par X.

On considère l'homomorphisme $\mathcal{Y}$ défini de X^* dans l'ensemble $\mathcal{D}$ des opérateurs différentiels qui à toute lettre x_i de X associe le champ de vecteurs $\mathcal{Y}(x_i) = Y_i$. Ainsi pour un mot $w \in X^*$, l'opérateur différentiel $\mathcal{Y}(w)$, que l'on note Y_w, est défini par:

$$\begin{cases} \mathcal{Y}(w) = Y_w = Y_{i_1} \circ Y_{i_2} ... \circ Y_{i_k} & \text{pour} \quad w = x_{i_1} x_{i_2} ... x_{i_k}, \\ \mathcal{Y}(\varepsilon) = Y_\varepsilon = Identité. \end{cases}$$

L'action d'un opérateur différentiel Y_w sur une fonction analytique f, définie sur la variété Q, sera notée $Y_w \circ f$.

Pour des entrées et le temps suffisamment petits, la sortie est donnée par la formule de PEANO-BAKER (*cf* M.Fliess[9] ou C.Reutenauer[22]) dite aussi **formule fondamentale** de M.FLIESS:

$$y(t) = \sum_{w \in X^*} (Y_w \circ h)_{|_{q(0)}} \int_0^t \delta_u w.$$

où $|_{q(0)}$ signifie l'évaluation en $q(0)$, $\int_0^t \delta_u w$ est l'intégrale itérée associée au mot w, relative à l'entrée u pendant l'intervalle de temps $[0, t]$. On rappelle que les intégrales itérées sont définies par récurrence sur la longueur du mot w comme suit:

$$\begin{cases} \text{si } w = \varepsilon, & \text{alors } \int_0^t \delta_u \varepsilon = 1, \\ \text{si } w = v x_i, & \text{alors } \int_0^t \delta_u w = \int_0^t (\int_0^\tau \delta_u v) u_i(\tau) d\tau. \end{cases}$$

(Cette définition, développée aussi par Hoang Ngoc Minh & G.Jacob dans [11], est symétrique de celle donnée par M.Fliess dans [8]).

En particulier: $\int_0^t \delta_u x_0 = \int_0^t u_0(\tau) d\tau = t$, puisque $u_0(\tau) \equiv 1$, pour tout τ.

Le comportement Entrée/Sortie du système (Σ) est entièrement défini par sa série génératrice g (*cf* M.Fliess[9]), qui est une série formelle en les variables non commutatives $x_0, x_1, ..., x_{m-1}$, donnée par:

$$(II.1.1) \qquad g = \sum_{w \in X^*} (Y_w \circ h)_{|q(0)} w.$$

$< g|w >$ étant le coefficient du mot w dans la série g, la sortie $y(t)$ peut s'écrire:

$$y(t) = \sum_{w \in X^*} < g|w > \int_0^t \delta_u w.$$

II.2. Champs de vecteurs et coordonnées locales.

La variété Q est de dimension N. Soient $z_1, z_2, ..., z_N$, un système de coordonnées locales. On notera $\mathbb{R}[[z_1, z_2, ..., z_N]]$ et $\mathbb{R}[z_1, z_2, ..., z_N]$ les $\mathbb{R}$-algèbres des séries et des polynômes commutatifs en les variables $z_1, z_2, ..., z_N$.

Le champ de vecteurs Y_i, vu au paragraphe II, peut alors s'écrire:

$$Y_i = \sum_{k=1}^{N} \theta_i^k(z) \frac{\partial}{\partial z_k}. \qquad \theta_i^k(z) \in \mathbb{R}[[z_1, z_2, ..., z_N]].$$

Ainsi, si f est une fonction analytique définie sur la variété Q, alors, l'action du champ de vecteurs Y_i sur la fonction f s'écrira:

$$Y_i \circ f = \sum_{k=1}^{N} \theta_i^k(z) \frac{\partial f}{\partial z_k}.$$

III. EVALUATION POUR UNE ENTREE DONNEE.

La formule fondamentale peut être interprétée comme suit: la sortie $y(t)$, pour une entrée u donnée, est l'*évaluation* de la série génératrice g pour cette entrée.

DEFINITION III.1[11]. – *On appelle <u>évaluation</u> d'un mot $w \in X^*$ pour l'entrée u, l'intégrale itérée:*

$$\mathcal{E}_u(w)(t) = \int_0^t \delta_u w.$$

On étend cette notion aux polynômes en posant, pour tout polynôme $P \in \mathbb{R}<X>$,

$$\mathcal{E}_u(P) = \sum_{w \in X^*} <P|w> \mathcal{E}_u(w).$$

Sous réserve de convergence, on peut étendre cette notion aussi aux séries en posant:

$$\mathcal{E}_u(S) = \sum_{w \in X^*} <S|w> \mathcal{E}_u(w).$$

Ainsi la sortie $y(t)$ du système (Σ), est l'*évaluation* de sa série génératrice g :

$$y(t) = \mathcal{E}_u(g) \qquad (ou \quad \mathcal{E}_u(g) = h(q(t))),$$

où $q(t) = \phi_{(u,t)}^{q(0)}$ est l'état du système, initialisé en $q(0)$, soumis à l'entrée u pendant le temps t.

Pour des entrées dont l'intégrale, dans un intervalle $[0,t]$, est bornée, nous avons le théorème suivant appelé théorème de l'unicité (*cf* M.Fliess[8]).

THEOREME III.2. – *Soient S et T deux séries. Si $\mathcal{E}_u(S) = \mathcal{E}_u(T)$, pour toute entrée u, alors $S = T$.*

LEMME III.3. – *Si S et T sont deux séries, alors* $\mathcal{E}_u(S \omega T) = \mathcal{E}_u(S).\mathcal{E}_u(T)$.

Preuve:

C'est une simple interprétation de la formule d'intégration par parties (*cf* G.Jacob & N.Oussous[13]).

IV. GEOMETRIE DU SYSTEME ET SERIES FORMELLES.

Dans ce paragraphe, nous allons montrer comment se transportent des notions géométriques du système dans l'algèbre des séries formelles. En particulier, nous allons voir comment se traduit la dérivation d'une fonction analytique, l'action d'un champ de vecteurs sur une fonction analytique ou encore le produit de deux fonctions analytiques en terme de séries formelles.

IV.1. Opérateurs différentiels.

Dans $\mathcal{C}^\omega(Q)$ (algèbre des fonctions analytiques définies sur Q) on a deux types d'opérateurs différentiels: les opérateurs liés au système et qui sont les champs de vecteurs Y_i, et les opérateurs liés à la variété et sont de la forme $\partial_i = \frac{\partial}{\partial z_i}$.

IV.1.1. Séries génératrices et codage des observations.

On appelle *algèbre d'observations* une sous-algèbre complète $\mathcal{O}_{q(0)}$ de l'algèbre $\mathcal{C}^\omega(Q)$ des fonctions analytiques au voisinage de $q(0)$, stable par les champs de vecteurs analytiques.

A tout système (Σ), on peut associer l'application (*cf* M.Fliess[9])

$$\sigma \ : \ \mathcal{O}_{q(0)} \longmapsto I\!R\ll X\gg,$$

définie par:

$$\forall h \in \mathcal{O}_{q(0)}, \qquad \sigma(h) \ = \ \sum_{w\in X^*} (Y_w \circ h)_{|_{q(0)}} w.$$

Si h est une observation, $\sigma(h)$ est donc la série génératrice du système (Σ) muni de l'observation h. Dans ce cas, l'évaluation de la série génératrice n'est autre que la fonction $h(q(t))$

$$\mathcal{E}_u(\sigma(h)) = h(q(t)).$$

où $q(t) = \phi^{q(0)}_{(u,t)}$, l'état du système, initialisé en $q(0)$, soumis à l'entrée u pendant le temps t.

IV.1.2 Codage des champs de vecteurs Y_i.

Voyons maintenant comment se traduit l'action d'un opérateur interne (un champ de vecteurs) sur une fonction analytique. On a:

$$\begin{aligned}\sigma(Y_i \circ h) \ &= \ \sum_{w\in X^*} (Y_w \circ (Y_i \circ h))_{|_{q(0)}} w = \ \sum_{w\in X^*} < \sigma(h) | w x_i > w \\ &= \ \sum_{w\in X^*} < x_i \triangleleft \sigma(h) | w > w = \ x_i \triangleleft (\sigma h)\end{aligned}$$

L'opérateur Y_i de $\mathcal{O}_{q(0)}$ est donc représenté par l'opérateur « $x_i\triangleleft$ » de simplification de la série $\sigma(h)$ à droite par la lettre x_i. On a ainsi reflété l'action des champs de vecteurs comme étant l'action "calcul de résiduels" sur des séries formelles.

En coordonnées locales, le champ de vecteurs Y_i s'écrit:

$$Y_i = \sum_{j=1}^{r} \theta_i^j(z_1, ..., z_r) \frac{\partial}{\partial z_j}$$

$$\Longrightarrow \qquad Y_i \circ z_j = \sum_{j=1}^{r} \theta_i^j(z_1, ..., z_r) \frac{\partial z_j}{\partial z_j} = \theta_i^j(z_1, ..., z_r).$$

En considérant chaque coordonnée locale z_j comme une observation, on définit les séries $Z_j = \sigma(z_j)$. Alors:

$$\sigma(Y_i \circ z_j) = x_i \triangleleft \sigma(z_j) = x_i \triangleleft Z_j.$$

D'où une méthode algorithmique pour le calcul des champs de vecteurs.

IV.1.3. Systèmes localement faiblement commandables.

DEFINITION. – *Soit* (Σ) *un système et soient* $Y_0, Y_1, ..., Y_{m-1}$ *les champs de vecteurs associés.* (Σ) *est dit localement faiblement commandable au point* q *si, et seulement si,*

$$\dim \mathcal{L}ie < Y_0, Y_1, ..., Y_{m-1} >_{|_q} = \dim T_q(Q).$$

Nous supposons à présent que le système (Σ) est localement faiblement commandable en q. Nous allons montrer que l'on peut définir un opérateur Δ_i sur les séries formelles vérifiant:

$$\Delta_i(\sigma_q(h)) = \sigma_q(\partial_i \circ h).$$

C'est-à-dire, à l'opérateur ∂_i sur les fonctions analytiques est associé un opérateur Δ_i sur les séries formelles.

On note $\Gamma(T(Q))$ l'ensemble des champs de vecteurs analytiques sur la variété Q.

THEOREME IV.1.1. – *Si le système* (Σ) *est localement faiblement commandable en* q, *on a:*

$$\sigma_q(h) = 0 \implies \forall \mathcal{Z} \in \Gamma(T(Q)), \quad \sigma_q(\mathcal{Z} \circ h) = 0.$$

Preuve

Pour montrer ce théorème, il nous faut montrer, pour tout champ de vecteurs $\mathcal{Z} \in \Gamma(T(Q))$, que:

$$[< \sigma_q(h) | w >= 0 \quad \forall w \in X^*] \implies [< \sigma_q(\mathcal{Z} \circ h) | w >= 0 \quad \forall w \in X^*].$$

Ce qui s'écrit encore:

$$\left.\begin{array}{l} \forall h \in \mathcal{C}^\omega(Q) \ \textit{telle que} \ \sigma_q(h) = 0, \\ \forall \mathcal{Z} \in \Gamma(T(Q)), \\ \forall w \in X^*, \end{array}\right\} \quad \text{on a} \quad (Y_w \circ \mathcal{Z} \circ h)_{|_q} = 0.$$

(i) On a clairement:

$$\sigma_q(h) = 0 \implies \mathcal{L}ie < Y_0, Y_1, ..., Y_{m-1} > \circ h_{|_q} = 0 \implies \Gamma(T(Q)) \circ h_{|_q} = 0 \implies \mathcal{Z} \circ h_{|_q} = 0.$$

(ii) Hypothèse de récurrence:

$$\mathcal{HR}_n \quad \left[\left.\begin{array}{l} \forall h \in \mathcal{C}^\omega(Q) \ \textit{telle que} \ \sigma_q(h) = 0, \\ \forall \mathcal{Z} \in \Gamma(T(Q)), \\ \forall w \in X^{\leq n}, \end{array}\right\} \quad \text{on a} \quad (Y_w \circ \mathcal{Z} \circ h)_{|_q} = 0.\right]$$

D'après (i), $\mathcal{HR}_1$ est vérifiée.

Supposons vérifiée $\mathcal{HR}_n$. Si $w \in X^n$, et $x \in X$, nous avons successivement:

$$Y_{wx} \circ (\mathcal{Z} \circ h) = Y_w \circ (Y_x \circ \mathcal{Z} \circ h) = Y_w \circ ([Y_x, \mathcal{Z}] \circ h) + Y_w \circ \mathcal{Z} \circ (Y_x \circ h).$$

(a) or on a: $\sigma_q(h) = 0$, $[Y_x, \mathcal{Z}] \in \Gamma(T(Q))$, $|w| = n$, et l'hypothèse $\mathcal{HR}_n$ entraine donc:

$$(Y_w \circ [Y_x, \mathcal{Z}] \circ h)_{|_q} = 0.$$

(b) et on a aussi: $\sigma_q(Y_x \circ h) = 0$, $\mathcal{Z} \in \Gamma(T(Q))$, $|w| = n$, et l'hypothèse $\mathcal{HR}_n$ entraine donc:

$$(Y_w \circ \mathcal{Z} \circ (Y_x \circ h))_{|_q} = 0.$$

On en déduit: $(Y_{wx} \circ \mathcal{Z} \circ h)_{|_q} = 0$. Et donc $\mathcal{HR}_n \implies \mathcal{HR}_{n+1}$.

L'hypothèse $\mathcal{HR}_n$ est donc vérifiée pour tout $n \in \mathbb{N}$, d'où le théorème.

□

COROLLAIRE. –

$$\sigma_q(h) = 0 \implies \forall i \in [1..n],\ \sigma_q(\partial_i \circ h) = 0.$$

On peut donc affirmer que si h et k sont deux fonctions analytiques alors,

$$\sigma_q(h) = \sigma_q(k) \implies \sigma_q(\partial_i \circ h) = \sigma_q(\partial_i \circ k).$$

Il existe donc une application Δ_i de $\sigma_q(\mathcal{O}_{q(0)})$ dans lui-même telle que $\sigma_q(\partial_i \circ h) = \Delta_i(\sigma_q(h))$.

IV.2. Algèbres des observations.

Dans ce qui suit, on va montrer que σ *transporte* la structure d'*algèbre d'observations* dans l'algèbre des séries formelles munie du mélange.

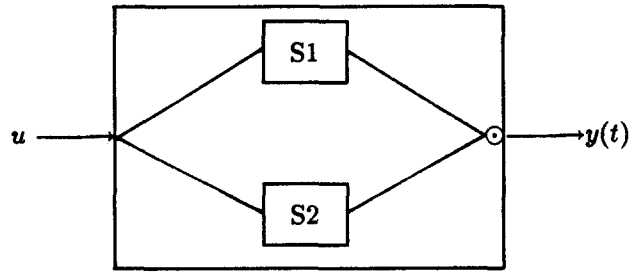
LEMME IV.2.1. – *σ est une application linéaire, c'est-à-dire:*

$$\forall h, k \in \mathcal{O}_{q(0)},\quad \forall \alpha, \beta \in \mathbb{R},\qquad \sigma(\alpha h + \beta k) = \alpha\sigma(h) + \beta\sigma(k).$$

LEMME IV.2.2. – *Soient $h, k \in \mathcal{O}_{q(0)}$, des observations. On a* $\sigma(h.k) = (\sigma(h))\ \text{ш}\ (\sigma(k))$.

Preuve:

Soit le système (S) suivant:



Ce système est formé de deux systèmes $(S1)$ et $(S2)$, qui possèdent la **même entrée** u et les sorties respectives $y_1(t)$ et $y_2(t)$, la sortie du système est le produit des deux sorties: $y_1(t).y_2(t)$.

On peut supposer que les deux systèmes $S1$ et $S2$ sont définis sur un même espace d'état et ont le même état initial (si nécessaire, plonger $S1$ et $S2$ dans l'espace produit de leurs espaces d'états et le nouvel état initial est $q^1(0) \times q^2(0)$). Si g_1 (resp. g_2) est la série génératrice du système $(S1)$ (resp. $(S2)$), alors la série génératrice du système (S) n'est rien d'autre que le mélange $g_1 \text{ш} g_2$.

Dans le cas présent:

$g_1 = \sigma(h_1)$, $g_2 = \sigma(h_2)$, et on a les correspondances suivantes, en posant $q(t) = \phi^{q(0)}_{(u,t)}$:

$$\sigma(h_1) \xrightarrow{\mathcal{E}_u} h_1(q(t)) = y_1(t) \qquad \text{et} \qquad \sigma(h_2) \xrightarrow{\mathcal{E}_u} h_2(q(t)) = y_2(t)$$

$$\sigma(h_1 h_2) \xrightarrow{\mathcal{E}_u} h_1 h_2(q(t)) = y_1 y_2(t)$$

$$\sigma(h_1) \text{ш} \sigma(h_2) \xrightarrow{\mathcal{E}_u} h_1 h_2(q(t)) = y_1 y_2(t).$$

On en déduit que, pour toute entrée u, $\mathcal{E}_u(\sigma(h_1 h_2)) = \mathcal{E}_u(\sigma(h_1) \text{ш} \sigma(h_2))$. Et d'après le théorème III.2,

$$\sigma(h_1 h_2) = \sigma(h_1) \text{ш} \sigma(h_2).$$

Donc σ est un morphisme de l'algèbre d'observations dans l'algèbre des séries formelles munie du mélange. □

Remarque:

$$\sigma(h) = 0 \iff \forall w \in X^*,\ Y_w \circ h = 0 \iff h \text{ inobservable.}$$

En d'autres termes, l'application $h \longmapsto \sigma(h)$ de $\mathcal{O}_{q(0)}$ dans $\mathbb{R}\ll X \gg$ est injective si, et seulement si, le système est localement observable (relativement à $\mathcal{O}_{q(0)}$). *i.e.* h inobservable $\implies h = 0$.

IV.3. Evaluation des coordonnées locales.

Nous explicitons ici la traduction des coordonnées locales de la variété Q dans l'algèbre des séries formelles.

IV.3.1. Structure d'algèbre commutative sur $\mathbb{R}\ll X\gg$.

En un point $q(0)$ on a (pour la dimension N) N coordonnées locales $z_1, ..., z_N$ où $z_i \in \mathcal{O}_{q(0)}$, $1 \leq i \leq N$. Nous rappelons que pour tout i, on a posé $Z_i = \sigma(z_i) \in \mathbb{R}\ll X\gg$. Les Z_i sont des séries propres. En effet:

$$< Z_i|\varepsilon > = (Y_\varepsilon \circ z_i)_{|_{q(0)}} = z_i|_{q(0)} = 0.$$

Dans $\mathbb{R}\ll X\gg$ les séries propres permettent de définir une topologie par le système fondamental de voisinages de zéro:

$$\mathcal{W}_n = \{ S \in \mathbb{R}\ll X\gg \mid \omega(S) \geq n \}.$$

Au produit des fonctions analytiques correspond, par σ, le produit de mélange des séries formelles (qui est un produit commutatif). Soit $\alpha = (i_1, i_2, ..., i_k)$, un **multi-indice**, $(k \geq 0)$, alors on pose (*cf* S.Abhyankar[1]) :

$$\begin{cases} |\alpha| = \sum_{j=1}^{k} i_j, \\ z^\alpha = z_1^{i_1} z_2^{i_2} ... z_k^{i_k}, \\ Z^{\text{ш}\,\alpha} = Z_1^{\text{ш}\,i_1} \text{ш}\, Z_2^{\text{ш}\,i_2} ... \text{ш}\, Z_k^{\text{ш}\,i_k} \end{cases}$$

où $\text{ш}\,i_j$ est la puissance $i_j^{\text{ième}}$ pour le produit de mélange. D'après le lemme IV.2, on a:

$$\sigma(z^\alpha) = Z^{\text{ш}\,\alpha}.$$

IV.3.2. Topologie dans $\mathcal{O}_{q(0)}$.

Puisque les z_i forment un système de coordonnées locales au point $q(0)$, alors toute fonction $f \in \mathcal{O}_{q(0)}$ admet un développement analytique:

$$f = \sum_\alpha c_\alpha z^\alpha$$

où α parcourt les multi-indices. Appelons *ordre* d'une fonction $h \in \mathcal{O}_{q(0)}$ (par rapport aux z_i) l'ordre $\Omega(h)$ du développement en série de h sur les z_i. La topologie sur $\mathcal{O}_{q(0)}$ peut être définie par le système fondamental de voisinages de zéro:

$$\mathcal{V}_n = \{h \in \mathcal{O}_{q(0)} \mid \Omega(h) \geq n\}.$$

Pour cette topologie $\mathcal{O}_{q(0)}$ est un espace complet.

THEOREME IV.3.1. – *L'application* $\sigma : \mathcal{O}_{q(0)} \longrightarrow (\mathbb{R}\ll X\gg, \text{ш})$, *est une fonction continue.*

Preuve:

On vérifie que, pour tout n, $\sigma^{-1}(\mathcal{W}_n)$ contient $\mathcal{V}_n$.

□

Puisque σ est continue, on en déduit:

LEMME IV.3.2. –

$$\sigma(\sum_\alpha c_\alpha z^\alpha) = \sum_\alpha c_\alpha Z^{\text{ш}\,\alpha}.$$

V. MOTS DE LYNDON

V.1. Ordre lexicographique et classes de conjuguaison.

Soit X un alphabet totalement ordonné, et soit X^* le monoïde libre engendré par X dont les éléments sont appelés des mots. On définit un ordre total sur X^* comme suit:

$$\forall\, u, v \in X^*, \quad u < v \quad \text{ssi} \quad \begin{cases} \text{ou} & (i)\ \exists\, w \neq \varepsilon \text{ tel que } uw = v, \\ & (ii)\ \exists x, y, z \in X^* \text{ et } a, b \in X \text{ tels que } u = xay,\ v = xbz \text{ et } a < b. \end{cases}$$

On a ainsi l'ordre lexicographique usuel sur les mots de X^*. Pour cet ordre, on a (*cf* Lothaire[18]), les deux propriétés suivantes:

$$\begin{cases} (1) & \forall w \in X^*, & u < v \iff wu < wv. \\ (2) & \text{Si } v \notin uX^*, \ \forall w, z \in X^*, & u < v \implies uw < vz. \end{cases}$$

On dira qu'un mot u est un *facteur* du mot v si il existe $x, y \in X^*$ tels que $v = xuy$.

Si $x = \varepsilon$ (resp. $y = \varepsilon$), on dira que u est le *facteur gauche* (resp. *droit*) de v, *propre* si $y \neq \varepsilon$ (resp. $x \neq \varepsilon$).

Deux mots u et v de X^* sont dits *conjugués* si il existe $x, y \in X^*$ *tels que* $u = xy$ *et* $v = yx$.

V.2. Mots de Lyndon.

Pour plus de détails sur les mots de Lyndon, on peut consulter les références [4], [18] ou [20].

DEFINITION V.2.1. – *Un mot* $w \in X^*$ *est un mot de Lyndon ssi:*

$$\text{ou} \quad \begin{matrix} (i) & \textit{il est strictement plus petit que tous ses conjugués,} \\ (ii) & \textit{il est strictement plus petit que ses facteurs droits propres.} \end{matrix}$$

On notera L *l'ensemble des mots de Lyndon sur* X.

Propriétés:

(1) Soit $w \in L\backslash X$ et soit m son plus long facteur droit propre dans L. Si $w = lm$, alors $l \in L$ et $l < lm < m$. Le couple $\sigma(w) = (l, m)$ est appelé *la factorisation standard* de w.

(2) $w \in L$ si, et seulement si, $\begin{cases} w \in X, \\ \text{ou} \\ w = lm \text{ avec } l, m \in L \text{ et } l < m. \end{cases}$

Ce qui donne un moyen algorithmique pour la construction des mots de Lyndon.

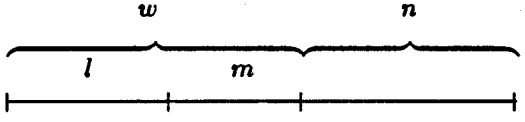
LEMME V.2.2. – *Soient* $w \in L\backslash X$ *et* $\sigma(w) = (l, m)$ *sa factorisation standard, et soit* n *un mot de Lyndon vérifiant* $w < n$. *Alors le couple* (w, n) *est la factorisation standard de* wn *si, et seulement si,* $n \leq m$.

Preuve:

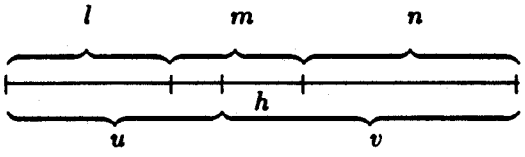
(a) Montrons que $\sigma(wn) = (w, n) \implies n \leq m$.

Supposons que $m < n$, alors d'après la propriété (2) $mn \in L$ et donc n n'est pas le plus long facteur droit propre de wn ce qui est en contradiction avec l'hypothèse: $(w, n) = \sigma(wn)$.

(b) On suppose $w < n \leq m$ et que $\sigma(wn) = (u, v)$ avec $v \neq n$.



On a alors nécessairement $|v| > |n|$. Donc $v = hn$ où h est un suffixe propre de w.



Soit alors k le plus petit suffixe, différent de ε, propre ou non de h. On a alors $k < m$. En effet:

$$k \leq h < hn = v < n \leq m.$$

Or $m \leq k$, puisque k est un suffixe propre de w, et que $\sigma(w) = (l, m)$. D'où contradiction.

□

V.3. Base de Lyndon.

A partir des mots de Lyndon, on peut construire une base (dite **base de LYNDON** ou de CHEN-FOX-LYNDON (*cf* Lothaire[18] ou Viennot[25])) de l'algèbre $\mathcal{L}ie\langle X\rangle$ des polynômes de Lie. Cette base est obtenue récursivement comme suit:

$$\begin{cases} c(x) = x \quad pour \quad x \in X, \\ c(w) = [c(l), c(m)], \quad pour\ w \in L\backslash X,\ tel\ que\ \sigma(w) = (l, m). \end{cases}$$

en utilisant les crochets de Lie. Cette définition nous donne un algorithme pour construire cette base (*cf* Viennot[25]).

V.4. Théorème de Lyndon.

THEOREME [18]. – *Chaque mot $w \in X^*$ peut être écrit de manière unique comme suit:*

$$w = l_1 l_2 \dots l_n \qquad (1)$$

où chaque $l_i \in L$ et $l_1 \geq l_2 \geq \dots \geq l_n$.

Pour la preuve de ce Théorème voir Lothaire[18] ou G.Melançon & C.Reutenauer[20].

Dans L, on peut définir différentes relations d'ordre total, en particulier, on peut définir l'ordre lexicographique et l'ordre lexicographique par longueur. Vu la définition de la base de Lyndon, toute relation d'ordre total sur L induit naturellement une relation d'ordre total sur la base de Lyndon de $\mathcal{L}ie\langle X\rangle$.

VI. RÉALISATION DES SYSTÈMES DYNAMIQUES.

Localement, on exprime (M.FLIESS) le problème de la réalisation comme suit:

Etant donné un comportement **E/S** donné par sa série génératrice sur un alphabet X, existe-t-il un système différentiel du type (Σ) qui admette la même série génératrice? Si oui, le décrire.

VI.1 Séries produites différentiellement.

DÉFINITION. – *Une série formelle $g \in \mathbb{R}\ll X \gg$ est dite produite différentiellement si, et seulement si, il existe $r \in \mathbb{N}$, une série formelle $h \in \mathbb{R}[\![z_1, \dots, z_r]\!]$, et un homomorphisme $\mathcal{Y}$ de X^* dans l'algèbre des opérateurs différentiels formels sur $\mathbb{R}[\![z_1, \dots, z_r]\!]$, tel que pour chaque lettre x_i de X, $\mathcal{Y}(x_i) = Y_i$ soit un champ de vecteurs formel tels que:*

$$\forall w \in X^* \quad \langle g|w\rangle = Y_w \circ h_{|_0}$$

où $|_0$ est l'évaluation en $z_1 = \dots = z_r = 0$.

On dira que (Y, h) est une **représentation différentielle** de g, de dimension r.

D'après la formule (II.1.1) il est clair que:

g est la série génératrice d'un système formel de la forme (Σ) si, et seulement si, g admet une représentation différentielle .

En conclusion:

Etudier les réalisations locales
revient à
Etudier les représentations différentielles.

VI.2. Théorème de Fliess.

THÉORÈME VI.2.1 [9]. – *Une série $g \in \mathbb{R} \ll X \gg$ est produite différentiellement si, et seulement si, son rang de Lie est fini, soit r. Dans ce cas r est égal à la plus petite dimension de ses représentations différentielles. Si (Y,h) et (Y',h') sont deux représentations différentielles de dimension r de g, alors il existe un automorphisme φ, continu, de $\mathbb{R}[z_1,...,z_r]$ tel que:*

$$\forall w \in X^*,\ \forall k \in \mathbb{R}[z_1,...,z_r],\quad h' = \varphi(h) \text{ et } \varphi(Y_w \circ k) = Y'_w \circ \varphi(k).$$

(Y_w désigne l'image de w par Y).

La réalisation (Y,h), unique à isomorphisme près, est dite réduite ou minimale.

Pour nous le problème de la réalisation consiste à calculer le rang de Lie r de la série génératrice g puis à déterminer les polynômes $Z_1,...,Z_r$ tels que l'on puisse exprimer g, qui au départ est une série formelle (finie) en les variables non commutatives $x_0,...,x_{m-1}$, comme combinaison linéaire des mélanges des polynômes $Z_1,...,Z_r$. On prendra ensuite comme coordonnées locales z_i *telle que* $\sigma(z_i) = Z_i$ pour $(1 \leq i \leq r)$. On obtient alors la fonction d'observation h telle que $\sigma(h) = g$. Ensuite, pour définir l'homomorphisme $\mathcal{Y}$, nous calculons des champs de vecteurs $Y_0,...,Y_{m-1}$, images respectives des lettres $x_0,...,x_{m-1}$ par $\mathcal{Y}$, et qui sont donnés par la formule:

$$Y_i = \sum_{j=1}^{r} (Y_i \circ z_j)\frac{\partial}{\partial z_j}, \quad \textit{avec} \quad \sigma(Y_i \circ z_j) = x_i \triangleleft Z_j \in \mathbb{R}[z_1,...,z_r].$$

VI.3. Construction de la réalisation.

On considère la sous-algèbre de Lie de l'algèbre $\mathcal{L}ie\langle X\rangle$, de codimension r, définie par:

$$\mathcal{A}(g) = \{\ P \in \mathcal{L}ie\langle X\rangle \ \mid g \triangleright P = 0\ \}$$

$\mathcal{A}(g)$ est engendrée par les polynômes de Lie qui annulent la série g.

On pose:

$$\mathcal{V}(\mathcal{A}(g)) = \{\ Q \in \mathbb{R}\ll X\gg \ \mid Q \triangleright \mathcal{A}(g) = 0\ \},$$

c'est l'ensemble des séries qui sont annulées par l'algèbre $\mathcal{A}(g)$.

Soit $(P_i)_{i\geq 1}$ une base de $\mathcal{L}ie\langle X\rangle$ telle que $P_1,...,P_r$ soit une base de $\mathcal{L}ie\langle X\rangle$ modulo $\mathcal{A}(g)$ et $P_{r+1},...,P_n,...$ soit une base de $\mathcal{A}(g)$. On définit les polynômes $Z_1,...,Z_r$ (*cf* C.Reutenauer[22]) comme étant des polynômes de terme constant nul vérifiant:

$$(VI.3)\quad \begin{cases} (i) & \langle Z_j | P_i \rangle = \langle Z_j \triangleright P_i | \varepsilon \rangle = \delta_{ij} \quad \text{pour } i \leq r, \\ (ii) & Z_j \in \mathcal{V}(\mathcal{A}(g)) \quad \text{pour } i > r. \end{cases}$$

où $\langle Z_j \triangleright P_i | \varepsilon \rangle$ est le coefficient de ε dans le simplifié de Z_j à gauche par le polynôme de Lie P_i.

On choisit comme base de Lie la base de CHEN-FOX-LYNDON définie dans le paragraphe précédent. Ce choix est dû, d'une part, au fait que les mots de Lyndon s'obtiennent à partir des éléments de la base en effaçant les crochets et les virgules et d'autre part, à cause des propriétés particulièrement intéressantes des mots de Lyndon.

On sait, d'après (MELANCON& REUTENAUER ou RADFORD) que les mots de Lyndon forment une base de **transcendance** de l'algèbre $\mathbb{R}\ll X\gg$ munie du mélange.

D'autre part, on a la relation très importante suivante:

$$\langle l_j | P_i \rangle = \langle l_j \triangleright P_i | \varepsilon \rangle = \delta_{ij}$$

où les l_j sont les mots de Lyndon correspondants aux éléments de la base de Lie. On a donc tendance à prendre comme polynômes Z_j les mots de Lyndon correspondants aux polynômes de Lie qui n'annulent pas g, mais il ne faut pas oublier la deuxième partie de (VI.3).

En définitive, on prend comme polynômes Z_j des combinaisons linéaires des mélanges des mots de Lyndon de degré inferieur au degré de la série g.

VI.4. Exemple.

Soit la série $g = x_1x_0x_1 + x_0x_1x_0x_1$. $X = \{x_0, x_1\}$. Les éléments de la base de $\mathcal{L}ie < X >$ jusqu'à l'ordre 4 (puisque la série est de degré 4) sont:

$$\begin{array}{llll} P_1 = x_0, & P_3 = [x_0, x_1], & P_4 = [x_0, [x_0, x_1]], & P_6 = [x_0, [x_0, [x_0, x_1]]], \\ P_2 = x_1, & & P_5 = [[x_0, x_1], x_1], & P_7 = [x_0, [[x_0, x_1], x_1]], \\ & & & P_8 = [[[x_0, x_1], x_1], x_1]. \end{array}$$

Et les mots de Lyndon correspondants sont:

$$\begin{array}{llll} l_1 = x_0, & l_3 = x_0x_1, & l_4 = x_0^2x_1, & l_6 = x_0^3x_1, \\ l_2 = x_1, & & l_5 = x_0x_1^2, & l_7 = x_0^2x_1^2, \\ & & & l_8 = x_0x_1^3. \end{array}$$

Si on calcule les simplifiés de g par les éléments de la base de $\mathcal{L}ie < X >$, On obtient:

$$\begin{array}{llll} g \triangleright x_0 = x_1x_0x_1, & g \triangleright [x_0, x_1] = -x_1 + x_0x_1, & g \triangleright [x_0, [x_0, x_1]] = -2x_1, & g \triangleright [x_0, [x_0, [x_0, x_1]]] = 0, \\ g \triangleright x_1 = x_0x_1, & & g \triangleright [[x_0, x_1], x_1] = -2, & g \triangleright [x_0, [[x_0, x_1], x_1]] = -2, \\ & & & g \triangleright [[[x_0, x_1], x_1], x_1] = 0. \end{array}$$

En ne gardant que les lignes et les colonnes non nulles de la matrice de Lie-Hankel, on obtient:

$$\mathcal{LH}_g = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -2 & 0 & 0 \\ -2 & 0 & 0 & 0 \\ -2 & 0 & 0 & 0 \end{bmatrix} \quad et \quad \mathcal{RL}_g = 4,$$

cette matrice est de rang 4 alors qu'elle a six lignes non nulles. Il existe donc des lignes linéairement dépendantes.

Les polynômes:

$$F_1 = P_4 - 2P_3 + 2P_2, \quad F_2 = P_6, \quad F_3 = P_7 - P_5 \quad \text{et} \quad F_4 = P_7 - P_5$$

annulent la série g. Donc ces polynômes, ainsi que tous les éléments de la base de $\mathcal{L}ie < X >$ de degré supérieur à 4 appartiennent à $\mathcal{A}(g)$. Les polynômes F_1 et F_3 sont obtenus à partir des relations de dépendances qui existent entre les lignes de $\mathcal{LH}_g$.

Le polynôme F_1 donne lieu à deux relations $Z_2 = l_2 - 2l_4$ et $Z_3 = l_3 + 2l_4$ sur les mots de Lyndon qui verifient:

$$< Z_i | P_j >= \delta_{ij}, \quad i = 2, 3 \quad et \quad j \geq 1.$$

La relation duale du polynôme F_4 est $Z_4 = l_7 + l_5$. On prend enfin $Z_1 = l_1$, puisque la ligne correspondant à P_1 est linéairement indépendante de toutes les autres. On a donc retenu comme base de $\mathcal{L}ie < X >$ modulo $\mathcal{A}(g)$ les polynômes de Lie P_1, P_2, P_3 et P_5.

	Z_1	Z_2	Z_3	Z_4
x_0	1	0	0	0
x_1	0	1	0	0
$[x_0, x_1]$	0	0	1	0
$[[x_0, x_1], x_1]$	0	0	0	1

où l'élément qui se trouve à la ligne P_i et à la colonne Z_j est le coefficient du mot vide (ε) dans le simplifié de Z_j à gauche par le polynôme de Lie P_i.

Il reste à montrer le (ii) de $(VI.3)$:

	F_1	F_2	F_3	F_4
$Z_1 = x_0$	0	0	0	0
$Z_2 = x_1 - 2x_0^2 x_1$	0	0	0	0
$Z_3 = x_0 x_1 + 2x_0^2 x_1$	0	0	0	0
$Z_4 = x_0 x_1^2 + x_0^2 x_1^2$	$-x_1$	0	0	0

d'autre part les $(Z_i)_{1 \leq i \leq 4}$ sont annulés par les polynômes de l'algèbre $\mathcal{A}(g)$ de degré supérieur à 4. Le seul problème qui reste est Z_4, puisqu'il n'est pas annulé par F_1. Mais, il suffit d'ajouter $\frac{1}{4} l_2 \sqcup l_2$ à Z_4 puisque $\frac{1}{4}(l_2 \sqcup l_2) \triangleright F_1 = x_1$. Donc, on prendra $Z_4 = l_5 + l_7 + \frac{1}{4}(l_2 \sqcup l_2)$. On remarquera que les Z_i sont des combinaisons linéaires des mélanges des mots de Lyndon de degré inférieur ou égal à 4.

Maintenant que nous avons les Z_i, nous pouvons écrire l'expression de g sur les mélanges de ces polynômes. On obtient alors:

$$g = \frac{1}{2} Z_2^{\sqcup 2} + Z_2 \sqcup Z_3 + \frac{1}{2} Z_3^{\sqcup 2} - 2Z_4.$$

Et la fonction d'observation est donc:

$$h = \frac{1}{2} z_2^2 + z_2 z_3 + \frac{1}{2} z_3^2 - 2z_4.$$

Pour le calcul des champs de vecteurs, nous utilisons le tableau suivant:

	$x_0 \triangleleft$	$x_1 \triangleleft$
$Z_1 = x_0$	1	0
$Z_2 = x_1 - 2x_0^2 x_1$	0	$1 - 2x_0^2 = 1 - Z_1^{\sqcup 2}$
$Z_3 = x_0 x_1 + 2x_0^2 x_1$	0	$x_0 + 2x_0^2 = Z_1 + Z_1^{\sqcup 2}$
$Z_4 = x_0 x_1^2 + x_0^2 x_1^2 + \frac{1}{4} x_1^2$	0	$x_0 x_1 + x_0^2 x_1 + \frac{1}{2} x_1 = Z_2 + 2Z_3$

Donc les champs de vecteurs sont:

$$\begin{cases} Y_0 = \dfrac{\partial}{\partial z_1}, \\ Y_1 = (1 - z_1^2) \dfrac{\partial}{\partial z_2} + (z_1 + z_1^2) \dfrac{\partial}{\partial z_3} + (2z_3 + z_2) \dfrac{\partial}{\partial z_4}. \end{cases}$$

Nous avons ainsi un moyen algorithmique pour calculer le réalisation minimale de tout système dynamique non linéaire de série génératrice finie. Cette méthode a déjà été implémentée sous le système de calcul formel MACSYMAinstallé sur les stations de travail SUN (*cf* G.Jacob & N.Oussous[14,15]). La suite de ce travail sera d'essayer d'étendre la méthode aux séries rationnelles qui correspondent aux systèmes bilinéaires (*cf* M.Fliess[5], G.Jacob[12] ou H.J.Sussmann[23]) qui sont beaucoup plus importants.

BIBLIOGRAPHIE

[1] S.S.ABHYANKAR. – Local analytic geometry. – *New York, London, Academic press*, 1964.

[2] J.BERSTEL & C.REUTENAUER. – Les séries rationnelles et leurs langages. – *Editions Masson*, 1984.

[3] D'ALESSANDRO, A.ISODORI & A.RUBERTI. – Realisation and structure theory of bilinear dynamical systems . – *SIAM J. Control*, t. **12**, 1974, p. 517 – 535.

[4] J.P.DUVAL. – Génération d'une section des classes de conjugaison et arbre des mots de Lyndon de longueur bornée. – *Publication du LITP Paris*, **88–20**, 1988.

[5] M.FLIESS. – Sur la réalisation des systèmes dynamiques bilinéaires . – *C. R. Acad. Sc., Paris*, t. **277**, **série A**, 1973, p. 923 – 926.

[6] M.FLIESS. – Sur divers produits de séries formelles . – *Bull. Soc. Math. Fr.*, t. **102**, 1974, p. 181 – 191.

[7] M.FLIESS. – Matrices de Hankel . – *J.Math. Pure Appl.*, t. **53**, 1974, p. 197 – 222.

[8] M.FLIESS. – Fonctionnelles causales non linéaires et indéterminées non commutatives . – *Bull. Soc. Math. France*, t. **109**, 1981, p. 3 – 40.

[9] M.FLIESS. – Réalisation locale des systèmes non linéaires, algèbres de Lie filtrées transitives et séries génératrices . – *Invent. Math.*, t. **71**, 1983, p. 521 – 537.

[10] C.HESPEL & G.JACOB. – Approximation of nonlinear systems by bilinear ones. – *Algebraic and Geometric Methods in Nonlinear Control Theory*, 1986, p. 511-520.

[11] HOANG NGOC MINH & G.JACOB. – Calcul symbolique et Séries de Volterra. – *Publication du LIFL Lille*, **IT–133**, 1988.

[12] G.JACOB. – Réalisation des systèmes réguliers (ou bilinéaires) et séries génératrices non commutatives. *Séminaire d'Aussois, In Outils et Modèles mathématiques pour l'automatique, l'analyse des systèmes et le traitement du signal*, coord. I.D.LANDAU, éd. CNRS, t. **1 – N° 567**, 1980, p. 325-357.

[13] G.JACOB & N.OUSSOUS. – Sur un résultat de REE: séries de Lie et algèbres de mélange. – *Publication du LIFL Lille*, **IT–103**, 1987.

[14] G.JACOB & N.OUSSOUS. – Un logiciel calculant la réalisation des systèmes analytiques de série génératrice finie. – *Publication du LIFL Lille*, **IT–123**, 1988.

[15] G.JACOB & N.OUSSOUS. – Algebraic computation of analytical minimal realisation. – *Publication du LIFL Lille*, **IT–139**, 1988.

[16] B.JAKUBCZYK. – Existence and uniqueness of realisations of nonlinear systems . – *SIAM J. Control Optimiz.*, t. **18**, 1977, p. 728 – 740.

[17] R.E.KALMAN. – Mathematical description of linear dynamical systems . – *SIAM J. Control*, t. **1**, 1963, p. 152 – 192.

[18] M.LOTHAIRE. – Combinatorics on words. – *Reading, Massachusetts*, 1983.

[19] MACSYMA. – Reference Manual, Version Ten. – *Massachusetts Institute of Technology and Symbolics*, 1983.

[20] G.MELANÇON & C.REUTENAUER. – Lyndon words, free algebras and shuffles. – *Publication du LITP Paris et UQAM Montreal, Quebec*, **87–63**, 1987.

[21] D.E.RADFORD. – A natural ring basis for the shuffle algebra and an application to group schemes . – *Journal of Algebra*, t. **58**, 1979, p. 432 – 454.

[22] C.REUTENAUER. – The local realisation of generating series of finite Lie rank. – *Algebraic and Geometric Methods In Nonlinear Control Theory*, **eds.** M.FLIESS and M.HAZEWINKEL, 1986, p. 33–43.

[23] H.J.SUSSMANN. – Minimal realisations and canonical forms for bilinear systems . – *J. Franklin Inst.*, t. **301**, 1976, p. 593 – 604.

[24] H.J.SUSSMANN. – Existence and uniqueness of minimal realisations of nonlinear systems . – *Math. Systems Theory*, t. **10**, 1977, p. 263 – 284.

[25] G.VIENNOT. – Algèbres de Lie libres et monoïdes libres . – *Lecture Notes In Mathematics, Springer-Verlag*, 691, 1978.

VIII. ANNEXE : PROGRAMMES ET EXEMPLES

PROGRAMMES

```
/*   Fonction qui renvoie le simplifie a droite du polynome <P> par la lettre*/
/* <x>, <P>  et <x> sont fournis en arguments.                                 */

SimDroLet(P,x) := if numberp(P) then 0
                  else
                    if atom(P) then if P=x then 1 else 0
                    else
                      if inpart(P,0) = "^^" then
                        (inpart(P,1)^^(inpart(P,2)-1)).SimDroLet(inpart(P,1),x)
                      else
                        if inpart(P,0) = "*" then
                          if numberp(inpart(P,1)) then
                            inpart(P,1)*SimDroLet(inpart(P,allbut(1)),x)
                          else print("SimDroLet --> ERREUR DANS POLYNOME")
                        else
                          if inpart(P,0) = "." then
                            inpart(P,1).SimDroLet(inpart(P,allbut(1)),x)
                          else
                            if inpart(P,0)="+" then
                              SimDroLet(inpart(P,1),x)
                                + SimDroLet(inpart(P,allbut(1)),x)
                            else print("SimDroLet --> ERREUR DANS POLYNOME")$


/*   Fonction qui renvoie le simplifie a droite du polynome <S> par le poly- */
/* nome <P>, <P> et <S> sont fournis en arguments.                            */


SimDroPol(S,P):= block(
                 setup_autoload("simdrolet.ldf",simdrolet),

                 if numberp(P) then P*S
                   else
                 if atom(P) then SimDroLet(S,P)
                     else
                   if inpart(P,0) = "+" then
                         SimDroPol(S,inpart(P,1))
                        + SimDroPol(S,inpart(P,allbut(1)))
                       else
                     if inpart(P,0) ="." then
                           SimDroPol(SimDroPol(S,inpart(P,allbut(1))),
                                                              inpart(P,1))
                       else
                      if inpart(P,0)= "*" then
                           if numberp(inpart(P,1)) then
                             inpart(P,1)*SimDroPol(S,inpart(P,allbut(1)))
                           else print("SimDroPol --> ERREUR DANS POLYNOME")
                         else
                       if inpart(P,0) = "^^" then
                             SimDroPol(SimDroPol(S,inpart(P,1)),
                                              inpart(P,1)^^(inpart(P,2)-1))
                           else print("SimDroPol --> ERREUR DANS POLYNOME"))$
```

```
/*   Cette fonction renvoie  le simplifie a gauche  du polynome  <P>  par la */
/* variable <x>.                                                              */

SimGauLet(p,x) := if numberp(p) then 0
          else
            if atom(p) then
              if p=x then 1
              else 0
            /* cas d'une puissance*/
            else
              if inpart(p,0) = "^^" then
                SimGauLet(inpart(p,1),x).(inpart(p,1)^^(inpart(p,2)-1))
              /* cas d'une multiplication par un scalaire*/
              else
                if inpart(p,0) = "*" then
                  /*if numberp(inpart(p,1)) then*/
                    inpart(p,1)*SimGauLet(inpart(p,allbut(1)),x)
                  /*else print("SimGauLet : ERREUR POLYNOME MAL ECRIT: ",p)*/
                else
                  if inpart(p,0) = "." then
                    SimGauLet(inpart(p,1),x).inpart(p,allbut(1))
                  else
                    if inpart(p,0)="+" then
                       SimGauLet(inpart(p,1),x)
                         + SimGauLet(inpart(p,allbut(1)),x)
                       else
                         print("SimGauLet --> ERREUR POLYNOME MAL ECRIT: ",p)$


/*  Cette fonction renvoie le simplifie a gauche du polynome <S> par le poly-*/
/* nome <P>, tous les deux fournis en arguments.                              */


SimGauPol(S,P):= block(
                setup_autoload("simgaulet.ldf",simgaulet),

                if numberp(P) then P*S
                  else
                if atom(P) then SimGauLet(S,P)
                    else
                  if inpart(P,0) = "^^" then
                        SimGauPol(SimGauPol(S,inpart(P,1)),
                                          inpart(P,1)^^(inpart(P,2)-1))
                      else
                    if inpart(P,0) ="." then
                       SimGauPol(SimGauPol(S,inpart(P,1)),
                             inpart(P,allbut(1)))
                      else
                     if inpart(P,0)= "*" then
                        inpart(P,1)*SimGauPol(S,inpart(P,allbut(1)))
                       else
                      if inpart(P,0) = "+" then
                      SimGauPol(S,inpart(P,1))
                                  + SimGauPol(S,inpart(P,allbut(1)))
                         else
                       print("SimGauPol --> ERREUR POLYNOME MAL ECRIT"))$
```

```
/*   Cette fonction renvoie  le simplifie du polynome <P>  par le mot de Lie */
/* <ML>, tous les deux fournies en arguments.                                 */

SimGauML(p,ML):= block(
            setup_autoload("simgaulet.ldf",simgaulet),

            if atom(ML) then SimGauLet(p,ML)
            else  SimGauML(SimGauML(p,first(ML)),last(ML))
                - SimGauML(SimGauML(p,last(ML)),first(ML)))$


/*   Cette fonction renvoie  le simplifie  a gauche  du polynome <S>  par le */
/* polynome de Lie <PL>.                                                      */

SimGauPL(S,PL) := block(
            setup_autoload("simgaulet.ldf",simgaulet),

            if atom(PL) then SimGauLet(S,PL)
            else
              if inpart(PL,0) = "[" then SimGauML(S,PL)
              else
                if inpart(PL,0) = "*" then
                  inpart(PL,1)*SimGauPL(S,inpart(PL,allbut(1)))
                else
                  if inpart(PL,0) = "+" then
                     SimGauPL(S,inpart(PL,1))
                   + SimGauPL(S,inpart(PL,allbut(1)))
                  else print("SimGauPL --> ERREUR DANS POLY. DE LIE")
            )$
```

```
decomp(u):=block(
        setup_autoload("simgaulet.ldf",simgaulet),

        if u=1 or atom(u) then
          (debut:u, suite:1)
        else
          if inpart(u,0) =="." or inpart(u,0) = "^^" then
            (decomp(inpart(u,1)), suite:SimGauLet(u,debut))$


EstMot(u) := if numberp(u) or atom(u) then true
             else
             if inpart(u,0) = "^^" then EstMot(inpart(u,1))
               else
               if inpart(u,0) = "." then
                   EstMot(inpart(u,1)) and EstMot(inpart(u,allbut(1)))
                 else false $


EstMon(u) := if EstMot(u) then true
             else
             if inpart(u,0) = "*" then
                 numberp(inpart(u,1)) and EstMot(inpart(u,allbut(1)))
               else false$


EstPol(p) := if EstMon(p) then true
             else
             if inpart(p,0) ="+" then
                 EstMon(inpart(p,1)) and EstPol(inpart(p,allbut(1)))
               else false $


MelMot(u,v) := block([debutu,suiteu,debutv,suitev,debut,suite],
               if EstMot(u) and EstMot(v) then
                 (if u=1 or v=1 then u*v
                  else
                    (decomp(u) , debutu :debut , suiteu:suite,
                     decomp(v),  debutv:debut,  suitev:suite,
                     debutu.MelMot(suiteu,v) + debutv.MelMot(u,suitev)
                    )
                 )
               else (print("MelMot --> LES MOTS SONT MAL ECRITS"),
                     print("u = ",u," v = ",v))
               )$


MelMon(m1,m2) := if EstMon(m1) and EstMon(m2) then
                   if EstMot(m1) then
                     if EstMot(m2) then MelMot(m1,m2)
                     else inpart(m2,1)*MelMot(m1,inpart(m2,allbut(1)))
                   else inpart(m1,1)*MelMon(inpart(m1,allbut(1)),m2)
                 else (print("MelMon --> ERREUR DANS LES MONOMES"),
                       print("m1 = ",m1,"m2 = ",m2))$
```

```
MelPol(p1,p2) := if EstPol(p1) and EstPol(p2) then
                   if EstMon(p1) then
                     if EstMon(p2) then MelMon(p1,p2)
                     else MelMon(p1,inpart(p2,1))
                            + MelPol(p1,inpart(p2,allbut(1)))
                   else MelPol(inpart(p1,1),p2)
                      + MelPol(inpart(p1,allbut(1)),p2)
                       else (print("MelPol --> ERREUR DANS LES POLYNOMES"),
                             print("P1 = ",p1,"      P2 = ",p2))$
```

```
/*   Cette fonction nous permet, etant donnes le cardinal m de l'alphabet et  */
/* le degre k jusqu'auquel on veut aller, de construire la base canonique de  */
/* l'algebre des polynomes non commutatifs. Cette base est ordonnee par       */
/* longueur et par l'ordre lexicographique pour chaque longueur.              */


BaseDeKX(m,k):=block([j,l,s,n,i],
      print(" "),
      print("      Calcul de la base canonique de R<X> pour m=",m," et k=",k),
      lis:[],
      if m=1 then
        (for i from 1 thru k+1 do
           (b[i]:(concat(x,0))^^(i-1),
            lis:endcons(b[i],lis)
           ),
         lis:rest(lis),
         dimtab:k+1
        )
       else
        (b[1]:1,
         for l from 0 thru k-1 do
           for j from m^l thru 2*m^l-1 do
             for s from 0 thru m-1 do
               (i:m*j+s-(m-2)*(m^(l+1)-1)/(m-1),
                n:j-(m-2)*(m^l-1)/(m-1),
                b[i]:b[n].concat(x,s),
                lis:endcons(b[i],lis)
               ),
         dimtab:i
        )
      )$
```

```
/*  Cette fonction permet de tester  si le monome de Lie <ll>  est inferieur */
/* au monome de Lie <mm> ou non.                                              */


estinf(ll,mm):=
        if atom(ll) then
          if atom(mm) then is(ll<=mm)
          else estinf(ll,first(mm))
        else
          if atom(mm) then estinf(first(ll),mm)
          else
            if is(first(ll)=first(mm)) then estinf(last(ll),last(mm))
            else estinf(first(ll),first(mm))$


/*   La fonction  <BaseDeLie>  permet de construire une base de Lie.  On lui */
/* passe comme parametres : m,  qui est le nombre de lettres de l'alphabet et*/
/* n, qui sera  le degre  jusqu'auquel on veut  aller. Dans notre cas n sera */
/* le degre  total de  la serie a etudier.                                    */


BaseDeLie(m,n):=block([j,ns,ur,i,tt:[]],local(aa,bb),
        blie:[],
        print(" "),
        print("    Calcul de la base de Lyndon de Lie<X> pour m=",m," et k=",n),
        for j:0 thru m-2 do assume(concat(x,j)<concat(x,j+1)),
          for i:0 thru m-1 do
            (lli[1,i]:concat(x,i),
             blie:endcons(lli[1,i],blie),
             mli[1,i]:concat(x,i)
            ),
        la[1]:m,
        if m>1 then
          (ns:2,
           while ns <= n do
             (i:0,
              if oddp(ns) then trait1(ns) else trait2(ns),
              la[ns]:i,
              ns:ns+1
             )
          )
       )$


/* Le parametre ns que l'on passe a la fonction <trait1> represente le numero*/
/* de la ligne que l'on veut construire; c'est aussi  le degre des <monomes> */
/* de Lie que cette ligne contient.                                           */


trait1(ks):=block([j],
        for j:1 thru (ks-1)/2 do const1(j,ks-j),
        for j:1 thru (ks-1)/2 do const2(ks-j,j)
       )$
```

```
/* <trait2> comme <trait1> prend pour parametre le numero de la ligne a */
/* construire, mais dans le cas ou ce numero est pair.                  */


trait2(ks):=block([j],
        for j:1 thru ks/2-1 do const1(j,ks-j),
        const3(ks/2),
        for j:1 thru ks/2-1 do const2(ks-j,j)
       )$


/* A la fonction <const1> on passe deux parametres qui representent les */
/* numeros des deux lignes a partir desquelles on va construire la ligne */
/* courante. Vous remarquerez qu'en meme temps on construit une liste <tt> */
/* qui va nous permettre d'eviter de construire des monomes dont les mots */
/* associes sont egaux.                                                  */

const1(a1,b1):=block([r,k,l],
       tt:[],
       r:0,
       for k:0 thru la[a1]-1 do
         for l:0 thru la[b1]-1 do
            (if estinf(lli[a1,k],lli[b1,l]) then
               (lli[ns,i]:[lli[a1,k],lli[b1,l]],
                blie:endcons(lli[ns,i],blie),
                mli[ns,i]:mli[a1,k].mli[b1,l],
                tt:endcons(mli[ns,i],tt),
                i:i+1
               )
             else
               (aa[a1,r]:k,bb[b1,r]:l,
                r:r+1
               )
            ),
       ur:r
      )$


/* Cette fonction permet de construire les monomes qui n'ont pas ete obtenus*/
/* par <const1>. On lui passe comme parametres les numeros des lignes a */
/* partir desquelles se fait la construction.                           */


const2(z,t):=block([a,r],
       for r:0 thru ur-1 do
         (a:mli[z,bb[z,r]].mli[t,aa[t,r]],
          if not(member(a,tt)) then
            (mli[ns,i]:a,
             lli[ns,i]:[lli[z,bb[z,r]],lli[t,aa[t,r]]],
             blie:endcons(lli[ns,i],blie),
             i:i+1,
             tt:endcons(a,tt)
            )
         )
      )$
```

```
/*  Cette fonction est utilisee  uniquement dans le cas ou ns est pair. Elle */
/* permet donc  de construire  des monomes  a partir de la ligne ns/2.       */


const3(ka):=block([a,j,k],
       for j:0 thru la[ka]-1 do
         for k:0 thru la[ka]-1 do
           if not(j=k) and estinf(lli[ka,j],lli[ka,k]) then
             (a:mli[ka,j].mli[ka,k],
              if not(member(a,tt)) then
                (mli[ns,i]:a,
                 lli[ns,i]:[lli[ka,j],lli[ka,k]],
                 blie:endcons(lli[ns,i],blie),
                 i:i+1,
                 tt:endcons(a,tt)
                )
             )
      )$
```

```
/*  Cette  fonction permet,  etant donnee  une serie <S>,  d'isoler le terme  */
/* constant de celle-ci.                                                      */

Constant(S) :=
        if numberp(S) then S
        else
          if atom(S) then 0
          else
            if inpart(S,0)="+" then
              if numberp(inpart(S,1)) then inpart(S,1)
              else 0
            else 0 $


/*  Cette fonction  prend comme argument  une serie <S>  et nous renvoie son  */
/* degre total.                                                               */

DegTo(S) :=
        if numberp(S) then 0
        else
          if atom(S) then 1
          else
            if inpart(S,0) = "^^" then inpart(S,2)*DegTo(inpart(S,1))
            else
              if inpart(S,0) = "." then
                DegTo(inpart(S,1)) + DegTo(inpart(S,allbut(1)))
              else
                if inpart(S,0) = "*" then DegTo(inpart(S,allbut(1)))
                else
                  if inpart(S,0) = "+" then max(DegTo(inpart(S,1)),
                                                DegTo(inpart(S,allbut(1))))
```

```
/*  Cette fonction construit  la matrice de Lie-Hankel et  le rang de Lie du */
/* polynome <S>. Ladite matrice est mise dans la matrice <mat> et le rang de */
/* Lie est mis dans <r>.                                                     */


matlyn(S,m) := block([i,j,jj,k,p,rep,tampon,l,liss],local(la),
        setup_autoload("consdegto.ldf",constant,degto),
        setup_autoload("basedelie.ldf",basedelie,trait1,trait2,const1,
                                       const2,const3,estinf),
        setup_autoload("basedekx.ldf",basedekx),
        setup_autoload("simgaulie.ldf",simgauml,simgaupl),
        setup_autoload("listmot.ldf",listmot),

        k:DegTo(S),
        BaseDeKX(m,k),
        iann:1,
        lpp:[],
        if m=1 then
          (tampon:SimGauML(S,x0),
           if tampon#0 then
             (lyn[1]:x0,
              lpp:endcons(x0,lpp),
              l:coefmatrix([tampon],lis),
              l:cons(constant(tampon),l[1]),
              MatLieHan:matrix(l),
              nbligne:1)
           else (Ann[iann]:x0,iann:iann+1),
                 lyn1[1]:x0
                )
        else
          (BaseDeLie(m,k),
           print(" "),
           print("     Construction de la matrice de Lie-Hankel"),
           nbligne:0,
           r:0,
           i:1,jj:1,
           dotexptsimp:false,
           for j:1 thru k do
             for n from 0 thru la[j]-1 do
               (tampon:SimGauML(S,lli[j,n]),
                mot:listmot(lli[j,n]),
                if tampon#0 then
                  (lyn[i]:mot,
                   i:i+1,
                   lpp:endcons(lli[j,n],lpp),
                   l: coefmatrix([tampon],lis),
                   l:cons(constant(tampon),l[1]),
                   l:matrix(l),
                   if r = 0 then (MatLieHan :l , r : 1)
                   else MatLieHan:addrow(MatLieHan,l),
                   nbligne:nbligne+1
                  )
                else (Ann[iann]:lli[j,n],iann:iann+1),
                lyn1[jj]:mot,
                jj:jj+1
               ),
           dotexptsimp:true
          ),
```

```
 iann:iann-1,

 print(" "),
 rep:read("     Voulez-vous afficher cette matrice<o/n>?: "),
 if rep=o then
   (print(" "),
    print("     La matrice de Lie-Hankel est:"),
    print(" "),
    print("MatLieHan= ",matliehan)
   ),
 ind:r:rank(MatLieHan),
 print(" "),
 print("     Le rang de Lie de S est : ",r),
 print(" "),
 read("          Tapez <c\;> puis <Return> pour continuer :"),
 clear()
)$
```

```
/*   Cette fonction renvoie <true>  si les deux listes fournies en arguments */
/* sont egales, et <false> sinon.                                             */


comparelist(l1,l2):=block([bool],
        bool:false,
        if l1=[] and l2=[] then bool:true
        else
          if length(l1)=length(l2) then
            if member(first(l1),l2) then
              (l2:delete(first(l1),l2,1),
               comparelist(rest(l1),l2)
              )
        )$



/*   Cette fonction renvoie  <true> si la liste  <l>  fournie en argument se */
/* trouve dans le tableau <t> , et <false> sinon.                             */


recherche(l,t,k1,k2):=block([i],
        i:k1,
        while i#(k2+1) and not(comparelist(l,t[i])) do i:i+1,
        if i=k2+1 then false  else true
       )$



/*   Cette fonction effectue la resolution  d'un systeme d'equations dans le */
/* but d'exprimer un polynome <P>  en fonction  des melanges des coordonnees */
/* locales <Zi>.                                                              */


resolution(P,indf,tablf):=block([i,r:0,listeq,listvar,liste:0,mat,systeq,sol],

        for i from 1 thru indf do
          (liste:coefmatrix([tablf[i]],lis),
           if r=0 then (mat:liste,r:1)
           else mat:addrow(mat,liste)
          ),

        systeq:0,
        for i from 1 thru indf do
          systeq:systeq+(a[i]*row(mat,i)),

        listvar:makelist(a[i],i,1,indf),

        liste:coefmatrix([P],lis),
        systeq:systeq-liste,

        listeq:makelist(systeq[1,i],i,1,dimtab-1),

        sol:solve(listeq,listvar)
       )$
```

```
/*   Cette fonction permet de calculer les relations de dependance entre les */
/* lignes de la matrice donnee en argument .                                  */


dependance(matrice):=block([i,listeq,listvar,systeq:0,an,F],local(alpha),

          for i from 1 thru nbligne do
            systeq:systeq+(alpha[i]*row(matrice,i)),

          listvar:makelist(alpha[i],i,1,nbligne),
          listeq:makelist(systeq[1,i],i,1,dimtab-1),

          %rnum:0,
          solve(listeq,listvar),
          lp:makelist(concat(P,i),i,1,nbligne),
          if %rnum#0 then
            (print("                Les relations de dependance sont :"),
             print(" "),
             listarith:false,
             for j:1 thru %rnum do
               (canon(%rnum,j),
                F:sum(alpha[j]*lp[j],j,1,nbligne),
                an:sum(alpha[j]*lpp[j],j,1,nbligne),
                FF[j]:ev(F),
                iann:iann+1,
                ann[iann]:ev(an),
                print("     ",concat(RD,j)," = ",ff[j])
               ),
             listarith:true
            )
          )$

/*  Cette  macro  permet de donner des valeurs canoniques aux parametres %rj  */

canon(n,j)::=buildq([n,j,i],
                for i:1 thru n do concat(%r,i)::(if i=j then 1 else 0))$



/*  Cette macro retourne la position de %rj dans la liste li.                 */

posi(li,j)::=buildq([li,j], block([test,k:1],
        test:inpart(li[k],2)=concat(%r,j),
        while k<= nbligne and not(test) do
          (k:k+1,
           test:inpart(li[k],2)=concat(%r,j)),
           k
          )
       )$



/* Cette fonction transforme un mot en une liste de lettres                   */
motlist(w):=if atom(w) then [w]
            else cons(first(w),motlist(rest(w)))$
```

```
/*  Cette fonction permet de calculer  une base de l'application definie par  */
/* les relations de dependance calculees par la fonction DEPENDANCE           */

noydef(table,nbrel):=block([i,j,sol,k,test,temp,nb,vect,listequ,syst],local(V),
         vect:makelist(concat(a,i),i,1,nbligne),
         MF:coefmatrix([table[1]],lp),
         if nbrel=1 then listequ:[MF.transpose(vect)]
         else
           (for j:2 thru nbrel do
              MF:addrow(MF,coefmatrix([table[j]],lp)),
            print("               La matrice de l'application lineaire F
                  definie par ces relations est :"),
                  print(" "),
                  print("     MF = ",MF),
                  Syst:MF.transpose(vect),
                  listequ:makelist(Syst[j,1],j,1,nbrel)
                 ),
             nb:nbrel,
             %rnum:0,
             sol:solve(reverse(listequ),vect),
             sol:first(sol),
             for j:1 thru nbligne-nb do
               (canon(nbligne-nb,j),
                V[posi(sol,j)]:ev(sol)
               ),
             VV:listarray(V)
            )$



/* Cette fonction definit le mot de Lyndon de plus haut degre dans le polynome*/
/* (coordonnee locale) <P>.                                                   */

lyndegmax(P):=block([],
        if (atom(P) or inpart(P,0)=".") then P
        else
          if inpart(P,0)="*" then inpart(P,2)
          else
            if inpart(P,0)="+" or inpart(P,0)="-" then
              if degto(inpart(P,1))>degto(inpart(P,allbut(1))) then
                lyndegmax(inpart(P,1))
              else lyndegmax(inpart(P,allbut(1)))
            else print("lyndegmax --> ERREUR DANS LE POLYNOME : ",P))$



/* Cette fonction  definit le polynome de Lie  correspondant au mot de Lyndon */
/*determine par la fonction <lyndegmax>.                                      */

repolie(P):=block([m,l],
        setup_autoload("duval.ldf",inversion,lexilong,motlyn,baselyn,
                  motlist,inflyn,conslyn,decomp),
        setup_autoload("consdegto.ldf",constant,degto),
        m:lyndegmax(P),
        dotexptsimp:false,
        l:motlist(m),
        dotexptsimp:true,
        decomp(l))$
```

```
/* Cette fonction permet de verifier la premiere condition sur les coordonnees*/
/*locales; a savoire <q_i|P_j>=delta_ij.                                        */

ortho(P,Plie,j):=block([i:1,bo:true,sim,con],
        setup_autoload("simgaulie.ldf",simgaupl,simgauml),
        setup_autoload("consdegto.ldf",constant,degto),

        while i<=r and bo do
          (sim:simgaupl(P,Plie[i]),
           con:constant(sim),
           if i=j then (bo:bo and ev(con#0),const:con)
           else bo:bo and ev(con=0),
           i:i+1
          ),
        bo
        )$


/*  Cette fonction permet de verifier si les poly. obtenus sont dans V(A(S))  */

dansag(Z):=block([k,j,ql,test,tamp],

        for j:1 thru r do
          (for k:1 thru iann do
             (test:simgaupl(Z[j],ann[k]),
              if test#0 then
                (dotexptsimp:true,
                 Z[j]:completer(Z[j],j,test,ann[k]),
                 dotexptsimp:false
                )
             ),
           print("     ",concat(q,j)," = ", Z[j])
          ),
        ql:makelist(concat(q,j),j,1,r),
        print("     Les coordonnees locales sont les polynomes : ",ql),
        print(" "),
        read("          Tapez <c\;> puis <Return> pour continuer :"),
        clear()
        )$


/* Cette fonction permet de determiner les mots de Lyndon et leurs melanges   */
/* qui sont utiles pour l'expression d'un polynome donne.                     */

LynUtiles(P,DP):=block([k:0,mm,n,fin:1,inddeb:1,indut,deg,list],

        setup_autoload("consdegto.ldf",constant,degto),
        setup_autoload("recherche.ldf",comparelist,recherche),
        setup_autoload("degpolie.ldf",degpolie),

        for n from 1 thru Nblyn do
            (deg:degto(lyn1[n]),
             if deg#0 and deg<= DP then
               (fin:0,k:k+1,
                tablyn[k]:[n],
                somdeglyn[k]:deg
               )
            ),
```

```
        indfinlyn:indut:k,
        while  fin=0 do
          (fin:1,
           for n from inddeb thru indfinlyn do
             (for mm from 1 thru indut do
                (deg:somdeglyn[mm]+somdeglyn[n],
                   if deg#0 and deg<=DP then
                     (list:endcons(first(tablyn[mm]),tablyn[n]),
                      if not(recherche(list,tablyn,indfinlyn+1,k)) then
                        (fin:0,k:k+1,
                         tablyn[k]:list,
                         somdeglyn[k]:deg
                        )
                     )
                )
             ),
           inddeb:indfinlyn+1,
           indfinlyn:k
          )
        )$


completer(cl,nn,t,f):=block([i,j,n,d,sol,indfinlyn,simp,comp,const:1,comp1,tamp,
                             tamp1,bool,expr,exprs,lequa,lvar,sole,expre],
                            local(a),
        setup_autoload("consdegto.ldf",constant,degto),
        setup_autoload("recherche.ldf",comparelist,recherche),
        setup_autoload("simgaulie.ldf",simgaupl,simgauml),
        setup_autoload("degpolie.ldf",degpolie),

        d:degto(t)+degpolie(f),
        lynutiles(t,d),
        for n:1 thru indfinlyn do
          (tamp:melliste(tablyn[n],lyn1),
           tabsimp[n]:simgaupl(tamp,f),
           tablemelyn[n]:tamp
          ),
        %rnum:0,remarray(a),
        kill(allbut(functions,arrays,
                    s,ss,m,rep,ind,liste,r,h,i,j,mm,lis,indfin,dimtab,bool,cl,
                    nbligne,kk,matliehan,iann,lpp,blie,mot,mat,lp,sol,f,mf,vv,
                    nblyn,z,k,ql,test,tamp,nn,t,n,d,indfinlyn,comp,const,comp1,
                    tamp1,expr,exprs,lequa,lvar,sole,expre)),
        print("     Expression de ce polynome sur ces melanges :"),
        sol:resolution(-ratsimp(t),indfinlyn,tabsimp),
        comp:constant(-t),
        for i:1 thru indfinlyn do
          comp:comp+a[i]*tablemelyn[i],
        j:1,
        while (j <= %rnum) do
          (canon(%rnum,j),
           comp1:ev(ratsimp(comp)),
           tamp1:expand(cl+comp1),
           simp:simgaupl(tamp1,f),
           if simp=0 then
             (bool:ortho(tamp1,Plie,nn),
              if bool then j:%rnum +1 else j:j+1
             )
```

```
          else j:j+1
        ),
      if const#0 then expand((1/const)*tampl)
      else expand(tampl)
      )$

/*  Cette fonction permet de calculer les coordonnees locales et de verifier  */
/* que celles-ci sont bien dans V(A(S)).                                       */

lyncoord(mat):=block([i,j,lp,sol,F,MF,VV,m,NBlyn],local(FF),

      dependance(mat),
      Nblyn:iann+nbligne-%rnum,print(" "),
      if %rnum#0 then
        (NoyDeF(FF,%rnum),
         print("     Les polynomes obtenus par ces relations sont :"),
         for j:1 thru r do
           (Z[j]:sum(VV[j][i]*Lyn[i],i,1,nbligne),
            Plie[j]:repolie(Z[j]),
            print("     ",concat(m,j)," = ",Z[j])
           )
        )
      else
        (print("     Les polynomes obtenus par ces relations sont :"),
         for j:1 thru r do
           (Z[j]:Lyn[j],
            Plie[j]:repolie(Z[j]),
            print("     ",concat(m,j)," = ",Z[j])
           )
        ),
      print(" "),
      print("Verifions que ces polynomes forment un systeme de coordonnees
             locales"),
      print("     sinon completons les."),
      dansag(Z)
     )$
```

```
/*  Cette fonction qui calcule les actions a droite des lettres <xj> sur les  */
/* coordonnees locales <Zi>.                                                   */


actionx(t):=block([i,y,j],
        setup_autoload("consdegto.ldf",constant,degto),
        setup_autoload("simdrolet.ldf",simdrolet),

        for j from 0 thru (m-1) do
          for i from 1 thru ind do
            (y:concat(x,j),
             matx[i,j]:simdrolet(t[i],y),
             degre[i,j]:degto(matx[i,j])
            )
      )$



/* Cette fonction permet, en appelant la fonction <exprime> sur chaque compo- */
/* sante de chacun  des chapms de vecteurs, d'exprimer celles-ci comme combi- */
/* naisons lineaires  des melanges  des coordonnees locales.  Ensuite,  elle  */
/* donne l'expression de chacun des champs de vecteurs.                       */


chvect(tabl):=block([i,j],

        for j from 0 thru (m-1) do
          for i from 1 thru ind do
            if not(numberp(tabl[i,j])) then
              chvec[i,j]:exprime(tabl[i,j],Plie,Z)
            else chvec[i,j]:tabl[i,j],

        print(" "),
        print ("               Les champs de vecteurs sont :"),
        print ("               ____________________________"),
        print (" "),

        dotdistrib:false,
        dotconstrules:false,
        for i from 0 thru m-1 do
          (mm:0,
           for j from 1 thru ind do
             if chvec[j,i]#0 then mm:mm+(chvec[j,i].(d/concat(dq,j))),
           print("     ",concat(Y,i)," = ",mm)
          ),
        dotconstrules:true,
        dotdistrib:true
       )$



/*Cette fonction calcule l'observation et les champs de vecteurs en calculant:*/
/*               -->   le rang de Lie, par <matlyn>                           */
/*               -->   les coordonnees locales, par <lyncoord>                */
/*               -->   la fonction d'observation, par <exprime>               */
/*               -->   les actions des xj sur les Zi, par <Actionx>           */
/*               -->   les champs de vecteurs par <chvect>                    */
```

```
calyn(S,m):=block([i,j,mm,lis,indfin,dimtab,bool,nbligne,kk,MatLieHan],
                  local(matx,degre,a,z,tableau,somdeg,tablemel,la,mli),

        if numberp(s) then
          (print("                        Rang de Lie = 0"),print(" "),
           print("          Il n'y a pas de champs de vecteurs ")
          )
        else
          (setup_autoload("matlyn.ldf",matlyn),
           setup_autoload("coordllyn.ldf",lyncoord,dependance,noydef,dansag,
                                          posi,canon),
           setup_autoload("action.ldf",actionx),
           setup_autoload("listeutile.ldf",coordutiles),
           setup_autoload("melliste.ldf",melliste,melutil,produit),
           setup_autoload("resolution.ldf",resolution),
           setup_autoload("exprime.ldf",exprime,puismel),


           matlyn(S,m),
           lyncoord(matliehan),
           Actionx(Z),
           h:exprime(S,Plie,Z),
           print(" "),
           print("                La fonction d'observation est :"),
           print("                _____________________________"),
           print(" "),
           print("     H = ",h),
           chvect(matx)
          ),
         read("   Tapez <c\;> puis <RETURN> pour continuer:")
         )$
```

```
/* Cette fonction permet  d'appliquer le champ de vecteurs  Yj  au polynome   */
/* commutatif P sur les variables qi.                                          */

Composition(P,j):=block([i],
          qq:0,
          for i:1 thru n do qq:qq+tab[i,j]*diff(P,concat(q,i)))$



/* Cette fonction permet de trouver le mot associe a une liste d'indices.     */

Motass(l):= if l=[] then 1
          else concat(x,first(l)).Motass(rest(l))$



/* Cette macro permet d'empiler une valeur dans la pile <pile>.               */

push(val,pile):=buildq([val,pile],pile:cons(val,pile))$



/* Cette macro permet de depiler une valeur de la pile <pile>.                */

pop(pile):=buildq([pile],
          block([temp:first(pile)], pile:rest(pile), temp))$



/* Cette fonction utilise les precedantes pour calculer la serie generatrice  */
/* correspondant au systeme trouve par l'algorithme de la realisation.        */

Verifier(tab,h,n):=block([i,posc:0,pilevide:true,constante,sa,qq,t,l,pile:[]],

          constante:sa:constant(S),
          t:[h,[],0],
          push(t,pile),
          while pile#[] do
            (l:pop(pile),
             if (l[3]<m) and (not(numberp(l[1]))) then
               (t:[l[1],l[2],l[3]+1],
                push(t,pile),
                Composition(l[1],l[3]),
                constante:constant(qq),
                sa:sa+constante*motass(cons(l[3],l[2])),
                t:[qq,cons(l[3],l[2]),0],
                push(t,pile)
               )
            ),
          print(" "),
          print("          La serie correspondant a vorte realisation est:"),
          print(" "),
          print(sa),
          print(" "),
          read("   Tapez <c\;> puis <return> pour continuer:")
        )$
```

EXEMPLES

PROGRAMME CALCULANT LA REALISATION

DES SYTEMES DYNAMIQUES

DE SERIE GENERATRICE FINIE

MENU PRINCIPAL

1- CALCULS INDEPENDANTS.

2- CALCULER LA REALISATION.

3- VERIFIER LA REALISATION.

4- FIN.

Tapez le numero de votre choix suivi de ; :

2;

Entrez le cardinal de votre alphabet termine par ; :

2;

Votre alphabet sera desormais le suivant :

[x0, x1]

Entrez votre serie terminee par ; :

x0.x1+x1.x0+x0.x1^^2+(x1^^2).x0;

```
        Calcul de la base canonique de R<X> pour m= 2  et k= 3


        Calcul de la base de Lyndon de Lie<X> pour m= 2  et k= 3


     Construction de la matrice de Lie-Hankel
n;


     Voulez-vous afficher cette matrice<o/n>?:


     Le rang de Lie de S est :  4


          Tapez <c;> puis <Return> pour continuer :
c;

     Les polynomes obtenus par ces relations sont :

      m1  =  x0

      m2  =  x1

      m3  =  x0 . x1

      m4  =  x0 . x1 . x1


     Verifions que ces polynomes forment un systeme de coordonnees locales

     sinon completons les.

      q1  =  x0

      q2  =  x1

      q3  =  x0 . x1

      q4  =  x0 . x1 . x1

     Les coordonnees locales sont les polynomes :  [q1, q2, q3, q4]


          Tapez <c;> puis <Return> pour continuer :
c;
```

La fonction d'observation est :

$$H = 2\,q4 - q2\,q3 + \frac{q1^2\,q2^2}{2} + q1\,q2$$

Les champs de vecteurs sont :

$$y0 = \frac{d}{dq1}$$

$$y1 = q3 \cdot \frac{d}{dq4} + q1 \cdot \frac{d}{dq3} + \frac{d}{dq2}$$

Tapez <c;> puis <RETURN> pour continuer:

c;

PROGRAMME CALCULANT LA REALISATION

DES SYTEMES DYNAMIQUES

DE SERIE GENERATRICE FINIE

MENU PRINCIPAL

1- CALCULS INDEPENDANTS.

2- CALCULER LA REALISATION.

3- VERIFIER LA REALISATION.

4- FIN.

Tapez le numero de votre choix suivi de ; :

3;

La serie correspondant a vorte realisation est:

$$x1^{\langle 2\rangle} \cdot x0 + x1 \cdot x0 + x0 \cdot x1^{\langle 2\rangle} + x0 \cdot x1$$

Tapez <c;> puis <return> pour continuer:

PROGRAMME CALCULANT LA REALISATION

DES SYTEMES DYNAMIQUES

DE SERIE GENERATRICE FINIE

MENU PRINCIPAL

1- CALCULS INDEPENDANTS.

2- CALCULER LA REALISATION.

3- VERIFIER LA REALISATION.

4- FIN.

Tapez le numero de votre choix suivi de ; :

2;

Entrez le cardinal de votre alphabet termine par ; :

2;

Votre alphabet sera desormais le suivant :

[x0, x1]

Entrez votre serie terminee par ; :

x1.x0;

```
     Calcul de la base canonique de R<X> pour m= 2  et k= 2


     Calcul de la base de Lyndon de Lie<X> pour m= 2  et k= 2


     Construction de la matrice de Lie-Hankel


     Voulez-vous afficher cette matrice<o/n>?:
o;

     La matrice de Lie-Hankel est:

                [  0   1  0  0  0  0  0 ]
MatLieHan=      [                       ]
                [ - 1  0  0  0  0  0  0 ]


     Le rang de Lie de S est :  2

          Tapez <c;> puis <Return> pour continuer :
c;

     Les polynomes obtenus par ces relations sont :

      m1  =  x1

      m2  =  x0 . x1

     Verifions que ces polynomes forment un systeme de coordonnees locales
     sinon completons les.

      q1  =  x1

     Calcul des melanges utiles pour le polynome dont le reste par x0 est : x1

     Expression de ce polynome sur ces melanges :

      q2  =  - x1 . x0

     Les coordonnees locales sont les polynomes :  [q1, q2]

          Tapez <c;> puis <Return> pour continuer :
c;
```

La fonction d'observation est :

$$H = -q2$$

Les champs de vecteurs sont :

$$y0 = (-q1) \cdot \frac{d}{dq2}$$

$$y1 = \frac{d}{dq1}$$

Tapez <c;> puis <RETURN> pour continuer:
c;

PROGRAMME CALCULANT LA REALISATION

DES SYTEMES DYNAMIQUES

DE SERIE GENERATRICE FINIE

MENU PRINCIPAL

1- CALCULS INDEPENDANTS.

2- CALCULER LA REALISATION.

3- VERIFIER LA REALISATION.

4- FIN.

Tapez le numero de votre choix suivi de ; :

3;

La serie correspondant a vorte realisation est:

x1 . x0

Tapez <c;> puis <return> pour continuer:

c;

PROGRAMME CALCULANT LA REALISATION

DES SYTEMES DYNAMIQUES

DE SERIE GENERATRICE FINIE

MENU PRINCIPAL

1- CALCULS INDEPENDANTS.

2- CALCULER LA REALISATION.

3- VERIFIER LA REALISATION.

4- FIN.

Tapez le numero de votre choix suivi de ; :

2;

Entrez le cardinal de votre alphabet termine par ; :

2;

Votre alphabet sera desormais le suivant :

[x0, x1]

Entrez votre serie terminee par ; :

x1.x0.x1;

```
     Calcul de la base canonique de R<X> pour m= 2  et k= 3

     Calcul de la base de Lyndon de Lie<X> pour m= 2  et k= 3

     Construction de la matrice de Lie-Hankel

     Voulez-vous afficher cette matrice<o/n>?:
o;

     La matrice de Lie-Hankel est:

                [  0   0   0   0  1  0  0  0  0  0  0  0  0  0  0 ]
                [                                                  ]
MatLieHan=      [  0   0  - 1  0  0  0  0  0  0  0  0  0  0  0  0 ]
                [                                                  ]
                [ - 2  0   0   0  0  0  0  0  0  0  0  0  0  0  0 ]

     Le rang de Lie de S est :  3

          Tapez <c;> puis <Return> pour continuer :
c;
```

Les polynomes obtenus par ces relations sont :

m1 = x1

m2 = x0 . x1

m3 = x0 . x1 . x1

Verifions que ces polynomes forment un systeme de coordonnees locales sinon completons les.

q1 = x1

Calcul des melanges utiles pour le polynome dont le reste par x0 est : x1

Expression de ce polynome sur ces melanges :

q2 = - x1 . x0

Calcul des melanges utiles pour le polynome dont le reste par x0 est :

x1 . x1

Expression de ce polynome sur ces melanges :

$$q3 = - x1^{\langle 2 \rangle} . x0 - x1 . x0 . x1$$

Les coordonnees locales sont les polynomes : [q1, q2, q3]

Tapez <c;> puis <Return> pour continuer :

c;

La fonction d'observation est :

$$H = q1\ q2 - 2\ q3$$

Les champs de vecteurs sont :

$$y0 = \left(-\frac{q1^2}{2}\right) \cdot \frac{d}{dq3} + (-q1) \cdot \frac{d}{dq2}$$

$$y1 = q2 \cdot \frac{d}{dq3} + \frac{d}{dq1}$$

Tapez <c;> puis <RETURN> pour continuer:

c;

PROGRAMME CALCULANT LA REALISATION

DES SYTEMES DYNAMIQUES

DE SERIE GENERATRICE FINIE

MENU PRINCIPAL

1- CALCULS INDEPENDANTS.

2- CALCULER LA REALISATION.

3- VERIFIER LA REALISATION.

4- FIN.

Tapez le numero de votre choix suivi de ; :

3;

La serie correspondant a vorte realisation est:

x1 . x0 . x1

Tapez <c;> puis <return> pour continuer:

c;

PROGRAMME CALCULANT LA REALISATION

DES SYTEMES DYNAMIQUES

DE SERIE GENERATRICE FINIE

MENU PRINCIPAL

1- CALCULS INDEPENDANTS.

2- CALCULER LA REALISATION.

3- VERIFIER LA REALISATION.

4- FIN.

Tapez le numero de votre choix suivi de ; :

```
2;
```

Entrez le cardinal de votre alphabet termine par ; :

```
2;
```

Votre alphabet sera desormais le suivant :

[x0, x1]

Entrez votre serie terminee par ; :

```
x0+x0^^4+x0.x1.x0^^2-x0.x1^^2-(x0^^3).x1+x0.x1^^3;
```

```
     Calcul de la base canonique de R<X> pour m= 2  et k= 4

     Calcul de la base de Lyndon de Lie<X> pour m= 2  et k= 4

     Construction de la matrice de Lie-Hankel

     Voulez-vous afficher cette matrice<o/n>?:
n;

     Le rang de Lie de S est :  5

          Tapez <c;> puis <Return> pour continuer :
c;

               Les relations de dependance sont :

                      p5
      rdl  =  p6  -   --
                      2

     Les polynomes obtenus par ces relations sont :

      m1  =  x0

      m2  =  x0 . x1

      m3  =  x0 . x0 . x1

      m4  =  x0 . x1 . x1

      m5  =  x0 . x1 . x1 . x1 + 2 (x0 . x0 . x0 . x1)

     Verifions que ces polynomes forment un systeme de coordonnees locales
     sinon completons les.

      q1  =  x0

      q2  =  x0 . x1

      q3  =  x0 . x0 . x1

      q4  =  x0 . x1 . x1

      q5  =  x0 . x1 . x1 . x1 + 2 (x0 . x0 . x0 . x1)

     Les coordonnees locales sont les polynomes :  [q1, q2, q3, q4, q5]

          Tapez <c;> puis <Return> pour continuer :
c;
```

La fonction d'observation est :

$$H = q5 - q4 - 2\,q1\,q3 + \frac{q1^2\,q2}{2} + \frac{q1^4}{24} + q1$$

Les champs de vecteurs sont :

$$y0 = \frac{d}{dq1}$$

$$y1 = \left(q4 + \frac{q1^3}{3}\right) \cdot \frac{d}{dq5} + q2 \cdot \frac{d}{dq4} + \frac{q1^2}{2} \cdot \frac{d}{dq3} + q1 \cdot \frac{d}{dq2}$$

Tapez <c;> puis <RETURN> pour continuer:
c;

PROGRAMME CALCULANT LA REALISATION

DES SYTEMES DYNAMIQUES

DE SERIE GENERATRICE FINIE

MENU PRINCIPAL

1- CALCULS INDEPENDANTS.

2- CALCULER LA REALISATION.

3- VERIFIER LA REALISATION.

4- FIN.

Tapez le numero de votre choix suivi de ; :

3;

La serie correspondant a vorte realisation est:

$$- x0^{\langle 3\rangle} \cdot x1 + x0 \cdot x1^{\langle 3\rangle} \quad x0 \cdot x1^{\langle 2\rangle} + x0 \cdot x1 \cdot x0^{\langle 2\rangle} + x0^{\langle 4\rangle} + x0$$

Tapez ‹c;› puis ‹return› pour continuer:

c;

PROGRAMME CALCULANT LA REALISATION

DES SYTEMES DYNAMIQUES

DE SERIE GENERATRICE FINIE

MENU PRINCIPAL

1- CALCULS INDEPENDANTS.

2- CALCULER LA REALISATION.

3- VERIFIER LA REALISATION.

4- FIN.

Tapez le numero de votre choix suivi de ; :

2;

Entrez le cardinal de votre alphabet termine par ; :

2;

Votre alphabet sera desormais le suivant :

[x0, x1]

Entrez votre serie terminee par ; :

x0+x0^^4+x0.x1.x0^^2-x0.x1^^2-(x0^^3).x1+x0.x1^^3;

Calcul de la base canonique de R<X> pour m= 2 et k= 4

Calcul de la base de Lyndon de Lie<X> pour m= 2 et k= 4

Construction de la matrice de Lie-Hankel

Voulez-vous afficher cette matrice<o/n>?:
n;

Le rang de Lie de S est : 5

Tapez <c;> puis <Return> pour continuer :
c;

Les relations de dependance sont :

$$rd1 = p6 - \frac{p5}{2}$$

Les polynomes obtenus par ces relations sont :

m1 = x0

m2 = x0 . x1

m3 = x0 . x0 . x1

m4 = x0 . x1 . x1

m5 = x0 . x1 . x1 . x1 + 2 (x0 . x0 . x0 . x1)

Verifions que ces polynomes forment un systeme de coordonnees locales sinon completons les.

q1 = x0

q2 = x0 . x1

q3 = x0 . x0 . x1

q4 = x0 . x1 . x1

q5 = x0 . x1 . x1 . x1 + 2 (x0 . x0 . x0 . x1)

Les coordonnees locales sont les polynomes : [q1, q2, q3, q4, q5]

Tapez <c;> puis <Return> pour continuer :
c;

La fonction d'observation est :

$$H = q5 \cdot q4 - 2\, q1\, q3 + \frac{q1^2\, q2}{2} + \frac{q1^4}{24} + q1$$

Les champs de vecteurs sont :

$$y0 = \frac{d}{dq1}$$

$$y1 = \left(q4 + \frac{q1^3}{3}\right) \cdot \frac{d}{dq5} + q2 \cdot \frac{d}{dq4} + \frac{q1^2}{2} \cdot \frac{d}{dq3} + q1 \cdot \frac{d}{dq2}$$

Tapez <c;> puis <RETURN> pour continuer:

c;

PROGRAMME CALCULANT LA REALISATION

DES SYTEMES DYNAMIQUES

DE SERIE GENERATRICE FINIE

MENU PRINCIPAL

1- CALCULS INDEPENDANTS.

2- CALCULER LA REALISATION.

3- VERIFIER LA REALISATION.

4- FIN.

Tapez le numero de votre choix suivi de ; :

3;

La serie correspondant a vorte realisation est:

$$-\,x0^{\langle 3\rangle}\,.\,x1 + x0\,.\,x1^{\langle 3\rangle} - x0\,.\,x1^{\langle 2\rangle} + x0\,.\,x1\,.\,x0^{\langle 2\rangle} + x0^{\langle 4\rangle} + x0$$

Tapez <c;> puis <return> pour continuer:

c;

RESUME

Dans ce travail, nous présentons d'abord une étude fondamentale des séries formelles et des opérations de base sur celles-ci. Nous nous sommes efforcés d'implémenter systématiquement tous les outils de manipulation des séries formelles non commutatives.

Ensuite, nous abordons un problème technique et très difficile de l'Automatique, à savoir le problème de la réalisation des systèmes dynamiques non linéaires. Nous apportons alors des éléments de démonstration d'une conjecture selon laquelles, pour tout polynôme non commutatif, on peut trouver la réalisation locale et minimale complètement polynomiale. Ces éléments consistent en une implémentation d'algorithmes permettant de construire cette réalisation.

Dans cet algorithme nous utilisons les mots de Lyndon pour la construction des coordonnées locales sur lesquelles seraient construites la fonction d'observation et les composantes des champs de vecteurs.

Notre algorithme est implémenté sur les stations de travail SUN avec le système de Calcul Formel MACSYMA.

Ce travail montre aussi l'intérêt de l'utilisation d'un système de Calcul Formel. Nous souhaitons intégrer ce logiciel dans une bibliothèque de programmes qui traitent les séries formelles non commutatives et les systèmes dynamiques non linéaires ce qui lui assurera une plus grande diffusion.