Calcul formel et parallélisme : bases de Gröbner booléennes, méthodes de calcul. Aapplications, parallélisation

Pascale Sénéchaud

THESE

Présentée par Pascale SENECHAUD

pour obtenir le titre de **DOCTEUR**

de l'**INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE**
(arrêté ministériel du 23 novembre 1988)

(Spécialité: **Mathématiques Appliquées**)

Calcul Formel et Parallélisme

BASES DE GRÖBNER BOOLEENNES METHODES DE CALCUL . APPLICATIONS PARALLELISATION

Date de soutenance : 15 Février 1990

Composition du jury :

Président :	D. Lazard
Rapporteurs :	J.H Davenport
	Y. Robert
Examinateurs :	J. Della Dora
	L. Trilling

Thèse préparée au sein du laboratoire LMC / Imag, U.A. au CNRS n°397.

INSTITUT NATIONAL POLYTECHNIQUE
G R E N O B L E

46 avenue F. Viallet - 38031 GRENOBLE Cedex -

Tél : 76.57.45.00

ANNEE UNIVERSITAIRE 1989

Président de l'Institut
Monsieur Georges LESPINARD

PROFESSEURS DES UNIVERSITES

ENSERG	BARIBAUD	Michel
ENSIEG	BARRAUD	Alain
ENSPG	BAUDELET	Bernard
INPG	BEAUFILS	Jean-Pierre
ENSERG	BLIMAN	Samuel
ENSHMG	BOIS	Philippe
ENSEEG	BONNETAIN	Lucien
ENSPG	BONNET	Guy
ENSIEG	BRISSONNEAU	Pierre
IUFA	BRUNET	Yves
ENSHMG	CAILLERI E	Denis
ENSPG	CAVAIGNAC	Jean-François
ENSPG	CHARTIER	Germain
ENSERG	CHENEVIER	Pierre
UFR PGP	CHERADAME	Hervé
ENSIEG	CHERUY	Arlette
ENSERG	CHOVET	Alain
ENSERG	COHEN	Joseph
ENSEEG	COLINET	Catherine
ENSIEG	CORNUT	Bruno
ENSIEG	COULOMB	Jean-Louis
ENSERG	COUMES	André
ENSIMAG	CROWLEY	James
ENSHMG	DARVE	Félix
ENSIMAG	DELLA DORA	Jean-François
ENSERG	DEPEY	Maurice
ENSPG	DEPORTES	Jacques
ENSEEG	DEROO	Daniel
ENSEEG	DESRE	Pierre
ENSERG	DOLMAZON	Jean-Marc
ENSEEG	DURAND	Francis
ENSPG	DURAND	Jean-Louis
ENSHMG	FAUTRELLE	Yves
ENSIEG	FOGGIA	Albert
ENSIMAG	FONLUPT	Jean
ENSIEG	FOULARD	Claude
UFR PGP	GANDINI	Alessandro
ENSPG	GAUBERT	Claude
ENSERG	GENTIL	Pierre
ENSIEG	GENTIL	Sylviane
IUFA	GREVEN	Hélène
ENSIEG	GUEGUEN	Claude
ENSERG	GUERIN	Bernard
ENSEEG	GUYOT	Pierre
ENSPG	JOST	Rémy
ENSPG	JOUBERT	Jean-Claud
ENSIEG	JOURDAIN	Geneviève
ENSIEG	LACOUME	Jean-Louis
ENSIEG	LADET	Pierre
ENSHMG	LESIEUR	Marcel
ENSHMG	LESPINARD	Georges
ENSPG	LONGEQUEUE	Jean-Pierre
ENSHMG	LORET	Benjamin
ENSEEG	LOUCHET	François
ENSEEG	LUCAZEAU	Guy
ENSIEG	MASSE	Philippe
ENSIEG	MASSELOT	Christian
ENSIMAG	MAZARE	Guy
ENSIMAG	MOHR	Roger
ENSHMG	MOREAU	René
ENSIEG	MORET	Roger
ENSIMAG	MOSSIERE	Jacques
ENSHMG	OBLED	Charles
ENSEEG	OZIL	Patrick
ENSEEG	PAULEAU	Yves
ENSIEG	PERRET	Robert
ENSHMG	PIAU	Jean-Mich
ENSERG	PIC	Etienne
ENSIMAG	PLATEAU	Brigitte
ENSERG	POUPOT	Christian
ENSEEG	RAMEAU	Jean-Jacq
ENSPG	REINISCH	Raymond
UFR PGP	RENAUD	Maurice
UFR PGP	ROBERT	André
ENSIMAG	ROBERT	François
ENSIEG	SABONNADIERE	Jean-Clau
ENSIMAG	SAUCIER	Gabriele
ENSPG	SCHLENKER	Claire
ENSPG	SCHLENKER	Michel
ENSERG	SERMET	Pierre
UFR PGP	SILVY	Jacques
ENSHMG	SIRIEYS	Pierre
ENSEEG	SOHM	Jean-Cla
ENSIMAG	SOLER	Jean-Lou
ENSEEG	SOUQUET	Jean-loui
ENSHMG	TROMPETTE	Philippe
ENSPG	VINCENT	Henri
ENSERG	ZADWORNY	François

SITUATION PARTICULIERE

PROFESSEURS D'UNIVERSITE

DETACHEMENT

ENSIMAG	LATOMBE	J.Claude	Détachement.....	21/10/1989
ENSHMG	PIERRARD	J.Marie	Détachement.....	30/04/1989
ENSIMAG	VEILLON	Gérard	Détachement.....	30/09/1990
ENSIMAG	VERJUS	J.Pierre	Détachement.....	30/09/1989
ENSPG	BLOCH	Daniel	Récteur à c/.....	21/12/1988

SURNOMBRE

INPG	CHIAVERINA	Jean	30/09/1989
ENSHMG	BOUVARD	Maurice...	30/09/1991
ENSEEG	PARIAUD	J.Charles	30/09/1991

PERSONNES AYANT OBTENU LE DIPLOME

d'habilitation à diriger des recherches

BECKER	M.	DANES	F.	GHIBAUDO	G.	MULLER	J.
BINDER	Z.	DEROO	D.	HAMAR	S.	NGUYEN TRONG	B.
CHASSERY	J.M.	DIARD	J.P.	HAMAR	R.	NIEZ	J.J.
CHOLLET	J.P.	DION	J.M.	LACHENAL	D.	PASTUREL	A.
COEY	J.	DUGARD	L.	LADET	P.	PLA	F.
COLINET	C.	DURAND	M.	LATOMBE	C.	ROGNON	J.P.
COMMAULT	..C.	DURAND	R.	LE HUY	H.	ROUGER	J.
CORNUEJOLS	G.	GALERIE	A.	LE GORREC	B.	TCHUENTE	M.
COULOMB	J.L.	GAUTHIER	J.P.	MADAR	R.	VINCENT	H.
COURNIL	M.	GENTIL	S.	MEUNIER	G.	YAVARI	A.R.
DALARD	F.						

CHERCHEURS DU C.N.R.S.

DIRECTEURS DE RECHERCHE CLASSE 0

LANDAU	Ioan
NAYROLLES	Bernard

DIRECTEURS DE RECHERCHE 1ère CLASSE

ANSARA	Ibrahim	KRAKOWIAK	Sacha
CARRE	René	LEPROVOST	Christian
FRUCHARD	Robert	VACHAUD	Georges
HOPFINGER	Emile	VERJUS	Jean-Pierre
JORRAND	Philippe		

DIRECTEURS DE RECHERCHE 2ème CLASSE

ALEMANY	Antoine	JOUD	Jean-Charles
ALLIBERT	Colette	KAMARINOS	Georges
ALLIBERT	Michel	KLEITZ	Michel
ARMAND	Michel	KOFMAN	Walter
AUDIER	Marc	LEJEUNE	Gérard
BERNARD	Claude	MADAR	Roland
BINDER	Gilbert	MERMET	Jean
BONNET	Roland	MICHEL	Jean-Marie
BORNARD	Guy	MEUNIER	Jacques
CAILLER	Marcel	PEUZIN	Jean-Claude
CALMET	Jacques	PIAU	Monique
CHATILLON	Christian	RENOUARD	Dominique
CLERMONT	Jean-Robert	SENATEUR	Jean-Pierre
COURTOIS	Bernard	SIFAKIS	Joseph
DAVID	René	SIMON	Jean-Paul
DION	Jean-Michel	SUERY	Michel
DRIOLE	Jean	TEODOSIU	Christian
DURAND	Robert	VAUCLIN	Michel
ESCUDIER	Pierre	VENNEREAU	Pierre

PERSONNALITES AGREEES A TITRE PERMANENT A DIRIGER DES TRAVAUX DE RECHERCHE

(DECISION DU CONSEIL SCIENTIFIQUE)

ENSEEG	HAMMOU	Abdelkader
	MARTIN-GARIN	Régina
	SARRAZIN	Pierre
	SIMON	Jean-Paul
ENSERG	BOREL	Joseph
ENSIEG	DESCHIZEAUX	Pierre
	GLANGEAUD	François
	PERARD	Jacques
	REINISCH	Raymond
ENSHMG	ROWE	Alain
ENSIMAG	COURTIN	Jacques
C.E.N.G	CADET	Jean
	COEURE	Philippe
	DELHAYE	Jean-Marc
	DUPUY	Michel
	JOUVE	Hubert
	NICOLAU	Yvan
	NIFENECKER	Hervé
	PERROUD	Paul
	PEUZIN	Jean-Claude
	TAIEB	Maurice
	VINCENDON	Marc

Laboratoire extérieurs :

C.N.E.T.	DEVINE	Rodericq
	GERBER	Roland
	MERCKEL	Gérard
	PAULEAU	Yves

XXXXXXXXXXXXXXXX

UNIVERSITE JOSEPH FOURIER
SCIENCES. TECHNOLOGIE. MEDECINE
GRENOBLE 1

Président de l'Université :

M. NEMOZ Alain

ANNEE UNIVERSITAIRE 1988 - 1989

MEMBRES DU CORPS ENSEIGNANT DE SCIENCES ET DE GEOGRAPHIE

PROFESSEURS DE 1ERE CLASSE

ADIBA Michel	Informatique
ANTOINE Pierre	Géologie I.R.I.G.M.
ARNAUD Paul	Chimie Organique
ARVIEU Robert	Physique Nucléaire I.S.N.
AURIAULT Jean Louis	Mécanique
AYANT Yves	Physique Approfondie
BARJON Robert	Physique Nucléaire I.S.N.
BARNOUD Fernand	Biochimie Macromoléculaire Végétale
BARRA Jean René	Statistiques - Mathématiques Appliquées
BECKER Pierre	Physique
BEGUIN Claude	Chimie Organique
BELORISKY Elie	Physique
BENZAKEN Claude	Mathématiques Pures
BERARD Pierre	Mathématiques Pures
BERNARD Alain	Mathématiques Pures
BERTRANDIAS Françoise	Mathématiques Pures
BERTRANDIAS Jean Paul	Mathématiques Pures
BILLET Jean	Géographie
BOEHLER Jean Paul	Mécanique
BRAVARD Yves	Géographie
CARLIER Georges	Biologie Végétale
CASTAING Bernard	Physique
CAUQUIS Georges	Chimie Organique
CHARDON Michel	Géographie
CHIBON Pierre	Biologie Animale
COHEN ADDAD Jean Pierre	Physique
COLIN DE VERDIERE Yves	Mathématiques Pures
CYROT Michel	Physique du Solide
DEBELMAS Jacques	Géologie Générale
DEGRANGE Charles	Zoologie
DEMAILLY Jean Pierre	Mathématiques Pures
DENEUVILLE Alain	Physique
DEPORTES Charles	Chimie Minérale
DOLIQUE Jean Michel	Physique des Plasmas
DOUCE Roland	Physiologie Végétale
DUCROS Pierre	Cristallographie
FINKE Gerde	Informatique
GAGNAIRE Didier	Chimie Physique
GAUTRON René	Chimie
GENIES Eugène	Chimie
GERMAIN Jean Pierre	Mécanique
GIDON Maurice	Géologie
GUITTON Jacques	Chimie
HICTER Pierre	Chimie
IDELMAN Simon	Physiologie Animale
JANIN Bernard	Géographie
JOLY Jean René	Mathématiques Pures

KAHANE André, détaché	Physique
KAHANE Josette	Physique
KRAKOWIAK Sacha	Mathématiques Appliquées
LAJZEROWICZ Jeanine	Physique
LAJZEROWICZ Joseph	Physique
LAURENT Pierre Jean	Mathématiques Appliquées
LEBRETON Alain	Mathématiques Appliquées
DE LEIRIS Joël	Biologie
LHOMME Jean	Chimie
LLIBOUTRY Louis	Géophysique
LOISEAUX Jean Marie	Sciences Nucléaires I.S.N.
LONGEQUEUE Nicole	Physique
LUNA Domingo	Mathématiques Pures
MACHE Régis	Physiologie Végétale
MASCLE Georges	Géologie
MAYNARD Roger	Physique du Solide
OMONT Alain	Astrophysique
OZENDA Paul	Botanique (Biologie Végétale)
PANNETIER Jean	Chimie
PAYAN Jean Jacques	Mathématiques Pures
PEBAY PEYROULA Jean Claude	Physique
PERRIER Guy	Géophysique
PIERRE Jean Louis	Chimie Organique
RENARD Michel	Thermodynamique
RIEDTMANN Christine	Mathématiques
RINAUDO Marguerite	Chimie C.E.R.M.A.V.
ROSSI André	Biologie
SAXOD Raymond	Biologie Animale
SENGEL Philippe	Biologie Animale
SERGERAERT Francis	Mathématiques Pures
SOUCHIER Bernard	Biologie
SOUTIF Michel	Physique
STUTZ Pierre	Mécanique
TRILLING Laurent	Mathématiques Appliquées
VAN CUTSEM Bernard	Mathématiques Appliquées
VIALON Pierre	Géologie

PROFESSEURS DE 2EME CLASSE

ARMAND Gilbert	Géographie
ATTANE Pierre	Mécanique
BARET Paul	Chimie
BERTIN José	Mathématiques
BLANCHI Jean Pierre	S.T.A.P.S.
BLOCK Marc	Biologie
BLUM Jacques	Mathématiques Appliquées
BOITET Christian	Mathématiques Appliquées
BORNAREL Jean	Physique
BORRIONE Dominique	Automatique informatique
BOUVET Jean	Biologie
BROSSARD Jean	Mathématiques
BRUANDET Jean François	Physique
BRUGAL Gérard	Biologie
BRUN Gilbert	Biologie
CERFF Rudiger	Biologie
CHIARAMELLA Yves	Mathématiques Appliquées
CHOLLET Jean Pierre	Mécanique
COLOMBEAU Jean François	Mathématiques (ENSL)
COURT Jean	Chimie
CUNIN Pierre Yves	Informatique

DAVID Jean	Géographie
DHOUAILLY Danielle	Biologie
DUFRESNOY Alain	Mathématiques Pures
GASPARD François	Physique
GIDON Maurice	Géologie
GIGNOUX Claude	Sciences Nucléaires
GILLARD Roland	Mathématiques Pures
GIORNI Alain	Sciences Nucléaires
GONZALEZ SPRINBERG Gérardo	Mathématiques Pures
GUIGO Maryse	Géographie
GUMUCHIAN Hervé	Géographie
HACQUES Gérard	Mathématiques Appliquées
HERBIN Jacky	Géographie
HERAULT Jeanny	Physique
HERINO Roland	Physique
JARDON Pierre	Chimie
KERCKHOVE Claude	Géologie
MANDARON Paul	Biologie
MARTINEZ Francis	Mathématiques Appliquées
MOREL Alain	Géographie
NEMOZ Alain	Thermodynamique C.N.R.S. C.R.T.B.T.
NGUYEN HUY Xuong	Informatique
OUDET Bruno	Mathématiques Appliquées
PAUTOU Guy	Biologie
PECHER Arnaud	Géologie
PELMONT Jean	Biochimie
PELLETIER Guy	Astrophysique
PERRIN Claude	Sciences Nucléaires I.S.N.
PIBOULE Michel	Géologie
RAYNAUD Hervé	Mathématiques Appliquées
REGNARD Jean René	Physique
RICHARD Jean Marc	Physique
RIEDTMANN Christine	Mathématiques Pures
ROBERT Danielle	Chimie
ROBERT Gilles	Mathématiques Pures
ROBERT Jean Bernard	Chimie Physique
SARROT REYNAULD Jean	Géologie
SAYETAT Françoise	Physique
SERVE Denis	Chimie
STOECKEL Frédéric	Physique
SCHOLL Pierre Claude	Mathématiques Appliquées
SUBRA Robert	Chimie
VALLADE Marcel	Physique
VIDAL Michel	Chimie Organique
VINCENT Gilbert	Physique
VIVIAN Robert	Géographie
VOTTERO Philippe	Chimie

MEMBRES DU CORPS ENSEIGNANT DE L'I.U.T. 1

PROFESSEURS DE 1ERE CLASSE

BUISSON Roger	Physique I.U.T. 1
CHEHIKIAN Alain	E.E.A. I.U.T.1
DODU Jacques	Mécanique Appliquée I.U.T. 1
NEGRE Robert	Génie Civil I.U.T. 1
NOUGARET Marcel	Automatique I.U.T. 1
PERARD Jacques	E.E.A. I.U.T. 1

PROFESSEURS DE 2EME CLASSE

BEE Marc	Physique I.U.T. 1
BOUTHINON Michel	E.E.A. I.U.T. 1
CHAMBON René	Génie Mécanique I.U.T. 1.
CHENAVAS Jean	Physique I.U.T. 1
CHILO Jean	Physique I.U.T. 1
CHOUTEAU Gérard	Physique I.U.T. 1
CONTE René	Physique I.U.T. 1
FOSTER Panayotis	Chimie I.U.T. 1
GOSSE Jean Pierre	E.E.A. I.U.T. 1
GROS Yves	Physique I.U.T. 1
HAMAR Roger	Chimie I.U.T. 1
KUHN Gérard, détaché	Physique I.U.T. 1
LEVIEL Jean Louis	Physique I.U.T. 1
MAZUER Jean	Physique I.U.T. 1
MICHOULIER Jean	Physique I.U.T. 1
MONLLOR Christian	E.E.A. I.U.T. 1
PERRAUD Robert	Chimie I.U.T. 1
PIERRE Gérard	Chimie I.U.T. 1
TERRIEZ Jean Michel	Génie Mécanique I.U.T. 1
TOUZAIN Philippe	Chimie I.U.T. 1
TURGEMAN Sylvain	Génie civil
VINCENDON Marc	Chimie I.U.T. 1
ZIGONE Michel	Physique I.U.T. 1

Je voudrais exprimer toute ma reconnaissance à Jean DELLA DORA, professeur à l'ENSIMAG, qui a dirigé cette thèse, pour ses conseils, ses encouragements tout au long de mon travail et pour la confiance qu'il m'a témoignée.

Je voudrais également remercier les autres membres du jury :

- le président :

Daniel LAZARD, professeur au LITP, que je remercie d'avoir accepté de présider cette thèse, d'avoir relu attentivement ce manuscrit et d'avoir fait des remarques constructives pour l'améliorer.

- les rapporteurs :

James DAVENPORT, professeur à l'université de Bath, grâce auquel les bases de Gröbner booléennes calculées ici sont vraiment des bases de Gröbner, et auquel j'adresse mes plus vifs remerciements,

Yves ROBERT, professeur à L'ENSL, que je remercie d'avoir accepté de juger ce travail. J'apprécie l'intérêt qu'il y a porté et l'effort qu'il a fait en lisant la première partie de cette thèse dont le sujet ne lui était pas familier.

- l'examinateur "candide", Laurent TRILLING, professeur à l'ENSIMAG, que je remercie d'avoir participé à ce jury en dépit de toutes les modifications d'horaires et de dates de soutenance mais également d'être toujours aussi souriant, avec ou sans baguette.....

Merci à Claire Discrescenzo pour ses relectures attentionnées ; à Joelle Prévost toujours prête à aider un utilisateur perdu devant sa console ; à Jean Michel Muller pour son aide et ses épiques randonnées ; à Evelyne Tournier pour son soutien ; à Jean Louis Roch sans le travail de qui, une partie du mien n'aurait pas été réalisée.

Combien l'amitié et le soutien de Gilles Villard m'ont été précieux tout au long de cette thèse. Combien il m'a été agréable de travailler avec Françoise Roch.

Je n'oublie pas les autres membres de l'équipe :

Abdel Barkatou, Guoting Chen, Mustafa Daoudi, Afonso Ferreira, Michel Gastaldo, Yvan Herreros, Jean Laurent Philippe, Shengrui Wang qui font souvent preuve d'un humour ravageur et acide à mes chastes (?) oreilles et qui savent mettre l'ambiance nécessaire pour conserver le moral de la troupe.

Merci à toute l'équipe de Calcul Formel et Parallélisme pour ses gâteaux et mes kilos en trop.

Sans parler de certains membres de l'équipe des EDP comme Christine Bernier, sans laquelle la tour IRMA manquerait de contestataire et grâce à laquelle ma connaissance sur la Russie s'est un peu enrichie.

Les encouragements et l'aide de Marc Sancandi, sa patience, sa constance mais aussi son humour m'ont permis de mener à bien ce travail. Douce et utile m'a été et m'est encore sa présence.

Je remercie enfin tous les membres du service de la scolarité de l'INPG et ceux du service de reprographie de l'IMAG pour la réalisation matérielle de cette thèse.

à maman

à la mémoire de papa

PLAN

PREMIERE PARTIE

PRESENTATION THEORIQUE DES BASES DE GRÖBNER

Introduction

Chapitre I Préliminaires

Chapitre II La notion de bases de Gröbner. Un algorithme de calcul

Chapitre III Une application importante : la résolution de systèmes d'équations non linéaires. Utilisation de différents systèmes de calcul formel

DEUXIEME PARTIE

APPLICATION DES BASES DE GRÖBNER LORSQUE $K = \mathbb{Z} / 2\mathbb{Z}$

Introduction

Chapitre I Préliminaires

Chapitre II Application à la logique

Chapitre III Application à la preuve formelle de circuits

Conclusion

TROISIEME PARTIE
PROGRAMMATION

Introduction

Chapitre I Structures de base de l'algorithme séquentiel

Chapitre II Parallélisation

Chapitre III Résultats expérimentaux de la parallélisation

Conclusion

Bibliographie

INTRODUCTION :

L'apparition des systèmes de Calcul Formel permettant d'effectuer sur ordinateur des calculs algébriques sur des polynômes a suscité un regain d'interêt pour des méthodes trop longues pour être menées à bien à la main. C'est le cas pour l'outil que représente les bases de Gröbner et qui nous intéresse dans cette thèse.

Ce travail est composé de trois parties :

••• La première partie consiste en une présentation théorique des bases de Gröbner. Cette présentation se veut accessible à des non-spécialistes et permet de se familiariser avec le mécanisme de l'algorithme de calcul des bases de Gröbner. Les notions de base essentielles à cette approche sont présentées. L'importance du choix de l'ordre est précisée et les critères utilisés par Buchberger [Buchberger79] sont donnés et une étude bibliographique des résultats de complexité est faite.

Comme exemple d'utilisation on présente la résolution de systèmes d'équations non linéaires dans le cas où l'ensemble des racines cherchées est fini.

Afin de mettre en évidence les propriétés des bases de Gröbner on utilise les systèmes de calcul formel Reduce, Maple et Macsyma. On donne à cette occasion les performances obtenues sur ces systèmes pour quelques exemples.

••• La deuxième partie présente des applications des bases de Gröbner lorsque le corps de base est $\mathbb{F}_2$. On se limite aux cas simples du calcul propositionnel et de la preuve de circuits combinatoires.

Le premier chapitre de cette partie est constitué de préliminaires destinés à mettre en place le formalisme sur $\mathbb{F}_2$ utilisé par la suite.
Le deuxième chapitre établit le lien existant entre les problèmes de logique et les bases de Gröbner et sert de présentation à la transformation de Stone[Stone36] et aux bases de Gröbner dans $\mathbb{F}_2[x_1,...x_n]$.
Le troisième chapitre correspond à l'utilisation de bases de Gröbner en preuve formelle de circuits. Pour des raisons de simplification on se limite aux cas des circuits combinatoires.On utilise ici les bases de Gröbner sur des circuits combinatoires simples puis on hiérarchise les circuits pour éviter l'introduction de beaucoup de variables intermédiaires. On présente alors un algorithme de preuve de circuits.
La comparaison des méthodes existantes et de la méthodes proposée est difficile car il n'existe pas de résultats satisfaisants sur la complexité des algorithmes de calcul de bases de Gröbner, que ce soit dans le cadre général ou dans le cadre des bases de Gröbner booléennes [Stillmann89]. La pratique montre en effet que l'on se situe très loin de la complexité théorique. De plus l'algorithme des bases de Gröbner fournit toujours une solution ce qui n'est pas forcément le cas des méthodes de compilation canonique.

••• La dernière partie concerne l'implantation d'algorithmes séquentiels et parallèles.

Le premier chapitre de cette partie est consacré à la mise en place de la notion de base de Gröbner booléenne et des structures utilisées dans les algorithmes. On montre comment à partir de polynômes booléens on construit une famille booléenne qui compose en partie une base de Gröbner de $\mathbb{F}_2[x_1,...x_n]$.
Il nous est paru souhaitable de séparer les problèmes qui se posent lors du calcul d'une base de Gröbner. Dans une première étape on s'affranchit des contraintes liées au grossissement des coefficients et à la multiplicité des monômes en travaillant avec des

polynômes booléens. On signale que les algorithmes sont généralisables au cas des polynômes à coefficients rationnels.
On construit un algorithme séquentiel adapté au cas des polynômes booléens. Hormis une étude des structures de bases, la comparaison de diverses méthodes de réduction met en évidence les problèmes de temps de calcul et de taille mémoire.

Les deux derniers chapitres présentent deux méthodes de parallélisation :

- la première utilise un anneau de processeurs. Cette méthode se situe à un niveau de parallélisation assez bas : les tâches indépendantes sont exécutées en parallèle. On effectue une étude qualitative et quantitative de cette première méthode.

- la deuxième méthode se situe à un niveau de parallélisme différent : on calcule en parallèle des sous-bases de Gröbner et l'on regroupe les résultats. On utilise pour celà une topologie d'hypercube. On obtient en fin d'algorithme la base de Gröbner minimale calculée de différentes manières suivant la répartition des sous-bases intermédiaires.
Pour cet algorithme des résultats quantitatifs et qualitatifs sont donnés. On conclut par une comparaison de ces deux algorithmes.

Nous soulignons l'influence de la répartition des données sur le temps d'exécution. On présente une méthode originale de répartition des polynômes basée sur la recherche de chemins de longueurs données dans un graphe orienté. Cette répartition nous permet d'obtenir des résultats interprétables et de conclure sur les différents algorithmes.

PREMIERE PARTIE :

PRESENTATION THEORIQUE DES BASES DE GRÖBNER

PREMIERE PARTIE : PRESENTATION THEORIQUE DES BASES DE GRÖBNER

INTRODUCTION

Le but de cette partie est essentiellement de faire comprendre ce que sont les bases de Gröbner à des lecteurs non spécialistes de la question sans toutefois rester vague ou oublier la rigueur mathématique.

Dans un premier chapitre, seront présentés des algorithmes classiques tels que celui d'Euclide et celui de Gauss. Il n'est bien évidemment pas question ici de les approfondir, mais d'en comprendre le principe afin de construire, suivant un mécanisme analogue, des algorithmes qui nous conduiront à la notion de base de Gröbner.

Il existe plus d'une façon de présenter la notion de base de Gröbner. Représentation d'un idéal, régles de réécriture, généralisation de l'algorithme de Gauss, de celui d'Euclide ; les facettes sont multiples. On tachera de les aborder toutes, mais on en gardera une comme fil conducteur, dont nous traçons ici les grandes lignes.

L'algorithme d'Euclide permet de déterminer le PGCD de polynômes à une variable et fournit un test simple et efficace d'appartenance d'un polynôme à un idéal de $\mathbb{Q}[x]$. Il se généralise au cas des polynômes à plusieurs variables. On obtient alors deux algorithmes sans lesquels peu de choses seraient calculables lorsque l'on manipule des polynômes à plusieurs variables.
Le premier algorithme, dit division d'Hironaka (1964) introduit une division "cohérente" entre polynômes à plusieurs variables. Pour celà il procède à une généralisation de la notion essentielle de degré en utilisant une relation d'ordre total sur $\mathbb{N}^n$. Le choix de cette relation dépend de l'information désirée sur les polynômes traités.
Cependant, le résultat de cette division ne permet pas à lui seul de savoir simplement si un polynôme appartient ou non à un idéal donné. Le second algorithme dit de Buchberger (1970) introduit la notion de base de Gröbner d'un idéal et permet de répondre à cette question. A partir d'une famille de polynômes générateurs d'un idéal, il en détermine une autre mieux adaptée au problème. La famille obtenue, dite base de Gröbner ou base standard, depend en général de l'ordre choisi.
Ces deux algorithmes généralisent celui d'Euclide et utilisent de plus une construction dont Macaulay avait prouvé la validité dès 1927.

Les chapitres qui suivent sont consacrés à la description des algorithmes d'Euclide, d'Hironaka, de Buchberger et aussi à celle des propriétés essentielles des bases de Gröbner et de certaines de leurs applications.

CHAPITRE I

PRELIMINAIRES

Ce chapitre a pour objet de mettre en place les notions de base essentielles pour la bonne compréhension des chapitres qui suivent et de se familiariser avec le mécanisme général de l'algorithme de calcul de base de Gröbner par le biais d'algorithmes plus courants tels que ceux de Gauss et d'Euclide.

Ce chapitre est inspiré du cours de DEA enseigné par Mme Lejeune-Jalabert [Lejeune85].

I- NOTATIONS ET DEFINITIONS PRELIMINAIRES

Dans la suite on note :

- $\mathbb{N}$ l'ensemble des entiers naturels,
- K un corps,
- $K[x_1,....x_n]$ l'anneau des polynômes à n indéterminées $x_1,....x_n$ sur K. Un *polynôme* est une somme de monômes ; un *monôme* est un produit d'un élément de K et d'indéterminées.

Un monôme s'écrit :

$$X^\alpha = a_\alpha x_1^{\alpha_1} x_2^{\alpha_2} \ldots\ldots x_n^{\alpha_n} \quad \text{avec} \quad \alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}^n \text{ et } a_\alpha \in K$$

Si on considère p un polynôme de $\mathbb{R}[x,y]$, p s'écrit :

$$p = \sum_\alpha X^\alpha = \sum_{(\alpha_1, \alpha_2) \in I} a_{(\alpha_1, \alpha_2)} x^{\alpha_1} y^{\alpha_2} \qquad \text{avec I une partie de } \mathbb{N}^2.$$

On introduit également les objets suivants :

- Soit $\mathcal{J}$ un sous ensemble de $K[x_1,....x_n]$. $\mathcal{J}$ est un *idéal* de $K[x_1,....x_n]$ si
  - i) $\mathcal{J}$ est un sous groupe additif de $K[x_1,....x_n]$,
  - ii) $\mathcal{J}$ est absorbant pour la multiplication. C'est à dire si

    pout tout élement a de $\mathcal{J}$ on a :

    a p $\in$ $\mathcal{J}$, pour tout polynôme p$\in$ $K[x_1,....x_n]$.

En pratique un idéal $\mathcal{J}$ n'est pas défini de façon aussi hermétique : il s'agit la plus part du temps d'un idéal engendré par une famille F de polynômes. Comme de plus une famille génératrice d'un idéal de $K[x_1,....x_i]$ est *finie* (Hilbert 1888), $\mathcal{J}$ est l'ensemble des combinaisons linéaires à coefficients polynomiaux des éléments de F. On note cet idéal (F).

Soit F = $\{p_1,...p_m\}$ une famille de polynômes. On caractérise f un élément de (F) par :

$$f = \sum_{i=0}^{m} g_i p_i \quad \text{avec } g_i \in K[x_1....x_n]$$

Par exemple sur $\mathbb{R}[x,y]$ l'idéal $\mathcal{J}$ engendré par le polynôme (x-1) est l'ensemble de tous les polynômes qui s'annulent en 1.

On dit d'un idéal $\mathcal{J}$ qu'il est *principal* s'il existe un élément privilégié f dans $\mathcal{J}$ tel que $\mathcal{J}$ soit exactement l'idéal engendré par f.

. Soit $p \in K[x_1,....x_n]$, $p = \sum a_\alpha x^\alpha$.

On appelle *support de p* et l'on note Supp(p) le sous ensemble de $\mathbb{N}^n$ défini par :

$$Supp(p) = \{ \alpha \in \mathbb{N}^n / a_\alpha \neq 0\}$$

• Pour représenter et manipuler les polynômes on se fixe un *"ordre sur les monômes"*. On considère $\mathbb{N}^n$ muni d'un ordre total que l'on note < (ou >) qui vérifie les hypothèses (HO) suivantes :

i) $\forall \alpha \in \mathbb{N}^n$ et $\forall \beta \in \mathbb{N}^n \ \beta \neq 0 \ \ \alpha < \alpha + \beta$

ii) $\forall (\alpha_1, \alpha_2) \in \mathbb{N}^n \times \mathbb{N}^n, \forall \beta \in \mathbb{N}^n \quad \alpha_1 < \alpha_2 \Leftrightarrow \alpha_1 + \beta < \alpha_2 + \beta$

Par exemple :

dans $\mathbb{R}[x,y]$

$\forall \alpha = (\alpha_1, \alpha_2) \in \mathbb{N} \times \mathbb{N}, \forall \beta = (\beta_1, \beta_2) \in \mathbb{N} \times \mathbb{N}$

les ordres suivants sur $\mathbb{N} \times \mathbb{N}$ vérifient les conditions (HO) :

l'ordre *lexicographique* défini par :

$\alpha > \beta$ si $\quad \alpha_1 > \beta_1$ ou si $\alpha_1 = \beta_1$ et $\alpha_2 > \beta_2$

l'ordre *diagonal* (dit aussi ordre *lexicographique du degré*) défini par :

$\alpha > \beta$ si $\quad \alpha_1 + \alpha_2 > \beta_1 + \beta_2$ ou si $\alpha_1 + \alpha_2 = \beta_1 + \beta_2$ et $\alpha_2 > \beta_2$

ou encore l'ordre *lexicographique inverse du degré* défini par :

$\alpha > \beta$ si $\quad \alpha_1 + \alpha_2 > \beta_1 + \beta_2$ ou si $\alpha_1 + \alpha_2 = \beta_1 + \beta_2$ et $\alpha_2 < \beta_2$

Exemple :

Soit les couples (5, 2) (2, 2) et (1, 3)

Pour l'ordre lexicographique :

$$(5, 2) > (2, 2) > (1, 3)$$

et on écrit $P = x^5y^2 + x^2y^2 + xy^3$.

Pour l'ordre diagonal :

$$(5, 2) > (1, 3) > (2, 2)$$

et on écrit $P = x^5y^2 + xy^3 + x^2y^2$.

Pour l'ordre lexicographique inverse du degré :

$$(5, 2) > (2, 2) > (1,3)$$

et on écrit $P = x^5y^2 + x^2y^2 + xy^3$.

Ici les couples sont rangés de la même façon pour l'ordre lexicographique et l'ordre lexicographique inverse du degré, c'est rarement le cas :

soit par exemple les couples (4,3) et (2,8).

Pour l'ordre lexicographique $(4, 3) > (2, 8)$
(car $4 > 2$).

Pour l'ordre lexicographique inverse du degré $(4, 3) < (2, 8)$
(car $2 + 8 > 4 + 3$).

On associe à un polynôme p les différents objets suivants :

i) *Exp(p)*, l'exposant de p (qui correspond à la notion de degré pour les polynômes à une variable), qui est le plus grand élément de Supp(p) pour l'ordre choisi, c'est à dire :

$$\text{Exp}(p) = \alpha \text{ avec } \alpha \in \text{supp}(p) \text{ tel que :}$$

$$\forall \beta \in \text{supp}(p) \quad \alpha > \beta \quad \text{avec } \alpha \neq \beta$$

Pour les polynômes p à une variable on note deg(p).

ii) *Init(p)*, partie initiale de p, qui correspond à la notion de monôme de tête pour les polynômes à une variable, c'est à dire :

$$\text{Init}(p) = a_\alpha X^\alpha \text{ avec } \alpha = \text{Exp}(p) .$$

Exemple :

soit p le polynôme $p = 7x^4y^3 + x^3y^4 + xy + 1$

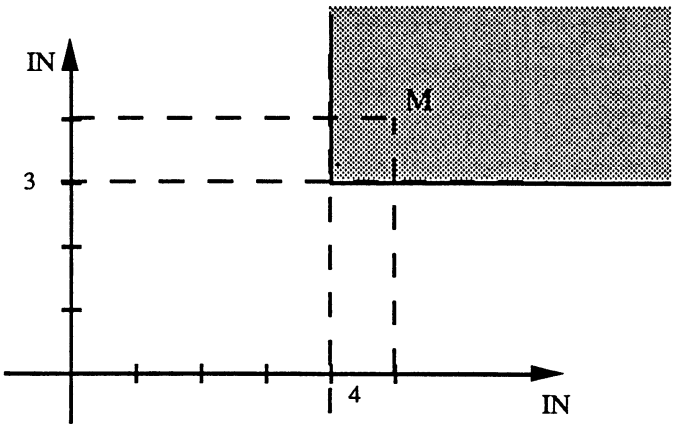
$\text{Supp}(p) = \{ (4, 3), (3, 4), (1,1), (0, 0) \}$ et

pour l'ordre lexicographique $\text{Exp}(p) = (4, 3)$ et $\text{Init}(p) = 7x^4y^3$

pour l'ordre diagonal $\text{Exp}(p) = (3,4)$ et $\text{Init}(p) = x^3y^4$

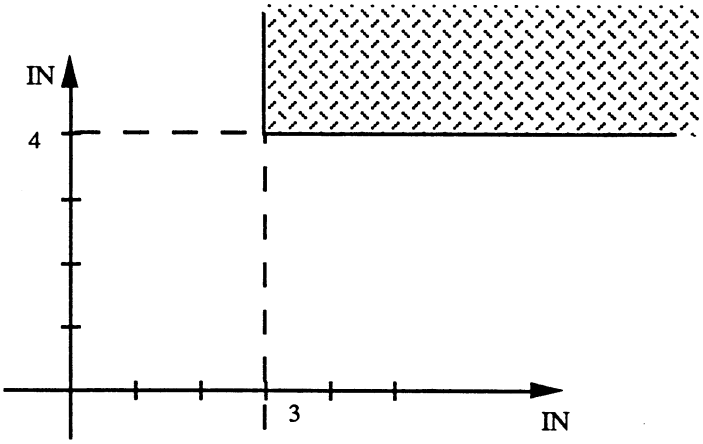
Il nous arrivera parfois de parler de monôme de tête pour les polynômes à plusieurs variables tels que les polynômes booléens.

On peut représenter les exposants de p sur les graphiques suivants:



C1 quadrant déterminé par (4,3)

M = (5,4) appartient à C1 , le monôme x^4y^3 divise x^5y^4



quadrant déterminé par (3, 4)

On appelle *quadrant déterminé par* α la partie de $\mathbb{N}^n$ qui contient tous les éléments de $\mathbb{N}^n$ qui s'écrivent $\alpha + a$ avec $a \in \mathbb{N}^n$.
Un quadrant déterminé par α ne contient pas forcément tous les éléments supérieurs à α.

Sur l'exemple, on peut remarquer que si $\underline{\alpha}$ est dans le quadrant déterminé par α alors X^{α} divise $X^{\underline{\alpha}}$.

Par la suite, dans les exemples si l'on ne précise pas l'ordre choisi, ce sera l'ordre lexicographique.

II- LES PARTIES STABLES DE $\mathbb{N}^n$. DEFINITIONS-PROPRIETES-EXEMPLES

Nous présentons ici cet objet essentiel dont les propriétes associées nous permettent de déduire l'existence et l'unicité d'une base de Gröbner.

II-1 Les parties stables

On appelle *partie stable de $\mathbb{N}^n$* un sous ensemble E de $\mathbb{N}^n$, non vide tel que :

$$\forall e \in E\ ,\ \forall\ \alpha \in \mathbb{N}^n\ ,\ \ e + \alpha \in E.$$

Exemple :

le quadrant déterminé par α est, pour tout α, une partie stable de $\mathbb{N}^n$.

On a le résultat suivant (que nous ne démontrerons pas) [Lejeune85]:

si E est une partie non vide de $\mathbb{N}^n$ alors E est une ***union finie de quadrants,*** *c'est à dire il existe un sous ensemble fini F de E tel que :*

$$E = \bigcup_{\alpha \in F} (\alpha + \mathbb{N}^n)$$

Si n =1 cela indique que toute partie de $\mathbb{N}$ admet un plus petit élément.

On dit alors que F est *une frontière* de E.

Il est clair qu'une partie stable peut avoir plusieurs frontières. En revanche il n'en existe qu'une de cardinal minimum.

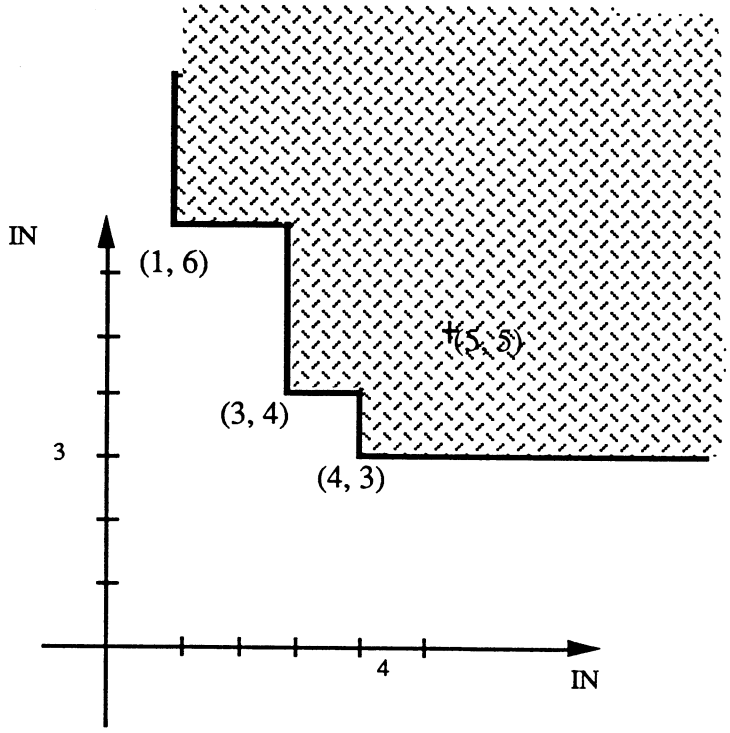
On appelle cette frontière privilégiée de E *l'escalier* de F.

Par exemple :

considérons la partie stable E de $\mathbb{N}^2$, union des 5 quadrants déterminés respectivement par :

$$\alpha = (3, 4);\ \beta = (4, 3),\ \gamma = (1, 6),\ \delta = (5, 5)$$

$F = \{\alpha, \beta, \gamma, \delta\}$ est une frontière de E et $\underline{F} = \{\alpha, \beta, \gamma\}$ en est l'escalier. On peut représenter E par :



II-2 Partition canonique de $\mathbb{N}^n$ associée à une famille finie de polynômes

Soit $F = (p_1,..,p_m)$ une famille de polynômes. A chaque polynôme p_i on peut associer, $Exp(p_i)$ (avec un ordre fixé). On a alors un ensemble d'éléments de $\mathbb{N}^n$ qui nous fournit une partition de $\mathbb{N}^n$ de la façon suivante :

à $Exp(p_1)$ on associe son quadrant c'est à dire la partie stable $\Delta_1 = Exp(p_1) + \mathbb{N}^n$,

à $Exp(p_2)$ on associe $\Delta_2 = (Exp(p_2) + \mathbb{N}^n) \setminus \Delta_1$,

à $Exp(p_3)$ on associe $\Delta_3 = (Exp(p_3) + \mathbb{N}^n) \setminus (\Delta_1 \cup \Delta_2)$,

.....

.....

....

à $Exp(p_m)$ on associe $\Delta_m = (Exp(p_m) + \mathbb{N}^n) \setminus (\cup \Delta_i)$ pour i dans $\{1,...m-1\}$.

A F on associe la partition Δ_1 .... Δ_m et $\underline{\Delta} = \mathbb{N}^n \setminus (\cup \Delta_i)$. Cette partition est dite *partition canonique* associée à F.

Par exemple :

soient les polynômes (avec l'ordre lexicographique) :

$p_1 = 4x^4y^3 + 3x^2y^2 + x^2 + 1$;

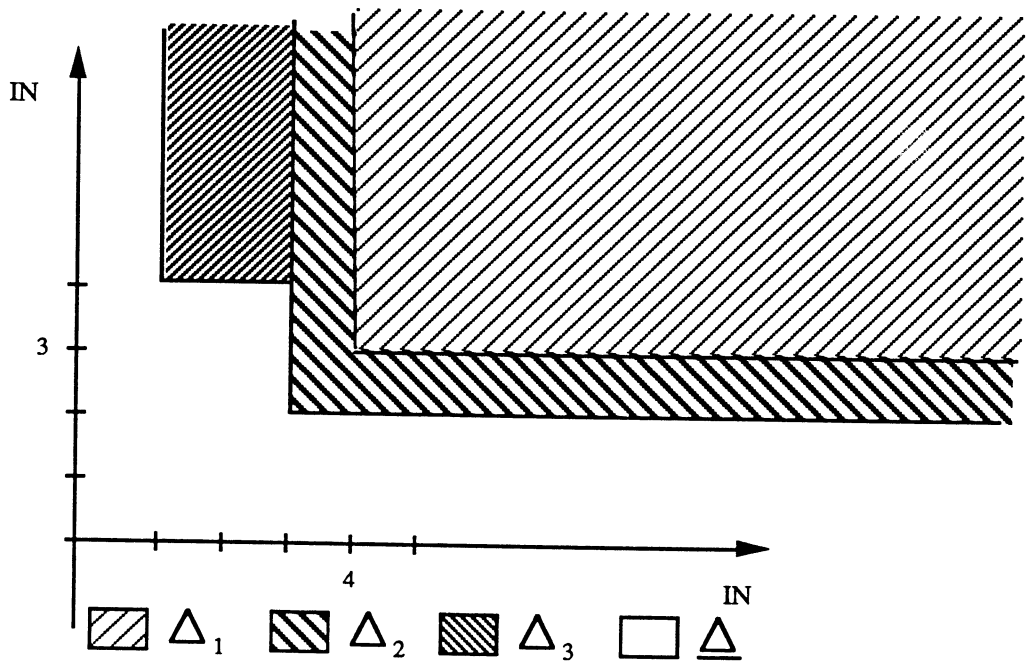
$p_2 = x^3y^2 + x^2y + xy + 9y + 8$;

$p_3 = 6xy^4 + xy + 9x$;

On a :

$$Exp(p_1) = x^4y^3$$
$$Exp(p_2) = x^3y^2$$
$$Exp(p_3) = xy^4$$

On peut alors représenter la partition canonique associée à la famille (p_1, p_2, p_3)



Remarques :

- la partition canonique associée à une famille dépend de l'ordre choisi sur les monômes,

- elle dépend également de l'indexage sur les polynômes. Les Δ_i sont différents, mais leur union et $\underline{\Delta}$ restent les mêmes.

Nous allons maintenant nous intéresser aux polynômes à une variable. Plus particulièrement, nous allons voir ce que deviennent les notions d'idéal et de division pour ces polynômes. Nous parlerons également de la technique utilisée dans l'algorithme de Gauss.

Puis nous généraliserons au cas des polynômes à plusieurs variables à l'aide des outils précédemment introduits.

III- LE CAS À UNE VARIABLE

III-1 Les ideaux de K[x]

Nous avons le résultat fondamental suivant :

Tout idéal de K[x] est principal .

C'est à dire que l'on peut décrire un idéal engendré par une famille de polynômes par un seul polynôme, que l'on appelle PGCD des polynômes de la famille.

Cet élément particulier nous fournit une caractérisation d'un idéal qui permet de tester de façon simple si un polynôme p appartient à cet idéal : il suffit de regarder si p est multiple de cet élément.

On peut remarquer que le PGCD d'un ensemble de polynômes est défini à un élément de K près.

L'application la plus directe de ce résultat est la suivante :

Si p_1 et p_2 sont deux polynômes et si l'on cherche l'ensemble R des racines communes à p_1 et p_2, on peut considérer l'idéal $\mathfrak{I}$ engendré par p_1 et p_2 et en particulier le polynôme privilégié g qui en est le générateur.

g qui est combinaison linéaire de p_1 et p_2 et générateur de $\mathfrak{I}$ s'annule exactement sur R et donc R est l'ensemble des racines de g.

De manière plus générale, trouver les racines communes d'un ensemble de polynômes à une variable est équivalent à trouver les racines d'un seul polynôme.

Le principal problème est alors de déterminer ce polynôme. Pour celà on utilise le fait que K[x] est un anneau euclidien c'est à dire que :

$$\forall\, f \in K[x]\,,\ \forall\, g \in K[x] \quad ,\ \exists\, (q, r) \in K[x] \times K[x] \quad \text{tels que :}$$

$$f = gq + r$$

Si de plus on impose soit $r = 0$, soit $\deg(r) < d$ (où d est le degré de g) on montre alors que l'on a unicité de cette décomposition.

A partir de cette division on écrit l'algorithme qui nous fournit le PGCD de plusieurs polynômes (Euclide III^{ième} siècle Av.JC).

III-2 L'algorithme d'Euclide

Soit un polynôme de K[x] et deg(p) son degré. L'algorithme d'Euclide, que l'on représente par la fonction PGCD, s'écrit :

PGCD(p_1, p_2) ==

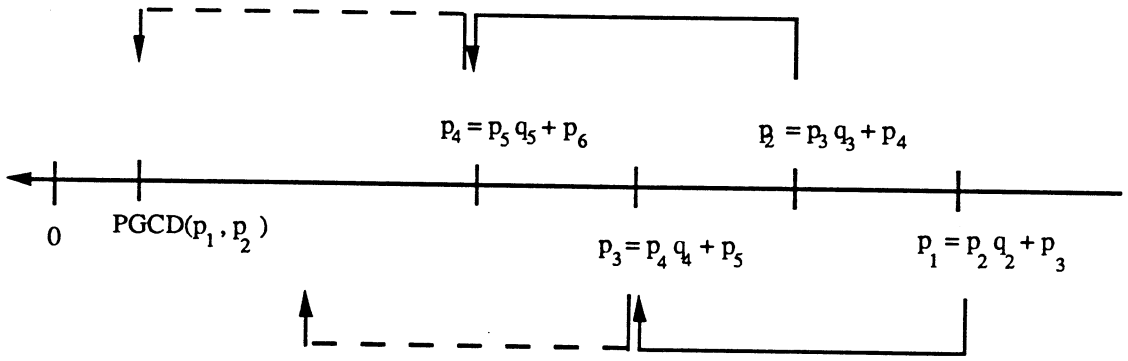
ENTREE : p_1 et p_2 deux polynômes de K[x],
tels que $\deg(p_1) \geq \deg(p_1)$
SORTIE : le PGCD de p_1 et de p_2.
{ variables intermédiaires }
q, r : deux polynômes ;
{ Initialisation }
tant que $p_2 \neq 0$
$p_1 := p_2\, q + r$ avec $r = 0$ ou $\deg(r) < \deg(p_2)$;
$p_1 := p_2$;
$p_2 := r$;

Sortir(p_1).

Fin de PGCD.

La boucle principale de l'algorithme ("tant que $p_2 \neq 0$") s'arrête ; sinon on construirait une suite infinie décroissante de degrés.

On peut schématiser l'algorithme par :



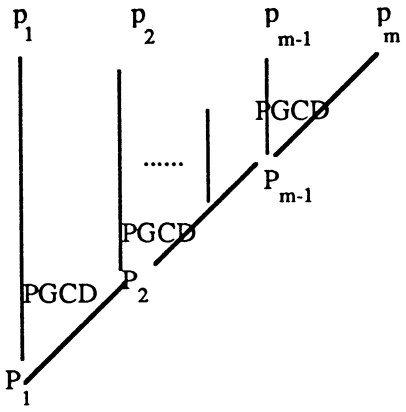
On le généralise pour construire récursivement le PGCD de toute une famille de polynômes :

$PGCD_F(F)$ ==

ENTREE : $F = \{p_1,...p_m\}$ une famille de polynômes de K[x] (m>2),
telle que $deg(p_1) \geq deg(p_2) \geq ....\geq deg(p_m)$.
SORTIE : le PGCD de p_1, p_2, ...p_m.
{ variables intermédiaires }
P_{m-1} : un polynôme ;
{ définition récursive}
si m=1 $PGCD_F := p_1$;
sinon
$PGCD_F := PGCD(p_1, p_2, ...p_{m-2}, PGCD(p_{m-1}, p_m))$;

Fin de $PGCD_F$.

On peut représenter le processus général de $PGCD_F$ par :



Remarques:

Le PGCD peut se calculer dans tout anneau euclidien ; le même algorithme est alors utilisé en particulier lorsque l'on travaille dans $\mathbb{Z}$.

Le coût C de cet algorithme correspond à m-1 fois le coût du PCDG de deux polynômes. Or l'algorithme du PGCD de deux polynômes de degrés respectifs d_1 et d_2 avec
$d_1 > d_2$ s'effectue en au moins d_1 multiplications de polynômes. Si l'on note M_i le coût d'une multiplication de deux polynômes de degré maximun d_i on peut écrire que :

$$C \leq \sum_{i=1}^{m-1} d_i M_i$$

IV- L'ALGORITHME DE GAUSS

On vient de voir, comment à partir d'un ensemble de polynômes à une variable, on peut trouver un seul polynôme qui permet de déterminer les racines communes des polynômes de l'ensemble donné.

Avant de voir comment on généralise cette méthode aux polynômes à plusieurs variables, nous rappellons ici l'algorithme de Gauss dont la technique s'apparente à la fois à celle utilisée dans la partie précédente (principe de divisions successives) et à celle que l'on utilisera pour "diviser" des polynômes à plusieurs variables.

L'algorithme de Gauss s'applique en fait à des polynômes à plusieurs variables bien particuliers : ce sont des polynômes d'exposants inférieurs à 1 quel que soit l'ordre choisi sur les monômes.

L'algorithme de Gauss, comme celui que l'on vient de voir, nous permet de caractériser un idéal. En effet, à partir d'une famille finie de polynômes génératrice d'un idéal, on en construit une autre, génératrice du même idéal, mais sur laquelle on détermine facilement les racines communes aux polynômes de départ, sous réserve d'existence d'au moins une racine.

Soient $p_1....p_m$ m polynômes à n variables $x_1,..,x_n$ à coefficients dans K on note, pour tout i de 1, m :

$$p_i = \sum_{j=1}^{n} a_{ij}x_j + a_{i0}$$

• Dans le cas où on a un ensemble fini de solutions on décrit l'algorithme de la façon suivante :

```
GAUSS ==
|  ENTREE : p1....pm  m polynômes de K[x1,..,xn],
|  SORTIE : p1....pm tel que le système associé soit triangulaire.
|      i := 1;
|      tant que i ≤ m
|              |  si ( aii ≠ 0)
|              |          |  pi := pi / aii ;
|              |          |  pour j = 0 à n
|              |          |          pj := pj - ajipi ;
|              |          |  i := i+1 ;
|              |  sinon
|              |          |  prendre k tel que aki ≠ 0
|              |          |  { k existe car on a un ensemble fini de solutions }
|              |          |  pi := pk ;
Fin de GAUSS.
```

A l'aide de cet algorithme on obtient un "système triangulaire", que l'on représente par :

$$\begin{array}{l} p_1 \\ p_2 \\ p_3 \\ \\ p_m \end{array} \begin{bmatrix} 1 & a_{12} & \ldots\ldots & a_{1n} & a_{10} \\ 0 & 1 & \ldots\ldots & a_{2n} & a_{20} \\ 0 & 0 & 1 & \ldots\ldots & \\ 0 & 0 & 0 & 1 & \ldots\ldots \\ \ldots & & & & \ldots\ldots \\ 0 & \ldots\ldots & & 0 & a_{m0} \end{bmatrix}$$

• Dans le cas où l'ensemble des solutions n'est pas fini, on utilise le même algorithme sachant toutefois qu'il existe i tel que $a_{ki} = 0$ pour tout k. On considère alors la variable x_i comme un paramètre et l'on passe au i suivant. Le système obtenu est encore triangulaire (au sens large) et l'on calcule en fait une base de l'espace des solutions.

L'analogie est simple avec l'algorithme donné dans la première partie : il s'agit bien dans les deux cas d'une "réécriture d'un polynôme modulo un autre ".

CHAPITRE II

LA NOTION DE BASES DE GRÖBNER UN ALGORITHME DE CALCUL

I- LA NOTION DE DIVISION

Nous avons vu comment à partir de la division euclidienne on associe à chaque idéal de l'anneau des polynômes à une variable, à coefficients dans un corps K, un élément privilégié (PGCD) qui permet de savoir de façon simple si un polynôme appartient à cet idéal.

Nous généralisons ici, la notion de division euclidienne, aux polynômes à plusieurs variables $K[x_1,...x_n]$, à l'aide des notions introduites dans le chapitreI de cette partie.

On introduit ici une division sur les polynômes de $K[x_1,...x_n]$ [Hironaka75], grâce au théorème (2-1) suivant :

Soit $\{p_1,...p_m\}$ une famille de m polynômes de $K[x_1,...x_n]$ et soit $\Delta_1,...\Delta_m$, $\underline{\Delta}$ la partition canonique associée (cf chapitre I)

Pour tout polynôme p de $K[x_1,...x_n]$ il existe $h_1,...h_m$ et $\underline{h}$ de $K[x_1,...x_n]$ uniques tels que :

i) $p = h_1p_1 + h_2p_2 + ...+h_mp_m + \underline{h}$

ii) $\forall i \in \{1,...m\}$ $Exp(p_i) + Supp(h_i) \subset \Delta_i$ et $Supp(\underline{h}) \subset \underline{\Delta}$

On a de plus que :

si $p \neq 0$ et $\underline{h} \neq 0$ alors $Exp(\underline{h}) \leq Exp(p)$

si $p \neq 0$ et $h_i \neq 0$ alors $Exp(h_i) + Exp(p_i) \leq Exp(p)$

L'unicité des polynômes décrits dans ce théorème résulte de la condition ii). Elle correspond en fait à celle imposée sur le degré du reste pour les polynômes à une variable.

L'existence résulte de l'algorithme suivant :

HIRONAKA ==

ENTREE : un polynôme p de $K[x_1,...x_n]$; une famille de polynômes $\{p_1,..p_m\}$ de $K[x_1,...x_n]$

SORTIE: $\{ h_1,...h_m \}$ et $h_0 = \underline{h}$

polynômes de de $K[x_1,...x_n]$ vérifiant le théorème (2-1)

{ initialisation }

pour i =0 à n

$h_i = 0$;

{ boucle principale }

tant que $p \neq 0$

soit $i \in \{ 0, 1,... m\}$ tel que $Exp(p) \in \Delta_i$ $(\Delta_0 = \underline{\Delta})$

{ i existe et est unique }

si $i \neq 0$

$p := p - (Init(p) / Init(p_i))\ p_i$;

$h_i := h_i + Init(p) / Init(p_i)$;

{ $Exp(p_i) + Supp(h_i) \subset \Delta_i$ }

sinon

$p := p - Init(p)$;

$h_0 := h_0 + Init(p)$;

fin d'HIRONAKA.

Ce processus se termine puisque Exp(p) décroît strictement à chaque étape et qu'il n'existe pas de suite infinie strictement décroissante dans $\mathbb{N}^n$.

L'opération principale de l'algorithme :

$$p = p - (Init(p) / Init(p_i))\ p_i \text{ et } h_i = h_i + Init(p) / Init(p_i),\ (i \neq 0),$$

correspond tout à fait à l'opération faite sur les différentes lignes d'une matrice traitée dans l'algorithme de Gauss ou encore au remplacement d'un polynôme par son reste dans l'algorithme d'Euclide.

On introduit les différentes définitions :

cet algorithme est dit *normalisation de p par rapport aux p_i* et h est *une forme normale de p par rapport aux p_i* . On note $\underline{h}$: $NormPF(p,\{p_1,..p_m\})$.

Si p est inchangé après normalisation on dira qu'il est *sous forme normale.*

Sur un exemple :

Soit la famille de polynômes de $\mathbb{Q}[x_1,x_2]$ suivante (on prend l'ordre lexicographique sur les monômes)

$p_1 = x_1^3$; $p_2 = x_1^2x_2 - x_2^3$; $p_3 = x_2^3 + x_2$ et soit $p = x_1^3x_2 + x_1$.
La forme normale de p modulo $F = \{p_1, p_2, p_3\}$ est alors $\underline{h} = x_1$
car $p = x_2 p_1 + \underline{h}$ avec les conditions ii) du théorème.

Il est bien évident que le résultat dépend de l'ordre choisi sur les monômes. Mais ils dépendent aussi de l'ordre d'indiciation des p_i. Par exemple :

La forme normale modulo $G = \{p_2, p_1, p_3\}$ est $\underline{h} = - x_1 x_2 + x_1$
car $p = x_1p_2 + x_1x_2^3 + x_1 = x_1p_2 + x_1p_3 - x_1 x_2 + x_1$ avec les conditions ii) du théorème.

On peut représenter ces résultats sur les graphiques suivant :

Pour F :

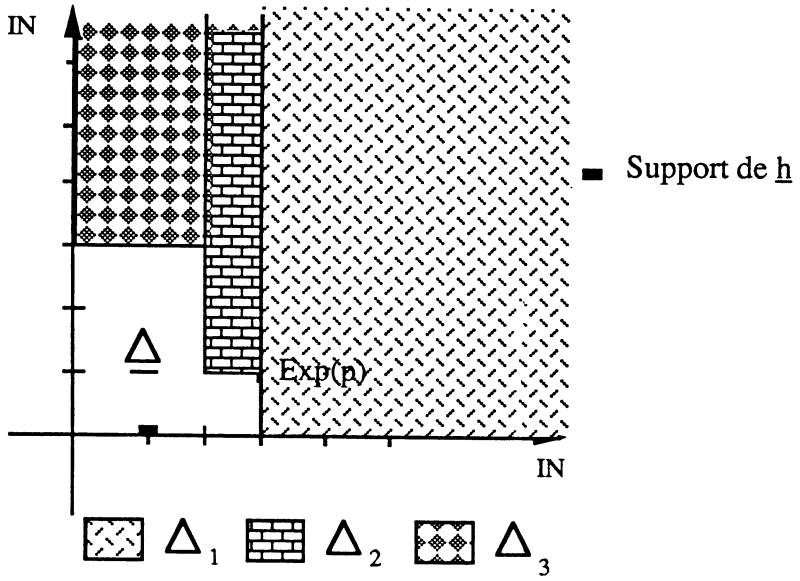


Figure1

Pour G :

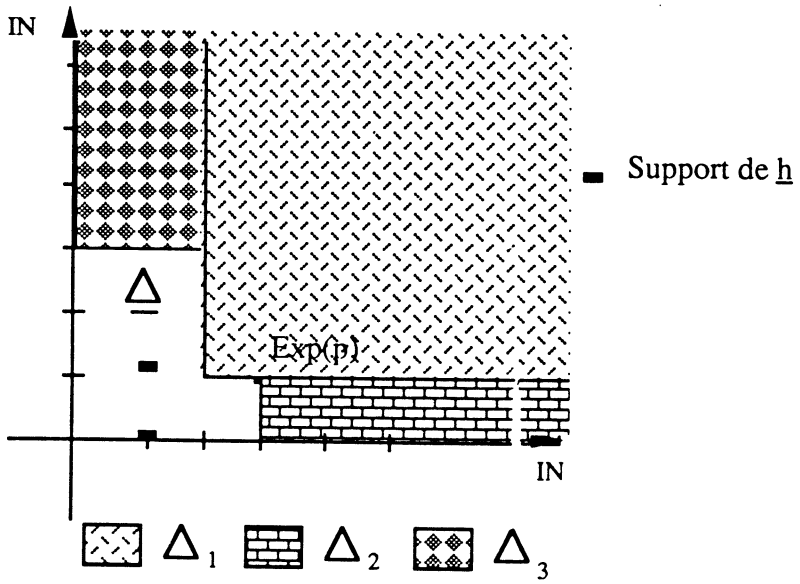


Figure2

II- BASE DE GRÖBNER D'UN IDEAL : DEFINITION

Soit $\mathfrak{J}$ un idéal de $K[x_1,...x_n]$ engendré par les polynômes de la famille $F = \{p_1, p_2, ...p_m\}$.

A F on associe le sous ensemble de $\mathbb{N}^n$, *Exp($\mathfrak{J}$)* défini par :

$$Exp(\mathfrak{J}) = \{ Exp(p) \text{ avec } p \in \mathfrak{J}, p \neq 0 \}$$

On a le résultat :

Exp($\mathfrak{J}$) est *une partie stable de $\mathbb{N}^n$* (cf chapitre I).

$\mathfrak{J}$ étant un idéal :

pour $\alpha \in Exp(\mathfrak{J})$ il existe $p \in \mathfrak{J}$ tel que $Exp(p) = \alpha$,

si $\delta \in \mathbb{N}^n$ $\alpha + \delta = Exp(X^\delta p)$ et $X^\delta p \in \mathfrak{J}$.

Exp($\mathfrak{J}$) possède une frontière $\mathfrak{F}$ et un escalier que l'on note $\underline{\mathfrak{F}}$; on dit aussi *escalier de $\mathfrak{J}$*.

On introduit la notion :

Une *base de Gröbner* (ou base standard) de $\mathfrak{J}$ est un ensemble de polynômes $\{g_1,...g_p\}$ de $\mathfrak{J}$ tel que $(Exp(g_1),..., Exp(g_p))$ soit une frontière de $\mathfrak{J}$, autrement dit tel que :

$$Exp(\mathfrak{J}) = \bigcup_{i \in \{1...p\}} (Exp(g_i) + \mathbb{N}^n)$$

Cette notion est bien définie, mais pour un idéal donné, il existe plusieurs ensembles de polynômes qui vérifient la définition.

Sous certaines conditions on a unicité (quitte à réindicer les g_i). On définit la notion de *base de Gröbner minimale* :

Une *base de Gröbner minimale* est une base de Gröbner de cardinal minimal, c'est à dire telle que les $Exp(g_i)$ forment l'escalier de $\mathfrak{J}$.

Remarques :

i) Une base de Gröbner est un système générateur de $\mathfrak{J}$,

ii) on n'a pas unicité de la base de Gröbner minimale (ne serait-ce qu'à cause des coefficients),

iii) une base de Gröbner minimale n'est pas un système générateur avec un nombre minimal d'éléments.

Puisqu'une base minimale n'est pas unique, on introduit de plus :

une *base de Gröbner réduite* de $\mathfrak{J}$ est une base de Gröbner minimale telle que :

pour tout i $Init(g_i) = X^{Exp(gi)}$,

tout élément g_i de cette base est sous forme normale par rapport à $\{g_1,...g_{i-1}, g_{i+1} ..., g_p\}$

Une base de Gröbner réduite est forcement minimale et on peut montrer que si $(g_1,..., g_p)$ et $(g'_1,.., g'_p)$ sont deux bases de Gröbner réduites alors il existe une permutation σ de $\{1,...,p\}$ telle que $g'_i = g_{\sigma(i)}$.

Autrement dit on a *unicité de la base de Gröbner réduite* quitte à réindexer les polynômes de la base.

Ceci implique entre autre que si l'on prend deux familles génératrices d'un même idéal, alors il n'existe qu'une base de Gröbner réduite (à l'indexage près).
La base de Gröbner réduite ne depend donc pas de la famille qui décrit l'idéal à caractériser.

Un des problèmes soulevé par l'exemple précédent est résolu : quelque soit l'ordre pris pour ranger les polynômes de la famille F la base de Gröbner réduite associée sera toujours la même. L'algorithme qui conduit, à partir d'un système de générateurs, à une base de Gröbner peut prendre plus ou moins de temps (cf Annexes).

On a également les résultats suivants :

i) Si G est une base de Gröbner d'un idéal $\mathfrak{J}$ alors la forme normale d'un polynôme quelconque par rapport à G est unique,

ii) si G et G' sont deux bases de Gröbner d'un même idéal pour le même ordre alors la forme normale de n'importe quel polynôme par rapport à G est la même que celle calculée par rapport à G',

iii) Soit G une base de Gröbner de $\mathfrak{J}$ on a équivalence entre :

$$p \in \mathfrak{J}$$

et la forme normale de p par rapport à G est nulle.

Dans l'exemple précédent :

$\{p_1, p_2, p_3, p_4\}$ avec

$$p_1 = x_1^3 ;$$
$$p_2 = x_1^2 x_2 - x_2^3 ;$$
$$p_3 = x_2^3 + x_2 ;$$
$$p_4 = x_1 x_2 ;$$
$$p_5 = x_2 ;$$

est une base de Gröbner associée à $\mathfrak{J}$.

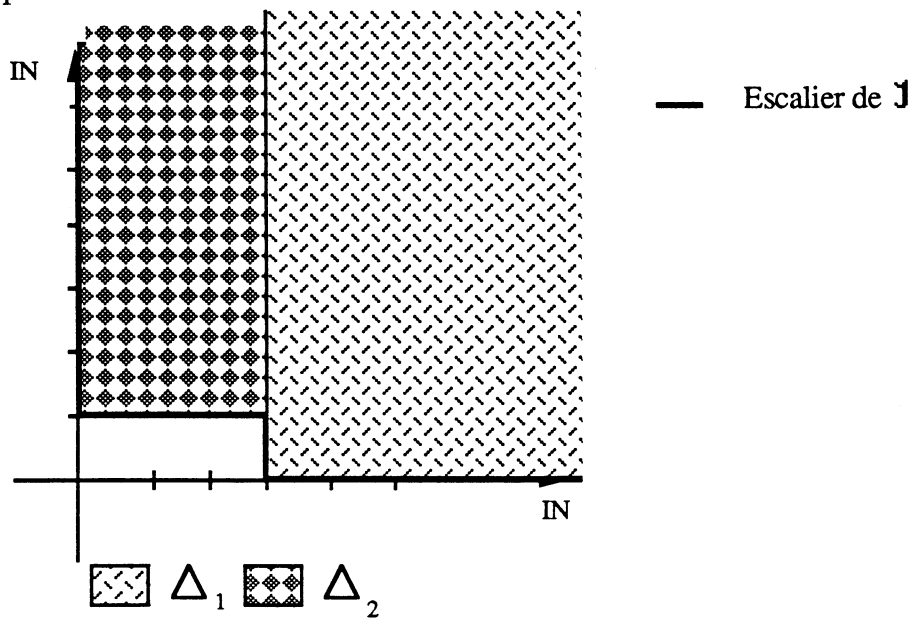
$G_1 = \{p_1', p'_2\}$ et $G_2 = \{p_1, p'_2\}$ avec :

$$p'_1 = x_1^3 + x_2 \; ; \; p'_2 = x_2$$

sont deux bases de Gröbner minimales.

G_2 est la base de Gröbner réduite.

Graphiquement :



En comparant ce graphique à la figure1 il est clair qu'un système de générateurs de l'idéal $\mathfrak{J}$ ne suffit pas pour décrire Exp($\mathfrak{J}$).

III- BASE DE GRÖBNER D'UN IDEAL : CALCUL

Les bases de Gröbner nous fournissent un test simple d'appartenance d'un polynôme à un idéal : si on a une base de Gröbner, p un polynôme quelconque sera dans l'idéal si sa forme normale par rapport aux éléments de la base est nulle.
Le problème est maintenant de déterminer une base de Gröbner d'un idéal donné par une famille génératrice (finie).

Soit $\mathfrak{J}$ un idéal, F une famille génératrice de $\mathfrak{J}$.

On utilise les notations :

Soit p et q deux polynômes de $K[x_1,...x_n]$ différents de zéro.
Soit $\alpha = (\alpha_1,...,\alpha_n) = Exp(p)$ et $\beta = (\beta_1,....,\beta_n) = Exp(q)$,
$Init(p) = c_p X^{\alpha}$ et $Init(q) = c_q X^{\beta}$
On pose $\delta = (\delta_1,...\delta_n)$ avec $\delta_i = Sup(\alpha_i , \beta_i)$ pour tous les indices i,

et on appelle :

spolynôme associé à p et q le polynôme défini par :

$$Spoly(p, q) = c_q X^{\delta-\alpha} p - c_p X^{\delta-\beta} q.$$

On dit que :

X^{δ} est *le plus petit commun multiple* de Exp(p) et Exp(q), on le note : PPCM(Exp(p), Exp(q)).

Remarque :

$$Exp(spoly(p, q)) < \delta.$$

On a le résultat fondamental suivant qui permet de construire un algorithme de calcul de bases de Gröbner :

Theorème (Buchberger) :

Un système de polynômes générateurs $G = \{g_1,..., g_p\}$ d'un idéal $\mathfrak{J}$ est une base de Gröbner pour $\mathfrak{J}$ si et seulement si :

pour tout (i,j) avec i <j, les formes normales par rapport à G des spolynômes $Spoly(g_i,g_j)$ sont nulles.

Nous ne démontrons pas ce résultat [Buchberger70].

Ce théorème nous fournit un moyen pour savoir si un ensemble de polynômes générateurs d'un idéal est ou n'est pas une base de Gröbner. Ce qui empêche un système de générateurs d'être une base de Gröbner ce sont les spolynômes de formes normales non nulles.

Pour construire un base de Gröbner associée à un idéal $\mathfrak{J}$ on opère de la façon suivante :

GRÖBNER ==

ENTREE : une famille $F = \{p_1,...p_m\}$ qui correspond à $\mathfrak{J}$
SORTIE: La base de Gröbner associée.

I := {1,...m} ;
tant que
il existe (i, j) de IxI tel que NormPF(Spoly(p_i, p_j), F) ≠0
p_{m+1} := NormPF(Spoly(p_i, p_j), F);
F := $\{p_1,...p_m, p_{m+1}\}$;
m :=m+1;
Sortir(F).

Fin de GRÖBNER.

Remarque :

i) Lorsque l'on ajoute un polynôme aux polynômes de départ, on a :
si NormPF(Spoly(p_i,p_j), $\{p_1,...p_m\}$) = 0
alors NormPF(Spoly(p_i,p_j), $\{p_1,...p_m, p_{m+1}\}$) = 0

ii) Ce processus de construction s'arrête. En effet si tel n'était pas le cas on construirait une suite infinie d'exposants et Exp($\mathfrak{J}$) en serait l'union infinie ; ce qui serait en contradiction avec le fait que Exp($\mathfrak{J}$) est une partie stable et a donc une frontière finie.

Si l'on considère l'exemple du paragraphe précédent on cherche une base de Gröbner de l'idéal engendré par $F = \{p_1, p_2, p_3\}$ avec :

$$p_1 = x_1^3 ;$$
$$p_2 = x_1^2x_2 - x_2^3 ;$$
$$p_3 = x_2^3 + x_2 ;$$

On forme :

Spoly(p_1, p_2) = $x_1x_2^3$; p_4 = NormPF(Spoly(p_1, p_2), F) = x_1x_2 ;

$F=F\cup\{p_4\}$;

Spoly(p_1, p_3) = 0 ; Spoly(p_1, p_4) = 0 ;

$Spoly(p_2, p_3) = x_1^2x_2^3 - x_2^5$; $p_5 = NormPF(Spoly(p_2, p_3), F) = 0$;

$Spoly(p_2, p_4) = -x_2^3$; $p_5 = NormPF(Spoly(p_3, p_4), F) = x_2$;

$Spoly(p_3, p_4) = x_1x_2$; $NormPF(Spoly(p_3, p_4), F) = 0$;
$Spoly(p_1, p_5) = 0$; $Spoly(p_4, p_5) = 0$;

$Spoly(p_2, p_5) = -x_2^3$; $NormPF(Spoly(p_2, p_5), F) = 0$;

$Spoly(p_3, p_5) = x_2$; $NormPF(Spoly(p_3, p_5), F) = 0$;

$\{p_1, p_2, p_3, p_4, p_5\}$ est une base de Gröbner associée à l'idéal engendré par : $\{p_1, p_2, p_3\}$.

IV- POUR LA BASE DE GROBNER REDUITE

L'algorithme précédent, ne donne pas une base de Gröbner minimale. Pour en construire une on procède de la façon suivante :

MINIMALE ==

ENTREE : $G = \{p_1,..p_q\}$ une base de Gröbner (obtenue, par exemple, à l'aide de l'algorithme précédent).

SORTIE : G minimale

tant qu'il existe $i \in \bigcup_{j \neq i} Exp(p_j) + \mathbb{N}^n$

$G := G - \{p_i\}$;

sortir(G).

fin de MINIMALE.

G contient un nombre fini d'éléments, et donc le processus s'arrête.

Pour tester si $Exp(p_i)$ appartient ou non à l'union précédemment citée, on introduit la notation :

on note $PGCD(m_1, m_2)$ où m_1 et m_2 sont deux monômes unitaires (sans coefficients) le plus grand monôme qui divise (au sens d'Hironaka) à la fois m_1 et m_2.

Dire que :

$$i \in \bigcup_{j \neq i} Exp(p_j) + \mathbb{N}^n$$

équivaut alors à dire que :

$$\text{si } NP_i \neq \varnothing \text{ alors } NormPF(p_i, NP_i) = 0$$

où $NP_i = \{ p_j \in G \,/\, PGCD(Exp(p_i), Exp(p_j)) = Exp(p_j) \}$.

Pour obtenir la base de Gröbner réduite, l'opération principale est la division d'Hironaka de p_i par $\{p_1,\ldots\ldots,p_{i-1}, p_{i+1},\ldots p_q\}$. Si le reste est nul on supprime p_i, s'il n'est pas nul on remplace p_i par ce reste.

Si on effectue systématiquement cette opération jusqu'à avoir tous les polynômes écrits sous forme normale par rapport aux autres, et si l'on divise les polynômes par le coefficient de leur partie initiale, on obtient la base de Gröbner réduite au bout d'un nombre fini d'étapes, plus précisement :

```
REDUITE ==
|
|  ENTREE : G = {p_1,..p_q} une base de Gröbner.
|  SORTIE : G réduite
|        { variable intermédiaire}
|
|        pour i =1 à q
|              | si NP_i ≠ ∅
|              |       | G := G - {p_i};
|
|        pour ( p_i dans G)
|              | p_i := NormPF(p_i, G - {p_i}) ;
|  sortir(G).
fin de REDUITE.
```

Buchberger [Buchberger83a] calcule la base de Gröbner réduite en utilisant le processus suivant :

G contient la base de Gröbner des entrées, et B les paires de polynômes qui n'ont pas encore été traitées. On utilise deux ensembles intermédaires : P et R.

R contient les polynômes susceptibles de ne pas être sous forme normale par rapport aux autres polynômes de G.
On normalise tour à tour ces polynômes que l'on stocke dans P. Quand R est vide, on remet G à jour en lui ajoutant les polynômes de P.

B est alors remis à jour et l'on considère un nouveau spolynôme que l'on normalise par rapport à B que l'on ajoute à P.

Certains polynômes de G peuvent ne pas être sous forme normale par rapport à ce nouveau polynôme, on les met alors dans R et on recommence.

Ce processus peut se décrire par :

GBuch (F) ==

ENTREE : une famille F de polynômes booléens.
SORTIE : la base de Gröbner booléenne réduite associée G.
{ variables intermédiaires }
R, G, P : familles de polynômes ;
B : ensemble de couples de polynômes ;
h : polynôme ;
{ Initialisation }
R := F ; P := ∅ ; G := ∅ ;
{ traitement des entrées : réduction préliminaire }
Réd (R, P, G, B) ;
{ itération principale }
tant que B ≠ ∅
Prendre C dans B : C = (p_1, p_2) ;
B := B - {C} ;
h := NormPF(Spoly(p_1, p_2), G) ;
Pm(h) := $X^{Exp(h)}$;
si h ≠ 0
{ pour g ∈ G on note Pm(g) = $X^{Exp(g)}$ }
G_0:= { g ∈ G / Pgcd(Pm(h), Pm(g)) = Pm(h) } ;
R := G_0 ; P := {h} ; G := G - G_0 ;
B := B - { (g_1, g_2) ∈ B / g_1 ∈ G_0 ou g_2 ∈ G_0} ;
Réd (R, P, G, B) ;
Sortir(G).

Fin de Gbuch.

La réduction Réd(R, P, G, B) est alors définie comme suit :

```
Réd(R, P, G, B) ==
    { variables intermédiaires }
    H, K : familles de polynômes ;
        tant que R ≠ ∅
        h := un élément de R ; R := R - {h} ;
        h := NormPF(h, P ∪ G) ;
         Pm(h) := X^Exp(h) ;
         h = (Pm(h) / Init(h)) h ;    { suppression du coefficient
                                        de tête }
        si h ≠ 0
            { pour g ∈ G on note  Pm(g) = X^Exp(g) }
            G0:= { g ∈ G / Pgcd(Pm(h), Pm(g)) = Pm(h) } ;
            P0:= { g ∈ P / Pgcd(Pm(h), Pm(g)) = Pm(h) } ;
            G := G - G0 ;
            P := P - P0 ;
            R := R ∪ P0 ∪ G0 ;
            B := B - { (g1, g2) ∈ B / g1 ∈ G0 ou g2 ∈ G0} ;
            P := P ∪ {h} ;
    G := G ∪ P ;
    B := B ∪ { (g, p) ∈ G x P / g ≠p } ;
    H := G ;
    K := ∅ ;
    tant que H ≠ ∅
        h := un élément de H ; H := H - {h} ;
        h := NormPF(h, G - {h} ) ;
        Pm(h) := X^Exp(h) ;
        h := (Pm(h) / Init(h)) h ;    { suppression du coefficient
                                        de tête }
        K := K ∪ {h} ;
    G := K.
```

Fin Réd.

Pour améliorer les temps d'exécution de cet algorithme on ne calcule pas tous les spolynômes. En effet si deux polynômes sont tels que :

$PGCD(X^{Exp(p)}, X^{exp(q)}) = 1$ alors la forme normale de Spoly(p, q) par rapport à une famille contenant p et q est toujours nulle (Buchberger critère2 [Buchberger83a]).

On dit alors que les monômes $X^{Exp(p)}$ et $X^{exp(q)}$ sont *étrangers*.

On suppose par la suite que ce test fait partie intégrante du calcul du spolynôme de deux polynômes : implicitement on ne fait pas le calcul lorsque le $PGCD(X^{Exp(p)},X^{exp(q)}) = 1$.

Au vu de la complexité de cet algorithme, dont nous faisons le point par la suite, Buchberger introduit un autre critère, le critère1, à appliquer sur les différents polynômes pour ne pas faire tous les calculs [Buchberger83a]. Ce critère est le suivant (avec les notions précédentes) :

si il existe un polynôme de G, p, qui vérifie les conditions suivantes :
i) $f_1 \neq p$, $f_2 \neq p$,
ii) lePPCM de $Exp(f_1)$ *et de* $Exp(f_2)$, *est multiple de* $Exp(p)$,
iii) les couples (f_1, p) *et* (f_2, p) *ne sont pas dans B,*

on ne calcule pas le spolynôme associé à f_1 *et* f_2.

Ce critère est un conséquence du théorème suivant [Buchberger 79] :

Un système de polynômes générateurs $G = \{g_1,..., g_p\}$ *d'un idéal* $\mathfrak{J}$ *est une base de Gröbner pour* $\mathfrak{J}$ *si et seulement si :*

pour tout f et g dans G il existe une suite finie $h_1,..., h_k$ *de polynômes de G telle que :*
i) $f = h_1$; $g = h_k$,
ii) le PPCM(Exp(g),exp(f)) est multiple de PPCM(Exp(h_1*),....,Exp(*h_k*)),*
iii) la forme normale de $Spoly(h_i,h_{i+1})$ *est nulle pour tout i de 1 à k.*

V- LA COMPLEXITE ; DIVERSES STRATEGIES

Il existe des bornes sur le nombre d'opérations à effectuer pour cette construction. Pour ce faire on a besoin de la notion de *degré* suivante :

> On appelle *degré d'un polynôme p* la somme des éléments du n-uplet formant Exp(p).

En l'état actuel de notre connaissance, on peut borner le degré maximal des polynômes de la base de Gröbner réduite et le nombre d'opérations à effectuer dans le corps de base K pour obtenir cette base [Lazard83] [Guisti85] [Möller & Mora84] : et ceci lorsque l'on connaît l'exposant maximal des polynômes donnés et que l'on se fixe soit l'ordre diagonal soit l'ordre du degré total.

Lazard [Lazard83] donne la borne suivante :

> Soit I l'idéal dont on cherche la base de Gröbner réduite.
> Soit d le degré maximal des polynômes donnés.
> Soit dg degré maximal des polynômes de la base de Gröbner réduite.
> Soit n le nombre de variables
> Soit $\underline{I}$.l'idéal engendré par les polynômes homogènes associés aux polynômes générateurs de I. On a :
>
> si dim($\underline{I}$) <= 0 alors
>
> $$dg \leq (n+1)\,(d-1) + 2$$

Guisti [Guisti85] donne, lorsque la dimension de $K[x_0, \ldots x_n] / I$ n'est pas supérieur à 1 la borne suivante

$$dg \leq (n+1)d^n$$

Mora et Möller [Möller & Mora84] utilisent la dimension s de l'idéal et donnent une borne du degré maximal des polynômes de n'importe quelle base de Gröbner, dgq (avec l'ordre du degré total) :

$$dgq \leq ((n+1)(d+1) +1)^{(n+1)\,2^{(s+1)}}$$

Guisti [Guisti85] nous fournit un algorithme de calcul qui nécessite (dans le cas de polynômes homogènes) :

$$O(dg^{(3(n+1))}) \text{ opérations sur K.}$$

Des études ont été menées dans le cas à deux variables.

En 1979 Buchberger [Buchberger79] donne une borne du nombre LO2 d'opérations sur K nécessaire à l'algorithme, dans le cas de l'ordre lexicographique inverse (avec des améliorations de l'algorithme GBuch) :

$$LO2 \leq 2(L + 16deg^2)^6$$

Où L est le nombre de polynômes donnés en entrée.

comme ($deg \leq 3d^2$) on a :

$$LO2 \leq 2\,(L + 48\,d^4)^6$$

Des améliorations ont été faites sur cette borne, en 83 Buchberger [Buchberger83b] donne, pour le cas de l'ordre total,

$$LO2 \leq 3/2\ (L + 2(d+2)^2)^4.$$

En 1985 Guisti nous donne une borne de DO2, nombre d'opérations nécessaires à l'algorithme pour l'ordre diagonal ou total [Guisti85] :

$$DO2 \leq O(d^6)$$

Il nous faut également signaler des études faites sur les polynômes à trois variables.

En particulier les travaux de Winkler [Winkler84], qui fournissent une borne sur le degré de tous les polynômes qui interviennent pendant l'algorithme de Buchberger. Si l'on note dd le degré minimal des polynômes de la famille donnée en entrée, tdeg le degré maximal des polynômes intervenant pendant l'algorithme et si l'on conserve les notations précédentes, on a :

$$tdeg \leq (8d + 1)\ 2^{dd}$$

Mais toutes ces bornes sont-telles significatives ?

Au vu de la variation du temps d'execution de l'algorithme en fonction des entrées, de l'ordre choisi sur les monômes, il semble en fait difficile de donner une complexité théorique de l'algorithme (cf exemples chapitre III).

Le nombre d'opérations nécessaires semble pour le moins incontrolable notamment au vu des importantes différences de temps entre la recherche de la base de Gröbner réduite d'un idéal donné par une certaine famille de polynômes et celle de la même base avec les mêmes polynômes initiaux mais indicés différemment (cf exemples).

Les études actuelles consistent à trouver des stratégies pour effectuer le moins d'opérations possibles. Buchberger est le premier à donner des critères sur les polynômes pour en éviter le traitement [Buchberger79].

Le premier fait remarquable, que nous avons déjà cité, est l'influence de l'ordre de sélection pour les paires (p_i, p_j) à traiter. Buchberger propose de traiter tout d'abord les polynômes p_i, p_j tels que le plus petit commun multiple de $Exp(p_i)$ et $Exp(p_j)$ soit minimal pour l'ordre choisi sur les monômes.
Si on décide de plus à chaque génération d'un nouveau polynôme p de normaliser s'il y a lieu p par rapport aux précédents ou de normaliser les précédents par rapport à p, il y aura une diminution notable du temps de calcul [Buchberger83a].

Nous avons vu un premier critère pour éviter le calcul de certains spolynômes (GBuch) il en existe d'autres, en particulier [Buchberger83a] :

Avec les notations de GBuch :

Soit p_1 et p_2 dans G. Si il existe $p \in G$ tels que Exp(p) est un diviseur du plus petit commun multiple de $Exp(p_1)$ et $Exp(p_2)$ et si on a traité les couples (p, p_1) et (p, p_2) alors on ne traite pas le couple (p_1, p_2)

Il existe plusieurs versions de ce critère [Buchberger79] nous nous contentons de celle énoncée ci-dessus [Buchberger83] qui permet, dans les situations favorables de ne calculer O(L) spolynômes au lieu de $O(L^2)$, ou L est le nombre de polynômes donnés.

Les plus récents travaux sur la question [Traverso & Donati89] ont pour but d'étudier les différents critères préexistants, et de trouver des méthodes heuristiques qui fournissent, en fonction de l'exemple traité, un choix sur les différentes combinaisons des différents critères qui optimise le temps de calcul.

Dans les algorithmes parallèles implantés nous n'avons pas pris en considération le critère1.
Il serait sans doute intéressant de le faire, surtout si l'on travaille avec des polynômes à coefficients dans $\mathbb{Q}$.
La principale motivation de ce choix est que le critère est intéressant si l'on peut le tester sur tous les polynômes p de G. Or dans notre modèle, en vertu de la localité des données, (cf troisième partie) on ne peut le faire efficacement.

CHAPITRE III

UNE APPLICATION IMPORTANTE : LA RESOLUTION DE SYSTEMES D'EQUATIONS NON LINEAIRES UTILISATION DE DIFFERENTS SYSTEMES DE CACUL FORMEL

Soit (S) un système d'équations algébriques nous nous intéressons ici uniquement au cas où (S) a **un nombre fini de solutions.**

I- LES SYSTEMES ALGEBRIQUES

Un système algébrique (S) correspond à la donnée de q polynômes à n variables à coefficients dans un corps. On définit (S) de la façon suivante :

$$(S)\begin{cases} p_1(x_1,x_2,\ .....,x_n) = 0 \\ p_2(x_1,x_2,\ .....,x_n) = 0 \\ ..... \\ ..... \\ p_q(x_1,x_2,\ .....,x_n) = 0 \end{cases}$$

On note F la famille des polynômes $p_1,....,\ p_q$ de $K[x_1,..,x_n]$, $\mathfrak{J}$ l'idéal engendré par F et G une base de Gröbner minimale de $\mathfrak{J}$.

Le problème posé consiste en la recherche d'un ou plusieurs zéros communs aux polynômes $p_1,..p_q$ dans une clotûre algébrique de K. Si K est algébriquement clos, cette recherche s'effectuera dans K.
On se borne ici, au cas où l'ensemble des racines cherchées est fini.

Par exemple si K = IR on cherche les racines dans l'ensemble des nombres complexes.

I-1 Un résultat d'existence

On a l'équivalence suivante [Lejeune85] :

$1 \in \mathfrak{J}$ si et seulement si (S) n'a pas de solution.

Exemple : le système,

$$(S)\begin{cases} x^2y - x^2 = 0 \\ x^3 - x^2 + y = 0 \\ xy^2 - xy + 2 = 0 \end{cases}$$

n'a pas de solution (dans n'importe quel corps) puisque 1 est dans l'idéal engendré par les polynômes définissant F.

I-2 Un procédé de résolution pour l'ordre lexicographique

Le procédé de construction est basé sur le résultat suivant [Trinks78] :

(1-2) $\mathfrak{J} \cap K[x_i,...x_n] = (F \cap K[x_i,...x_n])$

Plus précisément les polynômes à (n-i) variables de $\mathfrak{J}$ sont les mêmes que ceux qui sont combinaisons linéaires des polynômes de F qui ne contiennent que les (n-i) variables.

Plaçons nous dans le cas où i = n. On considère alors les polynômes en la variable x_n. Dans le cas de l'ordre lexicographique, on peut montrer [Trinks78] :

G contient exactement un polynôme g de $K[x_n]$ et $\mathfrak{J} \cap K[x_n] = g\,K[x_n]$
où $g\,K[x_n]$ est l'idéal de $K[x_n]$ engendré par g.

Ces résultats nous fournissent alors l'algorithme présenté ici récursivement :

```
RESLEX (n, F)==
|
| ENTREE : F famille des polynômes p1,....., pq de K[x1,..,xn].
| SORTIE : L'ensemble R des n-uplets qui annulent tous les pi.
|
|     si n=1
|         |  P:= PCCG(p1,....., pq) ;
|         |  RESLEX := { a / P(a) = 0 } ;
|     sinon
|         |  G := base de Gröbner minimale de F;
|         |  G := { g1,...gk};
|         |  si G ∩ K[xn] ≠ ∅
|         |     prendre g ∈ G ∩ K[xn]
|         |  sinon g := xn ;
|         |  R :={ a / g(a) = 0 };
|         |  Tant que (Card(R) ≠∅)
|         |      |  soit a∈ R ;
|         |      |  R := R - {a} ;
|         |      |  F := {g1(x1,...,xn-1,a),...gk(x1,...,xn-1,a)} ;
|         |      |  RESLEX :=RESLEX ∪ ({a}x RESLEX(n-1, F));
|         |      |  { x représente le produit cartésien des ensembles}
fin de RESLEX.
```

Exemple :

Avec l'ordre lexicographique tel que y > x :

Soit $F = \{ 3x^2y + 2xy + y + 9x^2 - 5x - 3 ; 2x^3y - xy - y + 6x^3 - 2x^2 - 3x + 3 ; x^3y + x^2y + 3x^3 + 2x^2 \}$

Une base de Gröbner minimale associée à F est :

$$G = \{ x^3 - 5/2x^2 - 5/2x ; y + x^2 - 3/2x - 3 \}$$

Le premier polynôme de G nous fournit trois racines :

$$a_1 = 0 ; a_2 = \frac{5 + \sqrt{65}}{4} ; a_3 = \frac{5 - \sqrt{65}}{4}$$

on remplace x par ces trois racines dans le deuxième polynôme et l'on obtient les trois solutions du système (S) :

$$(0 ; 3) ; (\frac{5 + \sqrt{65}}{4} ; - \frac{3 + \sqrt{65}}{4}) ; (\frac{5 - \sqrt{65}}{4} ; \frac{-3 + \sqrt{65}}{4})$$

Avec l'ordre lexicographique tel que x > y on obtient une base de Gröbner minimale :

$$G := \{ x - 1/4\ y^2 + 5/8y + 3/8 ; y^3 - 3/2y^2 - 8y + 21/2 \}$$

Le deuxième polynôme (en y) de G nous fournit les racines :

$$b_1 = 3 ; b_2 = \frac{3 + \sqrt{65}}{4} ; b_3 \quad \frac{-3 + \sqrt{65}}{4}.$$

on remplace y par ces trois racines dans le premier polynôme et l'on obtient les trois solutions du système (S) (les mêmes que précédemment, heureusement !!) :

$$(0 ; 3) ; (\frac{5 + \sqrt{65}}{4} ; \frac{3 + \sqrt{65}}{4}) ; (\frac{5 - \sqrt{65}}{4} ; \frac{-3 + \sqrt{65}}{4})$$

I-3 Pour un ordre quelconque

Le résultat (1-2) est encore valable. On peut procéder de manière analogue à la précédente, mais la détermination des polynômes à une variable considérés n'est plus immédiate. Il existe des méthodes pour déterminer ces polynômes. Leur inconvénient

majeur est qu'il faut alors résoudre des systèmes linaires pour déterminer les coefficients des polynômes (cf exemple).

En revanche pouvoir travailler avec un ordre autre que l'ordre lexicographique peut réduire la taille de coefficients intermédiaires (en particulier quand on a à traiter des polynômes à plus de trois variables) [Faugère & al 89].

Si on traite l'exemple ci-dessus avec l'ordre total et (y > x) on obtient :

$$G = \{ p_1 = x^2 + y - 3/2x - 3 ; p_2 = xy - y + x + 3 ; p_3 = y^2 - 5/2y - 4x - 3/2 \}$$

On détermine le polynôme p élément de l'idéal engendré par G à une seule variable $p = d_0 + d_1x + ... d_kx^k$:

On calcule les formes normales de 1, x, x^2,... par rapport à G notées $N(x^i,G)$.

On forme alors l'équation $d_0\ N(1,G) + ... d_i\ N(x^i,G) = 0$.

Si cette équation a une solution autre que la solution triviale elle nous fournit un système linéaire à résoudre qui nous permet de déterminer $d_0,..., d_i$ coefficients du polynôme cherché,
sinon on calcule $N(x^{i+1},G)$ et l'on forme l'équation :

$$d_0\ N(1,G) + ... + d_i\ N(x^i,G) + d_{i+1}\ N(x^{i+1},G) = 0.$$

Dans l'exemple,

$N(1, G) = 1$; l'équation n'a que la solution triviale,

$N(x,G) = x$; l'équation n'a que la solution triviale,

$N(x^2,G) = -y + 3/2x + 3$; l'équation n'a que la solution triviale,

$N(x^3,G) = -5/2y + 25/4x + 15/2$; et nous fournit l'équation :

$$d_0 + d_1\ x + d_2\ (-y + 3/2x + 3) + d_3\ (-5/2y + 25/4x + 15/2) = 0.$$

On a donc à résoudre le système linéaire suivant :

$5/2\ d_3 + d_2 = 0$

$25/4 d_3 + 3/2\ d_2 + d_1 = 0$

$15/2 d_3 + 3\ d_2 + d_0 = 0$

Le polynôme cherché est alors $p = x^3 - 5/2x^2 - 5/2x$ dont les racines sont :

$$a_1 = 0 ; a_2 = \frac{5 + \sqrt{65}}{4} ; a_3 = \frac{5 - \sqrt{65}}{4}$$

On remplace x successivement par a_1, a_2, a_3.

Pour a_1 on obtient les polynômes $y - 3$; $-y + 3$; $y^2 - 5/2y - 3/2$,

qui ont pour base de Gröbner la famille formée du polynôme $y-3$ qui a pour racine 3.

Le couple (0,3) est solution du système donné.

Pour a_2 on obtient de la même façon la solution (a_2,b_2) avec

$$b_2 = -\frac{3+\sqrt{65}}{4}$$

Pour a_3 on obtient la solution (a_3,b_3) avec

$$b_3 = \frac{-3+\sqrt{65}}{4}$$

On obtient bien les solutions données dans le cas de l'ordre lexicographique.

Sur cet exemple particulier, en utilisant le système de calcul formel REDUCE sur un MICROVAX II on calcule une base de Gröbner minimale en :

799 ms pour l'ordre lexicographique $y > x$

3451 ms pour l'ordre lexicographique $x > y$

2176 ms pour l'ordre total avec $y > x$

II- UTILISATION DE DIFFERENTS SYSTEMES DE CACUL FORMEL

La plus part des systèmes de calcul formel ont dans leur bibliothèque des fonctions qui permettent de calculer un base de Gröbner. Nous avons à notre disposition REDUCE, MAPLE et MACSYMA sur la même machine (MICROVAX II) ce que nous a permis une étude comparative qui est la suivante :

II-1 Utilisation de REDUCE

La fonction "GROEBNER" qui nous permet de calculer la base de Gröbner réduite associée à une famille de polynômes en tenant compte de l'ordre (spécifiable par l'utilisateur) a été mise en place par Gebauer, Hearn, Kredel, Möller, et étudié par Melenk, Möller et Neun [Melenk & al88a].

Sur ce système on peut faire appel à un switch qui permet d'avoir la trace des calculs intermédiaires (cf ANNEXES). On peut alors remarquer que tous les spolynômes ne sont pas calculés et que l'algorithme programmé tient compte de certains critères de Buchberger.

A chaque étape l'algorithme donne un hpolynôme, c'est à dire un spolynôme normalisé par rapport aux polynômes préexistants, et les nombres de spolynômes qu'il reste à calculer avant le calcul du hpolynôme. Vient ensuite l'étape de réduction finale.

Un autre switch nous permet d'avoir en plus les spolynômes calculés par l'algorithme (cf ANNEXES). Le temps en millisecondes nous assure que les bornes de complexité théorique sont loin d'être atteintes. On peut faire les quelques remarques d'ordre général suivantes :

- si on cherche la base de Gröbner associée à $\{p_1, p_2, p_3\}$ et celle associée à $\{p_1, p_2, p_3, p_4\}$ le temps mis pour calculer la première n'est pas forcément inférieur à celui mis pour calculer la deuxième (cf ANNEXES calculs 20 et 21),

- les temps de calculs sont très différents d'un ordre à l'autre, les polynômes obtenus aussi, en particuliers les coefficients sont plus petits avec l'ordre total,

- suivant l'indexage des entrées on peut noter une différence sensible dans le temps de calculs de la base de Gröbner réduite (cette remarque nous servira plus tard pour la parallélisation).

En dehors de la fonction "GROEBNER" il existe des fonctions permettant de faire la division d'Hironaka d'un polynôme par rapport à une famille.

II-2 Utilisation de MAPLE

L'utilisation de MAPLE est aussi simple que celle de REDUCE. La fonction à appeler pour calculer une base de Gröbner associée à une famille de polynôme est ici "GBASIS" qui tient également compte de l'ordre sur les monômes. La base de Gröbner obtenue n'est pas réduite. Elle est minimale et les polynômes obtenus sont normalisés les uns par rapport aux autres, mais les monômes de polynômes ne sont pas rangés dans l'ordre donné pour le calcul de la base et les polynômes ne sont pas unitaires.

Sinon, les remarques faites pour REDUCE sont les mêmes que pour MAPLE.
La comparaison des temps de calcul donnés pour MAPLE en secondes prouve que REDUCE est plus performant.
Maple nous permet de faire apparaître la place mémoire utilisée pour le calcul (en mots de 4 octets).
Il existe également des fonctions qui permettent d'effectuer la division d'Hironaka et l'on peut ainsi savoir si un polynôme appartient où non à un idéal donné.

II-3 Utilisation de MACSYMA

Les fonctions relatives aux calculs des bases de Gröbner sont plus nombreuses : on peut par exemple, savoir directement si un polynôme est dans un idéal donné.

L'utilisation de drapeaux permet la manipulation de variables booléennes qui fournissent les choix suivants :

- l'ordre sur les monômes (lexicographique ou total)
- la trace
- la division des coefficients
- l'obtention de la base de gröbner réduite
- la réduction en cours de calcul ....

On peut remarquer que la trace est moins lisible que celle obtenue par REDUCE, et qu'au niveau du temps de calcul on a les tableaux comparatifs suivants :

Avec les exemples des familles f1 et f2 données en annexe

i) l'ordre lexicographique avec $x > y > z$:

	MAPLE	MACSYMA	REDUCE
f1	9.97 s	43.52 s	0.56 s
f2	16.43 s	93 s	1.62 s

ii) l'ordre lexicographique avec $x < y < z$

	MAPLE	MACSYMA	REDUCE
f1	11.39 s	11.35 s	1.54 s
f2	26.67 s	29.4 s	3.76 s

iii) l'ordre total avec x >y >z

	MAPLE	MACSYMA	REDUCE
f1	3.59 s	12.8 s	0.255 s
f2	8.16 s	33.9 s	0.612 s

En dehors de ces trois systèmes il en existe d'autres : SCRATCHPAD II notamment qui fonctionne sur un IBM 4381 nous fournit des temps analogues à ceux de REDUCE.

ANNEXES

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ESSAI AVEC TRACE  %   REDUCE 3.3, 15-Jul-87 ...


                        % LES ENTREES : DEUX POLYNOMES  %


3: p1:=x**2*y*z+x*y+1;

        2
P1 := X *Y*Z + X*Y + 1

Time: 170 ms

4: p2:=3*x**2*y+x*y**2+4;

          2         2
P2 := 3*X *Y + X*Y  + 4

Time: 204 ms


5: on trgroeb;          % DEMANDE DE TRACE GENERALE %

Time: 34 ms

                       % CALCUL %


6: groebner({p1,p2});


h-polynom 3     pair ( 1,2 ) :

Z*X*Y**2 + 4*Z - 3*X*Y - 3


 computing time for h-polynom 0
```

```
Restpairs: 0

h-polynom 4     pair ( 2,3 ) :

Z*X - 3/4*X - 1/4*Y + 1

 computing time for h-polynom 17


Restpairs: 0

h-polynom 5     pair ( 3,4 ) :

Z + 3/16*X*Y**2 - 3/4*X*Y + 1/16*Y**3 - 1/4*Y**2 - 3/4

 computing time for h-polynom 17


Restpairs: 1


-------------------------------------------------------
 / reduction of polynom 4 and 5 leads to 0   ( 51  ms )
-------------------------------------------------------


-------------------------------------------------------
 / reduction of polynom 2 and 4 leads to 0   ( 102  ms )
-------------------------------------------------------

 calculation now in final reduction

 computing time for final calculation 0    milliseconds

 Number of Groebner Basis Polynomials :=   2

 The Groebner Basis Polynomials
```

```
Z + 3/16*X*Y**2 - 3/4*X*Y + 1/16*Y**3 - 1/4*Y**2 - 3/4


X**2*Y + 1/3*X*Y**2 + 4/3

Computing time for test 765 milliseconds
```

$$\{Z + \frac{3}{16}X*Y^2 - \frac{3}{4}X*Y + \frac{1}{16}Y^3 - \frac{1}{4}Y^2 - \frac{3}{4},$$

$$3*X^2*Y + X*Y^2 + 4\}$$

```
Time: 1258 ms

                    % DEMANDE DES SPOLYNOMES %

7: on trgroebs;

Time: 34 ms
              % CALCUL %

8: groebner({p1,p2});

S-polynom:

Z*X*Y**2 + 4*Z - 3*X*Y - 3


h-polynom 3     pair ( 1,2 ) :

Z*X*Y**2 + 4*Z - 3*X*Y - 3


 computing time for h-polynom 34


Restpairs: 0

S-polynom:

 - Z*X*Y**3 + 12*Z*X - 4*Z*Y - 9*X**2*Y - 9*X
```

```
h-polynom 4     pair ( 2,3 ) :

Z*X - 3/4*X - 1/4*Y + 1

 computing time for h-polynom 68


Restpairs: 0

S-polynom:

 - 4*Z - 3/4*X*Y**2 + 3*X*Y - 1/4*Y**3 + Y**2 + 3

h-polynom 5     pair ( 3,4 ) :

Z + 3/16*X*Y**2 - 3/4*X*Y + 1/16*Y**3 - 1/4*Y**2 - 3/4

 computing time for h-polynom 68


Restpairs: 1

S-polynom:

3/16*X**2*Y**2 - 3/4*X**2*Y + 1/16*X*Y**3 - 1/4*X*Y**2 + 1/4*Y - 1


------------------------------------------------------
 / reduction of polynom 4 and 5 leads to 0   ( 102  ms )
------------------------------------------------------

S-polynom:

- Z*X*Y**2 - 4*Z - 9/4*X**2*Y - 3/4*X*Y**2 +

3*X*Y
```

```
------------------------------------------------------
 / reduction of polynom 2 and 4 leads to 0   ( 153  ms )
------------------------------------------------------

 calculation now in final reduction

 computing time for final calculation 0    milliseconds

 Number of Groebner Basis Polynomials :=   2

 The Groebner Basis Polynomials

Z + 3/16*X*Y**2 - 3/4*X*Y + 1/16*Y**3 - 1/4*Y**2 - 3/4

X**2*Y + 1/3*X*Y**2 + 4/3

Computing time for test 1003 milliseconds

      3      2    3          1    3    1   2    3
{Z + ----*X*Y  - ---*X*Y + ----*Y  - ---*Y  - ---,
      16          4         16        4        4

    2         2
 3*X *Y + X*Y  + 4}

Time: 1479 ms

                    % FIN DE SESSION %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ESSAI AVEC DIFERENTS ORDRES %  REDUCE 3.3, 15-Jul-87 ...


                    % LES ENTREES : 5 polynomes / 3 variables %


3: p1:=x**2-2*x*z+5;

        2
P1 := X  - 2*X*Z + 5

Time: 153 ms

4: p2:=x*y**2+y*z**3;

                 3
P2 := Y*(X*Y + Z )

Time: 119 ms

5: p3:=3*y**2-8*z**3;

          2     3
P3 := 3*Y  - 8*Z

Time: 153 ms

6: p4:=2*x*z+x*y+y;

P4 := X*Y + 2*X*Z + Y

Time: 170 ms

7: p5:=x+2*y*z**3+3*z**2;

                3     2
P5 := X + 2*Y*Z  + 3*Z

Time: 221 ms

8: F1:={p1,p2}; F2:={p1,p2,p3};

 F3:={p1,p2,p3,p4}; F5:={p1,p2,p3,p4,p5};
```

% CALCULS %

% AVEC L'ORDRE LEXICOGRAPHIQUE SUR LES VARIABLES %

% avec X > Y > Z %

```
9: groebner(f1,{x,y,z});

  2
{X  - 2*X*Z + 5,

    2      3
 X*Y  + Y*Z ,

      3      2        4
 X*Y*Z  - 5*Y  - 2*Y*Z ,

  3    2   2  4    1     6
 Y  + ---*Y *Z  + ---*Y*Z }
       5           5

Time: 561 ms


10: groebner(f2,{x,y,z});

  2
{X  - 2*X*Z + 5,

    3     9    8    3    7    3    5
 X*Z  + -----*Z  - ----*Z  + ----*Z ,
        640        20        16

    2      3
 3*Y  - 8*Z ,

    3    3    8    2   7    1   5
 Y*Z  - ----*Z  + ---*Z  - ---*Z ,
        80         5        2
```

```
  9     32    8     80    6     1600    3
 Z  - -----*Z  + -----*Z  + -------*Z }
       3           3            9
```

Time: 1615 ms

```
11: groebner(f3,{x,y,z});

  2
{X  + 6*Y + 10*Z + 5,

 X*Y - 5*Y - 10*Z,

 X*Z + 3*Y + 5*Z,

    2
 3*Y ,

 Y*Z,

  2
 Z }
```

Time: 5134 ms

```
12: groebner(f4,{x,y,z});

{1}
```

Time: 11390 ms

% avec Z > Y > X %

```
13: groebner(f1,{z,y,x});

        4     2  3     1       5     3       3
{Z*Y + -----*Y *X  + -----*Y*X  + -----*Y*X  + Y*X,
       25            50           10

              2
  - 2*Z*X + X  + 5,

  2  4     1      6     15      4     75      2     125
 Y *X  + ---*Y*X  + -----*Y*X  + -----*Y*X  + ------*Y}
          8           8           8            8
```

```
Time: 1547 ms


14: groebner(f2,{z,y,x});

       1     11     32    10     3    9    32    8    3    7    32
{Z + ------*X   - ------*X   + -----*X  - -----*X  + ----*X  - -----*
     6250         9375         625        625        50         125

  6    2   5    32   4    3   3    5
 X  + ---*X  - ----*X  + ---*X  + ---*X,
       5        75        2        2

  2     1    11     64    10    17    9    512    8    13   7
 Y  - -----*X   + ------*X   - -----*X  + ------*X  - ----*X  +
       625         1875         375        1125        25

  128   6      5    26   3    64   2
 -----*X  - 3*X  - ----*X  - ----*X  - 10*X,
  75               3         9

    6         4         2            8   7       5        3    1000
 Y*X  + 15*Y*X  + 75*Y*X  + 125*Y + ---*X  + 40*X  + 200*X  + ------*
                                    3                          3

 X,

  12    64   11       10        9        8         7         6
 X   - ----*X   + 30*X   - 320*X  + 375*X  - 1600*X  + 2500*X  -
        3

  8000   5         4          2
 ------*X  + 9375*X  + 18750*X  + 15625}
   3

Time: 3757 ms


15: groebner(f3,{z,y,x});

      1    3    1
{Z + ----*X  + ---*X,
      10        2
```

```
      1   3    1   2    5       5
 Y - ---*X  + ---*X  - ---*X + ---,
      6        6        6       6

  4       2
 X  + 10*X  + 25}

Time: 16150 ms


16: groebner(f4,{z,y,x});

{1}

Time: 16830 ms


       % AVEC L'ORDRE DU DEGRE TOTAL %



     %    avec    z > Y > X  %

17: torder totaldegree;

INVLEX

Time: 68 ms


18: groebner(f1,{x,y,z});

  4        2  2         2       2
{X *Y + 8*X *Y  + 20*Y*Z  + 10*X *Y + 25*Y,

    3      2
 Y*Z  + X*Y ,

             2
  - 2*X*Z + X  + 5}

Time: 493 ms


19: groebner(f2,{x,y,z});
```

```
  4        2       2       2
{X  - 3*X*Y  + 20*Z  + 10*X  + 25,

       3      2
  - 8*Z  + 3*Y ,

  3    8     2
 Y  + ---*X*Y ,
       3

              2
  - 2*X*Z + X  + 5}
```

Time: 663 ms

```
20: groebner(f3,{x,y,z});

  2
{Z ,

 Y*Z,

  2
 Y ,

 2*X*Z + 10*Z + 6*Y,

 X*Y - 10*Z - 5*Y,

  2
 X  + 10*Z + 6*Y + 5}
```

Time: 6239 ms

```
21: groebner(f4,{x,y,z});

{1}
```

Time: 4471 ms

% avec X > Y > Z %

```
22: groebner(f1,{z,y,x});

   3      2    2
{Z *Y + Y *X,X  - 2*Z*X + 5}

Time: 255 ms


23: groebner(f2,{z,y,x});

   4     16      3     320   2
{Y  + ----*Z*Y  + -----*Y ,
       3             9

  2       3   3
 Y *X + ---*Y ,
         8

       3      2
  - 8*Z  + 3*Y ,

   2
  X  - 2*Z*X + 5}

Time: 612 ms


24: groebner(f3,{z,x,y});

   2
{Y ,

 X*Y - 5*Y - 10*Z,

  2
 X  + 6*Y + 10*Z + 5,

 Z*Y,

 Z*X + 3*Y + 5*Z,

  2
 Z }

Time: 3536 ms
```

```
25: groebner(f3,{z,y,x});

  2
{X  + 6*Y + 10*Z + 5,

 Y*X - 5*Y - 10*Z,

  2
 Y ,

 Z*X + 3*Y + 5*Z,

 Z*Y,

  2
 Z }

Time: 2550 ms


26: groebner(f4,{z,y,x});

{1}

Time: 9979 ms

                % FIN DE SESSION %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ESSAI AVEC DIFFERENTS INDICES SUR LES ENTREES %
REDUCE 3.3, 15-Jul-87 ...


                              % LES ENTREES %
1: p1:=x**2-2*x*z+5;

       2
P1 := X  - 2*X*Z + 5

2: p2:=x*y**2+y*z**3;

                  3
P2 := Y*(X*Y + Z )

3: p3:=3*y**2-8*z**3;

         2      3
P3 := 3*Y  - 8*Z

4: p4:=2*x*z+y*x+y;

P4 := X*Y + 2*X*Z + Y

                          % CALCULS %


7: groebner({p1,p4,p2,p3},{x,y,z});          % POUR P1,P2,P3,P4 %

  2
{X  + 6*Y + 10*Z + 5,

 X*Y - 5*Y - 10*Z,

 X*Z + 3*Y + 5*Z,

    2
 3*Y ,

 Y*Z,

  2
 Z }
```

```
Time: 4420 ms


8: groebner({p3,p4,p1,p2},{x,y,z});      % POUR P3,P4,P1,P2 %

  2
{X  + 6*Y + 10*Z + 5,

 X*Y - 5*Y - 10*Z,

 X*Z + 3*Y + 5*Z,

    2
 3*Y ,

 Y*Z,

  2
 Z }

Time: 5474 ms

% ON A OBTENU LE MEME RESULTAT (UNICITE DE LA BASE REDUITE)
   AVEC DES TEMPS DIFFERENTS  %


     % DE LA MEME FACON POUR P1,P2,P3 :%


9: groebner({p3,p2,p1},{x,y,z});      % POUR P3,P2,P1 %

  2
{X  - 2*X*Z + 5,

    3     9    8    3    7    3    5
 X*Z  + -----*Z  - ----*Z  + ----*Z ,
        640        20        16

    2      3
 3*Y  - 8*Z ,

    3    3    8    2   7    1   5
 Y*Z  - ----*Z  + ---*Z  - ---*Z ,
        80        5        2
```

```
 9     32   8     80   6     1600   3
Z  - ----*Z  + ----*Z  + ------*Z }
      3           3           9

Time: 1700 ms


10: groebner({p1,p3,p2},{x,y,z});      % pour P1,P3,P2 %

  2
{X  - 2*X*Z + 5,

   3     9    8     3    7     3    5
 X*Z  + -----*Z  - ----*Z  + ----*Z  ,
         640        20         16

   2      3
 3*Y  - 8*Z  ,

   3    3    8     2    7    1    5
 Y*Z  - ----*Z  + ---*Z  - ---*Z  ,
         80        5         2

 9     32   8     80   6     1600   3
Z  - ----*Z  + ----*Z  + ------*Z }
      3           3           9

Time: 1394 ms

          % FIN DE SESSION  %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

GLOSSAIRE :

DEUXIEME PARTIE :

APPLICATION DES BASES DE GRÖBNER LORSQUE $K = \mathbb{Z}/2\mathbb{Z}$

DEUXIEME PARTIE : APPLICATION DES BASES DE GRÖBNER LOSQUE K = $\mathbb{Z}/2\mathbb{Z}$

INTRODUCTION

Après avoir étudié théoriquement les bases de Gröbner nous en présentons quelques applications lorsque le corps de base est $\mathbb{Z}/2\mathbb{Z}$. Nous nous limiterons aux cas simples du calcul propositionnel et de la preuve de circuits combinatoires. Dans la conclusion, nous faisons le point sur ce qu'il pourrait être fait pour résoudre certains problèmes et généraliser certaines méthodes.

Le premier chapitre de cette partie, est formé de quelques préliminaires destinés à mettre en place le formalisme, restreint à $\mathbb{Z}/2\mathbb{Z}$, utilisé par la suite.

Le deuxième chapitre fait le lien entre les problèmes de logique, notamment ceux du calcul propositionnel, et des bases de Gröbner. Cette démarche est la même que celle utilisée pour prouver des circuits combinatoires. Les outils essentiels sont la transformation de Stone [Stone64] et les bases de Gröbner.

Le troisième chapitre correspond à l'utilisation des bases de Gröbner en preuve de circuits. On se restreint ici aux circuits combinatoires. Mais la reconnaissance du fait qu'un programme de preuve de circuits n'était autre qu'un programme de manipulation symbolique des objets d'un niveau sémantique plus ou moins élevé a été remarquée depuis un certain temps [Barrow84]. La connexion avec les langages actuels de manipulation algébrique tels que Macsyma et Reduce mettait ainsi en évidence une problèmatique commune qui se traduisait par des approches connues de formes normales, simplification, canonicité, etc....
On pourrait alors penser manipuler des objets plus généraux que les polynômes booléens. Cette approche rejoint certains travaux de logique [Kapur & Narendram85], et pourrait donner lieu à une étude future.

CHAPITRE I

PRELIMINAIRES

I- L'ALGORITHME LORSQUE LE CORPS DE BASE EST $\mathbb{Z}/2\mathbb{Z}$

Nous nous plaçons ici dans le cas de l'anneau $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n]$. Toutes les notions introduites dans la première partie sont encore valables puisque $\mathbb{Z}/2\mathbb{Z}$ est un corps commutatif.

La structure de l'algorithme de Buchberger reste la même ; les opérations de base sont simplifiées (en particulier puisque $x+x = 0$, pour tout x de $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n]$). Nous ne reprenons pas ici toute la structure de l'algorithme, nous regardons seulement ce que deviennent le calcul de la forme normale d'un polynôme par rapport à une famille de polynômes et celui d'un spolynôme.

Les objets que nous manipulons sont des polynômes de la forme :

$$\sum \prod_j x_j^{i_j}$$

Dans toute la suite, on se fixe un ordre : l'ordre lexicographique sur les variables.

I-1 La Normalisation

Il s'agit de la division d'Hironaka d'un polynôme par rapport à une famille de polynômes. Dans notre cas particulier elle devient :

Soit p un élément de $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n]$; Init (p) = X^{α}.

Soit F= { $p_1,....,p_m$ } une famille de polynômes avec Init(p_i) = X^{α_i}.

L'opération principale de l'algorithme de normalisation (cf première partie chapitreII) consiste à remplacer :

d'une part p par p - (Init(p) / (Init(p_i))p_i

et d'autre part h_i par h_i + Init(p) / Init(p_i).

Ici si α est multiple de α_i on remplace :

p par p + $X^{\alpha - \alpha_i}$ p_i

h_i par h_i + $X^{\alpha - \alpha_i}$

Ceci est effectué non seulement pour α tels que Init (p) = X^{α} mais aussi pour ceux du support de p tels que α est multiple de α_i

Cette opération est aussi appelée *application de réécriture* [Chazarain86].

Exemple :

Soit $p = x^2y^2 + xy^2 + y + 1$.

Soit $F = \{ p_1, p_2 \}$ avec $p_1 = xy^2 + y$ et $p_2 = x + 1$.

$p = (x + 1)\, p_1 + xy + 1 = (x + 1)\, p_1 + y\, p_2 + 1 + y$

p prend successivement les valeurs : $xy + 1$ et $1 + y$.

$1 + y$ est la forme normale de p par rapport à F.

Remarque :

La normalisation d'un polynôme par rapport à une famille est parfois considérée comme la clôture réflexive transitive de l'union des applications de réécriture[Chazarain86]. Dire alors que l'algorithme de division s'arrête c'est exactement dire que la relation associée à cette clôture est confluente : pour cela, il est nécessaire que l'ordre choisi sur les monômes vérifie les conditions (HO) de la première partie.

I-2 le calcul du spolynôme

Le calcul du spolynôme associé à deux polynômes différents est également simplifié puisque qu'il n'y a pas de coefficients qui interviennent devant les monômes.

Exemple :

Soit $p = x^2y^3 + xy + 1$ et soit $q = xy + 1$;

$Spoly(p, q) = x^2y^3 + xy + 1 + xy^2\,(xy + 1) = xy^2 + xy + 1$.

Au vu de ces applications la notion de base de Gröbner dans le cas de l'anneau $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n]$ a été étudiée par Sakai et Sato [Sakai & Sato88]. Stillman [Stillmann89] donne des bornes sur la complexité des algorithmes permettant de calculer de telles bases. Il montre que ces bornes sont nécessairement exponentielles en temps et en espace, ce qui confirme le comportement de l'algorithme dans un cas plus général.

II- LE CADRE GENERAL DES APPLICATIONS

II-1 Le cadre général. Notations

Soit $\mathfrak{B}^*$ l' anneau de Boole engendré par l'ensemble fini $\{x_1,..., x_n\}$, c'est à dire le plus petit anneau commutatif qui contient $\{x_1,..., x_n\}$ et tel que $X^2 = X$ pour tous ses éléments X.

On appelle les éléments de $\mathfrak{B}^*$ *les polynômes booléens.*

Nous verrons par la suite plus précisément pourquoi on est amené à considérer un tel anneau.

- **Les propriétés algébriques de $\mathfrak{B}^*$.**

Les polynômes booléens peuvent être considérés comme des représentants des classes d'équivalence de $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n]$ avec la relation d'équivalence définie par :

> deux polynômes de $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n]$ sont dans la même classe d'équivalence si leur différence est dans l'idéal I engendré par les polynômes $x_1^2 + x_1$, ..., $x_n^2 + x_n$.

En effet :

Soit $\mathfrak{B}$ l'ensemble de telles classes d'équivalence. $\mathfrak{B}$ est donc l'anneau quotient $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n] / I$.

$\mathfrak{B}$ et $\mathfrak{B}^*$ sont isomorphes

L'isomorphisme se construit de la façon suivante :

On considère l'homomorphisme d'anneau φ de $\mathbb{Z}[x_1,..,x_n]$ dans $\mathfrak{B}^*$ qui à chaque x_i de $\mathbb{Z}[x_1,..,x_n]$ associe x_i de $\mathfrak{B}^*$. Cet homomorphisme est surjectif et son noyau J est l'idéal engendré par $2, x_1^2 + x_1$, ..., $x_n^2 + x_n$.

On peut alors construire un homomorphisme d'anneau surjectif de $\mathbb{Z}[x_1,..,x_n] / J$ dans $\mathfrak{B}$. Comme de plus $\mathbb{Z}[x_1,..,x_n] / J$ est isomorphe à $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n] / I$, on peut construire un homomorphisme d'anneau surjectif Φ de$\mathfrak{B}$ dans $\mathfrak{B}^*$. Reste à avoir qu'il est injectif.

On note i l'application de $\mathbb{Z}[x_1,..,x_n]$ dans $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n] / I$.

Soient C_1 et C_2 deux éléments de $\mathfrak{B}$,
Soient p_1 et p_2 deux éléments de $\mathfrak{B}^*$ tels que $\Phi(C_1) = p_1$ et $\Phi(C_2) = p_2$, .

Comme φ est surjectif,

il existe q_1 et q_2 dans $\mathbb{Z}[x_1,..,x_n]$ tels que $\varphi(q_1) = p_1$ et $\varphi(q_2) = p_2$, et donc $\varphi(q_1) = \Phi(C_1)$ et $\varphi(q_2) = \Phi(C_2)$

Si $p_1 = p_2$ alors $\varphi(q_1 - q_2) = 0$

et donc $q_1 - q_2 \in I$.

On a , puisque φ est un homomorphisme d'anneau surjectif,

$\varphi^{-1}(\Phi(C_1)) - \varphi^{-1}(\Phi(C_2)) \in I$, et donc $i(\varphi^{-1}(\Phi(C_1)) - \varphi^{-1}(\Phi(C_2))) = 0$, c'est à dire par construction : $C_1 = C_2$.

Cet isomorphisme permet d'associer à une classe C de $\mathfrak{B}$ un polynôme (unique) de $\mathfrak{B}^*$. On confondra désormais $\mathfrak{B}^*$ et $\mathfrak{B}$.

II-2 La transformation de Stone

Cette transformation nous permet de passer d'une algèbre de Boole à un anneau de Boole et réciproquement. Les fonctions booléennes seront alors considérées comme des polynômes booléens.

Soit $\mathfrak{B}$ une *algèbre booléenne*. $\mathfrak{B}$ est alors un ensemble muni de deux opérateurs binaires notés respectivement $\wedge$ et $\vee$, d'un opérateur unaire noté ' et de deux éléments notés 0 et 1 tels que tout triplet a, b, c d'éléments de $\mathfrak{B}$ vérifie les sept propriétés suivantes (certaines sont redondantes) [Kuntzmann65] [Grätzer 78]:

i) $a \vee (b \vee c) = (a \vee b) \vee c$; $a \wedge (b \wedge c) = (a \wedge b) \wedge c$;

ii) $a \vee b = b \vee a$; $a \wedge b = b \wedge a$;

iii) $a \vee a = a$; $a \wedge a = a$;

iv) $a \wedge (b \vee a) = a$; $a \vee (b \wedge a) = a$;

v) $(a \vee b) \wedge c = (a \wedge c) \vee (b \wedge c)$;

vi) $a \vee 0 = a$; $a \vee 1 = 1$;

vii) $a \vee a' = 1$; $a \wedge a' = 0$.

Par exemple, l'ensemble des parties d'un ensemble est une algèbre de boole avec l'intersection des ensembles comme opérateur $\wedge$ et l'union comme opérateur $\vee$; les rôles de 0 et 1 sont joués par $\varnothing$ et la partie pleine ; le complémentaire d'une partie dans l'ensemble plein correspond à l'opérateur unaire '.

Un autre exemple est celui des treillis. On rappelle ici qu'un treillis est un ensemble ordonné $(E, \leq)$ tel que tout couple (a, b) d'éléments de E posséde une borne supérieure et une borne inférieure dans E notées respectivement sup(a, b) et inf(a, b).

On pose $a \vee b = \sup(a, b)$ et $a \wedge b = \inf(a, b)$,

si le treillis que l'on considère est de plus distributif complémenté c'est à dire si on a les propriétés suivantes :

$$\text{(distributif)}\ \left[\forall (a, b, c) \in E^3 \quad a \wedge (b \vee a) = a\ ;\ a \vee (b \wedge a) = a\ ;\right.$$

$$\text{(complémenté)} \left[\begin{array}{l} \text{(borné)}\ \left[\begin{array}{l}\text{E possède un plus petit et un plus grand élément}\\ \text{notés respectivement 0 et 1}\end{array}\right.\\ \left[\forall\, a \in E\,, \exists\, a' \in E \text{ tel que } a \vee a' = 1\ ;\ a \wedge a' = 0\ ;\right.\end{array}\right.$$

On peut alors montrer que E est une algèbre de Boole.

La notion de treillis est importante dans le cadre de l'étude des ensembles ordonnés. On peut également voir qu'une algèbre de Boole est un treillis distributif complémenté, et même la définir en tant que tel [Serfati73].

A partir de $\mathfrak{B}$ et de ses opérateurs on construit l'opérateur + défini de la façon suivante :

(S0) $\qquad a + b = (a \wedge b') \vee (a' \wedge b).$

Dans l'ensemble des parties d'un ensemble, ce nouvel opérateur correspond à la différence symétrique de deux ensembles.

On peut montrer que relativement à cette "addition" et à l'opérateur $\wedge$ l'ensemble $\mathfrak{B}$ est un anneau booléen.

Pour des commodités d'écriture, lorsque l'on considère $\mathfrak{B}$ comme un anneau booléen on note * l'opérateur $\wedge$ et $\mathfrak{R}(\mathfrak{B})$ l'ensemble $\mathfrak{B}$.

Réciproquement, soit $\mathfrak{R}$ un anneau booléen. $\mathfrak{R}$ est muni de deux opérateurs + et * ainsi que deux éléments particuliers 0 le neutre de l'addition et 1 celui du produit.

A l'aide de ces objets on construit les trois opérateurs $\vee$, $\wedge$ et ' donnés par le système (S1) suivant :

$$a \vee b = a + b + a*b\ ;$$
$$a \wedge b = a*b\ ;$$
$$a' = 1 - a\ .$$

Avec ces opérations de $\mathfrak{R}$ devient une algèbre de Boole que l'on note $\mathfrak{B}(\mathfrak{R})$.

Un ensemble muni d'une structure d'algèbre de Boole peut donc être également muni d'une structure d'anneau de Boole et réciproquement si l'ensemble considéré est muni d'une structure d'anneau de Boole il peut facilement être muni d'une structure d'algèbre de Boole.

Si l'on note $\mathfrak{R}$ la transformation qui à $\mathfrak{B}$ (algèbre de Boole) associe $\mathfrak{R}(\mathfrak{B})$ et $\mathfrak{B}$ celle qui à $\mathfrak{R}$ associe $\mathfrak{B}(\mathfrak{R})$ on a :

L'algèbre de Boole $\mathfrak{B}(\mathfrak{R}(\mathfrak{B}))$ est la même que $\mathfrak{B}$ et l'anneau $\mathfrak{R}(\mathfrak{B}(\mathfrak{R}))$ est le même que $\mathfrak{R}$,

$$\mathfrak{B} \overset{\mathfrak{R}}{\underset{\mathfrak{B}}{\Leftrightarrow}} \mathfrak{R}(\mathfrak{B}) \qquad \mathfrak{R} \overset{\mathfrak{R}}{\underset{\mathfrak{B}}{\Leftrightarrow}} \mathfrak{B}(\mathfrak{R})$$

La transformation qui permet, à partir d'une algèbre de Boole de se placer dans un anneau est dite *transformation de Stone.*

A l'aide du système (S1) on élimine les opérateurs $\vee$, $\wedge$ et ' et l'on introduit les opérateurs + et *. L'addition "+" verifie (S0) et l'algèbre de Boole peut être considérer comme un anneau avec les éléments 0 et 1.

Pour les algèbres booléennes et les anneaux booléens engendrés par des parties finies on a un résultat analogue :

Si $\{x_1,\ldots, x_n\}$ est une partie finie de $\mathfrak{B}$ (algèbre de boole) on note $\mathfrak{B}(x_1,\ldots, x_n)$ l'algèbre engendrée par $\{x_1,\ldots, x_n\}$: c'est la plus petite sous algèbre de $\mathfrak{B}$ contenant $\{x_1,\ldots, x_n\}$.
La transformation de Stone appliquée à $\mathfrak{B}(x_1,\ldots, x_n)$ nous fournit un sous anneau de $\mathfrak{R}$; le plus petit sous anneau de $\mathfrak{R}$ contenant $\{x_1,\ldots, x_n\}$ c'est à dire l'anneau engendré par $\{x_1,\ldots, x_n\}$ que l'on note $\mathfrak{R}(x_1,\ldots, x_n)$.
Avec la transformation inverse appliquée $\mathfrak{R}(x_1,\ldots, x_n)$ on obtient l'algèbre $\mathfrak{B}(x_1,\ldots, x_n)$.

Si on considère $\mathfrak{B}^*$ c'est à dire l'anneau de boole engendré par l'ensemble fini $\{x_1,..x_n\}$ ou encore $\mathfrak{B}$ l'anneau quotient $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n]/ I$ on peut les considérer comme l'algèbre boole engendrée par $\{x_1,\ldots, x_n\}$ avec les opérations introduites par (S1).

Nous allons dans les chapitres qui suivent nous servir de cette transformation et utiliser la notion de base de Gröbner pour résoudre des problèmes de logique simples puis appliquer ces méthodes à la preuve de circuit.

CHAPITRE II

APPLICATION À LA LOGIQUE

I- LE CALCUL PROPOSITIONNEL

I-1 approche syntaxique

Il s'agit ici de la structure de la logique symbolique la plus simple : une *proposition* est un énoncé déclaratif qui peut être vrai ou faux (ou exclusif !) et le calcul propositionel consiste en la manipulation de ces propositions.
Il existe deux propositions particulières qui sont : "vrai" et "faux", que l'on note respectivement $\mathcal{V}$et $\mathcal{F}$.

Par exemple : "le ciel est bleu", "la mer est verte", "il pleut" sont des propositions. Si l'on note x_1 la première proposition, x_2 la deuxième et x_3 la troisième, x_1, x_2, x_3 sont appelés des *atomes*.

A partir de ces atomes on construit des propositions composées.

Par exemple : " "le ciel est bleu" *et* "la mer est verte" ", " " *si* "il pleut" *alors* "le chat est à l'abri" " sont des propositions composées.

De telles propositions se construisent à l'aide des atomes et de plusieurs constructeurs (opérateurs) de bases qui sont les suivants :

$\wedge$: qui correspond au "et",

$\vee$: qui correspond au "ou" inclusif,

$\Rightarrow$: qui correspond à l'implication "si .... alors ....",

$\Leftrightarrow$: qui correspond à l'équivalence "si et seulement si",

' : qui correspond au "non".

Exemple :

Soit x_1 l'atome correspondant à la proposition : " il pleut"
Soit x_2 l'atome correspondant à la proposition : " il fait froid"
Soit x_3 l'atome correspondant à la proposition : " on se baigne"

alors la proposition " " *si* " il pleut" *et* "il fait froid" *alors* "on ne se baigne pas" " correspond à la proposition composée :

$$x_1 \wedge x_2 \Rightarrow x_3'$$

Les atomes et les propositions composées à partir de ces atomes forment ce qu'on appelle des *formes bien formées* ou des *formules.*

De manière plus précise on désigne par *formules du calcul propositionnel* les éléments définis récursivement (à l'aide des constructeurs) par :

i) Un atome est une formule

ii) Si F est une formule alors F' est une formule

iii) Si F et G sont deux formules alors :

$F \wedge G$ est une formule

$F \vee G$ est une formule

$F \Rightarrow G$ est une formule

$F \Leftrightarrow G$ est une formule

On peut remarquer qu'une formule peut devenir très vite compliquée et illisible. Il existe des régles de simplifications qui permettent de réduire le niveau d'imbrication et le nombre de constructeurs qui interviennent dans les formules. On parle alors suivant le cas de *forme normale conjonctive* ou *de forme normale disjonctive.*

Ces régles sont les suivantes (le signe d'égalité désignant la "réécriture") :

pour tout triplet (F, G, H) de formules :

i) $(F \Rightarrow G) = G \vee F'$,

ii) $(F \Leftrightarrow G) = (F \Rightarrow G) \vee (G \Rightarrow F)$,

qui permettent de n'avoir que les opérateurs $\wedge$, $\vee$ et ' ;

iii) $(F')' = F$

iv) $(F \vee G)' = (F' \wedge G')$,

v) $(F \wedge G)' = (F' \vee G')$,

qui sont les lois de Morgan et qui permettent de mettre les négations uniquement devant les atomes ;

vi) $F \vee (G \wedge H) = (F \vee G) \wedge (F \vee H)$ et $F \wedge (G \vee H) = (F \wedge G) \vee (F \wedge H)$,

vii) $F \vee \mathcal{F} = F$; $F \vee \mathcal{V} = \mathcal{V}$ et $F \wedge \mathcal{F} = \mathcal{F}$; $F \wedge \mathcal{V} = F$.

Si l'on applique successivement sur une formule F ces différentes lois, dont on désigne l'ensemble par (R), on peut obtenir :

i) sa forme normale conjonctive c'est à dire que l'on peut l'écrire :

$F = F_1 \wedge \ldots\ldots \wedge F_p$

avec $F_i = a_{i1} \vee \ldots\ldots \vee a_{iq}$ pour tous les indices i de 1 à p et

où pour tout a_{ij} il existe un atome x_k tel que $a_{ij} = x_k$ ou $a_{ij} = x_k'$,

ii) sa forme normale disjonctive c'est à dire que l'on peut l'écrire :

$F = F_1 \vee .....\vee F_p$

avec $F_i = a_{i1} \wedge .....\wedge a_{iq}$ pour tous les indices i de 1 à p et

où pour tout a_{ij} il existe un atome x_k tel que $a_{ij} = x_k$ ou $a_{ij} = x_k'$.

Toute formule peut s'écrire sous forme disjonctive (respectivement conjonctive). En particulier on peut considérer l'ensemble $\mathfrak{B}$ de toutes les formules comme muni de trois opérateurs $\vee$, $\wedge$ et ' et des deux éléments particuliers $\mathcal{F}$ et $\mathcal{V}$. On peut montrer qu'alors [Chang & Lee 70] :

$\mathfrak{B}$ est une algèbre de Boole.

Soit une partie finie de $\mathfrak{B}$ composée d'atomes $\{x_1,..., x_n\}$. On considère la sous algèbre engendrée par ces atomes, que l'on note $\mathfrak{B}(x_1,..., x_n)$, c'est l'ensemble de toutes les formules que l'on peut écrire avec les atomes $x_1,..., x_n$.

On peut alors considérer cet ensemble comme l'anneau de Boole engendré par $(x_1,..., x_n)$ grâce à la transformation de Stone (cf Chapitre I). On construit alors l'opérateur + qui vérifie

(S0) $\qquad F + G = (F \wedge G') \vee (F' \wedge G)$

Ce nouvel opérateur correspond en fait au "ou exclusif" que l'on note parfois $\oplus$.

Si l'on reprend les notations du chapitre I on note $\mathfrak{B}^*$ l'anneau de Boole engendré par $(x_1,..., x_n)$ On a vu que $\mathfrak{B}^*$ est isomorphe à $\mathfrak{B}$ (où $\mathfrak{B} = \mathbb{Z}/2\mathbb{Z}[x_1,..,x_n] / I$ avec I idéal engendré par $x_1^2 + x_1, ..., x_n^2 + x_n$).

On a donc le résultat suivant :

$\mathfrak{B}(x_1,..., x_n)$ est isomorphe à $\mathfrak{B}$ (en tant qu'anneau).

$\mathfrak{B}(x_1,..., x_n)$ et $\mathfrak{B}$ ont le même nombre d'éléments [Serfati73] (2^{2^n}) et sont donc bien isomorphes.

On peut donc lorsque l'on manipule des formules les considérer comme des éléments de $\mathfrak{B}$.

I-2 Approche sémantique

On a vu qu'à une proposition on peut associer une "valeur" : vrai ou faux (ou exclusif), que l'on appelle *valeur de vérité* associée à l'atome qui représente la proposition.

A une formule on peut alors associer une valeur de vérité en fonction des valeurs de vérité des atomes qu'elle contient.

On appelle cette valeur *interprétation de F.*

Si une formule contient n atomes elle aura 2^n interprétations possibles.

Une formule est vraie sous une interprétation si elle est évaluée à vrai ; sinon elle est dite fausse sous cette interprétation.

Les formules sont évaluées en sachant que :

si F et G sont deux formules alors :

i) si F est vraie alors F' est fausse et si F est fausse alors F' est vraie

ii) si F et G sont vraies alors la formule $F \wedge G$ est vraie ; sinon elle est fausse.

iii) si au moins l'une des formules F et G est vraie alors $F \wedge G$ est vraie ; sinon elle est fausse.

iv) la formule $(F \Rightarrow G)$ est fausse si F est vraie et G est fausse ; sinon elle est vraie.

v) la formule $(F \Leftrightarrow G)$ est vraie si F et G sont même valeurs de vérité ; sinon elle est fausse.

On dit que deux formules sont *équivalentes* si leurs valeurs de vérités sont les mêmes quelle que soit l'interprétation choisie pour F et G.

Le signe d'égalité introduit pour construire les formes conjonctive ou disjonctive d'une formule est cohérent avec la notion d'équivalence c'est à dire que :

Quelle que soit la loi du système (R) appliquée à une formule F, on obtient une formule équivalente. En particulier :

toute formule est équivalente à sa forme normale disjonctive ou conjonctive.

On dit d'une formule qu'elle est *valide* si elle est évaluée à vrai sous n'importe quelle interprétation. On l'appelle aussi une *tautologie.*

On dit d'une formule qu'elle est *inconsistante* si elle est fausse quelle que soit l'interprétation choisie.

Pouvoir déterminer si une formule est une tautologie est le problème essentiel du calcul propositionnel. En effet obtenir un tel résultat peut permettre de savoir si une affirmation est vraie, ou encore si sous un certain nombre d'hypothèses une conclusion est vraie.....

I-3 Prouver la validité d'une formule : aperçu des méthodes existantes

La méthode la plus directe pour déterminer si une formule F est ou n'est pas une tautologie est d'explorer toutes les interprétations possibles (2^n pour une formule contenant n variables !). Il en existe d'autres en particulier la recherche d'une forme "normale" associée à F, comme par exemple les formes disjonctives ou conjonctives, ou encore les formes de Lagrange qui sont en fait des formes disjonctives et conjonctives particulières contenant tous les atomes de F. Les méthodes sont nombreuses et sont en général basées sur le théorème de Shannon et sur les principes de résolution et d'unification que nous ne détaillerons pas ici [Chang & Lee70]. Parallèlement à ces méthodes que l'on qualifie de "preuve formelle" ils existent des méthodes statistiques qui peuvent donner une "idée" du résultat. Nous ne nous intéressons ici qu'aux méthodes de preuve formelle.

Une des premières personnes à avoir donné un algorithme de simplification des formules du calcul propositonnel est sans doute W.V.Quine dans les années 1950. L'algorithme contient une étape où l'on rajoute des variables aux fonctions traitées (détermination des implicants premiers). Comme l'auteur le dit lui-même cette étape peut malheureusement devenir très longue [Quine 52].

A partir de cette époque on voit apparaître dans la littérature d'autres méthodes. En particulier dans les années soixante Davis et Putnam [Davis-Putnam60] donne une méthode qui permet de dire d'une formule sous forme normale conjonctive si elle est ou non inconsistante. Il faut également noter les travaux de Dunham, Fridshal et Sward [Dunham & al59] ou encore ceux de Robinson [Robinson65].

Aujourd'hui les algorithmes restent basés sur les mêmes méthodes. Des améliorations ont été faites notamment à l'aide d'un algorithme analogue à celui de Davis et Putnam qui permet de tester la validité d'une formule donnée sous n'importe quelle forme. Il faut également noter la méthode des arbres syntaxiques [Chang & Lee70],[Dunham & al59].

D'autres méthodes existent : elles sont basées sur les systèmes de réécritures. Il est alors important de bien choisir les règles à utiliser. Certains systèmes ne sont pas confluents et en particulier celui donné par les règles d'une algèbre de Boole ne permet pas d'obtenir une forme normale de ce système. Ceci est lié à un problème d'unicité [Hullot80].

Il faut alors noter les travaux de Hsiang, qui utilise la structure d'anneau de Boole (ordonné) plutot que celle d'algèbre de Boole pour manipuler ses fonctions booléennes [Hsiang83].

Nous proposons ici un autre outil. Cet outil n'est pas révolutionnaire puisque qu'il peut être vu comme un système de réecriture, mais il apporte un formalisme nouveau et peut peut-être unifier les différents problèmes soulevés.

II- LE CALCUL PROPOSITIONNEL ET LES BASES DE GRÖBNER

Nous avons vu comment on pouvait considérer l'ensemble des formules du calcul propositionnel comme l'anneau des polynômes booléens. Puisque que l'on travaille dans $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n]$ on peut utiliser un algorithme de base de Gröbner mais pourquoi faire ?

II-1 Déduction et idéal [Chazarain86]

Soit F une formule et soit p son polynôme booléen associé. On a associé à F une valeur de vérité sous une certaine interprétation de ces atomes.

A un atome correspond une variable de $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n]$. A la valeur de l'atome correspond une valeur de la variable. Si la valeur de l'atome est vraie alors la variable correspondante est mise à 1 et sinon à 0. Une interprétation $(a_1,...a_n)$ des atomes $(x_1,...x_n)$ coïncide avec une affectation des variables $(x_1,...x_n)$ à un nuplet composé de 0 ou de 1 suivant les valeurs des a_i. On associe alors à un polynôme sa valeur calculée dans $\mathbb{Z}/2\mathbb{Z}$ en $(a_1,...a_n)$. On a alors :

F est valide si et seulement si p est identiquement égal à 1

F est inconsistant si et seulement si p est identiquement nul.

En effet :

Supposons de F soit valide. Soit $(a_1,...a_n)$ une interprétation de ses atomes alors $F(a_1,...a_n)$ est évaluée à vrai.

La transformation de Stone est telle que :

- $(a \wedge b)$ et $(a * b)$ ont des valeurs cohérentes en ce sens que si $(a \wedge b)$ est vrai sous une interprétation alors (a*b) est mis à 1 et à 0 sinon,
- a' et $(a + 1)$ ont des valeurs cohérentes : si a est faux a' est vrai ce qui correspond bien à dire que si a vaut 0 alors a+1 vaut 1; si a est vrai a' est faux ce qui correspond bien à dire que si a vaut 1 alors a+1 vaut 0.
- il en est de même pour $(a \vee b)$ et $a + b + a*b$ ainsi que $(a \vee b') \wedge (a' \vee b)$ et $a + b + 1$.

En particulier si l'on applique la transformation de Stone à F on obtient son polynôme booléen associé dont la valeur est 1 sous l'affectation des variables données. comme F est vraie pour toute interprétation son polynôme booléen associé est identique à 1.

La réciproque est vraie et de la même façon on montre que F est inconsistant si et seulement si p est identiquement nul.

Si l'on traduit le résultat précédent au niveau des idéaux :

F est valide si et seulement si $1 \in (p)$,

F est inconsistant si et seulement si (p) est l'idéal nul.

où p est un représentant quelconque de la classe de F (comme élément de $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n] / I$) et où (p) désigne l'idéal engendré par p dans $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n] / I$.

Si au lieu de vouloir montrer qu'une formule est valide on veut montrer qu'une formule est vraie sous un certain nombre d'hypothèses on peut utiliser le résultat suivant :

Soient $F_1,...F_p$ et F des formules ; soient $p_1,....p_p$ et q des représentants de chacunes de ces formules on a alors :

$$((F_1, ...,F_p) \Rightarrow F) \text{ si et seulement si } (q+1 \in (p_1 +1, ..., p_p + 1))$$

En effet :

les formules :

$$((F_1, ...,F_p) \Rightarrow F) \quad \text{et} \quad (F \vee (F_1') \vee (F_2') \vee ... \vee (F_p'))$$

sont équivalentes. Le polynôme qui représente la deuxième est dans la même classe que les polynômes qui s'écrivent :

$$q + \sum_{i=1}^{p} m_j(p_j + 1) \qquad \text{avec } m_j \in \mathbb{Z}/2\mathbb{Z}[x_1,..,x_n]$$

On démontre ce dernier résultat par récurrence sur p :

i) il est vrai pour p = 1 puisque si on applique la transformation de Stone à la formule :

$(F \vee F_1')$ on obtient :

$q + (p_1 + 1) + q*(p_1 + 1)$ soit $q + (p_1 + 1) * (q + 1)$.

ii) si le résultat est vrai pour p -1 , qu'en est -il pour p ?

Par la transformation de Stone on obtient (pour l'indice p) :

$$q + \sum_{j=1}^{p-1} m_j (p_j + 1) + (p_p + 1) ((q + 1) + \sum_{j=1}^{p-1} m_j (p_j + 1))$$

que l'on peut aussi écrire

$$q + \sum_{i=1}^{p} m_j(p_j + 1) \qquad \text{avec } m_j \in \mathbb{Z}/2\mathbb{Z}[x_1,..,x_n]$$

De ce résultat et du fait que F est valide si et seulement si $1 \in (p)$, on en déduit le résultat cherché.

On a donc transformé un problème de logique (déduction) en un problème d'appartenance d'un polynôme à un idéal de $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n] / I$. La théorie des Bases de Gröbner qui permet de résoudre de tels problèmes ne s'applique pas directement sur les anneaux quotients.
En revanche on peut utiliser les bases de Gröbner pour obtenir une caractérisation des idéaux de $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n]$.
Pour travailler dans cet anneau on utilise la remarque suivante :

tester si :

$P \in (P_1,...P_p)$, idéal de $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n] / I$,

est équivalent à tester si :

$p \in (p_1,...p_p, x_1^2 + x_1,....x_n^2 + x_n)$ dans $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n]$

où $p, p_1,...p_p$ sont des réprésentants respectifs des classes $P, P_1,...P_p$.

Soit (F) un idéal de $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n] / I$, on le transforme alors en un idéal J = (F, I) de $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_n]$ et l'on calcule G, la base de Gröbner (réduite) associée à J.

G est composée de certains polynômes de I et de polynômes booléens. Pour démontrer qu'un polynôme appartient à (F) on effectue la normalisation de ces polynômes par rapport aux polynômes de G.

II-2 Quelques exemples

Nous présentons ici quelques problèmes simples de logique que l'on peut résoudre à l'aide des bases de Gröbner.

II-2-1 Les coffres ciselés

On considère l'énoncé suivant :

> *Les coffres de la ville de Venise sont gravés par deux familles : Les Cellini et les Bellini.*
> *Chaque fois qu'un Cellini grave un coffre l'inscription qu'il a portée sur le coffre est fausse et chaque fois qu'un Bellini grave un coffre l'inscription portée est vraie.*
> *Les deux familles participèrent à la décoration de paires de coffres dont l'un était d'or et l'autre d'argent. Chaque coffre était alors l'oeuvre d'un seul homme et une paire pouvait être faite par deux hommes de familles différentes.*
> *On a la paire suivante :*
> *sur le coffre d'or est gravée l'inscription : les deux coffres de cette paire furent faits par des membres de la famille Cellini,*
> *sur le coffre d'argent est gravée l'inscription : les deux coffres ne furent pas faits par des membres de la même famille.*
> *Le problème est alors de déterminer qui a gravé les coffres.*

On a alors la modélisation suivante :

Chaque hypothèse est mise sous forme d'une formule à l'aide d'atomes qui désignent des propositions. On obtient alors :

les atomes :

x_1 = "Bellini a fait le coffre d'or"

y_1 = "Cellini a fait le coffre d'or"

x_2 = "Bellini a fait le coffre d'argent"

y_2 = "Cellini a fait le coffre d'argent"

- "chaque coffre est l'oeuvre d'un seul homme" se traduit alors par les deux formules :

i) $(x_1 \wedge y_1') \vee (x_1' \wedge y_1)$

ii) $(x_2 \wedge y_2') \vee (x_2' \wedge y_2)$

- l'inscription sur le coffre d'or se traduit par :

$y_1 \wedge y_2$

- l'inscription sur le coffre d'argent se traduit par :

$((x_1 \wedge x_2) \vee (y_1 \wedge y_2))'$

- "si le coffre d'or a été ciselé par un Bellini alors l'inscription qu'il porte est vraie" se traduit par :

iii) $x_1 \Rightarrow (y_1 \wedge y_2)$

- "si le coffre d'or a été ciselé par un Cellini alors l'inscription qu'il porte est fausse" se traduit par :

iv) $y_1 \Rightarrow (y_1 \wedge y_2)'$

- de façon analogue on obtient pour le coffre d'argent les implications :

v) $x_2 \Rightarrow ((x_1 \wedge x_2) \vee (y_1 \wedge y_2))'$,

vi) $y_2 \Rightarrow ((x_1 \wedge x_2) \vee (y_1 \wedge y_2))$

Les formules i) ii) iii) v) iv) et vi) donnent par la transformation de Stone les polynômes suivants :

$x_1 + y_1$; $x_2 + y_2$;

$x_1y_1y_2 + y_1y_2 + 1$;

$x_1x_2y_2 + y_2 + x_1x_2$;

y_1y_2 ;

$x_2y_1y_2 + x_2 + y_1y_2 + 1$.

Pour connaître les familles qui ont ciselé les coffres on cherche

quels polynômes parmi

$$x_1 + 1;\ y_1 + 1;\ x_2 + 1\ ;\ y_2 + 1$$

sont dans l'idéal engendré par :

$$x_1 + y_1 + 1\ ;\ x_2 + y_2 + 1\ ;\ x_1y_1y_2 + y_1y_2\ ;$$
$$x_1x_2y_2 + y_2 + x_1x_2 + 1\ ;\ x_2y_1y_2 + x_2 + y_1y_2\ ;\ y_1y_2 + 1;$$
$$x_1^2 + x_1\ ;\ x_2^2 + x_2\ ;\ y_1^2 + y_1\ ;\ y_2^2 + y_2$$

On calcule alors la base de Gröbner associée aux polynômes précédents, on obtient :

$$G = \{\ x_1 + 1,\ y_1,\ y_2 + 1,\ x_2\ \}.$$

ce que l'on peut interpréter en disant :

Bellini a gravé le coffre d'or (puisque $x_1 + 1$ est dans G)

Cellini a gravé le coffre d'argent (car $y_2 + 1$ est dans G)

II-2-2 Les purs, les pires et les loup-garous

On considère l'énoncé suivant :

> *l'action se deroule dans une île où les habitants sont soit des purs soit des pires. Les pires mentent toujours et les purs disent toujours la vérité. De plus certains habitants de l'île sont des loup-garous. Un loup-garou peut être indifféremment un pire ou un pur. Vous êtes prisonnier de trois habitants de l'île (A, B, C) dont vous savez qu'un seul est un loup-garou. Ils vous disent :*
> *A : "Je suis un loup-garou"*
> *B : "Je suis un loup-garou"*
> *C : "Un au plus de nous trois est un pur".*
>
> *Si vous devinez qui est qui, ils vous laissent la vie sauve. Que faites vous ?*

A votre place je modéliserais le problème et utiliserais les bases de Gröbner.

On introduit les atomes suivants :

x_1 = "A est un pur"

x_2 = "B est un pur"

x_3 = "C est un pur"

y_1 = "A est un pire"

y_2 = "B est un pire"

y_3 = "C est un pire"

z_1 = "A est un loup-garou"

z_2 = "B est un loup-garou"

z_3 = "C est un loup-garou"

On applique directement la transformation de Stone pour associer à chaque hypothèse un polynôme booléen :

" A, B,C sont soit des purs soit des pires" se traduit par les trois polynômes :

$$x_1 + y_1 \; ; \; x_2 + y_2 \; ; \; x_3 + y_3$$

"il n'y a qu'un seul loup-garou" se traduit par :

$$z_1z_2z_3 + z_1 + z_2 + z_3$$

"si A est un pur alors son affirmation est vraie" se traduit par :

$$x_1 \, z_1 + x_1 + 1$$

"si A est un pire alors son affirmation est fausse" se traduit par :

$$y_1 \, z_1 + 1$$

"si B est un pur alors son affirmation est vraie" se traduit par :

$$x_2 \, z_2 + x_2 + 1$$

"si B est un pire alors son affirmation est fausse" se traduit par :

$$y_2 \, z_2 + 1$$

l'affirmation de C se traduit par :

$$x_1x_2 + x_2x_3 + x_1x_3 + 1$$

"si C est un pur alors son affirmation est vraie" se traduit par :

$$x_1x_2x_3 + x_2x_3 + x_1x_3 + 1$$

"si C est un pire alors son affirmation est fausse" se traduit par :

$$x_1x_2y_3 + x_2x_3y_3 + x_1x_3y_3 + y_3 + 1$$

On calcule alors la base de Gröbner G associée aux polynômes :

$$x_1 + y_1 + 1 \; ; \; x_2 + y_2 + 1 \; ; \; x_3 + y_3 + 1 \; ;$$
$$z_1z_2z_3 + z_1 + z_2 + z_3 + 1 \; ; x_1 \, z_1 + x_1 \; ;$$
$$y_1 \, z_1 ; \; x_2 \, z_2 + x_2 \; ; \; y_2 \, z_2 \; ; \; x_1x_2 + x_2x_3 + x_1x_3 \; ;$$
$$x_1x_2x_3 + x_2x_3 + x_1x_3 \; ; \; x_1x_2y_3 + x_2x_3y_3 + x_1x_3y_3 + y_3 \; ;$$
$$x_1^2 + x_1 \; ; \; x_2^2 + x_2 \; ; \; x_3^2 + x_3 \; ; \; y_1^2 + y_1 \; ; \; y_2^2 + y_2 \; ;$$
$$y_3^2 + y_3 \; ; \; z_1^2 + z_1 \; ; \; z_2^2 + z_2 \; ; \; z_3^2 + z_3$$

$$G = \{ \, x_1, \, x_2, \, x_3 + 1, \, y_1 + 1, \, y_2 + 1, \, y_3, \, z_1, \, z_2, \, z_3 + 1 \}$$

Puis on détermine la nature de A, de B et de C en testant l'appartenance de différents polynômes à l'idéal engendré par G. Ici il est clair que les polynômes $x_3 + 1$, $y_1 + 1$, $y_2 + 1$, $z_3 + 1$ appartiennent à cet idéal ; ce que l'on peut traduire en disant que :

A est un pire, B est un pire, C est un pur et le loup-garou.

CHAPITRE III

APPLICATION A LA PREUVE FORMELLE DE CIRCUITS

INTRODUCTION

Nous venons de voir dans le chapitre précédent comment la validité de certaines implications logiques peut s'exprimer par un test d'appartenance d'un polynôme à un idéal. Pour cela on utilise essentiellement la transformation de Stone qui nous permet de considerer les fonctions booléennes comme des polynômes booléens. Cette démarche, appliquée aux circuits, va nous permettre de "prouver formellement" les circuits combinatoires. Nous allons tout d'abord expliquer en quelques mots en quoi consiste la preuve formelle de circuits combinatoires, et ce que peuvent apporter les bases de Gröbner dans ce domaine.

Il existe d'autres méthodes pour prouver formellement un circuit. La comparaison entre la méthode proposée ici et les méthodes préexistantes est difficile, car il n'existe pas de résultats satisfaisants sur la complexité des algorithmes de calcul de bases de Gröbner, que ce soit dans un cadre général (cf première partie) où dans le cas particulier des bases de Gröbner booléennes (cf le chapitre2 de la présente partie). En pratique on est bien loin de la complexité théorique, ce qui permet de penser que l'utilisation des bases de Gröbner booléennes pour "prouver" un circuit, n'est pas aussi innapropriée que l'on pourrait le croire. De plus, l'algorithme de calcul de bases de Gröbner fournit toujours une solution aux problèmes de preuve de circuit, ce qui n'est pas toujours le cas : il existe des méthodes de compilations canoniques qui n'aboutissent pas [Hullot80].

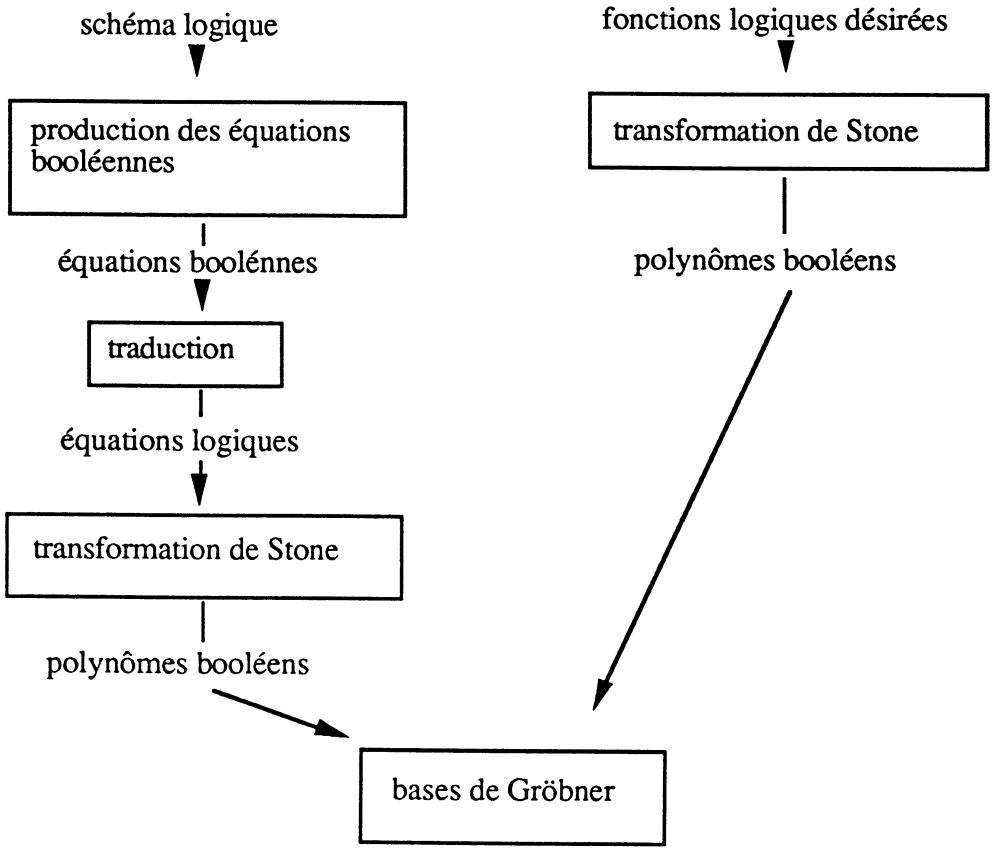
I- LA PREUVE DE CIRCUIT. DEFINITION

Soyons puriste ! l'expression "preuve de circuit" ne veut rien dire.... à moins que l'on m'ait toujours caché qu'un circuit était un théorème !!

Il s'agit en fait d'un abus de langage : on cherche à prouver qu'un circuit donné (par son schéma logique, par exemple) effectue correctement la ou les fonctions qui lui sont demandées. Pour cela on procède de la façon suivante :

> On suppose que l'information qui correspond au circuit combinatoire à prouver est donnée sous forme d'un schéma logique. A partir de ce schéma on extrait, avec un logiciel approprié, les fonctions booléennes représentant les sorties du circuit en fonction de ses entrées. On représente les fonctions censées être exécutées par le circuit comme des fonctions booléennes. Prouver le circuit consiste alors à vérifier que ces fonctions sont les mêmes que celles issues du schéma logique. Il s'agit d'un problème d'égalité de fonctions booléennes qui est en général résolu à l'aide de la forme canonique de telles fonctions, et qui reste toujours difficile (cf chapitreII partie I-3).

Le mécanisme de preuve de circuit proposé ici est le suivant :



La production des équations booléennes à partir du schéma logique et l'obtention par la transformations de Stone des polynômes booléens a fait l'objet d'un projet de troisième année [Hariz88].

Par une méthode analogue on peut "prouver" l'équivalence de deux circuits.

II- UNE PREMIERE APPROCHE. EXEMPLES

II-1 Circuits combinatoires à une sortie

Soit C un circuit. Soient $(a_1,...a_n)$ les entrées de C et b sa sortie. On décrit le circuit C à partir de son schéma logique en introduisant des variables intermédiaires $u_1,...u_p$, définies par le système (S) d'équations :

$$\begin{bmatrix} u_1 = A_1(a_1, ..., a_n) \\ u_2 = A_2(u_1, a_1, ..., a_n) \\ ..... \\ u_p = A_p(u_1, ..., u_{p-1}, a_1, ..., a_n) \end{bmatrix}$$

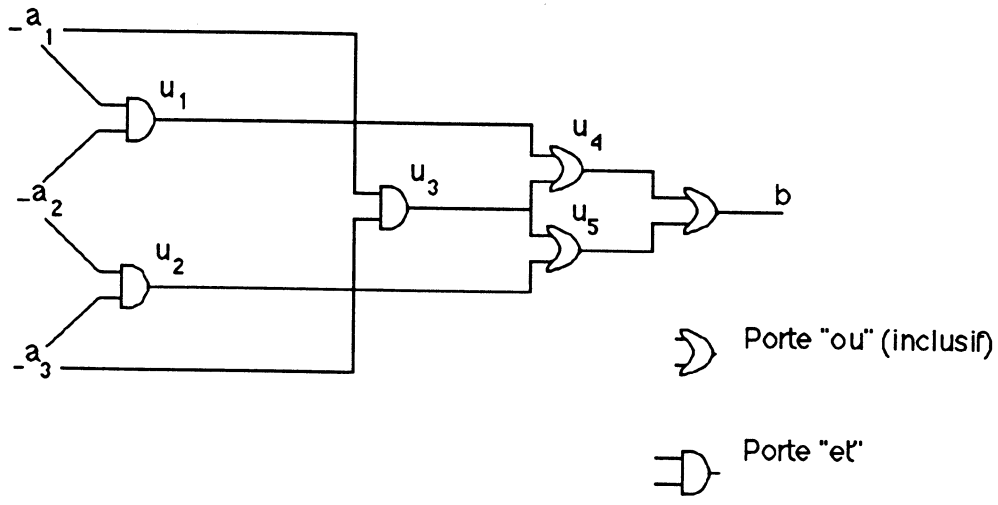
b s'écrit en fonction des variables a_i et u_i :

$$b = F(a_1,....., a_n,u_1, ...u_p).$$

On note $\mathfrak{B}(a_1,....., a_n, u_1,...u_p)$ l'algèbre de boole engendrée par $a_1,....., a_n, u_1,...u_p$. Les fonctions A_i et F sont à valeurs dans $\mathfrak{B}(a_1,....., a_n, u_1,...u_p)$: ce sont des fonctions booléennes.

Exemple :

soit le circuit représenté par le schéma logique suivant :



ce schéma nous fournit les équations :

$$\begin{bmatrix} u_1 = a_1a_2 ; \\ u_2 = a_2a_3 ; \\ u_3 = a_1a_3 ; \\ u_4 = u_1 \vee u_3 ; \\ u_5 = u_3 \vee u_2 ; \end{bmatrix}$$

avec les notations sur les opérateurs logiques introduites dans le chapitre précédent.

Soit G la fonction que le circuit doit réaliser. Pour prouver le circuit, il s'agit de prouver que :

$$F(a_1,....., a_n,u_1, ...u_p) = G(a_1,....., a_n)$$

Dans l'exemple qui précède :

$$G(a_1, a_2, a_3) = a_1a_2 \vee a_2a_3 \vee a_1a_3 ;$$

Pour prouver le circuit donné on cherche à montrer que :

$$a_1a_2 \vee a_2a_3 \vee a_1a_3 = u_4 \vee u_5$$

sachant que

$$\begin{bmatrix} u_1 = a_1a_2 ; \\ u_2 = a_2a_3 ; \\ u_3 = a_1a_3 ; \\ u_4 = u_1 \vee u_3 ; \\ u_5 = u_3 \vee u_2 ; \end{bmatrix}$$

Le système (S) représentant le circuit peut s'écrire sous forme d'un système de polynômes booléens, en appliquant la transformation de Stone aux fonctions A_i. Soient P_i les polynômes booléens associés aux fonctions A_i (S) s'écrit :

$$\begin{bmatrix} u_1 + P_1(a_1, ..., a_n) = 0 ; \\ u_2 + P_2(u_1, a_1, ..., a_n) = 0 ; \\ ..... \\ u_p + P_p(u_1, ..., u_{p-1}\ a_1, ..., a_n) = 0 ; \end{bmatrix}$$

Si l'on associe à F et G leurs polynômes booléens déterminés par la transformation de Stone $\underline{F}$ et $\underline{G}$ il s'agit alors de prouver que :

$$\underline{F}(a_1,....., a_n,u_1, ...u_p) = \underline{G}(a_1,....., a_n)$$

sachant que les u_i vérifient le système (S)

On note un tel problème ($\mathcal{P}$).

Soit I l'idéal engendré par $a_1^2 + a_1,....u_p^2 + u_p$ dans $\mathbb{Z}/2\mathbb{Z}[a_1,..,u_p]$. Grâce aux résultats des chapitres précédents et en posant $a_1 = x_1$, ..., $a_n = x_n$,..., $u_p = x_k$, on sait que :

$$\mathfrak{B}(a_1,....., a_n, u_1,...u_p) \text{ et } \mathbb{Z}/2\mathbb{Z}[x_1,..,x_k] / I \text{ sont isomorphes.}$$

Prouver un circuit peut être considéré comme un problème d'appartenance d'un polynôme à un idéal du quotient $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_k] / I$, puisque :

résoudre le problème ($\mathcal{P}$)
est équivalent à montrer que :
$\underline{F} + \underline{G} \in \mathcal{J}$
où $\mathcal{J}$ est l'idéal engendré par les polynômes de (S). Soit $\mathcal{J} = (Q_1,...Q_p)$ avec

$$\begin{bmatrix} Q_1 = u_1 + P_1(a_1, ..., a_n) \ ; \\ Q_2 = u_2 + P_2(u_1, a_1, ..., a_n) \ ; \\ ..... \\ Q_p = u_p + P_p(u_1, ..., u_{p-1}\ a_1, ..., a_n) \ ; \end{bmatrix}$$

En effet en terme de logique propositionnel pour résoudre ($\mathcal{P}$) il s'agit de prouver la validité d'une implication : le système (S) correspond à un ensemble d'hypothèses et il faut alors prouver que la conclusion correspondant à $\underline{F} + \underline{G} = 0$ est vraie. Or prouver la validité d'une implication est équivalent à prouver qu'un polynôme appartient à un idéal (cf chapitre2).

Nous avons déjà vu que la théorie des bases de Gröbner qui permet de résoudre des problèmes de test d'appartenance d'un polynôme à un idéal ne s'applique pas directement sur les anneaux quotients mais que l'on a le résultat suivant :

tester si :

$P \in (P_1,...P_p)$, idéal de $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_k] / I$,

est équivalent à tester si :

$p \in (p_1,...p_p, x_1^2 + x_1,....x_n^2 + x_n)$ dans $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_k]$

où $p, p_1,...p_p$ sont des réprésentants respectifs des classes $P, P_1,...P_p$.

On procède alors comme on l'a fait dans le cas du calcul propopsitionnel :

Soit (P) un idéal de $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_k] / I$, on le transforme alors en un idéal $J = (P, I)$ de et l'on calcule G, la base de Gröbner (réduite) associée à J.

Dans l'exemple il s'agit de montrer que :

$$P = u_4u_5 + u_5 + u_4 + a_1a_2 + a_2a_3 + a_1a_3$$

appartient à l'idéal engendré par :

$$P_1 = u_1 + a_1a_2 \ ; \ P_2 = u_2 + a_3a_2 \ ;$$
$$P_3 = u_3 + a_3a_1 \ ; \ P_4 = u_4 + u_1u_3 + u_1 + u_3 \ ;$$
$$P_5 = u_5 + u_2u_3 + u_2 + u_3 \ ;$$

Les polynômes P_1, P_2, P_3, P_4, P_5 et ceux qui engendrent I forment un base de Gröbner (non minimale) de l'idéal qui leur est associé dans $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_k]$ et grâce à la normalisation de P par ces polynômes on obtient :

$P = u_1u_2u_3 + u_3u_2 + u_3u_1 + u_3 + u_2u_1 + u_2 + u_1 + a_1a_2 + a_2a_3 + a_1a_3$

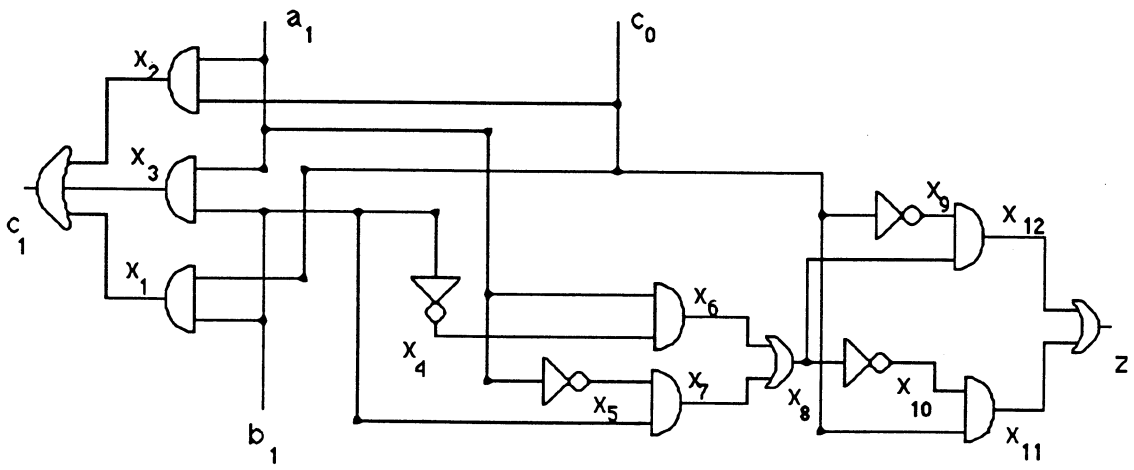
en normalisant P avec P_4 et P_5.

P = 0 en le normalisant par P_1, P_2, P_3.

II-2 Circuits combinatoires à plusieurs sorties

•• Jusqu'ici, les cicuit combinatoires considérés n'avait qu'une sortie. Il est simple de généraliser ce processus lorsque que l'on a plusieurs sorties indépendantes : il s'agit alors de démontrer que plusieurs polynômes (un par sortie) appartiennent à l'idéal engendré par les polynômes provenant que la description du circuit.

Exemple : (cellule de base d'un additionneur)



Les sorties de ce circuit s'écrivent en fonction des entrées (a_1, b_1, et c_0), et des variables intermédiaires :

$c_1 = x_1x_2x_3 + x_3x_2 + x_3x_1 + x_3 + x_2x_1 + x_2 + x_1$;

$z = x_{12}x_{11} + x_{12} + x_{11}$;

Les fonctions supposées exécutées par le circuit sont :

$c_1 = a_1b_1 + a_1c_0 + b_1c_0$;

$z = a_1 + b_1 + c_0$;

Il s'agit alors de vérifier si les polynômes :

$x_1x_2x_3 + x_3x_2 + x_3x_1 + x_3 + x_2x_1 + x_2 + x_1 + a_1b_1 + a_1c_0 + b_1c_0$;

$x_{12}x_{11} + x_{12} + x_{11} + a_1 + b_1 + c_0$;

appartiennent à l'idéal engendré par :

$x_1 + b_1c_0$; $x_2 + a_1c_0$; $x_3 + a_1b_1$; $x_4 + b_1 + 1$; $x_5 + a_1 + 1$;

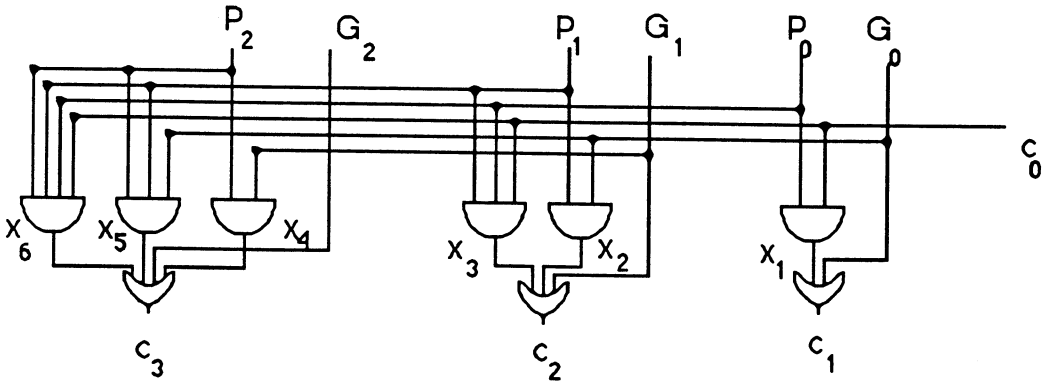
$x_6 + a_1x_4$; $x_7 + b_1x_5$; $x_8 + x_6x_7 + x_7 + x_6$; $x_9 + c_0 + 1$;

$x_{10} + x_8 + 1$; $x_{11} + c_0x_{10}$; $x_{12} + x_9x_8$;

•• Le circuit peut également avoir ses sorties liées les unes aux autres. Il est possible que les fonctions que doit réaliser le circuit soient écrites de telle sorte que la $i^{ième}$ sortie s'exprime en fonction des précédentes.

Soit par exemple :

(additeur Carry look ahead)



Le système (S) fournit par le schéma est le suivant :

$$\begin{bmatrix} x_1 + P_0c_0 = 0 \\ x_2 + P_1G_0 = 0 \\ x_3 + P_1P_0c_0 = 0 \\ x_4 + P_2G_1 = 0 \\ x_5 + P_1P_2G_0 = 0 \\ x_6 + P_1P_2P_0c_0 = 0 \end{bmatrix}$$

Les sorties, en fonction des entrées s'écrivent alors :

$c_1 = x_1G_0 + G_0 + x_1$;

$c_2 = x_2x_3G_1 + x_2G_1 + x_3G_1 + x_3x_2 + x_3 + x_2 + G_1$;

$c_3 = x_4x_5x_6G_2 + x_4x_5x_6 + x_6x_4G2 + x_4x_5G_2 + x_5x_6 + x_5G_2 + x_4x_5 + x_5 + x_4 + x_6G_2 + x_6 + G_2$;

Le circuit est censé calculer :

$c_1 = G_0P_0c_0 + P_0c_0 + G_0$;

$c_2 = G_1P_1c_1 + P_1c_1 + G_1$;

$c_3 = G_2P_2c_2 + P_2c_2 + G_2$;

Il s'agit de montrer que :

i) $x_1G_1 + x_1 + G_0P_0c_0 + P_0c_0 = 0$; sachant que les x_i vérifient le système (S)

ii) $x_2x_3G_1 + x_2G_1 + x_3G_1 + x_3x_2 + x_3 + x_2 + G_1P_1c_1 + P_1c_1 = 0$; sachant que

- les x_i vérifient le système (S),
- $c_1 = G_0P_0c_0 + P_0c_0 + G_0$,

iii) $x_4x_5x_6G_2 + x_4x_5x_6 + x_6x_4G_2 + x_4x_5G_2 + x_5x_6 + x_5G_2 + x_4x_5 + x_5 + x_4 + x_6G_2 + x_6 + G_2P_2c_2 + P_2c_2 = 0$; sachant que

- x_i vérifient le système (S),
- $c_1 = G_0P_0c_0 + P_0c_0 + G_0$
- $c_2 = G_1P_1c_1 + P_1c_1 + G_1$

Si l'un de ces trois points est faux alors le circuit ne calcule pas les fonctions boolénnes escomptées.

Pour chacun des trois points on utilise la méthode donnée précédemment. pour les circuits à sorties indépendantes. On modifie à chaque fois le système (S) en lui ajoutant le ou les polynômes adéquates.

Remarque :

dans l'exemple donné, si l'on travaille dans $\mathbb{Z}/2\mathbb{Z}[x_1,..,x_k]$ / I directement comme il est expliqué dans la troisième partie avec les variables rangées correctement, on peut montrer que les polynômes ajoutés au système (S) ont leurs monômes de têtes étrangers à ceux des polynômes de (S). Ceci est en particulier dû au fait que le polynôme que l'on ajoute à chaque fois est un polynôme où les variables x_i n'interviennent pas et où les variables d'entrées sont indépendantes de celles utilisées par la sortie précédente. A chaque étape l'idéal est représenté par une base de Gröbner (non minimale) ce qui en évite le calcul ; ce n'est malheureusement pas toujours le cas.

De manière plus générale, on note a_i les variables d'entrée du circuit, u_i les variables intermédiaires et c_i les sorties. Les calculs de bases de Gröbner s'effectuent alors $\mathbb{Z}/2\mathbb{Z}[a_1,.., a_n, u_1,..,u_p, c_1,..,c_l]$ (en ajoutant aux polynômes générateurs les polynômes qui engendrent I).

On suppose disposer :

- d'un algorithme de calcul de base de Gröbner noté Grob ;
- d'un procédure de division d'Hironaka qui donne 0 si le polynôme est pas l'idéal et un polynôme non nul sinon. On peut reprendre la procédure NormPF introduite dans la première partie ;
- d'une procédure notée Extract qui à partir du schéma logique du circuit nous fournit l'ensemble des polynômes booléens le décrivant ;
- d'une procédure notée Sortie donnant les polynômes $\underline{G}(i)$ et $\underline{F}(i)$ l'un correspondant à la $i^{ième}$ fonction que doit réaliser le circuit et l'autre correspondant à la description de la $i^{ième}$ sortie écrite en fonction des entrées et des variables intermédiaires.

l'algorithme qui correspond à la preuve d'un circuit C s'écrit :

PR(C) ==

```
ENTREE : un circuit.
SORTIE : la réponse à : le circuit fait-il ce qu'on lui demande ?

    { variables intermédiaires }
    S : une famille de polynômes booléens ;
    P : un polynôme ;
    Fin : une variable booléenne ;
    Rep : une chaine de caractères ;
    { initialisation }
    Fin := Faux ;
    S := Extract ;
    i := 1;
    Sortie ; { initialisation de F(1) et G(1) }
    { boucle sur les sorties }
    tant que ((Fin = Faux) et (i ≤ l))      { l est le nombre de sorties }
        P := F(i) + G(i) ;
        si NormPF(P,S) ≠ 0     Fin = Vrai
        sinon
            P := c_i + G(i) ;
            S := S ∪ P ;
            S := Grob(S) ;
            i := i+1;
            Sortie ; { initialisation de F(i) et G(i) }
    si (Fin = Vrai ) rep :=" le circuit est faux sur la sortie numero i"
    sinon  rep := "le circuit est correct "
    Sortir(rep).
fin de Pr(C).
```

Remarque :

à chaque étape on ajoute un polynôme à une base de Gröbner que l'on calcule à nouveau. On peut adapter, dans ce cas précis, l'algorithme classique de calcul de la base de Gröbner d'un idéal, en utilisant l'algorithme MGBuch2 décrit dans le chapitre2 de la partie3, qui à partir de deux bases de Gröbner en calcule une autre.

Dans l'exemple :

si l'on applique NormPF à

$x_1G_0 + x_1 + G_0P_0c_0 + P_0c_0$ et à la famille de polynômes représentant le circuit.

on obtient 0 car $x_1 + P_0c_0$ fait partie de la base de Gröbner donnée par le système S.

On rajoute au système S l'équation $c_1 = G_0P_0c_0 + P_0c_0 + G_0$. Autrement dit on ajoute aux polynômes donnés au départ, $c_1 + G_0P_0c_0 + P_0c_0 + G_0$. On a encore une base de Gröbner.

si l'on applique NormPF à

$x_2x_3G_1 + x_2G_1 + x_3G_1 + x_3x_2 + x_3 + x_2 + G_1P_1c_1 + P_1c_1$ et à la nouvelle base de Gröbner

on obtient 0 par les polynômes donnant les expressions de x_2, de x_3 et de $G_0P_0c_0$.

On ajoute alors à la base de Gröbner le polynôme $c_2 + G_1P_1c_1 + P_1c_1 + G_1$

La normalisation de

$x_4x_5x_6G_2 + x_4x_5x_6 + x_6x_4G_2 + x_4x_5G_2 + x_5x_6 + x_5G_2 + x_4x_5 + x_5 + x_4 + x_6G_2 + x_6 + G_2P_2c_2 + P_2c_2$ par la dernière base de Gröbner obtenue donne 0.

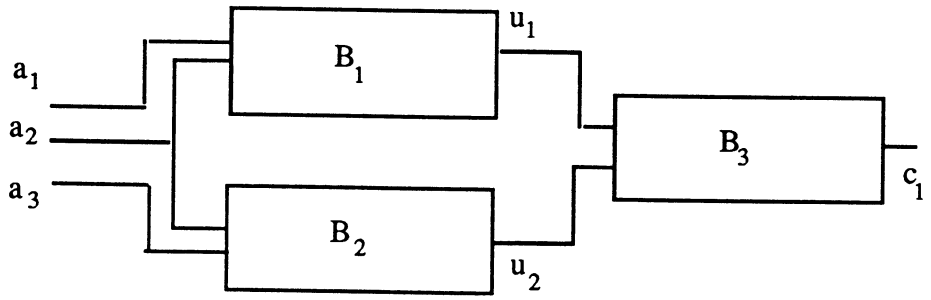
III- UTILISATION DE LA HIERARCHIE

Le principal inconvénient de cette méthode est l'introduction de variables intermédiaires lors de la description du circuit. Dans le premier exemple on en introduit 5 dans le deuxième 11 et dans le troisième 6 et ce ne sont que les "petits" circuits !

Pour éviter l'introduction d'un grand nombre de variables en même temps et pour utiliser le fait que certaines parties du circuit peuvent être déjà connues on adapte le procédé décrit plus haut à une représentation hierarchisé du circuit.

On représente un circuit non plus à partir de son circuit logique (niveau le plus bas) mais par des boites et des interconnexions entres ces boites. Les boites intervenant ici, correspondent à des circuits déjà prouvés.

Soit l'exemple suivant :



Supposons avoir déjà prouver que les boites B_1, B_2 et B_3 représentent trois circuits corrects. Elles correspondent à trois polynômes booléens P_1, P_2 et P_3 qui décrivent leurs sorties en fonction (uniquement) de leur entrées.

On a alors les variables a_1, a_2, a_3 (les entrées du circuit) u_1 et u_2 deux variables intermédiaires et c_1 la sortie.

Pour prouver le circuit, il s'agit alors de prouver que

$c_1 = \underline{G}(a_1, a_2, a_3)$ où $\underline{G}$ est le polynôme booléen correspondant à la fonction qui doit réaliser le circuit, sachant que :

$$u_1 = P_1(a_1, a_2)$$
$$u_2 = P_2(a_2, a_3)$$
$$c_1 = P_3(u_1, u_2)$$

De la même façon que précédemment, résoudre ce problème est équivalent à montrer que :

le polynôme booléen $P_3(u_1, u_2) + \underline{G}(a_1, a_2, a_3)$ est dans l'idéal engendré par

$$u_1 + P_1(a_1, a_2)$$
$$u_2 + P_2(a_2, a_3)$$

On peut facilement généraliser ce procédé à des circuits combinatoires plus compliquées, avec plus de blocs, plus d'entrées et plus de sorties.

Lorsqu' un circuit est donné par son schéma logique on détermine les premières boites. Ces boites sont choisies selon différents critères : soit le choix est arbitraire, soit il est dicté par une connaissance partielle du circuit.

Lorsque le circuit est hierarchisé, à une étape on suppose que les blocs donnés sont déjà prouvés et l'on applique la démarche précédente. Si les blocs ne sont pas prouvés, on réapplique la démarche précédente sur chacun des circuits correspondant aux différentes boites.

Remarque :

> On peut prouver les différents blocs d'un circuit en parallèle, regrouper les résultats pour prouver le circuit total. La notion de hiérarchisation de circuit correspond peut-être à une détermination des tâches indépendantes dans un circuit tout comme la parallélisation d'algorithme correspond souvent (tout dépend du niveau de parallélisation) à un découpage des taches indépendantes de l'algorithme.

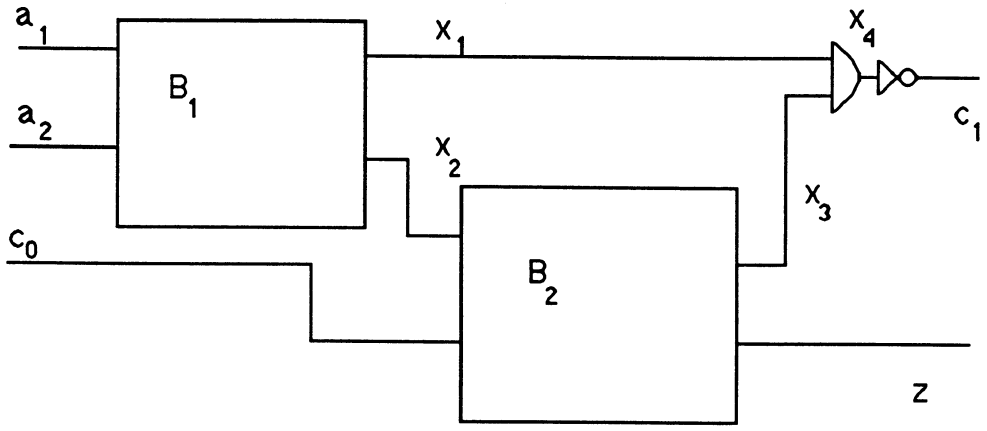
On suppose connaître les procédures suivantes :

> - une procédure notée HIER qui à partir d'un circuit fournit les ensembles de polynômes, chaque polynôme correspondant à une boite prouvée ou non du circuit. Si la boite n'est pas prouvée l'information est donnée par une variable booléenne associée à chaque polynôme (vraie si le circuit correspondant a été prouvé) ;
>
> - les procédures utilisées par l'algorithme Rp donné dans la première partie.

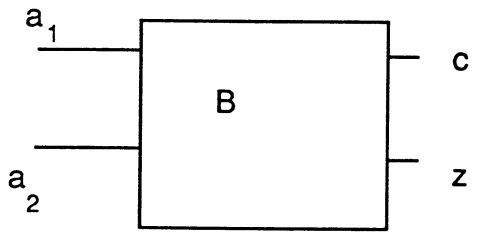
On propose la démarche générale suivante, appliquée à un circuit hiérarchisé :

```
PrHiér(C) ==
    ENTREE : Un circuit hiérarchisé.
    SORTIE : la réponse à : le circuit fait-il ce qu'on lui demande ?
        { variables intermédiaires }
        S : une famille de polynômes booléens ;
        P : un polynôme;
        Fin : une variable booléenne ;
        Rep : une chaine de caractères ;
        { initialisation }
        T : un tableau de booléens ;
        Fin := Faux ;
        S := Hier ; { initialisation de S et de T }
        rep:= " le circuit est correct " ;
        si (T[j] = vrai pour tout j)
            { boucle sur les sorties}
            i := 1;
            Sortie ; { initialisation de F(1) et G(1)}
            tant que ((Fin = Faux) et (i ≤ l))
                { l est le nombre de sorties }
                P := F(i) + G(i) ;
                si NormPF(P,S) ≠ 0      Fin = Vrai
                sinon
                    P := c_i + G(i) ;
                    S := S ∪ P ;
                    S := Grob(S) ;
                    i := i+1;
                    Sortie ; { initialisation de F(i) et G(i) }
            si (Fin = Vrai ) rep :=" le circuit est faux sur la sortie
            numero i de la boite numéro j " ;
            sinon  rep:= " le circuit est correct " ;
        sinon
            Tant que ((il existe j tel que T[j] = Faux) et (rep = "le
                    circuit est correct"))
                soit C le circuit correspond au bloc numéro j ;
                PrHiér(C) ;
                T[j] = Vrai ;
        Sortir(rep).
fin de PrHiér(C).
```

Exemple : (cellule d'addition)



Les boites B1 et B2 sont identiques et sont données par :



Avec les équations :

$$c = a_1.a_2 + 1 ;$$
$$z = a_1 + a_2 ;$$

Si on suppose que ces deux blocs ont été préalablement prouvés, on peut décrire la cellule d'addition par les équations :

$$\begin{cases} x_1 = a_1a_2 + 1; \\ x_2 = a_1 + a_2 ; \\ x_3 = x_2c_0 + 1; \\ x_4 = x_3x_1 ; \\ z = x_2 + c_0 ; \\ c_1 = x_4 + 1; \end{cases}$$

Il s'agit alors de montrer, pour "prouver" le circuit, que :

$$z = a_1 + a_2 + c_0 ;$$
$$c_1 = a_1a_2 + c_0a_2 + c_0a_1 ;$$

On utilise la même méthode que précédemment : on teste si les polynômes

$$x_2 + a_1 + a_2 \quad \text{et} \quad x_4 + a_1a_2 + c_0a_2 + c_0a_1 + 1$$

sont des éléments de l'idéal engendré par :

$x_1 + a_1.a_2 + 1$;

$x_2 + a_1 + a_2$;

$x_3 + x_2c_0 + 1$;

$x_4 + x_3x_1$;

CONCLUSION :

Nous venons de voir comment les bases de Gröbner peuvent fournir un outil pour résoudre des problèmes simples de logique (calcul propositionnel) et de preuve de circuit (combinatoire). Il s'agit ici d'une première approche et il serait intéressant d'approfondir certains points tels que :

Pour la logique :

> quel est l'apport des bases de Gröbner en logique multivaluée ? Sur ce sujet on peut citer les travaux de Chazarain [Chazarain & Riscos88] ou encore ceux de Kapur [Kapur & Narendram85] et ceux de Hiang [Hiang83].

Pour les circuits :

> comment généraliser l'approche utilisant les bases de Gröbner à des circuits non combinatoires, c'est à dire avec des rébouclages des sorties sur les entrées ? Il faudrait sans doute utiliser les travaux précédemment cités. $\mathbb{Z}/2\mathbb{Z}$ ne suffirait peut-être plus, il faudrait peut-être se placer dans $\mathbb{Z}/3\mathbb{Z}$pour caractériser les états d'une variable. Les diverses représentations des circuits peuvent également servir [Bryant86]. Bref il y a encore beaucoup à faire.

L'inconvénient de la méthode donnée ici, se situe au niveau de la taille des données. Pour un additionneur de deux nombres de k bits la réprésentation de Stone fournit des polynômes dont la taille croît exponentiellement avec k. Malheureusement, comme le souligne Hsiang [Hsiang83] ou Hullot [Hullot80], rester dans une algèbre de Boole dans le cas général peut poser des problèmes de confluence. Pour certains circuits, il est peut-être suffisant de se placer dans une algèbre de Boole, mais de manière générale, si l'on veut mettre au point une méthode qui permette de prouver tous les circuits, la représentation la plus appropriée semble être celle de l'anneau de Boole.

Aux problèmes de taille des données, le parallélisme semble être une bonne réponse. Comme on l'a déjà remarqué si le circuit est hierarchisé le parallelisme devient inhérent, il serait intéressant d'étudier les algorithmes dans cette voie.

TROISIEME PARTIE :

PROGRAMMATION

TROISIEME PARTIE : PROGRAMMATION

INTRODUCTION

Nous avons vu sur des exemples simples que le calcul d'une base de Gröbner pouvait faire intervenir une masse impressionnante de calculs et ceci indépendamment de la croissance des coefficients intermédiaires.

Le parallélisme peut être une nouvelle voie pour répondre à ce type de problème. Nous nous attachons, en particulier dans la suite de cette partie, à la parallélisation des algorithmes de calcul de bases de Gröbner. On pourrait d'ailleurs adapter les techniques décrites à toute une classe de problèmes pour lesquels le parallélisme n'est pas naturel.

La mise en place des algorithmes parallèles dépend de l'architecture de la machine utilisée ainsi que du modèle choisi pour la parallélisation. Les divers algorithmes étudiés ont été mis en place sur un hypercube de la série T de Floating Point Systems [FPS] à 32 processeurs. Ils s'inscrivent dans le cadre plus général du projet PAC qui consiste en la réalisation, sur une machine de type MIMD, d'une bibliothèque d'algorithmes de base en arithmétique exacte.

.La machine

Un hypercube est un d.cube binaire dont les sommets sont constitués par les processeurs de la machine et les arêtes par leurs connexions. Cette architecture a été adoptée par de nombreux constructeurs (Hillis avec la connection machine, Telmat avec le super-node). Elle permet notamment la mise en œuvre d'un grand nombre de processeurs tout en gardant un réseau d'interconnexion simple.

Rappelons qu'un d.cube binaire est un graphe orienté dont les 2^d sommets, numérotés de 0 à 2^d-1, sont reliés entre eux deux à deux si en écriture binaire leurs numéros diffèrent exactement d'un bit.
Une caractéristique essentielle d'une telle structure est qu'elle permet d'accéder à d'autres modèles d'interconnexion, tels que l'anneau, le tore....L'anneau, dont nous reparlerons, est obtenu grâce à un cycle qui traverse chaque sommet exactement une fois. De tels cycles sont construits à l'aide de codes de Gray.

.Le modèle d'architecture

Le comportement du FPS T40 est du type MIMD selon la classification de Flynn[Flynn66]. Chaque processeur à une mémoire propre suffisamment importante (1Mo) et communique avec ses voisins par envois de messages. Le protocole de communication est basé sur les rendez-vous entre processeurs synchronisés. Dans notre cas, on suppose que les calculs ne s'effectuent pas en parallèle avec les communications et que les canaux peuvent envoyer et recevoir des données différentes simultanément.

Sur cette machine tous les nœuds contiennent le même programme. Chacun d'eux se reconnaît par son "numéro binaire" et peut alors savoir quelles sont les tâches qui lui sont assignées. Ce mécanisme nous impose de décrire les algorithmes avec un formalisme très particulier. Par exemple, les procédures sont paramétrées par le numéro du processeur concerné, les communications sont traduites par les fonctions ENVOI et RECEPTION dont nous préciserons plus tard les paramètres.

.Les booléens

En l'état actuel de nos connaissances sur le calcul formel et le parallélisme, il nous a paru souhaitable de pouvoir traiter séparément les différents problèmes qui peuvent se poser lors du calcul d'une base de Gröbner.

Dans cet ordre d'idées il serait intéressant dans une première étape de s'affranchir des contraintes liées au grossissement des coefficients ou encore à la multiplicité des

monômes. Ceci peut être facilement réalisé si l'on accepte, par exemple, de travailler avec des polynômes à coefficients 0 ou 1 et tels que $x_i^2=x_i$ pour tout i de 1 à n, c'est à dire si l'on se place dans l'anneau de Boole $\mathbb{B}$, où $\mathbb{B}$ est le quotient de l'anneau $\mathbb{Z}/2\mathbb{Z}[x_1,...,x_n]$ par l'idéal engendré par les polynômes $x_1^2 + x_1$ ..... $x_n^2 + x_n$. (cf 2ième partie).

Malheureusement pour nous il n'existe pas dans cet ensemble de structure d'ordre possédant les propriétés nécessaires à la définition d'une base de Gröbner.
Néanmoins, on montre dans la suite que l'on peut déterminer, à partir de polynômes booléens, une famille particulière elle même booléenne que l'on appelle par abus de langage "base de Gröbner booléenne" et qui compose en partie une base de Gröbner d'un idéal de $\mathbb{Z}/2\mathbb{Z}[x_1,...,x_n]$. Son calcul est effectivemment réalisé dans $\mathbb{B}$ et évite donc les inconvénients mentionnés plus haut.

En outre ceci ne doit pas être perçu comme un exercice de style inutile car de nombreuses applications font appel au calcul de ces "pseudo" bases de Gröbner non standard (cf deuxième partie).
De plus l'adaptation des algorithmes au calcul dans $\mathbb{Q}[x_1,...,x_n]$ ne nécessite que des modifications de codages et d'opérations de bases : le principe des méthodes parallèles reste inchangé.

Parmi les précédentes études liées à la parallélisation des bases de Gröbner ont peut citer essentiellement les travaux de Ponder [Ponder88] et de Melenk et Neun [Melenk & Neun88a]. En ce qui concerne les travaux de Melenk et de Neun il s'agit d'une vectorisation des opérations de base. Les travaux de Ponder n'ont pas vraiment abouti en raison sans doute d'un manque de machines pour tester les diverses méthodes de parallélisation proposées.

La partie qui suit est composée de trois chapitres :

Le premier est consacré à la mise en place de la notion de base de Gröbner booléenne et des structures utilisées dans les algorithmes.

Le deuxième est consacré à différentes techniques de parallélisation, et plus particulierement à la description de ces méthodes.

Le troisième regroupe les résultats des implantations sur le T40.

CHAPITRE I

STRUCTURES DE BASES - L'ALGORITHME SEQUENTIEL

Ce chapitre est consacré aux structures de bases utilisées dans toute la partie concernant la programmation et à l'étude de l'algorithme séquentiel.

I- L'ENSEMBLE $\mathfrak{B}$ ET LES BASES DE GRÖBNER

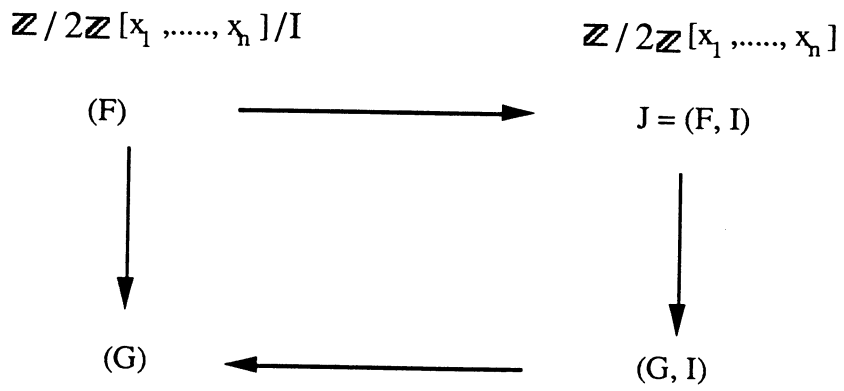
Les objets manipulés sont les polynômes booléens. Le domaine de travail est, pour mémoire, l'anneau quotient $\mathbb{Z}/2\mathbb{Z}[x_1,..., x_n] / I$, où I est l'idéal engendré par les polynômes $x_1^2 + x_1,...., x_n^2 + x_n$. On note $\mathfrak{B}$ cet anneau.

Les bases de Gröbner à calculer sont celles associées à un idéal J de $\mathbb{Z}/2\mathbb{Z}[x_1,.., x_n]$ engendré par une famille F de polynômes de $\mathfrak{B}$ à laquelle sont adjoints les polynômes générateurs de I (cf 2ième partie).

Les éléments de $\mathfrak{B}$ sont de la forme :

$$(\mathcal{F}) \qquad P = \sum m_i \qquad \text{avec } m_i = \prod x_j^{k_j} \quad j \in \{1...n\} \text{ et } k_j \in \{0, 1\}$$

On cherche G telle que (G,I) soit une base de Gröbner de J comme l'indique la figure1 suivante :



avec les notations introduites dans la deuxième partie et où l'on note "chemin1" la flêche qui associe (F) à (G) et "chemin2" les trois autres flêches qui relient (F) à (G) par l'intermédiaire de J et de (G, I).

Ne serait-il pas alors possible de travailler directement avec des objets de la forme ($\mathcal{F}$) en utilisant la règle de simplification suivante :

$$\forall x \in \mathbb{Z}/2\mathbb{Z}[x_1,..., x_n] \qquad x^2 = x$$

En effet si l'on considère la figure1 il semble plus simple de choisir le "chemin 1" plutôt que le "chemin 2" pour obtenir G à partir de F.

On se heurte cependant aux problèmes i) et ii) suivants :

i)

si l'on choisit dans $\mathbb{Z}/2\mathbb{Z}[x_1,..., x_n]$, l'ordre lexicographique :

$$x_1 < x_1^2 < .... < x_2 < x_1 x_2 < x_2^2 < ....$$

celui-ci est transformé dans $\mathbb{B}$ par l'application canonique de $\mathbb{Z}/2\mathbb{Z}[x_1,.., x_n]$ sur $\mathbb{B}$ en :

$$x_1 < x_2 < x_1x_2 < ....$$

Si le premier ordre est compatible avec le produit des monômes, comme le veut la théorie (hypothèse (HO) sur l'ordre chapitre1 partie théorique), ce n'est malheureusement pas le cas pour le second.

Considérons en effet la suite d'inégalités suivantes :

$$x_1 < x_2 < x_1x_2,$$

après multiplication par x_2, on obtient :

$x_1 x_2 < x_2^2 < x_1 x_2$ soit $x_1 x_2 < x_2 < x_1 x_2$.

Ce qui est faux.

En résumé :

> *l'application canonique ne transporte pas les propriétés associées à l'ordre qui sont nécessaires à la définition même d'une base de Gröbner, ce qui rend à priori impossible tout calcul dans $\mathbb{B}$.*

Les principales opérations des différents algorithmes classiques de calcul de bases de Gröbner, sont le calcul du spolynôme associé à deux polynômes et la normalisation d'un polynôme par rapport à un autre. Ce problème d'ordre n'intervient que lorsque l'on multiplie un monôme par un polynôme. Pour l'éviter on réordonne les monômes de tous polynômes issus d'un produit monôme-polynôme avec l'ordre suivant :

$$x_1 < x_2 < x_1x_2 < .....$$

sachant que seule cette opération nécessite ce traitement particulier.

Calculons, par exemple, le spolynôme SP associé aux polynômes P_1 et P_2 suivants :

$$P_1 = x_7x_6 + x_5x_3 + x_5 + x_4$$
$$P_2 = x_5x_6 + x_3x_2$$

SP vaut alors $x_5 P_1 + x_7 P_2$.

Ce calcul fait intervenir le produit monôme-polynôme $x_5\ P_1$ qui vaut :

$$x_5\ (x_7x_6 + x_5x_3 + x_5 + x_4) = x_5x_7x_6 + x_5x_3 + x_5 + x_5x_4$$

et qui nécessite un réarrangement des monômes, puisque avec l'ordre choisi :

$$x_5 < x_5x_4 < x_5x_3$$

soit $\quad x_5\ P_1 = x_5x_7x_6 + x_5x_4 + x_5x_3 + x_5,$

SP vaut alors $\ x_7x_2x_3 + x_5x_4 + x_5x_3 + x_5.$

Les produits qui interviennent dans les algorithmes seront tous suivis d'un réarrangement des monômes des polynômes résultats. Ceci sera fait sans opération coûteuse grâce au codage particulier choisi pour représenter les monômes en mémoire (voir plus loin).

ii)

Pour obtenir une base de Gröbner il est nécessaire, pour chaque polynôme booléen p, de calculer les spolynômes associés à p et aux polynômes $p_i = x_i^2 + x_i$ pour tout les indices i tels que la variable x_i soit présente dans le monôme de tête de p.

En effet :

soit une famille $F = (x_2x_3 + x_1 + x_2)$.

On calcule les spolynômes $Spoly(\ x_2x_3 + x_1 + x_2\ ,\ x_3^2 + x_3) = x_1x_3$

$Spoly(\ x_2x_3 + x_1 + x_2\ ,\ x_2^2 + x_2) = x_3x_2 + x_1x_2 + x_2$.

Le deuxième spolynôme obtenu est normalisé et devient $x_1x_2 + x_1$.

En pratique on ne considère que les polynômes booléens et on calcule ces spolynômes particuliers connaissant l 'opération à effectuer alors : il suffit de multiplier le polynôme p par le monôme x_i.

De cette façon, comme l'illustre la figure1, à une famille F d'éléments de $\mathbb{B}$, qui correspond à la donnée d'un idéal de $\mathbb{B}$, on associe une autre famille G telle que :

i) G est génératrice de l'idéal donné par F,

ii) $G \cup \{\ x_1^2 + x_1,...., x_n^2 + x_n\}$ est une base de Gröbner de l'idéal engendré par $F \cup \{\ x_1^2 + x_1,...., x_n^2 + x_n\ \}$.

Ce dernier résultat se déduit des différents points suivants :

i) $x_1^2 + x_1,...., x_n^2 + x_n$, appartiennent à une base de Gröbner.

ii) le résultat est l'ordre des divisions d'Hironaka intervenant dans l'algorithme (voir 1ième partie). Ici, tout se passe comme si on travaillait dans $\mathbb{Z}/2\mathbb{Z}[x_1,..., x_n]$: la division d'Hironaka par $x_1^2 + x_1,...., x_n^2 + x_n$ sur les polynômes en cours de traitement est activée dès qu'elle est nécessaire.

Par abus de langage nous appelerons G "base de Gröbner booléenne de l'idéal engendré par F".

En pratique le travail s'effectue sur des éléments de $\mathbb{B}$. Ceci permet alors d'avoir des structures plus simples à manipuler et des calculs et une occupation mémoire moindres.

Remarque :

Les familles de polynômes données par les problèmes de preuve de circuits (cf deuxième partie, chapitre3) sont particulières puisque si l'on considère l'ordre induit par les variables $x_1,..x_k$, les monômes de tête des polynômes booléens sont étrangers. En effet ces derniers sont de la forme : $x_l + P(x_{l-1},.., x_1)$.

II- LA REPRESENTATION ET LE CODAGE DES STRUCTURES DE DONNEES ET LEURS OPERATIONS

II-1 La représentation et le codage des monômes

II-1-1 Description

Le monôme, structure de données de base, est représenté par un entier construit de la façon suivante :

le monôme un est représenté par 000...01
le monôme x_j est représenté par 0.....10...01
↑
$j+1^{\text{ième}}$ bit

et plus généralement, on définit l'opération de codage suivante:

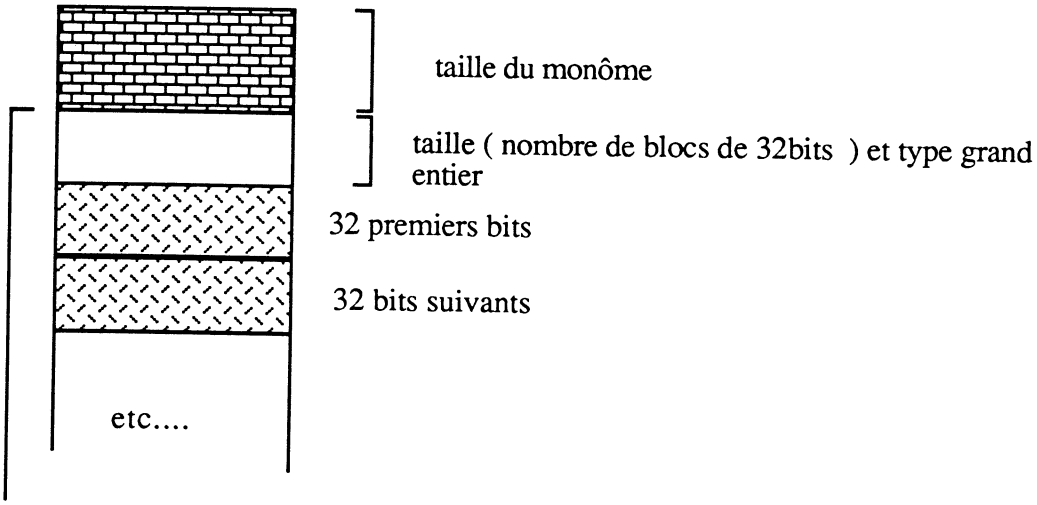
le monôme $\prod x_j^{k_j}$ $j \in \{1...n\}$ et $k_j \in \{0, 1\}$

est représenté par $(\sum k_j 2^j) + 1$

Par exemple le monôme $x_1x_2x_5x_6$ est représenté par :

$1 + 2 + 2^2 + 2^5 + 2^6$ c'est à dire 103.

De cette façon, le nombre de monômes manipulables est limité par la longueur du codage d'un entier dans le langage utilisé (ici 32). Pour éviter cette restriction, nous utilisons une structure analogue à celle mise en place pour les entiers en précision infinie [Roch89]. Un monôme se décrit par :

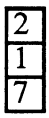


Un monôme est un bloc comprenant :

- un en-tête formé de la taille de ce monôme, c'est à dire du nombre de blocs de 32 bits nécessaires pour sa définition complète.

- une liste formée d'un grand entier. Les variables du monôme sont partitionnées en groupe de 32. Chaque ensemble de 32 variables constitue un bloc de 32 bits du grand entier. Un grand entier est une liste principalement formée d'un premier élément privilégié qui correspond à sa taille (blocs de 32 bits nécessaires à sa définition) et à son type, et d'éléments suivants, qui correspondent chacun à un bloc de 32 bits. Ces blocs de 32 bits sont en fait les éléments de la décomposition du nombre écrit en base 2^{32}.

Par exemple, sans tenir compte des types,



représente le monôme $x_1\, x_2$

représente le monôme $x_1\, x_2\, x_{34}$.

Mettre un en-tête au monôme dans sa représentation, semble superflu, mais donne un moyen pour le differentier d'un grand entier, sans test sur les types.

Ce choix de représentation a été motivé par les remarques suivantes :

il faut pouvoir traduire les opérations de bases de manière simple et efficace. Avec la représentation choisie, le produit de deux monômes correspond à un "ou inclusif" sur les bits (excepté le dernier) des entiers représentant les deux monômes.

on aurait pu coder les monômes autrement avec, par exemple, des nombres pairs, excepté 1 toujours représenté par 1. Mais alors un monôme quelconque m pourrait être représenté à la fois par son propre codage et celui du produit 1.m, ce qui compliquerait les tests d'égalité très fréquents dans la somme puisque $x + x = 0$.

d'autres choix sont possibles comme celui qui consiste à représenter les monômes par des nombres pairs, y compris 1 représenté par 0. Dans ce cas l'ambiguité précédente n'existerait plus, mais 0 serait la représentation 0 et 1, ce qui nécessiterait un artifice supplémentaire pour les différentier.

Dans toute la suite on note C l'application codage choisie. C est alors la bijection de l'ensemble des monômes booléens sur les entiers impairs qui à un monôme m associe son codage C(m).

II-1-2 Codage et opérations

Comme nous venons de le souligner, le choix de la représentation est intimement lié aux opérations de base nécessaires aux algorithmes. Rappelons ici les principales opérations internes sur les monômes et interprétons les sur leurs codages.

On utilise dans ce paragraphe les notations des opérations booléennes introduites dans la deuxième partie. Soient m_1 et m_2 deux monômes quelconques de $\mathfrak{B}$.

. Le Produit de m_1 et m_2

On le note $m_1 * m_2$.

On a alors :

$$C(m_1 * m_2) = C(m_1) \vee C(m_2)$$

Exemple :

$7 \vee 5 = 7$ soit $(x_1x_2) * x_2 = x_1x_2$

. Le "Pgcd" de m_1 et m_2

On le définit et le note de la façon suivante :

$$Pgcd(m_1, m_2) = m_3$$

où m_3 est exactement formé des variables communes à m_1 et m_2, c'est à dire :

$$C(Pgcd(\, m_1, m_2\,)) = C(\, m_1) \wedge C(\, m_2\,).$$

Cette opération permet de déterminer si deux monômes sont étrangers et si un monôme est multiple d'un autre (cf 1ière partie).

Exemple :

$7 \wedge 5 = 5$ soit $Pgcd(\,(\,x_1x_2\,), x_2\,) = x_2$

. La "Div" de m_1 et m_2

On la définit et la note de la façon suivante :

$$Div(m_1, m_2) = m_3$$

où m_3 est exactement constitué des variables de m_1 auxquelles on ote celles de m_2. Pour effectuer cette opération, il est nécessaire d'avoir au moins une variable de m_2 dans m_1.

On a alors :

$$C(\, Div(\, m_1, m_2)\,) = (\,(C(\, m_1) \text{ xor } C(\, m_2)\,) \vee 1) \wedge C(\, m_2)$$

Cette opération est utilisée pour calculer le spolynôme associé à un couple de polynômes et pour effectuer la normalisation d'un polynôme par rapport à un autre.

Exemples :

$(7 \text{ xor } 5) \vee 1) \wedge 7 = 3$ soit $\text{Div}((x_1x_2), x_2) = x_1$

$(15 \text{ xor } 25) \vee 1) \wedge 15 = 7$ soit $\text{Div}((x_1x_2x_3), x_3x_4) = x_2x_1$

. L'ordre dans $\mathbb{B}$

L'ordre choisi sur les monômes booléens est

$$x_1 < x_2 < x_1x_2 < \ldots.$$

Sur les représentations des monômes il devient

$$3 < 5 < 7 < \ldots$$

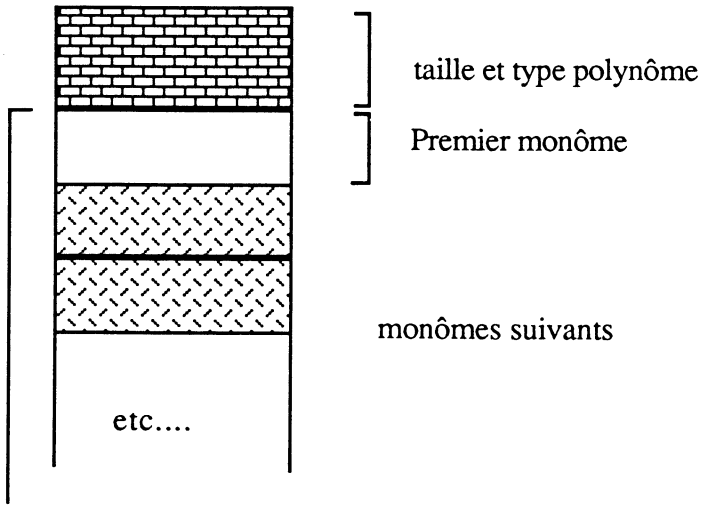
Nous travaillons alors avec l'ordre canonique sur les entiers ce qui offre bien des avantages. Les différents tests sur l'ordre des monômes nécessaires à l'algorithme notamment les réarrangements des monômes d'un polynôme se font très rapidement.

II-2 La représentation et le codage des polynômes

Après avoir mis en place une structure pour les monômes, nous pouvons construire celle qui correspond aux polynômes.

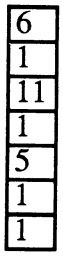
II-2-1 Description

On utilise le même modèle que précédemment. La représentation choisie est alors la suivante :



Un polynôme est alors une liste composée principalement de ses différents monômes rangés par ordre décroissant. Le premier élément de la liste est privilégié, c'est un entier de 32 bits qui représente à la fois la taille de la liste (c'est à dire le nombre d'entiers de 32 bits intervenant) et le type polynôme. Ce type est construit de la même façon que celui des grands entiers : on met à 1 les bits de poids fort dont on est sûr qu'ils ne seront pas affectés par une modification de la taille du polynôme. On peut alors différentier les grands entiers, les monômes et les polynômes.

Par exemple, le polynôme $x_1x_3 + x_2 + 1$ est représenté (sans tenir compte du type) par :



II-2-2 Codage et opérations

Nous allons maintenant répertorier les opérations qui font intervenir les polynômes et les transcrire au niveau du codage.

On a deux types d'opérations : celles dont les opérandes peuvent être ou des monômes ou des polynômes et celles qui ne font intervenir que des polynômes.

On représente ici un polynôme par la liste de ses monômes (supposés distincts) rangés par ordre décroissant.

Soient m_1 un monôme, p_1 et p_2 deux polynômes.

p_1 se représente par $(m_1^1, m_2^1, ..., m_q^1)$
et p_2 par $(m_1^2, m_2^2, ..., m_k^2)$.

On note Réord(p), où p est un polynôme, la fonction qui ordonne les monômes de p par ordre décroissant.

Les opérations du premier type sont les suivantes :

. Le produit monôme-polynôme

C'est le produit standard. On effectue simplement, sur les différents monômes du polynôme à traiter, l'opération produit monôme-monôme, décrite plus haut, et l'on en fait la somme comme décrit plus loin.

On note $m_1 * p_1$ le produit de m_1 par p_1 et on le définit, avec les notations précédentes, par :

$$m_1 * p_1 = m_1 * (m_1^1, m_2^1, ..., m_q^1)$$
$$= \text{Somme}((m_1 * m_1^1, m_1 * m_2^1, ..., m_1 * m_q^1))$$

. La somme monôme-monôme

Il s'agit simplement de la construction, soit d'une liste ordonnée à partir de deux éléments distincts, soit de la liste vide si les deux éléments sont égaux.

. La somme monôme-polynôme

On parcourt la liste des monômes du polynôme à traiter en les comparant au monôme à ajouter. Si ce dernier ne figure pas dans la liste on l'intercale entre deux monômes, de façon compatible avec l'ordre choisi ; sinon on supprime dans la liste l'élément égal au monôme à ajouter.

Les opérations du deuxième type s'écrivent à l'aide de celles déjà données et sont les suivantes :

. La somme de deux polynômes

Il s'agit de réunir deux listes de monômes. On utilise la même démarche que celle utilisée dans la somme monôme-polynôme sans toutefois itérer cette dernière opération sur les monômes d'un des deux polynômes à ajouter. Puisque les listes traitées sont ordonnées, on fusionne l'une dans l'autre en traitant éventuellement les extrémités des listes (en particulier, si il y a égalité de monômes).

On note les différentes sommes de la même façon : a + b, en supposant savoir reconnaître la somme correspondant aux types de a et de b.

. Le calcul du spolynôme associé à deux polynômes booléens

On note cette opération sur les polynômes p_1 et p_2 : $Spoly(p_1, p_2)$.
On la définit et on la décrit avec les opérations précédentes :

$Spoly(p_1, p_2)$ ==
| Si $Pgcd(m_1{}^1, m_1{}^2) \neq 1$ alors
| $Spoly(p_1, p_2) := Div(m_1{}^1, m_1{}^2) * p_2 + Div(m_1{}^2, m_1{}^1) * p_1$;
| sinon $Spoly(p_1, p_2) := 0$;

Fin de Spoly.

Sur l'exemple du I :

$p_1 = x_7x_6 + x_5x_3 + x_5 + x_4$

$p_2 = x_5x_6 + x_3x_2$

$Pgcd(x_7x_6, x_5x_6) = x_6$ et est donc différent de 1

$Div(x_7x_6, x_5x_6) = x_7$

$Div(x_5x_6, x_7x_6) = x_5$

$Spoly(p_1, p_2)$ vaut alors $x_5 * p_1 + x_7 * p_2$.

soit $x_7x_2x_3 + x_5x_4 + x_5x_3 + x_5$.

. Le calcul du spolynôme associé à un polynôme booléen et à un $x_i^2+x_i$

On note cette opération : Spoly(p, i).
Soit m_1 le mônome de tête de p, on a :

```
Spoly(p, i) ==
        | Si Pgcd(m1, xi) ≠ 1 alors
        |     Spoly(p1, i) := xi * p ;
        | sinon Spoly(p1, i) := 0 ;
Fin de Spoly.
```

. La normalisation d'un polynôme par rapport à un autre

On note la normalisation de p_1 par rapport à p_2 : Norm(p_1, p_2).

Elle est basée sur le mécanisme suivant :

- on considère les monômes de p_1. Si l'un d'eux contient le monôme de tête de p_2 c'est a dire si
il existe i de {1, 2, ...,q} tel que Pgcd(m_i^1, m_1^2) = m_1^2
alors
p_1 est remplacé par p_1 + Div(m_i^1, m_1^2) * p_2,

et se décrit par l'algorithme suivant :

```
Norm(p1,p2) ==
        | ENTREE : deux polynômes p1 et p2.
        | SORTIE : p1 normalisé par rapport à p2.
        |     { variables intermédiaires}
        |     q : entier   { longueur de p1 }
        |     i     : indice ;
        |     { Initialisation }
        |     i := 1 ;
        |     q := longueur de p1 ;
        |     tant que (i<= q)
        |         | si (Pgcd(mi1, m12) = m12)
        |         |     | p1 := p1 + Div(mi1, m12) * p2 ;
        |         |     | i := 1 ;
        |         | sinon       i := i +1 ;
        |         | q := longueur de p1
        |     Sortir(p1).
fin de Norm.
```

Sur un exemple :

$p_1 = x_7x_6 + x_5x_3x_2 + x_5x_2 + x_4$
$p_2 = x_5x_2 + x_3x_2 + x_2$

$Pgcd(x_7x_6, x_5x_2) = 1$ et est donc différent de x_5x_2

$Pgcd(x_5x_3x_2, x_5x_2) = x_5x_2$
$Div(x_5x_3x_2, x_5x_2) = x_3$

p_1 devient $p_1 + x_3 * p_2$
soit $p_1 = x_7x_6 + x_5x_2 + x_4$

$Pgcd(x_5x_2, x_5x_2) = x_5x_2$
$Div(x_5x_2, x_5x_2) = 1$

p_1 devient $p_1 + p_2$
soit $p_1 = x_7x_6 + x_4 + x_3x_2 + x_2$

p_1 ne contient plus de monôme contenant x_5x_2 et donc :

$Norm(p_1, p_2) = x_7x_6 + x_4 + x_3x_2 + x_2$

II-3 La représentation et le codage des familles de polynômes

La dernière structure utilisée est celle associée aux familles de polynômes qui est l'objet manipulé le plus complexe.

II-3-1 Description

La structure utilisée est calquée sur celle des polynômes. Une famille de polynôme est une liste composée de polynômes différents. Le premier élément de la liste est un entier correspondant à la dimension de la famille, c'est à dire au nombre de blocs de 32 bits nécessaires à sa définition, les éléments suivants constituent les polynômes de la famille.

II-3-2 Codage et opérations

Au niveau des opérations de bases de l'algorithme séquentiel la structure de famille de polynômes sert à écrire la normalisation d'un polynôme par rapport à une famille, ce qui correspond à une division d'Hironaka.

. La normalisation d'un polynôme par rapport à une famille

Cette opération est faite grâce à la normalisation d'un polynôme par rapport à un autre et à son itération sur les différents polynômes de la famille. L'itération est terminée lorsque le polynôme donné est sous forme normale par rapport à chaque polynôme de la famille.

Soient F = $\{p_1,....p_r\}$ une famille de r polynômes et p un polynôme. Supposons que l'on veuille normaliser p par rapport à F. On note NormPF(p, F) la normalisation de p par F. On procède alors de la façon suivante :

```
NormPF(p, F) ==
|  ENTREE : un polynôme p et une famille de polynômes F.
|  SORTIE : p normalisé par rapport à F.
|       { variables intermédiaires}
|       Préf : polynôme ;
|       i     : indice ;
|       q : entier { longueur de F }
|       { Initialisation }
|       Préf := 0 ; { le polynôme nul }
|       i := 1 ;
|       q := longueur de F ;
|       { itération principale }
|       tant que  (Préf ≠ p)
|            |  Préf := p ;
|            |  i := 1 ;
|            |  tant que ((i<= q)
|            |       |  p := Norm(p,p_i) ;
|            |       |  si (Préf ≠ p)  alors i :=q ;
|            |       |  i := i + 1 ;
|       Sortir(p).
fin de Norm.
```

Après chaque normalisation de p par rapport à un polynôme de F, on peut-être amené à normaliser le résultat par rapport à un polynôme de la famille déja rencontré. Le prédicat préf = p sera vrai si p est sous forme normale par rapport à F.

Par exemple :

Soit $F = \{p_1,....p_3\}$ avec :

$p_1 = x_4x_6x_7 + x_7x_6 + x_4x_3 + 1$

$p_2 = x_5x_6 + x_6$

$p_3 = x_7x_6 + x_5x_6 + x_4$

et $p = x_8x_7 + x_7x_6x_5 + x_7x_6x_4 + x_6x_7 + x_5x_6 + x_6 + x_4x_3$

on itère l'opération de normalisation et p vaut successivement :

$p = \text{Norm}(p, p_1) = x_7x_8 + x_7x_6x_5 + x_6x_5 + x_6 + 1$

$p = \text{Norm}(p,p_2) = x_7x_8 + x_7x_6 + 1$

$p = \text{Norm}(p, p_3) = x_7x_8 + x_6x_5 + x_4 + 1$

$p = \text{Norm}(p, p_1) = x_7x_8 + x_6x_5 + x_4 + 1$

$p = \text{Norm}(p,p_2) = x_7x_8 + x_6 + x_4 + 1.$

Puis p est inchangé par les normalisations successives par rapport à p_1, p_2, p_3.

donc $\text{Norm}(p, F) = x_7x_8 + x_6 + x_4 + 1.$

La structure de famille de polynôme sert également au niveau de la parallélisation d'algorithme (voir plus loin).

III- L'ALGORITHME SEQUENTIEL ET LE CAS BOOLEEN

Dans la partie théorique, nous avons vu comment, pour réduire le nombre d'opérations de l'algorithme, Bucherger applique des critères sur les monômes des polynômes à traiter. Nous ne tenons pas compte du critère1 pour deux raisons :

> la première est que, dans notre cas, le codage des structures de données nous fournit des opérations simples, dont le temps d'exécution est comparable à celui nécessaire pour appliquer le critère précédent.
> la deuxième est que notre but est la parallélisation des algorithmes et que ce critère est difficile à mettre en place sur les algorithmes parallélisés, ne fournit que des résultats partiels si l'on utilise une machine à mémoire distribuée.

Tous les algorithmes décrits ici fournissent, une famille G réduite telle que (G, I) soit une base de Gröbner. Dans les chapitres qui suivent nous dirons encore plus simplement la base de Gröbner. Si nous utilisons une base de Gröbner non réduite, nous le préciserons.

On note Pm(p) la fonction qui associe à un polynôme p son monôme de tête (c'est à dire Exp(p)) et G * F celle qui associe aux ensembles F et G le produit cartésien F x G auquel on ote les couples (f,g) tels que f = g. Cette dernière fonction est dite produit pseudo-cartésien. On note p_i le polynôme $x_i^2 + x_i$ pour tous les indices i et PI l'ensemble qu'ils forment.

Le mécanisme de l'algorithme est le suivant :

> R contient les polynômes de G susceptibles d'être réduits par d'autres polynômes de G. Tant que R n'est pas vide, on en prend les éléments, on les réduits, on les stocke dans P, et on les retire de R.
>
> Une fois R vide, on réunit les éléments de P avec ceux de G dans G et on remet B1 et B2 à jour. Puis pour chaque couple d'éléments de B1 et de B2 on calcule le spolynôme h associé que l'on met dans P.
>
> Enfin, on supprime de G les polynômes susceptibles d'être réduits par h pour les mettre dans R et l'on recommence la réduction sur R.

On appelle GBuch l'algorithme de Buchberger [Bucherger83a], que l'on adapte au cas booléen,

plus précisément :

```
GBuch (F) ==
|  ENTREE : une famille F de polynômes booléens.
|  SORTIE : la base de Gröbner booléenne réduite associée G.
|      { variables intermédiaires }
|      R, G, P : familles de polynômes ;
|      B1, B2 : ensembles de couples de polynômes ;
|      h : polynôme ;
|      { Initialisation }
|      R := F ; P := ∅ ; G := ∅ ;
|      { traitement des entrées : réduction préliminaire }
|      Réd (R, P, G, B1, B2) ;
|      { itération principale }
|      tant que ((B1 ≠ ∅) ou (B2 ≠ ∅))
|          |  Si (B1 ≠ ∅)
|          |      |  Prendre C dans B1 : C := (p1, p2) ;
|          |      |  B1 := B1 - {C} ;
|          |      |  h := NormPF(Spoly(p1, p2), G) ;
|          |  Sinon
|          |      |  Prendre C dans B2 : C := (p, pi) ;
|          |      |  B2 := B2 - {C} ;
|          |      |  h := NormPF(Spoly(p, i), G) ;
|          |  si h ≠ 0
|          |      |  G0:= { g ∈ G / Pgcd(Pm(h), Pm(g)) = Pm(h) } ;
|          |      |  R := G0 ; P:={h} ; G := G - G0 ;
|          |      |  B1 := B1 - {(g1,g2) ∈ B1 / g1∈ G0 ou g2∈ G0};
|          |      |  B2 := B2 - {(g1,pi) ∈ B2 / g1∈ G0};
|          |      |  Réd (R, P, G, B1, B2) ;
|      Sortir(G).
Fin de GBuch.
```

G contient la base de Gröbner des entrées, B1 les paires de polynômes booléens qui n'ont pas encore été traitées, et B2 les paires composées d'un polynôme booléens et d'un p_i. B1 est une matrice creuse, triangulaire inférieure, que l'on stocke ligne par ligne. Informatiquement on créé deux tableaux à une dimension notés C et L tels que :
si $B1_{ij}$ est le $l^{ième}$ élément à stocker on a C[l] = j et L[l] = i.
B2 est un tableau dont chaque élémént correspond à un polynôme booléen p et contient un entier permettant de savoir quelles paires (p, p_i) n'ont pas encore été traitées.

La mise à jour de B1 correspond à des manipulations classiques sur les matrices creuses triangulaires rangées ligne par ligne [Lascaux & Théodor86].

La réduction Réd(R, P, G, B1, B2) est alors définie comme suit :

```
Réd(R, P, G, B1, B2) ==
|   { variables intermédiaires }
|   H, K : familles de polynômes ;
|   tant que R ≠ ∅
|       |  h := un élément de R ; R := R - {h} ;
|       |  h := NormPF(h, P ∪ G) ;
|       |  si h ≠ 0
|       |      |  G0:= { g ∈ G / Pgcd(Pm(h), Pm(g)) = Pm(h) } ;
|       |      |  P0:= { g ∈ P / Pgcd(Pm(h), Pm(g)) = Pm(h) } ;
|       |      |  G := G - G0 ;
|       |      |  P := P - P0 ;
|       |      |  R := R ∪ P0 ∪ G0 ;
|       |      |  B1 := B1 - {(g1,g2) ∈ B1 / g1∈ G0 ou g2∈ G0};
|       |      |  B2 := B2 - {(g1,pi) ∈ B2 / g1∈ G0};
|       |      |  P := P ∪ {h} ;
|   G := G ∪ P ;
|   B1 := B1 ∪ (G * P) ;        { * désigne le produit pseudo-
|                                 cartésien }
|   B2 := B2 ∪ (P * PI) ;
|   H := G ;
|   K := ∅ ;
|   tant que H ≠ ∅
|       |  h := un élément de H ; H := H - {h} ;
|       |  h := NormPF(h, G - {h} ) ;
|       |  K := K ∪ {h} ;
|   G := K.
Fin Réd.
```

On construit un autre algorithme, MGBuch qui diffère de GBuch au niveau de la sélection de G_0 et de P_0. Dans MGBuch, on teste si Pm(h) est contenu dans l'un quelconque des monômes des polynômes de P ou de G (et non plus uniquement dans leur monôme de tête). Ceci permet de simplifier la fonction Réd en supprimant sa dernière boucle. En

revanche celà augmente le temps de calcul au niveau de la construction de P_0 et de G_0. Nous pouvons décrire MGBuch à partir de GBuch en remplaçant :

G_0 par $G_0 := \{ g = g_p + \ldots + g_1 \in G / \exists\ g_i$ tel que $Pgcd(Pm(h), g_i) = Pm(h) \}$

P_0 par $P_0 := \{ g = g_p + \ldots + g_1 \in P / \exists\ g_i$ tel que $Pgcd(Pm(h), g_i) = Pm(h) \}$

où les g_i sont les monômes de g

Réd(R, P, G, B1, B2) par MRéd(R, P, G, B1, B2) avec :

MRéd(R, P, G, B1, B2) ==

```
{ variables intermédiaires }
H : familles de polynômes ;
tant que R ≠ ∅
        h := un élément de R ; R := R - {h} ;
        h := NormPF(h, P ∪ G) ;
        si h ≠ 0
                G0 := { g = gp +....+ g1 ∈ G / ∃ gi tel que
                                Pgcd(Pm(h), gi) = Pm(h) } ;
                P0 := { g = gp +....+ g1 ∈ P / ∃ gi tel que
                                Pgcd(Pm(h), gi)) = Pm(h) } ;
                G := G - G0 ;
                P := P - P0 ;
                R := R ∪ P0 ∪ G0 ;
                B1 := B1 - {(g1,g2) ∈ B1 / g1∈ G0 ou g2∈ G0};
                B2 := B2 - {(g1,pi) ∈ B2 / g1∈ G0};
                P := P ∪ {h} ;
G := G ∪ P ;
B1 := B1 ∪ (G * P) ;     { * désigne de produit
                                pseudo-cartésien }
B2 := B2 ∪ (P * PI) ;
```

Fin MRéd.

Ces deux algorithmes ont été implantés sur les transputers T414 [INMOS] de la machine T40. Leur simulation tend à prouver que le deuxième algorithme est meilleur que le premier au niveau du temps d'exécution. Il semble donc, que le fait d'augmenter le nombre de polynômes à réduire dans R soit compensé par le fait que l'on supprime les normalisations de chaque polynôme h de G par rapport à G - {h}. Les normalisations sont plus nombreuses dans les boucles internes du deuxième algorithme. Ces normalisations sont faites par rapport à $G \cup P$ qui contient alors plus de polynômes, ce que permet de réduire "plus" les polynômes et ce qui diminue globalement le nombre d'opérations.

On peut donner les quelques résultats suivants qui fournissent les temps d'exécution en millisecondes des deux algorithmes :

F	G	GBuch	MGBuch
$x_2x_4 + x_2x_3 + x_1x_2 + x_2$ $x_1x_2x_4x_5 + x_2x_1 + x_1$	$x_1x_3x_5$ $x_2x_1 + x_1$ $x_1x_4 + x_1x_3$ $x_2x_4 + x_2x_3 + x_2 + x_1$	681	679
$x_2x_4 + x_2x_3 + x_1x_2 + x_1$ $x_2x_5x_4 + x_4x_5 + x_1x_4 + 1$ $x_1x_2x_{33} + x_1x_4 + 1$ $x_1x_{33} + x_1x_2$	1	606	673
$x_4x_5x_3 + x_4x_5 + x_2x_4 + x_4$ $x_4 + x_3x_2 + x_1$ $x_4x_6 + x_3x_4x_5 + x_2x_3 + 1$	$x_2x_5 + x_5 + x_2 + 1$ $x_1x_2 + x_2 + x_1 + 1$ $x_1x_3 + x_3 + x_1 + 1$ $x_3x_5 + x_5 + x_2 + 1$ $x_2x_3 + x_3$ $x_3x_6 + x_6 + x_3 + 1$ $x_1x_6 + x_6 + x_1x_5 + x_5$ $x_2x_6 + x_6 + x_2 + 1$ $x_4 + x_3 + x_1$	3580	905

Ces deux mécanismes de réduction sont accompagnés d'une gestion mémoire importante notamment au cours de la construction des deux familles de polynômes G_0 et P_0.
Nous aurions pu simuler leurs constructions afin de ne pas déplacer effectivement les polynômes en mémoire, en gérant des variables booléennes. Ces dernières auraient alors représentées l'appartenance d'un polynôme aux familles à déterminer. Nous avons préféré, pour des raisons de simplicité, programmer les deux méthodes telles qu'elles sont présentées, ce qui comporte une phase de gestion mémoire puisque la structure qui représente une famille de polynômes est dynamique.

L'importance de cette gestion mémoire est mise en évidence par la comparaison de ces méthodes de réduction avec la procédure BRéd suivante :

```
BRéd(R, G, B1, B2) ==
|
|       { variables intermédiaires }
|       n      : nombre de polynômes de R;
|       Créé : une variable boléenne
|       i      : un indice ;
|       créé := vrai;  i := 1 ;
|       tant que ((créé = vrai) ou  (i ≠ n))
|       |       h := un élément de R ; R := R - {h} ;
|       |       h':= NormPF(h, R ∪ G) ;
|       |       si ( (h'=h)
|       |             créé := faux ;
|       |       sinon
|       |             créé :=vrai ;
|       |       R := R ∪ {h'} ;
|       |       i := i+1;
|       G := G ∪ R ;
|       B1 := B1 ∪ (G * R) ;
|       B2 := B2 ∪ (R * PI) ;
fin de BRéd.
```

Dans la comparaison de MRéd et de BRéd on remarque que même si l'on augmente le nombre de réductions dans BRéd, son temps d'exécution global est sensiblement inférieur à celui nécessaire pour exécuter MRéd. Plus précisement on a les résultats expérimentaux suivants :

Pour 8 polynomes à 6 variables :

MRéd : 973 ms BRéd : 444 ms

Pour 16 polynômes à 10 variables :

MRéd : 961 ms BRéd : 551 ms

Pour 32 polynômes à 11 variables :

MRéd : 2906 ms BRéd : 2737 ms

Pour 64 polynômes à 12 variables :

MRéd : 36422 ms BRéd : 12686 ms

Le problème de l'évaluation de la complexité de tels algorithmes reste entier, d'autant plus que le temps d'exécution varie beaucoup d'une méthode à l'autre. Nous venons de le voir, en comparant diverses réductions. Considérons toutefois quelques exemples.

Prenons par exemple, les polynômes à 5 variables :

$$P_1 = x_2x_4 + x_2x_3 + x_1x_2 + x_2$$
$$P_2 = x_1x_2x_4x_5 + x_2x_1 + x_1$$
$$P_3 = x_4x_5 + x_3x_4 + x_2 + 1$$
$$P_4 = x_3x_4x_5 + x_3x_4 + x_3$$
$$P_5 = x_4 + x_3 + x_1x_2.$$

Pour différents jeux de données construits à partir de ces polynômes les temps d'exécution T en millisecondes (ms) de MGBuch sont :

$F = \{P_1, P_2\}$	T = 677 ms
$F = \{P_1, P_2, P_3\}$	T = 1444 ms
$F = \{P_1, P_2, P_5\}$	T = 204 ms
$F = \{P_1, P_2, P_4\}$	T = 1144 ms
$F = \{P_1, P_2, P_3, P_4\}$	T = 907 ms
$F = \{P_1, P_2, P_3, P_5\}$	T = 408 ms
$F = \{P_1, P_2, P_3, P_4, P_5\}$	T = 472 ms.

Faire varier le nombre de polynômes ou le nombre de monômes ne semble pas significatif. Si maintenant on fait varier le nombre de variables, qu'obtient-on ?

Prenons l'exemple suivant :

- à 6 variables :

$$P_6 = x_2x_4 + x_2x_3 + x_1x_2 + x_2$$
$$P_7 = x_1x_2x_4x_5\,x_6 + x_5x_6 + x_4 + x_1x_2 + x_2.$$

Le temps d'exécution en millisecondes de MGBuch avec $F = \{P_6, P_7\}$ est alors 6991 ms ce qui est largement supérieur à 677 ms obtenu pour 5 variables et 2 polynômes.
Ceci provient essentiellement du fait que cet exemple à 6 variables nécessite plusieurs réductions.

- à 7 variables (pris parmi les 64 premières)

$$P_8 = x_4x_{64}x_{17} + x_2x_3 + x_1$$
$$P_9 = x_{17}x_{23} + x_2 + 1$$

Le temps d'exécution en millisecondes de MGBuch avec $F = \{P_8, P_6\}$ est alors 1508ms qui est inférieur au temps obtenu pour 6 variables, ce qui s'explique par le fait que l'exemple à 7 variables nécessite une seule réduction.

Faire varier le nombre de variables n'est donc pas significatif.

Le résultat est indépendant de l'ordre des polynômes qui forment la famille F (cf 1ière partie), on peut donc également s'intéresser à la variation du temps d'exécution en fonction de celui-ci.

Si on considère, par exemple, la famille F formée par P_1, P_2, P_3, P_4, P_5 on obtient un temps d'exécution de MGBuch de 472 ms.
Si l'on permute les polynomes de F et de que l'on exécute MGBuch avec F = {P_1, P_4, P_5, P_3, P_2 }on obtient un temps de 514 ms et avec F = {P_3, P_2, P_4, P_1, P_5 } un temps de 270 ms.

Conclusion :

A partir de ces quelques essais nous pouvons faire les remarques suivantes :

i) l'importance du choix de la réduction :

BRéd correspond à un mécanisme de réduction plus simple que ceux utilisés pour Réd et MRéd. Il utilise peu de gestion mémoire, mais en revanche, il normalise les polynômes de manière systèmatique.
L'appel à la procédure de normalisation d'un polynôme relative à une famille est beaucoup plus fréquent, mais il s'agit ici de polynômes booléens dont la normalisation consiste en des opérations simples et rapides (cf chapitre1)

ii) les temps d'exécutions dépendent :

- de la répartition des variables dans les monômes, ce qui est normal vu la construction des algorithmes.
- de l'ordre d'entrée des données ce qui sera utilisé pour la parallélisation des algorithmes.

Par la suite, de manière plus générale, on appelle RGBuch les algorithmes qui fournissent une base de Gröbner réduite, sans préciser le choix de la réduction.

LISTE DES OPERATIONS INTRODUITES DANS LE CHAPITRE I :

CHAPITRE II

PARALLELISATION

INTRODUCTION

Ce chapitre est consacré à la parallélisation de l'algorithme de Buchberger, RGBuch. Nous avons déjà introduit le modèle d'architecture et la machine, nous présentons ici un cadre pour les algorithmes.

• La répartition des données

La première phase des différents algorithmes décrits dans ce chapitre consiste toujours en la distribution des données.

On répartit les polynômes donnés dans la mémoire locale de chaque processeur. On procède de la façon suivante :

Supposons que l'on ait m polynômes à répartir sur p processeurs. On a :

$$m = qp + r$$

p-r processeurs contiennent k = q polynômes

r processeurs contiennent k = q + 1 polynômes.

Chaque processeur contient alors k polynômes dans sa mémoire locale. Ces k polynômes sont dits polynômes de référence.

Nous avons constaté l'importance du choix de l'ordre des entrées sur le temps d'exécution de l'algorithme séquentiel sans toutefois pouvoir prévoir quel est celui qui le minimise. Pour les algorithmes parallèles, le phénomène se retrouve. Il est évident que la répartition de m polynômes sur p processeurs peut être faite de nombreuses façons. Cependant, lorsque l'on décidera de comparer les temps de calcul quand p varie, on choisira de manière arbitraire une stratégie de répartition que l'on conservera pour toutes les valeurs de p.

• le principe de localité

Un processeur ne peut modifier que les données qu'il possède dans sa mémoire locale. Ceci nous permet de ne perdre aucune information sur les polynômes lors de la normalisation et de la réduction.

• Un mot sur la complexité

Nous avons déjà parlé de la complexité de l'algorithme séquentiel : nous retrouvons les mêmes problèmes au niveau des algorithmes parallèles. Nous tenons toutefois compte dans la parallélisation du coût des communications. Le temps T_n nécessaire pour

communiquer n données de taille unitaire d'un processeur à un autre, contigües en mémoire est :

$$T_n = \beta + n\,\tau$$

c'est à dire la somme d'un temps d'initialisation du canal à occuper (β) et d'un terme proportionnel à la taille (n) des données à transférer [Saad86].

. L'écriture des algorithmes

Nous en avons déjà dit un mot, ajoutons à celui-ci que nous ne présenterons pas en détail la totalité des algorithmes. Nous nous contenterons souvent de les décrire de façon globale, sans donner avec précision l'algorithme effectué par chaque processeur : ceci s'avèrerait sans doute laborieux tant à l'écriture qu'à la lecture.

Certaines notations introduites dans le chapitre précédents sont réutilisées pour les description des algorithmes, notamment le produit pseudo-cartésien des ensembles.

. La parallélisation des algorithmes

Nous présentons dans la suite deux techniques différentes de parallélisation d'algorithmes de calcul de base de Gröbner.

- la première est la plus naturelle, elle consiste à découper l'algorithme choisi en tâches indépendantes pour les traiter en parallèle et utilise un anneau de processeurs pour le transfert des données.
- la deuxième repose sur l'utilisation de la topologie d'hypercube et consiste principalement à calculer des "sous-bases de Gröbner" sur chaque processeur et à regrouper les résultats.

Ces techniques de parallélisation sont valables quelque soit le corps de base sur lequel on travaille. On a choisi dans un premier temps, de ne considérer pour l'implantation de ces méthodes que pour les bases de Grobner booléennes (cf chapitre précédent). Les polynômes et les bases de Grobner que l'on manipule sont booléens ; on ne le precise plus par la suite. Si on ne se restreint plus aux polynômes booléens nous le signalerons.

I- UTILISATION DE L'ANNEAU

Avant de nous intéresser à la parallélisation d'algorithmes du type RGBuch qui fournissent la base de Gröbner réduite associée à une famille de polynômes donnés, nous montrons comment on parallélise un algorithme plus simple (noté SGBuch) qui ne fournit qu'une base de Gröbner non réduite. Cette étude préalable nous permet de mettre en évidence les difficultés que l'on rencontre pour paralléliser RGBuch et de donner des techniques de parallélisation adéquates sur l'anneau. L'algorithme séquentiel associé à cette parallélisation diffèrera alors, par construction, de celui présenté précédemment.

I-1 Pour une base de Gröbner non réduite. Parallélisation de SGBuch (PSGBuch)

Soit l'algorithme séquentiel suivant :
SGBuch (F) ==

```
ENTREE : une famille F de polynômes.
SORTIE : une base de Gröbner non réduite associée G.
    R, P : familles de polynômes ;
    B1, B2: ensembles de couples de polynômes ;
    h : polynôme ;
    R := F ; P := ∅ ;
    B1 := F * F ;
    B2 = F * PI ;
    tant que R ≠ ∅
            { étape de niveau 1 notée TL }
            tant que ((B1 ≠ ∅) ou (B2 ≠ ∅))
                    { étape de niveau 0 notée ITL }
                    Si (B1 ≠ ∅)
                            Prendre C dans B1 : C := (p1, p2) ;
                            B1 := B1 - {C} ;
                            h := NormPF(Spoly(p1, p2), G) ;
                    Sinon
                            Prendre C dans B2 : C := (p, pi) ;
                            B2 := B2 - {C} ;
                            h := NormPF(Spoly(p, i), G) ;
                    Si h ≠ 0  P := P ∪ {h} ;
            B1 := (F * P) ∪ (P * P) ;      B2 := (P * PI) ;
            F := F ∪ P ;
            R := P ;
    Sortir(F) .
Fin de SGbuch.
```

I-1-1 Les grandes lignes d'une parallélisation de SGBuch : TL,"module de base"

La parallélisation de SGBuch (PSGBuch) débute avec la distribution des données, c'est à dire avec la répartition de F en familles de référence entre les différentes mémoires. On rappelle que notre modèle est construit sur les deux hypothèses suivantes :

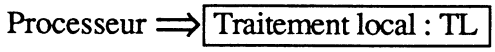
i) les processeurs ne peuvent modifier que leurs propres données,

ii) les calculs et les communications ne se font pas au même instant.

Un processeur ne peut donc effectuer des manipulations que sur ses propres polynômes de référence.

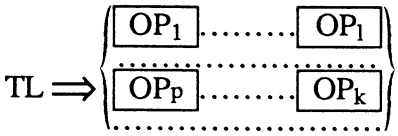
La première phase de calcul de l'algorithme PSGBuch, consiste à activer TL (cf description de SGBuch précédente) dans chaque processeur sur ses polynômes de référence. On a le schéma suivant :

pour un processeur :



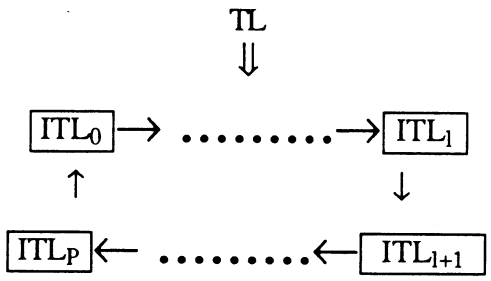
TL sera notre "module de base" c'est à dire la plus petite entité indivisible de l'algorithme, et ceci en raison des remarques suivantes :

On aurait pu en effet imaginer activer ce traitement local (TL) grâce à sa décomposition en opérations indépendantes effectuées en parallèle sur un réseau de processeurs. Soit schématiquement :



Si l'on étudie de plus près TL on remarque que :

TL est principalement construit sur l'itération de ITL dont les opérations sont le calcul de spolynômes et la normalisation. Le $k^{ième}$ appel de ITL noté ITL_k est paramétré par les résultats de ITL_{k-1}. On pourrait avoir un réseau de processeurs effectuant chaque ITL_k en pipelinant les donnés et en se fixant un nombre arbitraire d'appel à ITL, avec par exemple :



ITL deviendrait alors notre "module de base".
A ce stade, décomposer ITL en opérations indépendantes pour avoir dans les processeurs un volume de calcul encore moins important serait possible mais peu intéressant (à moins de vectoriser les opérations de bases !).

En effet la famille $F \cup P$ ne serait pas alors mise à jour dans le processeur où l'on effectuerait le calcul du spolynôme. On devrait alors mettre au point un algorithme dans lequel le temps de calcul serait négligeable par rapport au temps de communication.
De plus nous avons à notre disposition, un nombre fixé de processeurs, ce qui limite d'autant le nombre de spolynômes que l'on peut calculer simultanément .

On pourrait tout de même concevoir un algorithme parallèle avec ITL comme brique de base. Le principal inconvenient serait le transfert des données. Le temps de communication d'une donnée serait plus important que le traitement même de cette donnée puisque dans le pire des cas le calcul ne couterait rien et pour chaque activation des fonctions de communications les temps d'initialisation du canal s'accumuleraient.

Finalement une décomposition excessive de TL nous rendrait trop tributaire de la machine de par l'introduction de temps de communication dont le coût risquerait de devenir prohibitif.

Une autre hypothèse, qui fait partie du cadre choisi pour décrire les algorithmes parallèles, est de supposer que le transfert des données ne se fait que sur les familles de polynômes. Une justification plus précise de cette hypothèse est donnée lors de la description des procédures de communication nécessaires après avoir effectué TL dans chaque processeur.

Dans les paragraphes qui suivent on précise ce qu'est TL. On montre comment grâce à un processus itératif construit à l'aide de ce module de base, d'une procédure de normalisation globale et de procédures de communication on construit finalement une base de Gröbner associée à F. La procédure de normalisation globale utilise :

- un procédé local de normalisation pendant lequel les processeurs normalisent des polynômes par rapport à des familles de polynômes contenus dans leurs mémoires,
- une procédure de communication qui permet le transfert des polynômes à normaliser.

Cette normalisation globale est nécessaire pour réduire la taille des données à traiter.

Le premier appel de la procédure TL traite des données équidistribuées (initiales). Avant les autres appels successifs de TL il pourrait sembler intéressant de redistribuer les polynômes à traiter pour tenter d'équilibrer l'algorithme afin que chaque processeur effectue le même volume de calcul. Malheureusement, l'équidistribution des données n'est pas une condition suffisante pour cela en particulier en ce qui concerne le volume de calcul imposé par la normalisation. Comme de plus elle implique des communications supplémentaires, rien ne nous assure un gain sur le temps d'exécution de l'algorithme et nous avons choisi de ne pas l'implanter.

I-1-2 Le calcul des spolynômes et la normalisation

Après la phase de répartition des données chaque processeur i contient une famille de référence FP_i.

. le traitement local - la brique de base

On définit ici le traitement local $TL_i(FP)$ où i désigne le numéro du processeur dans lequel on se place et qui contient une famille quelconque FP. Il permet de calculer les spolynômes normalisés associés aux entrées et s'écrit :

$TL_i(FP)$ ==

```
ENTREE : Une famille FP quelconques de polynômes.
SORTIE : SPN_i spolynômes associés aux polynômes de FP
         normalisés par rapport à FP_i.
  { variables intermédiaires }
  B1, B2: ensembles de couples de polynômes ;
  h : polynôme ;
  { Initialisation }
  B1 := FP * FP_i ;        { produit pseudo-cartésien de FP par FP_i }
  B2 := (FP * PI) ∪ (FP_i * PI) ;
  SPN_i := ∅;
  { itération principale }
  tant que ((B1 ≠ ∅) ou (B2 ≠ ∅))
        Si (B1 ≠ ∅)
              Prendre C dans B1 : C := (p1, p2) ;
              B1 := B1 - {C} ;
              h := NormPF(Spoly(p1, p2), G) ;
        Sinon
              Prendre C dans B2 : C := (p, pi) ;
              B2 := B2 - {C} ;
              h := NormPF(Spoly(p, i), G) ;
        Si h ≠ 0
              SPN_i := SPN_i ∪ {h} ;
  Sortir(SPN_i).
```

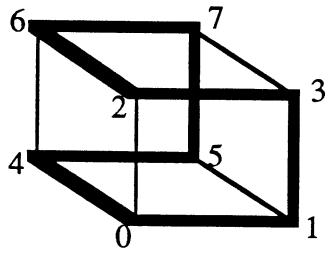
fin de$TL_i(FP)$.

Lorsqu'on applique TL_i sur FP_i, on calcule les spolynômes associés aux k polynômes de référence et on les normalise par rapport à ces derniers. Il s'agit ensuite de calculer les spolynômes manquants, c'est à dire ceux associés aux polynômes de

référence qui sont localisés dans des mémoires locales différentes. On introduit alors un mécanisme de communication qui permet la rencontre de tels polynômes.

. la phase de communication

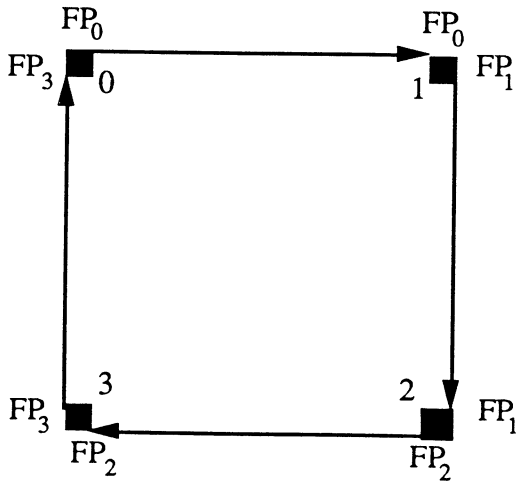
On utilise une topologie d'anneau. Comme il est souligné dans l'introduction, dans un d-cube à $p = 2^d$ sommets, il est possible de trouver un cycle de longueur $L = 2^k$ avec k inférieur ou égal à d, qui traverse chaque sommet exactement une fois en les reliant par une seule arête. On peut dire que les processeurs de l'hypercube, sont disposés en anneau et numéroté de 0 à L-1 (on suppose que $L \geq 2$).



avec la numérotation suivant le code de Gray
pour un anneau de longueur 8
dans un hypercube de dimension 3.

Les algorithmes qui utilise cette topologie, peuvent être facilement paramétrés par le nombre de processeurs utilisés, suivant la longueur du cycle choisi.

Les familles de polynômes de référence contenues dans les mémoires locales circulent sur l'anneau : chaque processeur envoie ses polynômes de référence à son voisin. Sur un anneau à quatre sommets on peut schématiser les transferts de la façon suivante (avec la numérotation de 0 à 3) :



On rappelle que le temps T_n, nécessaire pour communiquer n données de taille unitaire consécutives en mémoire est la somme de deux termes : β qui correspond à

un temps d'initialisation du canal à occuper et n τ proportionnel à n. Il est donc important qu'aux objets manipulés correspondent des données consécutives en mémoire. La représentation des monômes, des polynômes et des familles de polynômes, (décrite dans le chapitre précédent) nous le permet. Pour éviter la multiplication des temps d'initialisation du canal on cherche à communiquer la plus grande structure possible qui soit contigüe en mémoire. Les transferts ne se font donc que sur les familles de polynômes.

On note Com_i la phase de communication pour le $i^{ième}$ processeur de l'anneau et mod l'opération modulo. Elle a pour paramètres deux familles quelconques FE et FR :

Com_i (FE, FR) ==

```
| j := (i -1) mod L ;
| k := i mod L ;
| l := (i +1) mod L ;
| ENVOYER(k, FE, l) ;          { L est la longueur de l'anneau }
| RECEVOIR(k, FR, j) ;
```

fin de com_i.

où les fonctions ENVOYER et RECEVOIR correspondent à la modélisation des communications et permettent de communiquer d'un processeur à l'autre. Leurs paramètres sont :

i) le processeur concerné,

ii) la famille de polynômes à envoyer ou recevoir, suivant le cas,

ii) le processeur où va (si c'est ENVOYER) ou d'où vient (si c'est RECEVOIR) la famille à transférer.

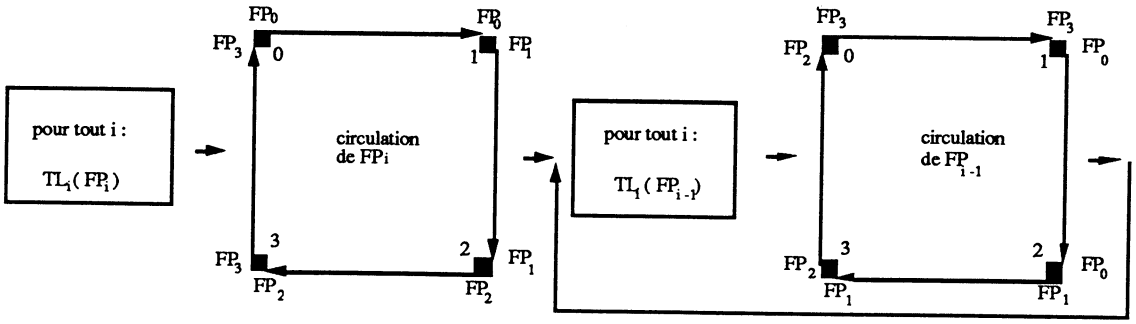
Dans la suite de ce paragraphe i correspond au numéro du processeur sur l'anneau. On a, pour cette première communication, FE = FP_i et FR = FP_{i-1} (sauf pour i = 0 car alors FR vaut FP_{l-1} et pour i = L-1 car alors FE vaut FP_0).

. l'itération des deux phases précédentes

Chaque processeur a maintenant dans sa mémoire locale ses propres polynômes de référence (FP_i), ceux de son voisin dans l'anneau (FP_{i-1}) et enfin les polynômes résultant du traitement local (SPN_i).
On leur ajoute les spolynômes associés aux couples de FP_i * FP_{i-1}, que l'on normalise par rapport aux FP_i. Cette phase est effectuée à l'aide de $TL_i(FP_{i-1})$. On cumule alors les résultats de $TL_i(FP_i)$ et de $TL_i(FP_{i-1})$ dans SPN_i.

On enchaine ensuite l'étape de communication et le module de base TL_i pour calculer les spolynômes associés à tous les polynômes donnés en entrée (répartis en familles de polynômes de référence).
L'étape de communication consiste pour chaque processeur à envoyer à son voisin (toujours le même) les polynômes reçus à l'étape précédente. On applique com_i ; FE et FR prennent alors successivement les valeurs FP_{i-1} et FP_{i-2} puis FP_{i-2} et FP_{i-3}..... TL_i s'applique ensuite aux polynômes juste reçus.

Le test d'arrêt de l'itération est simple : il suffit d'avoir effectuer la phase de communication L/2 fois. On note la procédure correspondant à cette itération TSL_i. On peut la schématiser par :



L'algorithme de chaque processeur est alors :

TSL_i ==

ENTREE : Les FP_i.
SORTIE : SPN_i spolynômes associés aux polynômes de F normalisés localement par rapport aux FP_i.

{ variables intermédiaires }
FE, FR, SPN_i : familles de polynômes ;
j, q : deux indices ;
{ Initialisation }
j := 0 , q := 1 ;
FE := FP_i ;
si i = 0
    FR := FP_{L-1} { L est la longueur de l'anneau }
sinon
    FR := FP_{i-1} ;
TL_i (FE) ; { SPN_i est alors initialisé }
tant que (q <= ((L / 2) -1))
    Com_i (FE, FR);
    FE := FR ;
    SPN_i = $SPN_i \cup TL_i$ (FE) ;
    j := j+1 ;
    si (i = 0 et j = 1) j := i - (L - 1) ;
    si (i = j) j := i - (L -1) - 1 ;
    FR := FP_{i-j-1} ;
Sortir(SPN_i).

fin deTSL_i.

Pour des commodités d'écriture on utilise dans la description ci-dessus les variables intermédiaires FE et FR qui contiennent en fait des valeurs que le processeur ne connaît pas encore. Cette "affectation " réserve de la place mémoire dans le processeur considéré pour recevoir la donnée envoyée ulterieurement.

Lorsque TSL_i est terminée on dispose de tous les spolynômes, comme c'était le cas dans l'algorithme séquentiel. Néanmoins, ils ne sont pas complétement normalisés par rapport à tous les polynômes donnés en entrée ; c'est le but de la phase suivante.

. la normalisation parallèle

C'est la dernière phase de cette première étape, on note TN_i la suite d'opérations que doit alors effectuer le processeur i.

Les polynômes à normaliser doivent l'être par rapport à des polynômes qui sont répartis entre les différents processeurs. On utilise la même procédure de communication que précédemment. Le processeur i envoie au processeur i+1 tous les polynômes calculés disponibles dans SPN_i. On active la phase de communication Com_i sur les SPN_i puis les on normalise en parallèle par rapport aux polynômes de référence. On répète ces deux phases :

- on fait circuler le long de l'anneau les polynômes de SPN_i,
- on les normalisent localement par rapport aux polynômes de référence.

On poursuit jusqu'à obtenir, dans la mémoire locale de chaque processeur, une partie des spolynômes associés aux entrées et normalisés par rapport à celles-ci.

Le mécanisme utilisé est le même que celui de TSL_i si on remplace la phase de calcul TL_i par une phase de normalisation.

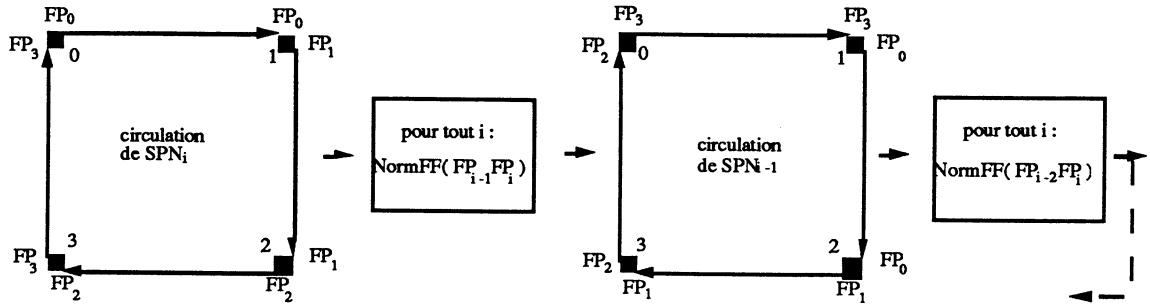
Le test d'arrêt de cette série d'itérations n'est pas aussi simple que celui de TSL_i. En effet pour que l'itération s'arrête, il est nécessaire d'avoir dans tous les processeurs les polynômes résultats normalisés non seulement localement mais aussi par rapport aux polynômes de référence des autres processeurs. On normalise les SPN_i par rapport aux FP_i puis par rapport aux FP_j par circulation. Nous présentons ici, le processus de terminaison de la phase de normalisation.

- Terminaison de la normalisation :

Si jamais un processeur terminait sa tâche avant les autres il bloquerait l'anneau en empêchant la circulation des données.
Pour éviter un tel comportement on force l'arrêt synchrone des processeurs, en envoyant avec les polynômes à normaliser un message qui donne l'état de normalisation des polynômes reçus. Ce message précise si les polynômes qui lui sont associés sont ou non sous forme normale par rapport aux polynômes de référence déjà rencontrés.
Le processeur considéré normalise s'il y a lieu les polynômes qu'il vient de recevoir, fait circuler l'information reçue, et précise de plus, l'état de normalisation des polynômes qu'il envoit par rapport à ses polynômes de référence.
Le processus de normalisation s'arrête au bout d'un nombre entier de tours lorsque les polynômes sont enfin sous forme normale par rapport à tous les polynômes de référence. On note ce test global TNG_i qui est mis à faux dans le processeur i si il existe un processeur qui n'a pas

terminé et à vrai sinon ; en fait TNG_i a la même valeur dans tous les processeurs.

On peut schématiser cette normalisation globale par :



NormFF appliquée à deux familles FP et FQ est la procédure qui permet de calculer par rapport à FQ les formes normales des polynômes contenus dans FP.

L'algorithme de chaque processeur est alors :

TN_i ==

ENTREE : Les SPN_i.
SORTIE : SPN_i spolynômes associés aux polynômes de F normalisés localement par rapport à tout F
{ variables intermédiaires }
FE, FR, : familles de polynômes ;
{ Initialisation }
FE := SPN_i ;
si i = 0
    FR := SPN_{L-1} { L est la longueur de l'anneau }
  sinon
    FR := SPN_{i-1} ;
{ les SPN_i sont déja normalisés par rapport à FP_i}
tant que (non (TNG_i))
    Com_i (FE, FR) ;
    FE := FR ;
    PE = NormFF(PR , FP_i) ;
    j := j+1 ;
    si (i = 0 et j = 1) j := i - (L - 1) ;
    si (i = j) j := i - (L -1) - 1 ;
    FR := SPN_{i-j-1} ;
Sortir (SPN_i).

fin deTN_i.

I-1-3 L'algorithme parallèle complet : PSGBuch

Nous venons de décrire les opérations effectuées en parallèle qui correspondent à l'itération principale de l'algorithme séquentiel.

Il s'agit maintenant de répéter ces opérations en modifiant leurs entrées et en mettant B1 et B2 à jour grâce au traitement local. A chaque étape, et dans chaque processeur les polynômes de référence sont modifiés : on leur ajoute les polynômes issus de l'étape de calcul précédente, présents dans chaque mémoire locale considérée. On fait en sorte de ne pas calculer des spolynômes plusieurs fois : B1 ne contient pas les paires qui proviennent des polynômes de référence qui se sont déjà rencontrés pendant la phase précédente (pendant laquelle tous les spolynômes associés aux polynômes de référence on été calculés grâce à une circulation sur l'anneau de longueur L composée de L/2 communication élémentaire d'un processeur à son voisin). Dans chaque processeur B1 et B2 correspond à l'état local de l'algorithme.
L'algorithme se termine lorsque tous les polynômes issus d'une étape sont nuls. Le test d'arrêt, comme le précédent, se fait par l'envoi d'un message pour synchroniser les processeurs, on le note TGG.

Pour écrire PSGBuch de façon globale on utilise le fait que chaque processeur connaît sa position dans l'anneau. On peut représenter PSGBuch de la façon suivante :

PSGBuch ==

ENTREE : Les polynômes de F répartis en FP_i.
SORTIE : Une base de Gröbner booléenne non réduite associée à F.

pour tout i faire { sur tous les processeurs de l'anneau }
    TGG := faux ;
    { boucle principale }
    tant que (non (TGG))
        TSL_i ;{on a dans chaque processeur SPN_i }
        TN_i ;
        $FP_i := FP_i \cup SPN_i$;
    Sortir (SPN_i).

fin dePSGBuch.

I-1-4 Quelques remarques

Il est possible d'imaginer des variantes de cet algorithme. On peut, par exemple, calculer tous les spolynômes de chaque étape en faisant circuler sur un anneau les polynômes de référence, puis les normaliser en une seule fois en les faisant circuler à nouveau sur l'anneau. Cette méthode a été en fait la première implantée sur le T40.

Les résultats expérimentaux ont montré qu'il était préférable de normaliser les polynômes localement au fur et à mesure de leur construction. Ceci permet d'avoir des

données à transférer d'un processeur à l'autre moins conséquentes, donc un temps de communication moindre et de calculer moins de spolynômes à chaque étape.

Nous allons maintenant utiliser les techniques de parallélisation de SGBuch pour paralléliser l'algorithme MGBuch donné dans le premier chapitre de cette partie.

I-2 Pour la base de Gröbner réduite. Parallélisation de RGBuch : P1RGBuch

La différence entre les algorithmes séquentiels se situe au niveau de la procédure de réduction (MRéd ou encore BRéd) qui n'intervient que dans les algorithmes calculant la base de Gröbner réduite. Les structures des algorithmes parallèles sont identiques et la principale difficulté pour paralléliser RGBuch est donc l'introduction de cette procédure. Le seul critère à respecter pour l'implantation de cette procédure est d'éviter la perte d'informations [Watt85][Davenport85].

Le modèle choisi pour les algorithmes, en particulier le fait que les processeurs qui contiennent des données différentes ne peuvent modifier que leurs propres données, nous en protège en partie. Pour être tout à fait sûr de ne pas supprimer de polynômes qui ne devraient pas l'être, on ne parallélise pas la procédure de réduction choisie (MRéd ou BRéd).

En pratique on introduit la procédure de réduction (non parallélisée, avec B1 et B2 remis localement à jour) à deux niveaux différents. Ce choix est dicté par des raisons de simplicité et pour éviter la manipulation de grosses données. Réd est insérée après chaque phase de normalisation parallèle et après chaque affectation $FP_i := FP_i \cup SPN_i$. On obtient alors dans chaque processeur les morceaux (réduits pour chacun d'eux) d'une base de Gröbner. Il nous reste alors à réduire tous ces morceaux entre eux. Pour celà on appelle TN_i qui normalise les morceaux de la base de Gröbner en les faisant circuler sur l'anneau.

Nous verrons également d'autres choix possibles sur des exemples (cf chapitre suivant), en particulier nous ferons différents tests suivant le choix de la procédure (MRéd, Bréd) et suivant le choix de son lieu d'insertion dans l'algorithme.

Cet algorithme paralléle ne correspond pas à un algorithme séquentiel tel que MGBuch, qui active la procédure de réduction après chaque calcul d'un spolynôme. L'algorithme séquentiel correspondant est constitué du module de base TL qui normalise les polynômes calculés par rapport à l'ancienne famille de polynômes de référence, mais qui n'active une réduction que lorsque tous les spolynômes associés à une famille sont calculés contrairement à l'algorithme séquentiel présenté dans le chapitre précédent. Nous verrons comment se comporte ces algorithmes séquentiels dans le chapitre consacré aux résultats.

II- UTILISATION DE L'HYPERCUBE

L'étude ménée ici est directement faite sur RGBuch puisque l'introduction de la procédure de réduction ne présente pas les mêmes problèmes que dans l'approche précédente.: il s'agit de la parallélisation de MGBuch.

Dans la suite on dit d'un processeur qu'il est actif si il effectue un certain nombre de tâches et qu'il est passif sinon.

Afin d'écrire les procédures de communications nécessaires à cet algorithme, nous pouvons faire les remarques suivantes :

Nous avons déjà souligné le fait qu'un processeur pouvait envoyer ou recevoir de ses processeurs voisins sur un anneau. En effet chaque processeur connaît son voisin (sur l'anneau) grâce au code de Gray. Ici, nous utilisons le fait que chaque processeur connaît son numéro binaire, que l'on appelle par la suite numéro absolu, et que deux processeurs sont directement reliés entre eux si leurs numéros respectifs diffèrent d'un bit.
Un processeur k se repère par son numéro absolu que l'on représente par un d-uplet de 0 et de 1 $(k_{d-1},....,k_0)$, où D = d -1 est la dimension de l'hypercube dans lequel on travaille (un d-cube !). On a alors :

$$k = \sum_{i=0}^{i=d-1} k_i 2^i$$

Si l'on communique directement, du processeur k de numéro absolu $(k_{d-1},....,k_0)$ au processeur j de numéro absolu $(j_{d-1},...., j_0)$, on utilise les fonctions standard RECEVOIR et ENVOYER en précisant le canal sur lequel se fait l'envoi, c'est à dire le bit qui diffère d'un processeur à l'autre. Si par exemple, on veut transformer D_k de k en une donnée D_j de j on écrit :

c étant le numéro du bit qui différencie k et j

ENVOYER(c, k_c, D_k, j_c)

RECEVOIR(c, j_c, D_j, k_c)

L'algorithme parallèle donné ici noté P2RGBuch correspond à une conception du parallélisme différente de celle développée dans la partie I-. En effet, au lieu de diviser l'algorithme en tâches indépendantes, on le considère lui même comme une tâche et ce sont principalement des manipulations sur des données qui permettent d'introduire le parallélisme.
Les deux points essentiels sur lesquels repose l'algorithme sont un processus de "divide and conquer" et l'indépendance entre les résultats et l'ordre des données.
Pour des raisons de clarté l'algorithme est tout d'abord décrit sur 4 processeurs puis sur 8 puis sur 16 et enfin sur un hypercube de n'importe quelle dimension. La technique de parallélisation repose sur le résultat suivant :

II-1 Résultat théorique préliminaire

Soient F = $\{p_1,...p_q\}$ une famille de q polynômes correspondant à la donnée d'un idéal I et G une base de Gröbner de I.

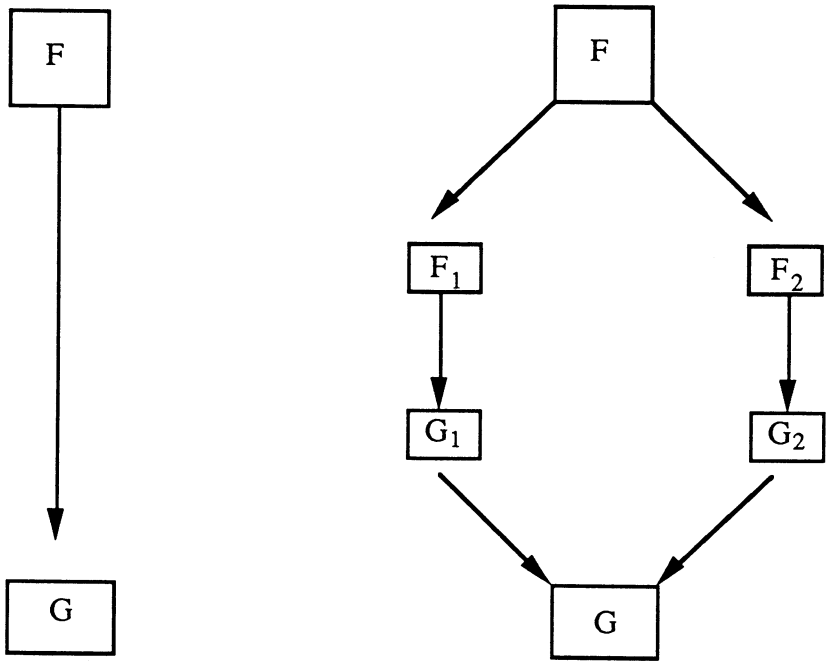
Si l'on considère une partition de F en deux ensembles F_1 et F_2, associés respectivement aux idéaux I_1 et I_2, avec $F_1 = \{p_1,...p_l\}$ et $F_2 = \{p_{l+1},...p_q\}$, on a le résultat suivant :

Si G_1 et G_2 sont des bases de Gröbner associées respectivement à I_1 et I_2 alors une base de grobner G_{12} associée à l'idéal engendré par l'union $G_1 \cup G_2$ est également une base de Gröbner pour I.
Si de plus toutes les bases considérées sont réduites alors, par unicité, G_{12} est la base de Gröbner (réduite) associée a I.

La démonstration de ce résultat s'appuie essentiellement sur le fait que les idéaux engendrés par $G_1 \cup G_2$ et I sont les mêmes.

Les bases de Gröbner G_1 et G_2 sont appelées "sous-bases" de G. Ce résultat est également valable pour les bases de Gröbner booléennes.

On peut schématiser ce résultat en disant qu'il y a équivalence entre le schéma$_1$ et le schéma$_2$ avec :



Schéma$_1$ Schéma$_2$

Figure1

A priori, ce résultat n'apporte rien de particulier puisque qu'il est nécessaire de recalculer la base de Gröbner de $G_1 \cup G_2$ pour avoir celle de I.
On utilise alors le fait que les familles G_1 et G_2 ont déjà une structure de base de Gröbner et l'on adapte l'algorithme de Buchberger MGBuch à ce cas particulier pour pouvoir en bénéficier.

Avant de décrire cette procédure que l'on note RGBuch2 et qui correspond à la "brique de base" de l'algorithme nous pouvons regarder, sans entrer dans les détails, la stratégie utilisée pour mettre en place P2RGBuch et les problèmes posés alors.

II-2 Les grandes lignes de l'algorithme P2RGBuch

Si l'on veut superposer la décomposition du schéma$_2$ sur deux processeurs il suffit de répartir les données sur les deux processeurs (en familles de polynômes de référence), de calculer en parallèle deux sous-bases (avec MGBuch, par exemple) de regrouper les résultats dans un processeur et d'y effectuer MGBuch2.
Cette manipulation va de pair avec le modèle associé à l'algorithme (les processeurs ne peuvent modifier que leurs données, les communication et les calculs ne s'effectuent pas en parallèle).
Le problème de cette étape, problème qui subsiste encore, est que pour la dernière étape un seul processeur active MGBuch2. Dans la dernière phase de l'algorithme complet il en sera de même : une seule moitié des processeurs activera MGBuch2....

Si maintenant on cherche à appliquer la même approche avec quatre processeurs on obtient :

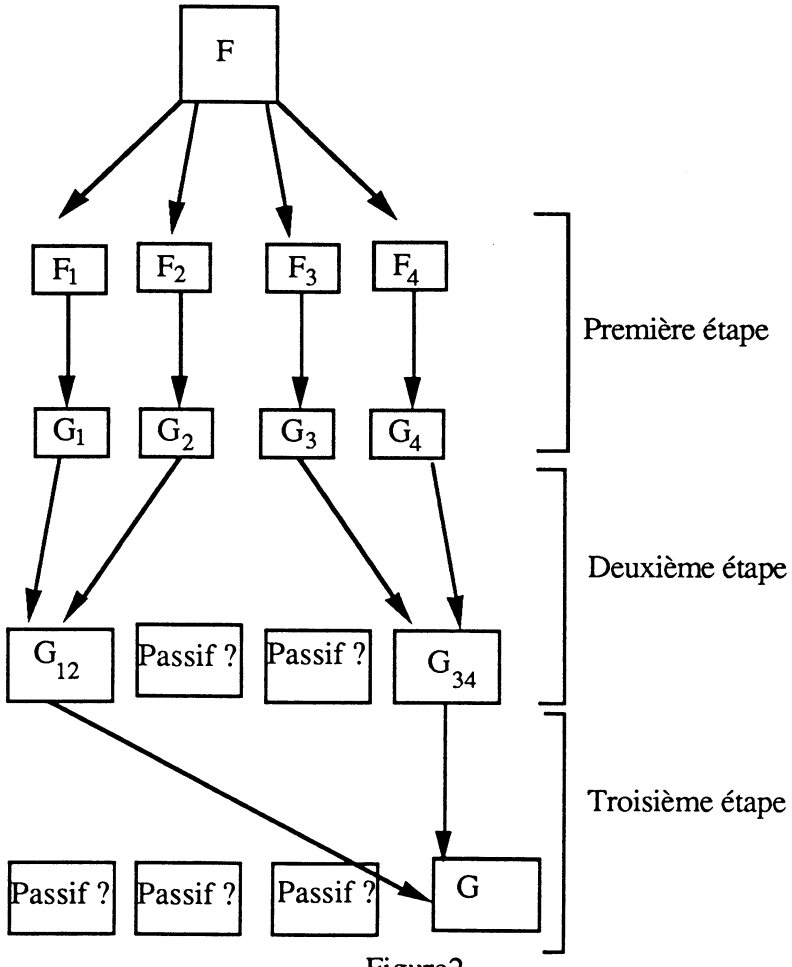


Figure2

A chaque étape seule la moitié des processeurs actifs à l'étape précédente l'est encore. Pour avoir tous les processeurs actifs on utilise le fait que le résultat cherché est indépendant de l'ordre de entrées. Les processeurs passifs deviennent actifs et calculent le résultat associé à une autre répartition des données.
Cette répartition fournit également une base de Gröbner de ces entrées (la même que celle de l'autre répartition si on cherche la base de Gröbner réduite) en un temps de calcul qui peut être très différent (cf le chapitre précédent). L'algorithme s'achève lorsque l'un des processeurs actifs à la derniere étape a terminé sa série de calculs. C'est le premier qui a terminé qui conditionne la terminaison globale.
On peut représenter les répartitions schématiquement par :

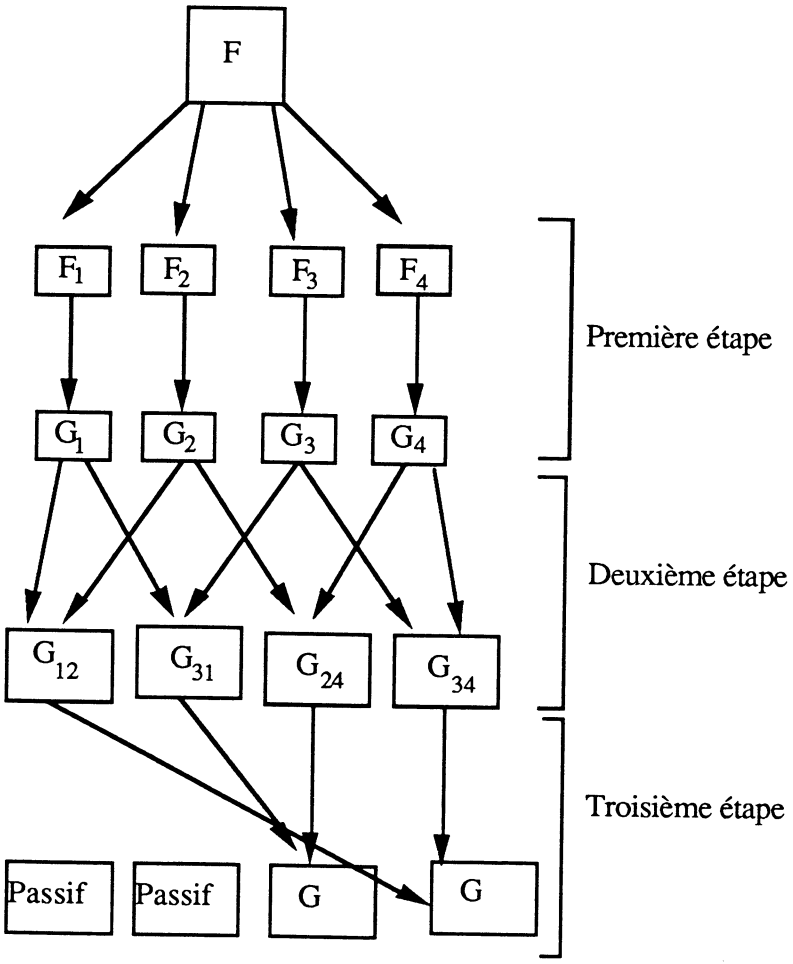


Figure3

Sur quatre processeurs, à l'aide de MGBuch, MGBuch2 et de fonctions de communications, on montrera, dans la suite, comment on peut calculer deux bases de Gröbner (identiques) associées aux polynômes initialement répartis sur les processeurs de deux façons différentes en des temps d'exécution différents. Pour ce faire on utilisera la procédure Hyp4 composée des deuxième et troisième étapes de la figure3.

On généralise ce procédé à 8, 16, 32,...2^d processeurs. Après chaque étape de calcul les processeurs sont partitionnés en ensembles disjoints qui travaillent indépendamment les uns des autres (en parallèle). Succinctement l'algorithme se déroule de la façon suivante :

pour huit processeurs :

On passe d'un ensemble à huit processeurs à deux ensembles à quatre processeurs :

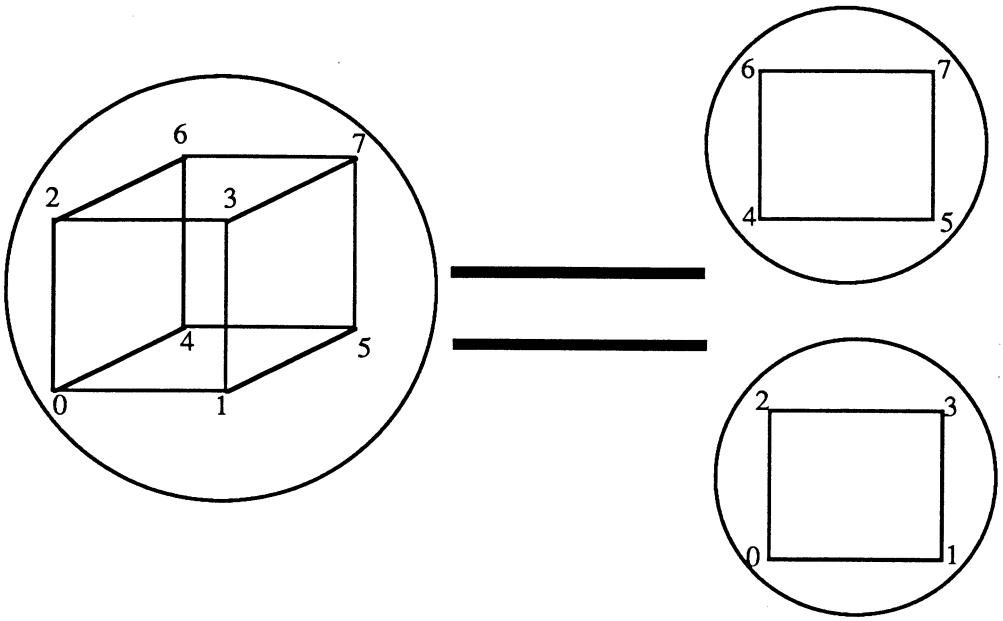


Figure4

On suppose que la famille F est répartie sur les huit processeurs en huit familles de polynômes de référence, et que l'on a calculé en parallèle huit sous-bases. A l'aide d'une procédure de communication, on fabrique deux répartitions différentes de ces sous-bases afin de pouvoir considérer le cube à huit processeurs comme deux cubes disjoints à quatre processeurs (figure4). A chacun de ces cubes est associé une répartition des sous-bases. Ces répartitions correspondent chacune à différentes manières de regrouper les huit sous-bases en quatre couples différents, sans éléments communs.

Si par exemple on a les sous-bases G_0, G_1...G_7, les couples :

- (G_0,G_1), (G_2,G_3), (G_4,G_5), (G_6,G_7) forment une répartition possible et peuvent correspondent à un cube à 4 processeurs,

- (G_0,G_7), (G_1, G_6), (G_2, G_5), (G_3,G_4) en forment une autre.

Puis, pour chaque groupe de 4 processeurs on active MGBuch2 puis Hyp4 pour obtenir la base de Gröbner associée à F.

De manière plus générale sur un cube à 2^d sommets l'algorithme P2MGBuch est composé de d-1 étapes. Les d-2 premières étapes sont analogues ; la $k^{ième}$ consiste à effectuer en parallèle le calcul de MGBuch ou de MGBuch2 (si k = 1 MGBuch,

MGBuch2 sinon), puis à répartir les données pour passer d'un hypercube de dimension k à deux hypercubes de dimension k-1. La d-1 $i^{ième}$ étape active Hyp4 sur chaque groupe indépendant de 4 processeurs.

II-3 Construction de P2RGBuch

II-3-1 Le traitement local MGBuch2

On utilise ici MGbuch, en raison de la facilité de son adaptation au problème posé. Il s'agit d'obtenir à partir de deux bases de Gröbner réduites celle associée à leur union. Les principales différences entre MGBuch et MGBuch2 se situent au niveau de la réduction MRéd et de la construction de B1 et B2. On écrit MGBuch2 à l'aide d'une procédure de réduction de la façon suivante :

MGBuch2 ==

ENTREE : Deux bases de Gröbner G_i et G_j.
SORTIE : la base de Gröbner (réduite) associée G_{ij}.
R, G_{ij}, P : familles de polynômes ;
B1, B2 : ensembles de couples de polynômes ;
h : polynôme ;
R := G_i ; P := G_j ; $G_{ij} = \varnothing$;
{ traitement des entrées : réduction préliminaire }
MRéd2 (R, P, G_{ij}, B1, B2) ;
tant que ((B1 $\neq \varnothing$) ou (B2 $\neq \varnothing$))
  Si (B1 $\neq \varnothing$)
    Prendre C dans B1 : C := (p_1, p_2) ;
    B1 := B1 - {C} ;
    h := NormPF(Spoly(p_1, p_2), G_{ij}) ;
  Sinon
    Prendre C dans B2 : C := (p, $\underline{p_i}$) ;
    B2 := B2 - {C} ;
    h := NormPF(Spoly(p, i), G_{ij}) ;
  si h $\neq$ 0
    G_0:= { g = (g_p +....+ g_1) $\in$ G_{ij} / $\exists$ g_k tel que
      Pgcd(Pm(h),g_k) = Pm(h) } ;
    R := G_0; P:={h} ; G_{ij} := G_{ij} - G_0 ;
    B1 := B1 - {(g_1,g_2) $\in$ B1 / $g_1 \in G_0$ ou $g_2 \in G_0$};
    B2 := B2 - {(g_1,$\underline{p_i}$) $\in$ B2 / $g_1 \in G_0$};
    MRéd (R, P, G_{ij}, B1, B2) ;
Sortir(G_{ij}).

Fin de MGbuch2.

MRéd2, procédure de réduction, est définie par :

MRéd2(R, P, G_{ij}, B1, B2) ==

```
{ variables intermédiaires }
H : famille de polynômes ;
tant que R ≠ ∅
        h := un élément de R ; R := R - {h} ;
        h := NormPF(h, P ∪ G_ij) ;
        si h ≠ 0
                G_0:= { g = g_p +....+ g_1 ∈ G_ij / ∃ g_k tel que
                                Pgcd(Pm(h), g_k) = Pm(h) } ;
                P_0:= { g = g_p +....+ g_1 ∈ P / ∃ g_k tel que
                                Pgcd(Pm(h), g_k) = Pm(h) } ;
                G_ij := G_ij - G_0 ;
                P := P - P_0 ;
                R := R ∪ P_0 ∪ G_0 ;
                B1 := B1 - {(p_1,p_2) ∈ B1 / p_1∈ G_0 ou p_2∈ G_0};
                B2 := B2 - {(p_1,p_i) ∈ B2 / p_1∈ G_0};

                P := P ∪ {h} ;
G_ij := G_ij ∪ P ;
B1 := B1 ∪ (P * G_ij) ; B2 := B2 ∪ (P * PI) ;
```

Fin MRéd2.

Au premier appel de MRéd2, P a pour valeur une des bases de Gröbner de l'entrée et R l'autre, si bien que les polynômes sont normalisés entre eux et mis dans P. Puis l'on forme l'ensemble B1 qui correspond au produit pseudocartésien P * R ainsi que B2. Les polynômes issus d'une même base ne sont pas alors traités puisque leurs spolynômes calculés puis normalisés par rapport à G_{ij} sont nuls.

II-3-2 Cas de quatre processeurs : description de Hyp4

P2MGBuch débute par le calcul d'une sous-base de Gröbner réduite dans chaque processeur. Pour le processeur i, où i désigne le numéro absolu (i est compris 0 et 3) on note cette sous-base G_i. G_i est construite à partir des polynômes de référence avec MGBuch. On active ensuite Hyp4 , qui nous permet d'obtenir deux répartitions des sous-bases puis la base de Gröbner des entrées.

Les différentes phases de Hyp4 se déroulent suivant les étapes 2 et 3 de la figure3 : chaque processeur envoie sa sous-base à son voisin à l'intérieur d'un anneau à 4 noeuds ; on note ce mécanisme 1PPL.

Chaque processeur contient alors deux sous-bases à partir desquelles il calcule la sous-base associée à l'aide de MGBuch2. Puis grâce à une autre procédure de communication noté 2PPL les sous-bases de deux processeurs diagonalement opposés sur le cube se

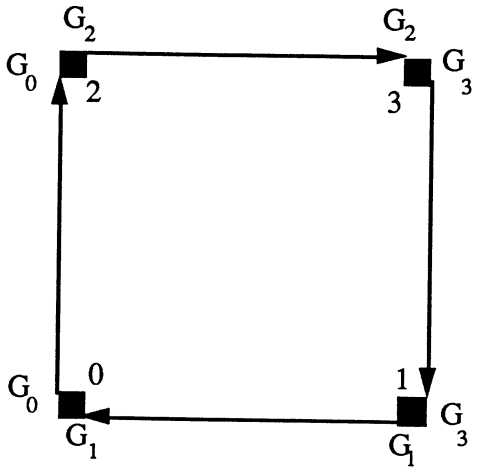
rencontrent pour entamer la derniere étape de calcul et obtenir la base de Gröbner associée aux polynômes d'entrée. On peut schématiser P2MGBuch par :

- calcul de G_i (Première étape de la figure3) :

Processeur i $\Longrightarrow$ Traitement MGBuch G_i

- appel à Hyp4 :

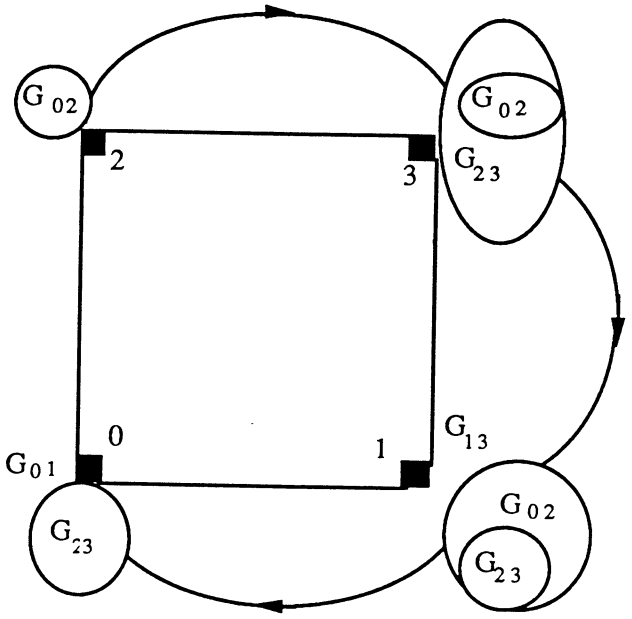
- premiers transferts 1PPL (deuxième étape de la figure3) :



- calcul de $G_{i,j}$ (avec (i, j) valant (0, 1), (1, 3), (2, 0),(3, 2))

Processeur i $\Longrightarrow$ Traitement MGBuch2 $G_{i,j}$

- derniers transferts 2PPL (troisième étape de la figure3) :



- derniers calculs : MGBuch2 pour le calcul de la base finale dans les processeurs 0 et 1 .

Le numéro absolu défini pour chaque processeur va nous permettre d'écrire les communications à faire sur chaque groupe de quatre processeurs de façon générique.

Rappelons ici que chaque processeur a le même programme et qu'il repère les tâches qu'il doit exécuter : cela permet d'introduire de façon simple le parallélisme. Dans les algorithmes nous avons choisi de signaler, à chaque fois que cela est possible, si les processeurs communiquent en parallèle ou non. En ce qui concerne le calcul, on ne signale rien : les calculs sont toujours parallèles ! on décrit les procédures de communications de la façon suivante :

```
1PPL ==
|   ENTREE : une sous-base G_k pour chaque processeur k.
|   SORTIE : deux sous-bases G_k et G_j pour chaque processeur k.
|       { variables intermédiaires }
|       E, R, G_j : familles de polynômes ;
|       { initialisation }
|        E := G_k ;
|       { chaque processeur k → (k_{d-1}, ....k_0) }
|       si k_0 = 0        { tous les processeurs pairs }
|               |  { communication Bas-Haut }
|               |   ENVOYER(1, 0, E, 1) ;
|               |  RECEVOIR(1, 1, R, 0)
|       sinon              { tous les processeurs impairs }
|               |  { communication Haut-Bas}
|               |  ENVOYER(1, 1, E, 0) ;
|               |  RECEVOIR(1, 0, R, 1) ;
|       { les communications Bas-Haut et Haut-Bas  se font en parallèle }
|       si k_1 = 0
|               |  { communication Droite-Gauche }
|               |  ENVOYER(0, 1, E, 0) ;
|               |  RECEVOIR(0, 0, R, 1)
|       sinon
|               |  { communication Gauche-Droite}
|               |  ENVOYER(0, 0, E, 1) ;
|               |  RECEVOIR(0, 1, R, 0) ;
|       { les communications Droite-Gauche et Gauche-Droite se font en
|       parallèle }
|       G_j := R ;
|  Sortir(G_k, G_j).

Fin de 1PPL.
```

Par exemple sur l'anneau à 4 sommets le transfert de 0 vers 2 est effectué en parallèle avec celui de 3 à 1, puis les transferts de 1 à 0 et de 3 à 2 sont activés également en parallèle.

2PPL ==

ENTREE : une sous-base G_{kj} pour chaque processeur k.
SORTIE : deux sous-bases G_{kj} et G_{im} pour chaque processeur k.
{ variables intermédiaires }
E, R1, R2, G_{im} : familles de polynômes ;
{ initialisation }
E := G_{kj} ;
R1 := ∅ ;
{ chaque processeur k → (k_{d-1}, ...k_0) }
si k_1 = 1
{première communication Gauche-Droite }
ENVOYER(0, 0, E, 1) ;
RECEVOIR(0, 1, R1, 0) ;

si k_0 =1 { tous les processeurs qui viennent de recevoir une sous-base l'envoie avec leur propre sous-base }
{ une communication Haut-Bas}
ENVOYER(1, 1, E, 0) ;
RECEVOIR(1, 0, R2, 1) ;
ENVOYER(1, 1, R1, 0) ;
RECEVOIR(1, 0, R1, 1) ;
si k_1 = 0 { tous les processeurs qui viennent de recevoir les deux sous-bases envoie R2 et garde R1}
{ communication Droite-Gauche}
ENVOYER(0, 1, R2, 0) ;
RECEVOIR(0, 0, R1, 1) ;

G_{im} := R1 ;
Sortir(G_{ki}, G_{jm}).

Fin de 2PPL.

Ainsi écrites ces procédures de communications permettent d'effectuer Hyp4 sur un hypercube de dimension d (D = d-1) à n = 2^d sommets : les processeurs travaillent indépendamment par groupes de quatre car les communications n'utilisent que les canaux 0 et 1 des processeurs.

Avec ces procédures et MGBuch2 on peut écrire Hyp4 de la façon suivante :

Hyp4 ==

ENTREE : une sous-base G_k pour chaque processeur k.
SORTIE : La base de Gröbner G (associée à l'union de sous-bases)
(réduite) pour les processeurs k tels que $k_1 = 0$.
{ variables intermédiaires }
G, G_{kj},G_{im} : familles de polynômes ;
{ initialisation }
G := ∅ ;
{ chaque processeur k → (k_{d-1}, ...k_0) }
pour tout k faire { pour tous les processeurs }
{ communications }
1PPL(G_k) ;
{ on obtient deux sous-bases par processeurs G_k et G_j }
G_{kj} := MGBuch2 (G_k, G_j) ;
{ communications }
2PPL(G_{kj}) ;
{ on obtient deux sous-bases par processeurs G_{kj} et G_{im}}
if $k_1 = 0$
G := MGBuch2 (G_{kj}, G_{im}) ;
Sortir(G).

Fin de Hyp4.

Hyp4 s'arrête dès que l'un des processeurs qui activent MGBuch2 après 2PPL, à la fin de la procédure, a terminé son calcul. Nous reviendrons plus tard, sur ce test d'arrêt.

Hyp4 nous sert alors soit comme dernière étape de l'algorithme P2RGBuch sur un nombre de processeurs supérieur à 4 soit pour écrire P2RGBuch sur 4 processeurs (et alors d = 2).

Nous nous attachons maintenant à décrire à l'aide de Hyp4 et de procédures de communications, l'algorithme P2RGBuch sur huit processeurs.

II-3-3 Cas de huit processeurs : description de Hyp8.

On suppose que chaque processeur contient une sous-base de Gröbner engendrée au cours d'une précédente étape. Cette sous-base peut provenir :

- d'un appel à MGBuch et alors le résultat de Hyp8 est la base de Gröbner de l'union de toutes les familles de références : il s'agit alors de P2MGbuch sur huit processeurs (avec d = 3),

- ou d'une procédure utilisant plus de huit processeurs et dans ce cas on vient d'activer MGBuch2.

Hyp8 est composée de deux phases :

- la première consiste à construire deux répartitions différentes des huit sous-bases de manière à avoir 2 cubes indépendants, on la note REP(D). Cette procédure est paramétrée par D (D = d-1 où d est la dimension de l'hypercube considéré). Ceci nous permet par la suite de définir très simplement le passage d'un hypercube de dimension d à un hypercube de dimension d-1. Ici, on passe de la dimension 3 à la dimension 2.

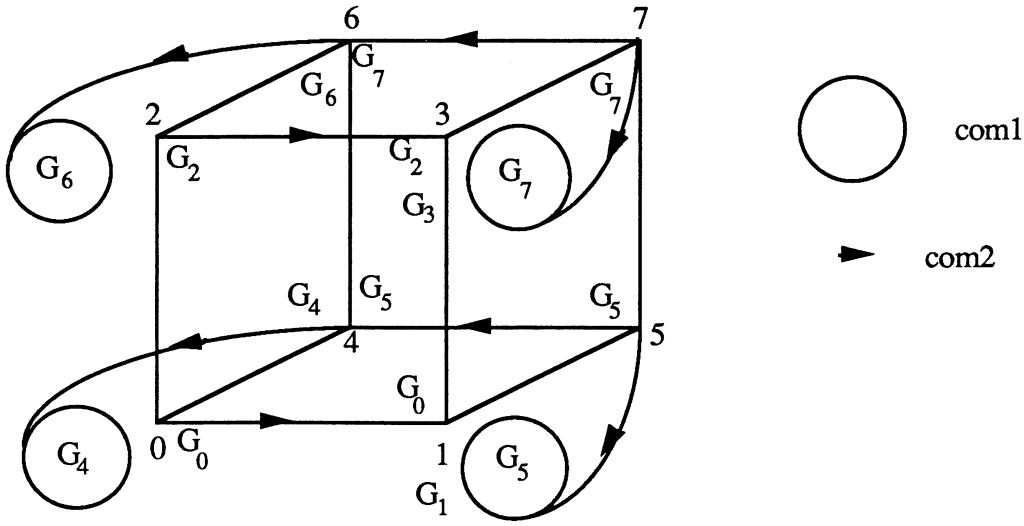
- la deuxième consiste à obtenir les résultats de Hyp4 sur chaque bloc de 4 processeurs en parallèle, ce que fait Hyp4 elle même avec d = 3 (grâce à l'utilisation exclusive de canaux 0 et 1).

Pour d = 3 , D = 2 et REP(D) se déroule de la façon suivante :

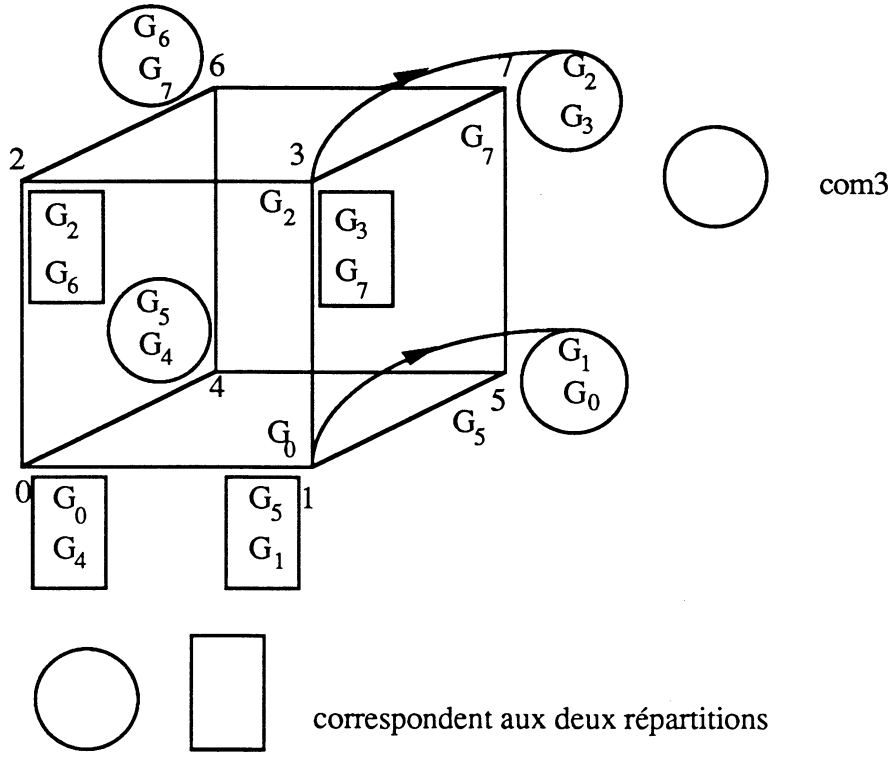
Après une première étape de communication (com1), on a sur le cube "avant" une répartition des données, puisque chaque processeur du cube "arrière" envoie ses données dans le processeur qui lui correspond dans le cube "avant". Pour obtenir l'autre répartition sur le cube "arrière" on active les deux phases de communication com2 et com3 représentées ci-dessous.

com1 et com2 sont activées en parallèle puisque chaque processeur peut envoyer et recevoir simultanément sur les canaux différents.

On schématise com1 et com2 sur la même figure par :



puis on active la troisième étape de communication :



On peut déjà généraliser ces manipulations dans la procédure suivante :

REP(D) ==

ENTREE : une sous-base G_k pour chaque processeur k.
SORTIE : deux sous-bases G_k et G_m pour chaque processeur k telles que l'on ait une répartition différente sur chaque groupe de 4 processeurs.
{ D = d-1 }
{ variables intermédiaires }
E, R1, R2 : familles de polynômes ;
{ initialisation }
E := G_k ;
R1 := ∅ ;
{ chaque processeur k → (k_D, ...k_0) }
{première communication com1 ; on envoie d'un hypercube de dimension d -1 sur un hypercube de même dimension }
ENVOYER(D, 1, E, 0) ;
RECEVOIR(D, 0, R1, 1) ;
{ deuxième communication com2, en parallèle avec com1, à l'intérieur d'un même hypercube}
si k_D =1
    ENVOYER(D-2, 1, E, 0) ;
    RECEVOIR(D-2, 0, R1, 1)
sinon { en parallèle avec les communications précédentes }
    ENVOYER(D-2, 0, E, 1) ;
    RECEVOIR(D-2, 1, R2, 0) ;
{ troisième communication com3, transfert de paires }
si k_{D-2} = 1 { les processeurs qui viennent de recevoir les deux sous-bases envoie R2 et E }
    ENVOYER(D, 0, E, 1)
    RECEVOIR(D, 1, R1, 0) ;
    ENVOYER(D, 0, R2, 1) ;
    RECEVOIR(D, 1, E, 0) ;
G_k := E ;
G_m := R1 ;
Sortir(G_k, G_m).

Fin de REP(D).

On a alors, pour D = 2, deux répartitions des données :
une sur le cube "avant" (0, 1, 2, 3)
l'autre sur le cube "arrière " (4, 5, 6, 7).

On active alors Hyp4, dernière étape de Hyp8.

Hyp8 s'écrit alors :

Hyp8 ==

ENTREE : une sous-base G_k pour chaque processeur k.
SORTIE : La base de Gröbner G (associée à l'union de sous-bases)
réduite pour les processeurs k tels que $k_1 = 0$.
{ variables intermédiaires }
G, G_k : familles de polynômes ;
{ initialisation }
G = ø ;
{ chaque processeur k → $(k_{d-1}, ...k_0)$ }
pour tout k faire { pour tous les processeurs }
{ communications }
REP(2) ;
{ on obtient deux sous-bases par processeurs G_k et G_j }
Hyp4 ; { d = D+1 = 3 et alors G est calculé dans 4 processeurs }
Sortir(G).

Fin de Hyp8.

Hyp8 s'achève lorsque l'un des quatre processeurs encore actifs à la dernière étape a terminé son calcul.
L'algorithme P2MGBuch sur huit processeurs s'écrit alors avec MGBuch activée sur les polynômes de référence de chaque processeur puis avec Hyp8 sur les sorties de MGBuch.

L'algorithme se généralise de façon simple sur un hypercube de n'importe quelle dimension .

II-3-4 Généralisation à n processeurs : description de Hypn

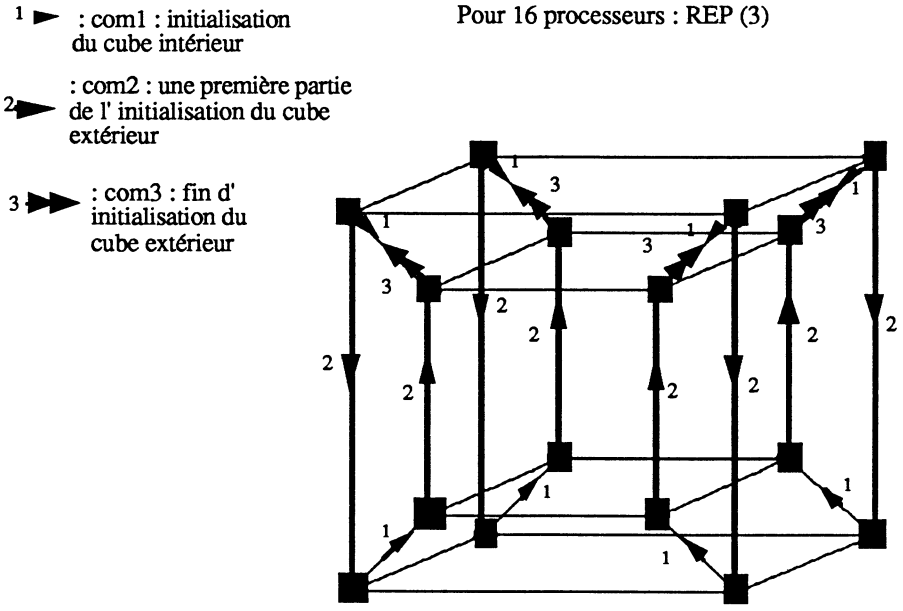
Sur un hypercube de dimension d (à $n = 2^d$ processeurs) la première étape de P2RGBuch consiste en le calcul de sous-bases de Gröbner en parallèle ; on active tous les processeurs avec MGBuch. On effectue ensuite Hypn qui est constituée de :

- une phase de répartition : il nous faut répartir les résultats en deux ensembles disjoints. Pour cela, on utilise la procédure REP(D) où D = d-1 qui nous permet de considérer l'hypercube comme essentiellement composé de deux hypercubes de dimension d-1,

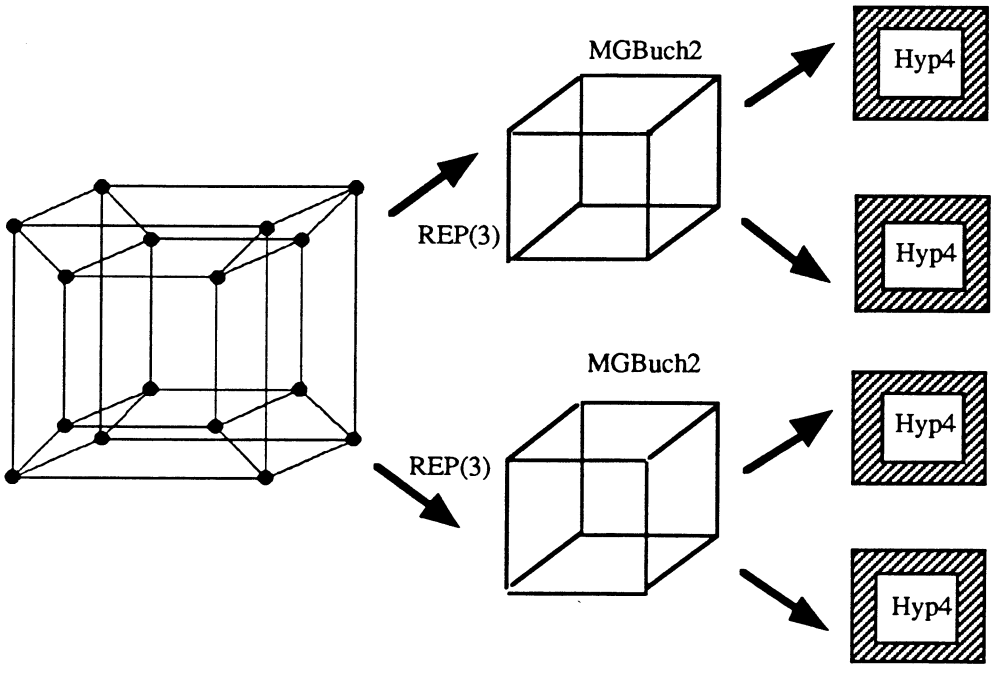
- une phase de calcul des nouvelles sous-bases : on effectue MGBuch2 sur les nouvelles entrées de chaque processeurs,
- une nouvelle phase de répartition REP(D-1),
- une nouvelle phase de calcul.....
- etc....

Cette itération s'achève lorque REP est paramétrée par 2. Auquel cas on effectue Hyp4 qui nous fournit pour chaque groupe de 4 processeurs deux bases de Gröbner (identiques) associées aux entrées.

On peut schématiser les trois phases de communication de REP sur 16 processeurs par (com1 et com2 se sont en parallèle) :



On peut schématiser Hyp16 de la façon suivante :



Hypn s'écrit :

Hypn ==

```
ENTREE : une sous-base G_k pour chaque processeur k.
SORTIE : La base de Gröbner G (associée à l'union de sous-bases)
         réduite pour les processeurs k tels que k_1 = 0.
  { variables intermédiaires }
  G: familles de polynômes ;
  D : entier ; { dimension initiale de l'hypercube }
  { initialisation }
  G := ∅ ;
  D := log2(n) -1 ;
  { chaque processeur k → (k_D, ....k_0) }
  pour tout k faire      { pour tous les processeurs }
      tant que D >= 2
          { communications }
          REP(D) ;
          { on obtient deux sous-bases par processeurs G_k
            et G_j }
          MGBuch2 { sur les sorties de REP }
          D := D-1 ;
      Hyp4 ;  { G est alors calculer dans 2 fois par blocs de 4
            processeurs }

  Sortir(G).
```

Fin de Hypn.

II-3-5 Quelques remarques

L'algorithme P2RGBuch sur n processeurs nous fournit alors la base de Gröbner associée à ses entrées calculée de n/2 manières différentes. Parmi les processeurs qui effectuent le dernier calcul de MGBuch2, celui qui termine le premier conditionne l'arrêt de l'algorithme complet.

Du moins, il devrait en être ainsi, malheureusement le modèle d'architecture que nous avons à notre disposition ne nous le permet pas. Le protocole de rendez-vous et le fait que les calculs et les communications se font séquentiellement nous en empêche. Il faudrait mettre en place un processus qui permette à chaque processeur de recevoir (et surtout de lire) un message en permanence, qui ralentirait considérablement l'algorithme.

En pratique c'est l'utilisateur qui envoie une commande d'arrêt dès qu'un processeur lui fournit un résultat. Les temps, utilisés pour les comparaisons, sont alors ceux du processeurs qui a terminé le premier.

Nous pouvons remarquer que le nombre de communications (c'est à dire le nombre de β) est le même quelles que soient les données traitées.

Il serait intéressant d'avoir un processus aléatoire de répartition des données, que ce soit sur tout l'hypercube ou plus simplement au niveau de chaque processeur. Il n'a pas été mis en place, par manque de temps, mais aussi de moyen : communiquer aléatoirement sur une machine telle que le T40 demande la mise en place de procédures de communications non triviales .

LISTE DES OPERATIONS INTRODUITES DANS LE CHAPITRE II :

CHAPITRE III

RESULTATS EXPERIMENTAUX DE LA PARALLELISATION

0- AVERTISSEMENT

On a l'habitude de comparer les performances d'un algorithme parallèle en calculant les rapports des temps d'exécution obtenus pour des nombres différents de processeurs.
Cette méthode qui donne de bons critères de comparaison dans le cas de problèmes purement 'déterministes' (Gauss par exemple) ne nous a pas semblé convenir dans le cas d'un problème tel que le calcul d'une base de Gröbner. En effet il est bien connu que les temps de calcul (même en séquentiel) sont totalement imprédictibles et quasiment aléatoires suivant l'exemple traité.

Nous avons donc laissé de coté les critères 'standards' de comparaison et nous avons essayé de nous intéresser directement aux temps d'exécution des algorithmes dans le cas séquentiel et parallèle (anneau ou hypercube). Il nous a alors paru évident que le comportement asymptotique (c'est à dire pour des temps d'exécution en séquentiel très importants) devait être un des critères principaux de comparaison.
Il ne sert en effet à rien de se limiter à des comparaisons du type "l'algorithme parallèle s'exécute 2 fois plus vite que son homologue séquentiel" si le gain le temps de calcul n'est que de quelques dixièmes de secondes. En revanche il est beaucoup plus important de savoir comment se comporte le gain de temps lorsque les temps de calcul sont élevés.

I- LA REPARTITION DES DONNEES

Comme souligné dans le chapitre I, la répartition des données a une influence sensible sur le temps d'exécution des algorithmes de calcul d'une base de Gröbner. Influence d'autant plus notable qu'il s'agit ici de répartir les polynômes entre différents processeurs.

I-1 Le choix d'une répartition

Nous avons choisi une stratégie de répartition qui, comme nous l'avons déjà noté dans l'introduction, permet de faire varier le nombre de processeurs de façon à obtenir des résultats interprétables. Dans un premier temps on suppose que les données sont équidistribuées.

Regardons tout d'abord différentes répartitions :

sur quatre processeurs : soit F = { $p_1, p_2,...p_8$} avec :

$p_1 = x_4x_5 + x_3x_4 + x_2$; $p_5 = x_6x_8 + x_5x_4 + x_3$;

$p_2 = x_5 + x_2x_3$; $p_6 = x_6x_8$;

$p_3 = x_5 + x_4$; $p_7 = x_7 + x_6x_5$;

$p_4 = x_4x_5 + x_3 + x_1$; $p_8 = x_8x_7 + x_6$.

On répartit les polynômes de manière équilibrée sur les différents processeurs. Soit la répartition suivante :

le processeur numéro 0 contient les polynômes p_3, p_4
le processeur numéro 1 contient les polynômes p_1, p_2
le processeur numéro 2 contient les polynômes p_5, p_6
le processeur numéro 3 contient les polynômes p_7, p_8

l'algorithme parallèle P1RGBuch s'exécute alors en : 5082ms

On peut également considérer la répartition suivante :

le processeur numéro 0 contient les polynômes p_6, p_2
le processeur numéro 1 contient les polynômes p_5, p_3
le processeur numéro 2 contient les polynômes p_1, p_7
le processeur numéro 3 contient les polynômes p_8, p_4

l'algorithme parallèle P1RGBuch s'exécute alors en : 7051ms

D'autres répartitions sont possibles telles que :

le processeur numéro 0 contient les polynômes p_1, p_2
le processeur numéro 1 contient les polynômes p_3, p_4
le processeur numéro 2 contient les polynômes p_5, p_8
le processeur numéro 3 contient les polynômes p_7, p_6

l'algorithme parallèle P1RGBuch s'exécute alors en : 5126ms

ou encore :

le processeur numéro 0 contient les polynômes p_1, p_3
le processeur numéro 1 contient les polynômes p_4, p_2
le processeur numéro 2 contient les polynômes p_5, p_6
le processeur numéro 3 contient les polynômes p_7, p_8

l'algorithme parallèle P1RGBuch s'exécute alors en : 6528ms

Il existe encore d'autres répartitions.

Ce que l'on remarque surtout est la variation du temps d'exécution de l'algorithme. Ce temps pourrait passer du simple au double !

Une des explications possibles est la suivante :

> avant d'entamer les procédures de communication et de calculer d'autres polynômes, l'algorithme P1RGBuch consiste, sur chaque processeur en le calcul des spolynômes et en la normalisation de ces derniers par rapport à ceux dit de référence. On appelera par la suite cette étape *étape d'initialisation.* Si l'on réparti les polynômes en faisant en sorte de calculer le plus grand nombre de spolynômes au cours de cette étape, les communications et les calculs qui s'ensuivent s'en trouvent nettement diminués.

Prenons le cas extrême où parmi les données se trouvent 4 polynômes p_1, p_2, p_3, p_4 tels que :

$Pgcd(Pm(p_1) , Pm(p_2)) = 1$, $Pgcd(Pm(p_3) , Pm(p_4)) = 1$,

tandis que :

$Pgcd(Pm(p_1) , Pm(p_3)) \neq 1$, $Pgcd(Pm(p_2) , Pm(p_4)) \neq 1$.

Si p_1 et p_2 sont dans un même processeur et p_3 et p_4 dans un autre, les deux processeurs ne calculent que les spolynômes existants entre les polynômes p_1, p_2, p_3, p_4 et les p_i pendant l'étape d'initialisation ce qui correspond au calcul minimun de cette étape. Il faut alors activer l'étape de communication pour commencer effectivement le calcul des spolynômes ; étape que l'on aurait pu éviter en distribuant différemment les polynômes de façon à avoir p_1 et p_3 dans un même processeur et p_2 et p_4 dans un autre.

On retrouve ce phénomène dans l'exemple donné.

On met alors au point une stratégie de répartition essentiellement basée sur "l'état relatif des monômes de têtes des différents polynômes". Cette stratégie est également utilisée pour répartir les polynômes et utiliser P2RGBuch ; cet algorithme débutant par un calcul de bases de Gröbner sur chaque processeur il est alors fait en sorte que ce calcul soit efficace.

I-2 La représentation

On peut représenter la famille de polynômes par un graphe orienté G. Plus précisement on procède de la façon suivante :

Soit A l'ensemble des arêtes du graphe et X l'ensemble de ses sommets.

A chaque sommet s_i de X correspond le monôme de tête m_i de chaque polynôme p_i de F.

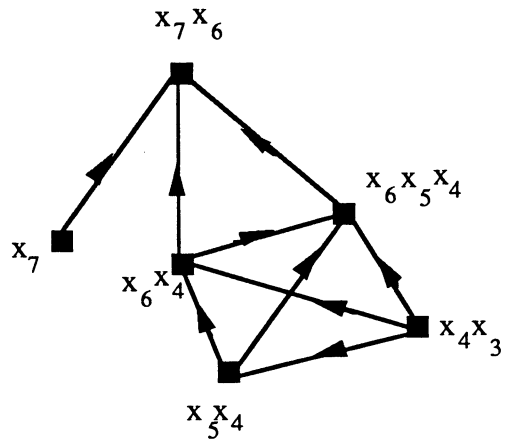
Un élément $a = (s_1, s_2)$ où s_1 et s_2 deux sommets de X est dans A si et seulement si :

$$Pgcd(m_1,m_2) \neq 1 \text{ et } m_1 < m_2$$

Exemple :

soit $F = \{x_7 + x_6, x_7x_6 + x_2, x_4x_3 + x_4 + x_2, x_4x_5 + x_2, x_4x_5x_6 + x_3, x_4x_6 + x_2\}$

F est représentée par le graphe suivant :



En pratique, le graphe correspondant à une famille de polynômes est construit à l'aide de deux tableaux : T qui correspond aux indices des polynômes de la famille rangés suivant l'ordre décroissant de leurs monômes de tête et ADJ qui contient les adjacents de chaque sommet.

De tels graphes ne sont pas toujours connexes.

La répartition des polynômes dans différents processeurs peut alors se ramener à la construction de classes de sommets telles que les monômes correspondants à ces sommets aient des variables communes, c'est à dire à la recherche de composantes connexes ou encore de chemins dans le graphe.

i) utilisation de la connexité :

le cas idéal serait alors le suivant : Le graphe est formé de p composantes connexes équilibrées c'est à dire :

m = qp+r ou m est le nombre de sommets du graphe,

(p-r) composantes contiennent q sommets, et r en contiennent q+1 ;

Malheureusement ce cas ne se produit que rarement.

L'idée pourrait être la suivante :

déterminer un "degré" de connexité entre des différentes composantes connexes. Ce degré pourrait par exemple nous donner la longueur du plus long chemin dans une composante de son plus petit sommet jusqu'à un point d'articulation du graphe.
Cette idée pourrait être approfondie dans le cas où les polynômes de la famille correspondante ont des monômes de têtes sans "beaucoup de variables communes" ce qui recouvre qu'une classe d'exemples très réduite.

ii) utilisation de la notion de chemin :

plus simple à aborder cette notion est liée à l'exploration d'un graphe par un examen exhaustif des sommets.
Un *chemin* dans un graphe orienté est une suite de sommets telle qu'il existe un arc entre deux sommets consécutifs toujours orienté dans le sens "ascendant".
La représentation informatique du graphe conditionne le choix des algorithmes de parcours nécessaires à la construction de chemins.

C'est cette dernière notion que l'on utilise et on met au point la stratégie ci-dessous.

I-3 Stratégie à p et l fixés

On utilise la notion de chemin dans un graphe pour construire p "chemins" disjoints de longueur l où l est la partie entière du quotient de m/p m étant le nombre de sommets.

On note *"chemin"* tout parcours d'un sommet à un autre sans tenir compte de l'orientation.

Chaque "chemin" nous fournit un ensemble de polynômes à stocker dans la mémoire d'un même processeur que l'on appelle *la donnée d'un processeur*. Cette stratégie correspond à une équidistribution des données.

Pour celà on procède à la recherche d'un chemin de longueur maximum l à l'aide d'un algorithme classique d'exploration en profondeur [Xuong] que l'on note EXP(l). Il se peut que le chemin C obtenu soit de longueur c_l inférieur à l.

Divers cas se présentent :

i) cl = l, le chemin C correspond à la donnée d'un processeur, on le supprime dans le graphe en utilisant une procédure qui modifie G en remettant à jour A et X que l'on note MODIF(G, C),

ii) cl < l, on explore le graphe de manière à déterminer un "chemin" orienté des sommets les plus hauts (à partir du dernier sommet de C) vers les sommets les plus bas qui prolonge C à l'aide de la procédure PROL(C) qui utilise elle aussi un procédé d'exploration en profondeur. Si on a réussi à prolonger le chemin intial pour obtenir un "chemin" C de longueur l il correspond à la donnée d'un processeur, on le supprime du graphe. Sinon, on le met de coté en marquant les sommets du graphes par lesquels il passe a l'aide de la procédure MARC(C).

A partir de ces diverses constructions on construit alors l'algorithme de répartition suivant :

```
REP ==
|  ENTREE : un graphe G =(A,X).
|  SORTIE : p "chemins" de longeur l.
|      i := 1;
|      Tant que (il existe s de X non marqué)
|          |  { la recherche de C_i se fait sur les sommets non marqués
|          |   de G, en commençant par le plus petit }
|          |  C_i := EXP(l) ;
|          |  c_li := longueur du "chemin" C_i ;
|          |  si (c_li = l)  MODIF(C,G) ;
|          |  sinon
|          |      |  PROL(C_i) ;
|          |      |  c_li := longueur du chemin C_i ;
|          |      |  si (c_li = l)  MODIF(C,G) ;
|          |      |  sinon MARC(C) ;
|      TRAITFIN ;
|      Sortir(C_1,...C_p).
fin de REP.
```

La procédure TRAITFIN correspond au regroupement des sommets marqués restant dans le graphe en p-s chemins de longueur de l, si s est le nombre de "chemins" déjà construits. On conserve les "chemins" auquels appartiennent les sommets restant. A partir de ces "chemins" on construit des chemins de longueurs l en les "concaténant".

Il reste encore r sommets dans le graphe, on les regroupe au mieux avec les "chemins" déjà construits afin d'obtenir l'équivalent de (p-r) composantes contenant q sommets, et r en contenant q+1, et l'on obtient la répartition désirée.

Dans l'exemple précité si l'on veut répartir les 6 polynômes sur 3 processeurs, on obtient 3 chemins de longueur 2 :

$$(x_4x_3,\ x_5x_4)\ ;\ (x_6x_4,\ x_6x_5x_4)\ ;\ (x_7,\ x_6x_7).$$

Cette stratégie n'est pas forcement la meilleure. Quelle est d'ailleurs la meilleure ? au vu des exemples traités il est difficile de répondre à cette question. On peut tout de même affirmer que la répartition obtenue à l'aide de la stratégie donnée ici, fait en général partie de celles qui fournissent les meilleurs temps d'exécution.

Sur l'exemple des quatre processeurs, avec F = { $p_1, p_2,...p_8$} où :

$p_1 = x_4x_5 + x_3x_4 + x_2$; $p_5 = x_6x_8 + x_5x_4 + x_3$;

$p_2 = x_5 + x_2x_3$; $p_6 = x_6x_8$;

$p_3 = x_5 + x_4$; $p_7 = x_7 + x_6x_5$;

$p_4 = x_4x_5 + x_3 + x_1$; $p_8 = x_8x_7 + x_6$.

la stratégie mise au point nous fournit la répartition suivante :

le processeur numéro 0 contient les polynômes p_2, p_3
le processeur numéro 1 contient les polynômes p_4, p_1
le processeur numéro 2 contient les polynômes p_7, p_8
le processeur numéro 3 contient les polynômes p_5, p_6

l'algorithme parallèle P1RGBuch s'exécute alors en : 4896ms ce qui est tout à fait comparable aux temps d'exécution de P1RGBuch obtenus avec les autres répartitions.

I-4 D'autres stratégies possibles

Le temps ne nous a pas permis d'aller très loin dans l'étude des différentes stratégies possibles, pourtant intéressantes tant au niveau des résultats pouvant être obtenus qu'au niveau de la recherche elle-même de ces stratégies.

Au lieu de fixer la longueur et le nombre de "chemins" à obtenir (i.e le nombre de processeurs), on peut faire en sorte que ce soit la stratégie elle-même qui nous fournisse le nombre et la longueur des "chemins" à construire. En effet on peut chercher parmi les sommets non marqués du graphe, le chemin le plus long. Une telle stratégie nous fournit alors une répartition qui n'est pas forcément équilibrée mais qui peut dans certains cas s'avérer meilleure :

Exemple :
on considère un exemple à 8 polynômes.

Si l'on impose une répartition à l et q fixés 2 et 4 respectivement, l'exécution de P1RGBuch s'effectue en 14326ms (sur 4 processeurs avec 2 polynômes chacun).

Si on utilise une stratégie analogue à celle décrite ci-dessus on obtient l = 4 et q = 2. L'exécution de P1RGBuch s'effectue en 8565ms (sur 2 processeurs avec 4 polynômes chacun). La répartition est équilibrée, mais il se peut que ce ne soit pas le cas.

On peut remarquer ici que l'optimum du temps d'exécution n'est pas obtenu pour le plus grand nombre de processeurs. Nous allons voir, par la suite, que plus d'un critère entre en jeu pour étudier expérimentalement les algorithmes parallèles et que les résultats de la parallélisation sont étonnants à bien des niveaux.

I-5 Des familles et des processeurs

Nous avons découpé une famille de polynômes, en différentes familles, puis les familles construites ont été réparties entre les processeurs de manière simple en stockant la première famille dans le processeur numéro 0 puis la deuxième dans le processeur numéro 1, etc...

Soit l'exemple suivant :

sur quatre processeurs : soit $F = \{ p_1, p_2,...p_8 \}$ avec :

$p_1 = x_1x_2 + x_1$; $p_5 = x_4x_5x_8 + x_2x_3 + x_2$;

$p_2 = x_1x_3 + x_2$; $p_6 = x_5x_7 + x_6x_5$;

$p_3 = x_3x_2 + x_2x_1$; $p_7 = x_6x_8 + x_4x_5 + x_3$;

$p_4 = x_2x_4x_5 + x_3 + x_1$; $p_8 = x_8x_9x_{10} + x_2x_3$.

Avec la stratégie précédente,

le processeur numéro 0 contient les polynômes p_1, p_2
le processeur numéro 1 contient les polynômes p_3, p_4
le processeur numéro 2 contient les polynômes p_5, p_6
le processeur numéro 3 contient les polynômes p_7, p_8

l'algorithme s'exécute en 35355 ms

Si on repartit les mêmes familles sur des processeurs différents par exemple si :

le processeur numéro 0 contient les polynômes p_7, p_8
le processeur numéro 1 contient les polynômes p_3, p_4
le processeur numéro 2 contient les polynômes p_5, p_6
le processeur numéro 3 contient les polynômes p_1, p_2

Le temps d'exécution de l'algorithme est alors : 32380 ms

ou encore si :

le processeur numéro 0 contient les polynômes p_3, p_4
le processeur numéro 1 contient les polynômes p_5, p_6
le processeur numéro 2 contient les polynômes p_7, p_8
le processeur numéro 3 contient les polynômes p_1, p_2

Le temps d'exécution de l'algorithme est alors : 38914ms

On voit apparaître sur cet exemple des variations de temps non négligeables. Ceci est dû à la "proximité" dans l'anneau des polynômes ayant des variables communes, et à l'ordre des réductions qu'induit la répartion des polynômes sur l'anneau.

Pour étudier les variations de temps d'exécution des algorithmes en fonction du nombre de processeurs mis en oeuvre, nous gardons la stratégie décrite plus haut.

Nous reparlerons du comportement des algorithmes en fonction de la répartition des données.

II- LE CAS DE L'ANNEAU

II-1 Le cadre de l'étude

•• L'étude qui suit a été faite avec des exemples assez homogènes c'est à dire avec des polynômes à une quinzaine de variables, et un nombre fixé des monômes par polynômes (de 1 à 5). On se limite à de tels exemples pour diverses raisons :

i) leurs comportements sont anologues même si on enregistre certaines fluctuations,

ii) le temps de calcul reste raisonnable ce qui permet d'exécuter l'algorithme sur plus d'exemples.

•• Pour des exemples moins homogènes il semble, au vu de l'étude faite ici, que ce soient les anneaux de taille conséquente (la taille d'un anneau étant le nombre de processeurs qu'il contient) qui seront les plus performants.

•• Les temps moyens dont on parle par la suite sont obtenus en calculant la moyenne des temps d'exécution d'un certain nombre d'exemples (de 7 à 10 suivant le cas) ayant le même nombre de polynômes.

•• Nous avons choisi d'étudier le comportement de l'algorithme en fonction du nombre de polynômes d'entrée. Il est bien évident que ce n'est pas le seul critère à considérer : l'influence du nombre de monômes et du nombre de variables ne sont pas négligeables. Ce choix nous permet d'avoir des exemples comparables comme on le souligne plus haut et ce critère est très facilement quantifiable.

•• Nous montrons ensuite, comment on peut faire la synthèse de résultats obtenus.

Dans toute la suite, on note $\mathcal{A}(n)$ l'anneau de dimension n (qui contient 2^n processeurs).

II-2 La réduction

Nous avons déjà parlé du choix de la réduction, dans le cas où l'algorithme est séquentiel, en comparant diverses réductions : MRéd, Bréd, Réd. Nous choisirons par la suite de conserver BRéd, au vu des tests effectués sur l'algorithme séquentiel, mais également au vu des résultats comparatifs suivants :

On note A1 l'algorithme P1RGBuch implanté avec MRéd comme procédure de réduction et A2 le même algorithme avec comme procédure de réduction BRéd.

(i) avec 4 processeurs on obtient la base de Gröbner d'un exemple à 24 polynômes en :

avec A1 15612 ms

avec A2 10689 ms

(ii) avec 8 processeurs on obtient la base de Gröbner d'un exemple à 24 polynômes en :

avec A1 17734 ms

avec A2 12549 ms

(iii) avec 32 processeurs on obtient la base de Gröbner d'un exemple à 32 polynômes en :

avec A1 122905 ms

avec A2 45448 ms

Les temps d'exécution sont liés à la taille des exemples traités. En général il semble que se soit avec A2 que l'on calcule la base de Gröbner le plus rapidement.

En dehors du choix de la réduction, nous avons également le choix au niveau du lieu d'implantation de cette réduction au sein même de l'algorithme. Nous avons retenu, pour leurs comparaisons, deux possibilités différentes :

on active la procédure de réduction, le plus souvent possible, ce qui nous fournit l'algorithme RM (sans réduction des entrées) ;

on ne réduit que les polynômes obtenus par rapport aux polynômes de référence et on effectue à la fin du programme une réduction globale, grâce à l'algorithme RI.

on obtient alors les résultats suivants (avec Bréd) :

On reprend les exemples précédents

(i) temps d'exécution de l'algorithme RM : 10689 ms
temps d'exécution de l'algorithme RI : 13116 ms

(ii) temps d'exécution de l'algorithme RM : 12549 ms
temps d'exécution de l'algorithme RI : 14512 ms

(iii) temps d'exécution de l'algorithme RM 45448 ms
temps d'exécution de l'algorithme RI : 128140 ms

INTERPRETATION :

en augmentant le nombre de réductions faites à chaque étape, on diminue la taille globale des données en éliminant au fur et à mesure les polynômes calculés. Ceci semble entraîner, à la fois une diminution des données à traiter par la suite, mais également du nombre global d'itérations de TL. La diminution du nombre de ces itérations, lorqu'elle est notable, semble largement compenser l'augmentation du nombre de calculs effectués à chacune d'elles. De plus, même si le nombre d'itérations ne diminue pas on note une diminution du nombre de spolynômes calculés et donc une diminution de nombre total des normalisations.

Plus précisement dans les exemples :

(i) le plus grand nombre de spolynômes calculés à l'intérieur d'un processeur :

pour RM : 280 pour RI : 390

le plus grand nombre d'appels à la fonction Norm dans un même processeur :

pour RM : 1848 pour RI : 2100

le nombre d'itérations, quelque soit l'algorithme : 2

(ii) le plus grand nombre de spolynômes calculés à l'intérieur d'un processeur :

pour RM : 212 pour RI : 267

le plus grand nombre d'appels à la fonction Norm dans un même processeur :

pour RM : 1088 pour RI : 1225

le nombre d'itérations, quelque soit l'algorithme : 2

(iii) le plus grand nombre de spolynômes calculés à l'intérieur d'un processeur :

pour RM : 349 pour RI : 349

le plus grand nombre d'appels à la fonction Norm dans un même processeur :

pour RM : 2487 pour RI : 3184

le nombre d'itérations, quelque soit l'algorithme : 1

On utilise par la suite l'algorithme correspondant à RM.

II-3 La réduction des entrées

L'algorithme RM proposé ci dessus ne débute pas par une réduction des entrées. Pour obtenir un algorithme séquentiel performant il est pourtant nécessaire de le faire :

sur un exemple à 48 polynômes le temps d'exécution de l'algorithme $\mathcal{A}(1)$ sans réduction des entrées est 1289 secondes. Si l'algorithme comprend la réduction des entrées ce temps devient de 35.6 secondes soit 36 fois plus faible !!!.

Pour l'algorithme parallèle la réduction des entrées peut être faite de diverses manières :

i) on peut répartir les données dans les différents processeurs et réduire dans chaque processeur les données réparties. On appelle cette première stratégie REP-RED.

ii) on peut réduire les entrées dans un processeurs puis répartir les polynômes résultats de la réduction. On appelle cette deuxième stratégie RED-REP.

Sur deux processeurs l'exemple précédent nous fournit les temps suivants :

avec la stratégie REP-RED : 40.49 secondes
avec la stratégie RED-REP : 26.17 secondes

Il est certain que si les entrées ne nécessitent pas de réduction l'opération est alors inutile mais c'est le pire des cas.

Pour étudier la parallélisation des algorithmes avec réduction des entrées faut-il comparer les algorithmes avec la première ou la deuxième stratégie ?

On considère les "petits exemples" suivants (les 4 premiers sont composés de 4 polynômes les 4 suivants de 8) où AR correspond aux algorithmes avec réduction SR aux algorithmes sans réduction et où l'on note les deux stratégies possibles sur les algorithmes parallèles. Les temps donnés sont en secondes :

$\mathcal{A}(0)$		$\mathcal{A}(1)$			$\mathcal{A}(2)$		
SR	AR	SR	AR		SR	AR	
			RED-REP	REP-RED		RED-REP	REP-RED
3.02	1.85	4.16	3.72	4.15	9.05	10.14	9.09
2.93	1.78	3.18	3.43	3.20	8.55	5.32	8.55
3.48	2.34	2.75	1.95	2.76	9.10	5.37	9.13
11.36	11.18	9.56	9.56	9.58	26.70	26.73	26.59
4.46	2.80	5.41	2.64	3.34	4.68	5.68	4.71
5.56	1.88	4.07	2.09	3.03	5.92	3.43	5.88
5.70	1.00	4.85	1.76	4.57	5.77	2.62	5.68
6.42	1.03	4.79	1.76	4.55	7.65	2.40	7.50

INTERPRETATION :

La réduction locale est beaucoup moins efficace puisque que l'on répartit les données. Elle s'effectue alors sur moins de polynômes et n'est pas complête en ce sens qu'il faudrait que les polynômes de différents processeurs se rencontrent. La réduction est naturellement séquentielle et au niveau du parallélisme il semble qu'il faille comparer les algorithmes avec la deuxième statégie.

La réduction est alors considérée comme un prétraitement des données. Elle permet de réduire la taille des entrées et l'étude du parallélisme est faite sur des données de taille moins conséquente.

Ce qui nous intéresse dans cette étude ce ne sont pas les performances des algorithmes en elles-même. On veut évaluer l'apport du parallélisme sur un algorithme de type RBuch et plus précisement son apport "asymptotique" (c'est à dire pour des temps d'exécution en séquentiel très importants).

Dans l'étude qui suit on ne réduit pas les entrées, les données traitées fournissent alors un temps d'exécution de l'algorithme séquentiel important. Ceci nous permet de simuler le comportement asymptotique des algorithmes parallèles et séquentiels avec réduction des entrées.

II-4 L'occupation mémoire

Le C, langage utilisé ici, nous a permis de faire des allocations dynamiques de la mémoire.

Pour chaque opération élémentaire du programme, la place non utilisée est libérée, ce qui permet de travailler avec des données de taille conséquente. Il est bien évident, que cette gestion mémoire augmente le temps de calcul par rapport à un programme qui n'utiliserait que des variables fixes en mémoire.

On pourrait envisager de n'utiliser la gestion mémoire que pour des calculs la nécessitant, nous avons pas implanté un tel processus.

On s'intéresse sur un exemple aux variations de la taille mémoire utilisée dans le programme, de la taille maximum occupée, et de la taille qui serait occupée si l'on n'avait pas de gestion mémoire.

Exemple :

Sur 4 processeurs on répartit les polynômes suivants :

$x_4x_5 + x_3x_2 + x_2$;

$x_5 + x_3x_2$;

x_5+x_4 ;

$x_4x_5 + x_3 + x_1$;

$x_6x_8 + x_5x_4 + x_3$;

$x_7 + x_6x_5$;

$x_8x_7 + x_6$;

On choisit, pour ses performances, un algorithme utilisant BRéd et RM, que l'on notera ici P.

La base de Gröbner de cet exemple est obtenue en trois étapes : trois exécutions de la boucle principale (cf description de PSGBuch dans le chapitre précédent) que l'on note E1, E2 et E3. Chacune de ces trois étapes peut être décomposée en 7 phases :

phase1 : calcul des spolynômes associés aux polynômes de référence.

phase2 : normalisation des spolynômes par rapport aux polynômes de réfence.

phase3 : calcul des spolynômes et normalisation de ces derniers sur l'anneau.

Remarquons que les phases de 1 à 3 constituent ce que l'on a noté plus haut TSL_i.

phase4 : réduction des polynômes calculés (utilisation de BRéd).

phase5 : normalisation des polynômes réduits par rapport aux polynômes de référence.

phase6 : les polynômes circulent sur l'anneau et y sont normalisés.

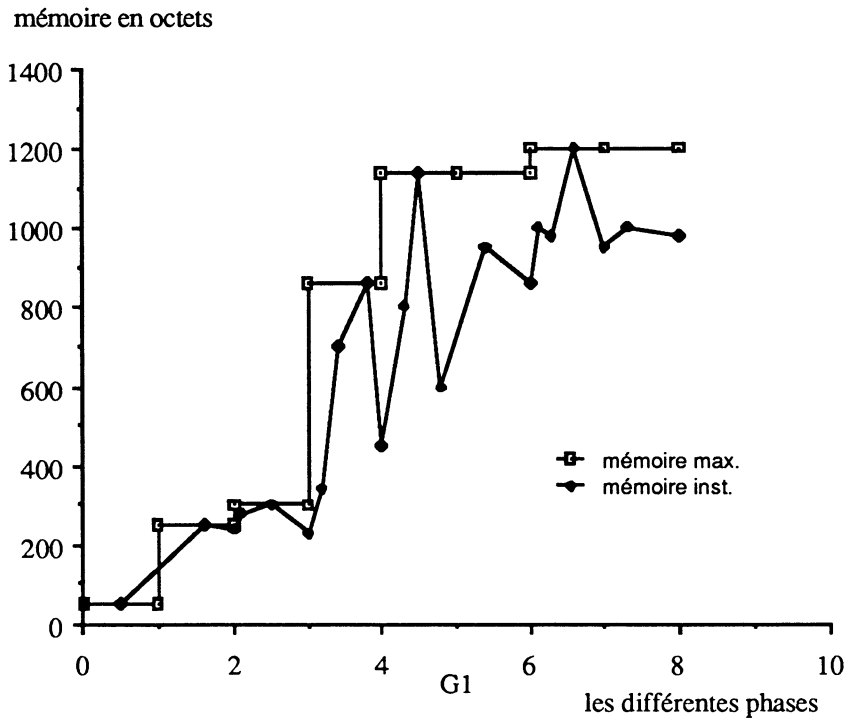
phase7 : nouvelle réduction des polynômes calculés (utilisation de BRéd)

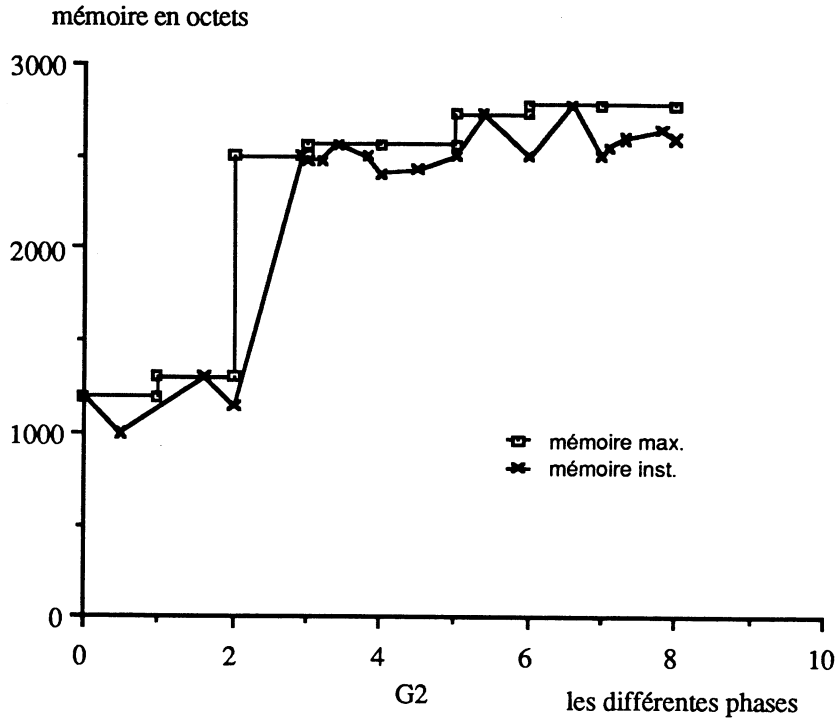
Les phases 5 et 6 constituent TN_i introduit plus haut.

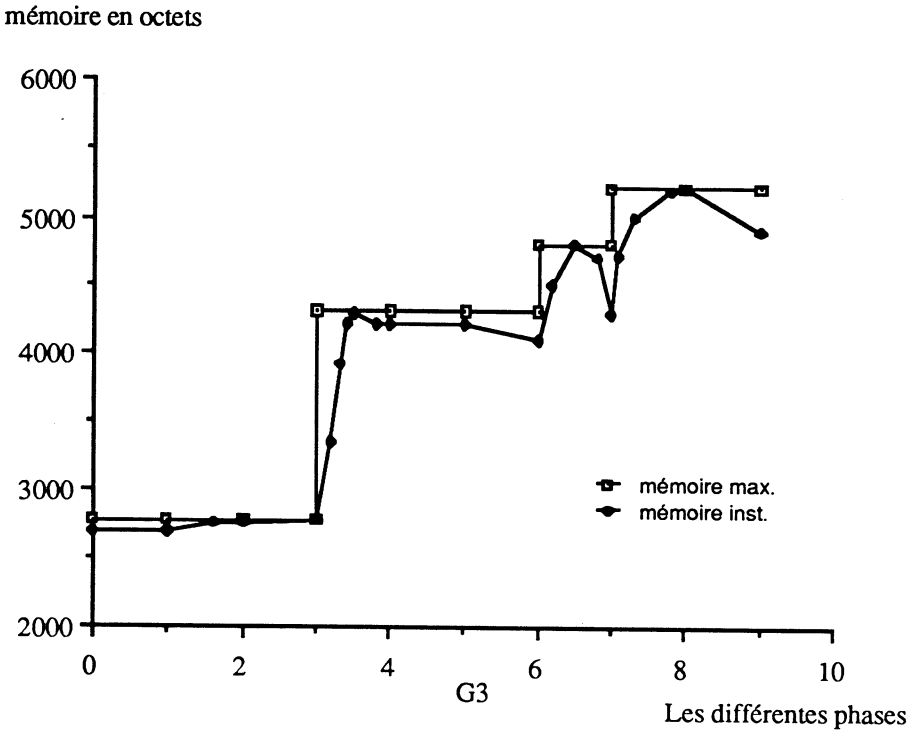
Pour l'étape E3 on ajoute une huitième phase qui correspond à la réduction de la base de Gröbner complete (utilisation de TN_i).

L'étape E1 est effectuée avec les entrées comme polynômes de référence, les étapes E2 et E3 avec comme polynômes de référence les polynômes résultats de l'étape précédente et des polynômes de référence de l'étape précédente. La phase1 est alors légèrement différente pour éviter les calculs redondants.

Pour chacune de ces trois étapes, on étudie ci-dessous **l'évolution des tailles mémoires maximale et instantanée pour les 7 (ou 8) phases successives.** A chaque étape Ei correspond un graphique Gi, qui représente ces variations mémoire en octets au cours du déroulement de l'étape considérée sur le processeur 1. En abcisse sont représentées les phases à 1 a 8 (au plus).







INTERPRETATION :

i) Les phases qui provoquent une augmentation sensible de la place mémoire utilisée par l'algorithme sont surtout les phases 3 et 6.

La phase3 est la phase de calcul de tous les spolynômes sur l'anneau, il est normal que cette étape consomme de la place mémoire pour stocker ses résultats. La phase6 correspond à la phase de normalisation sur l'anneau. Les processeurs

recoivent les données ce qui entraîne, une augmentation de la place mémoire utilisée à cet instant.

ii) Sur les deux premiers graphiques G1 et G2 on s'aperçoit que les normalisations entrainent une variation notable de la place mémoire occupée. Ces variations sont dues au fait que les polynômes simplifiés peuvent contenir plus de monômes que les polynômes avant simplifications.

iii) Si l'on calcule la taille mémoire nécessaire à un algorithme analogue sans libération mémoire, on trouve que les données manipulées par le processeur 1 occupent une place mémoire de 5.10^4 octets pour l'exemple donné (de taille : 54 octets).

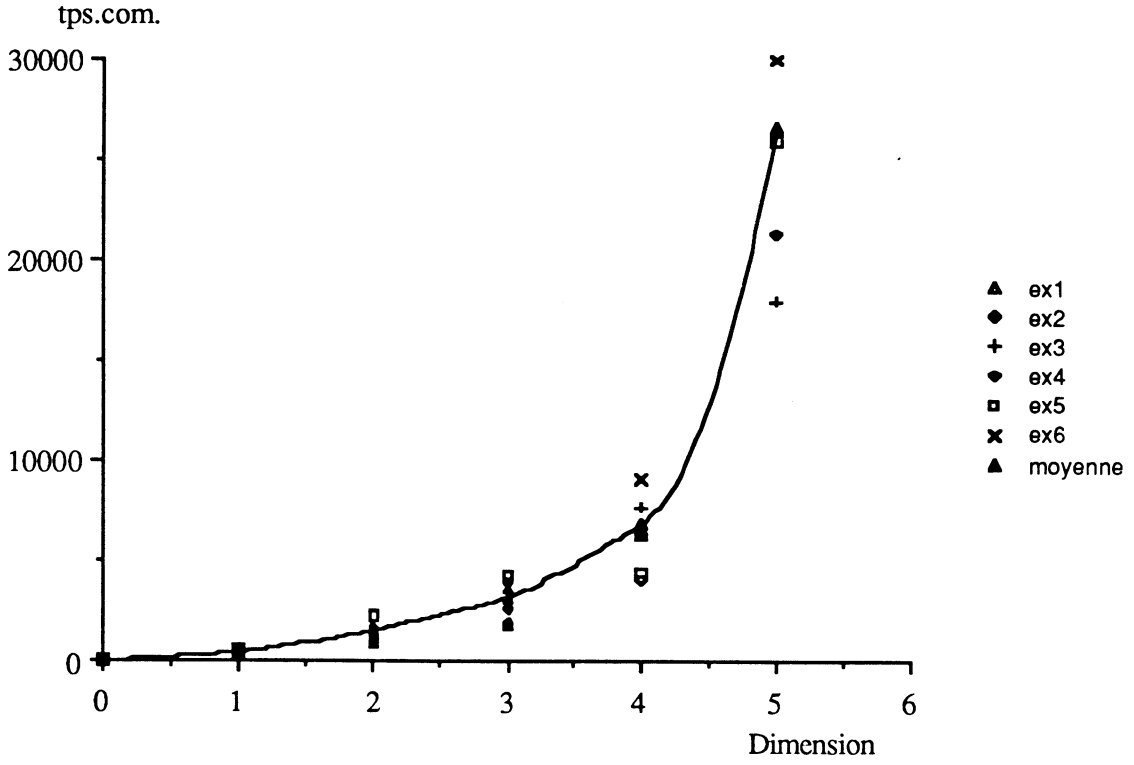
II-5 Le temps d'exécution

Nous avons déjà remarqué dans un exemple précédent, qu'en augmentant le nombre de processeurs, un gain de temps n'était pas systématiquement observé.

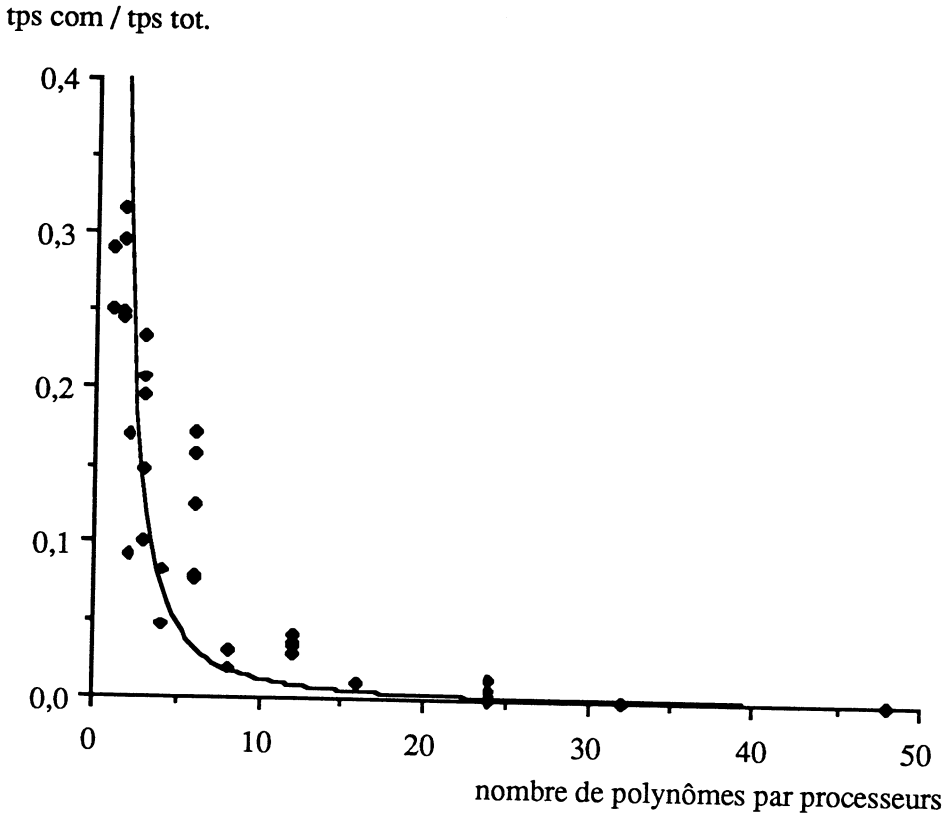
On explique l'absence de gain dans ce cas particulier par une prépondérance des temps de communication sur les temps de calculs. Sur certains exemples, on a pu constater que obtention d'une base de Gröbner imposait de nombreuses communications. Pour des exemples demandant peu d'opérations arithmétiques, l'efficacité de l'exécution paralléle peut alors être très mauvaise.

II-5-1 Influence du temps de communication

Le temps de communication augmente en fonction du nombre de processeurs et peut devenir non négligeable devant le temps de calcul. Par exemple, si l'on représente en fonction de la dimension de l'anneau utilisé, le temps de communication (en moyenne sur plusieurs exemples) :



Le temps de communication augmente en fonction du nombre de processeurs utilisés. Pour évaluer l'influence du temps de communication sur l'ensemble du programme en fonction des données on représente le rapport temps de communication / temps total en fonction du nombre de polynômes par processeurs.



Remarque :

Losrqu'on augmente le nombre de processeurs on augmente le temps de communications pour deux raisons essentielles :

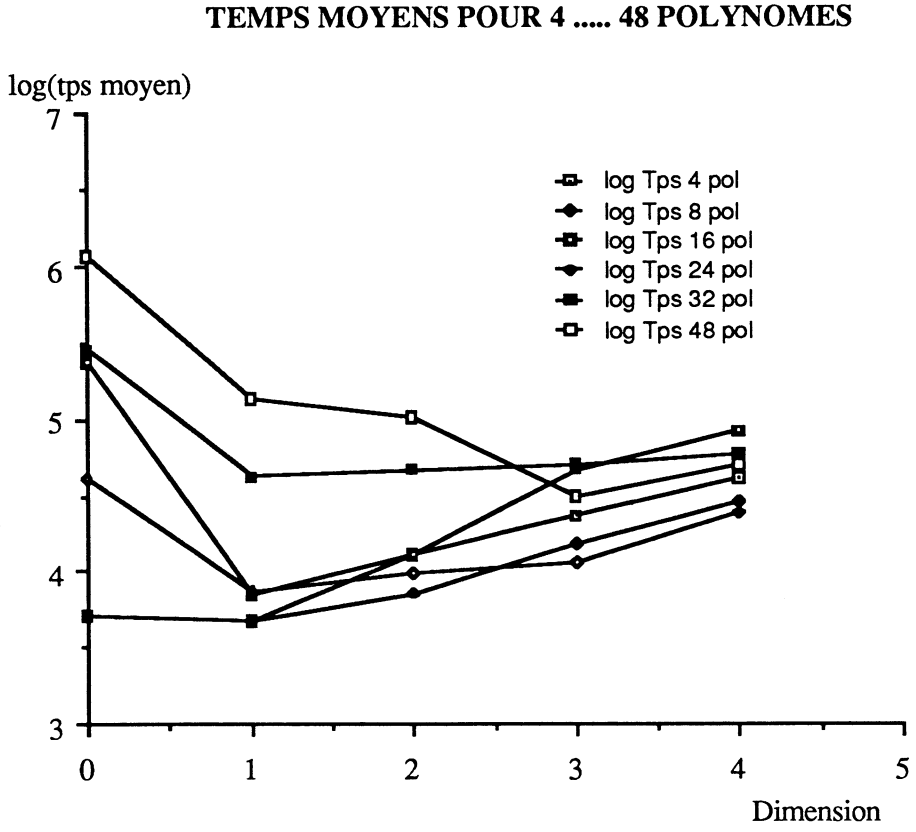
- la première est qu'en augmentant le nombre de processeurs, on augmente la taille de l'anneau, et les procédures de normalisations sur l'anneau sont plus longues.

- la deuxième est que la taille des données transférées est modifiée. En effet si l'on normalise les polynômes sur l'anneau, la taille des données à transferer varie en fonction des polynômes "normalisateurs" contenus dans chaque processeur.

Sur le deuxième graphique on remarque que plus la taille des données augmente plus le temps de communication devient négligeable vis à vis du temps total.

II-5-2 Variation du temps d'exécution total

.. Nous débutons cette étude par l'interprétation de la **variation du temps moyen en fonction de la dimension de l'anneau.**



On représente en fonction de la dimension de l'anneau le logarithme décimal du temps moyen pour exécuter un essai à n polynômes. La valeur n est celle figurant dans la légende.

INTERPRETATION :

i) un examen global de ce graphique montre qu'un anneau de dimension faible peut être, suivant la taille des données, plus performant qu'anneau de dimension élevée. C'est en effet le cas pour un petit nombre de polynômes où le programme s'exécute 10 fois plus lentement sur 16 processeurs que sur 1. En revanche ce classement tend à s'inverser lorsque le nombre de polynômes initiaux augmente (voir la courbe tracée pour 48 polynômes).

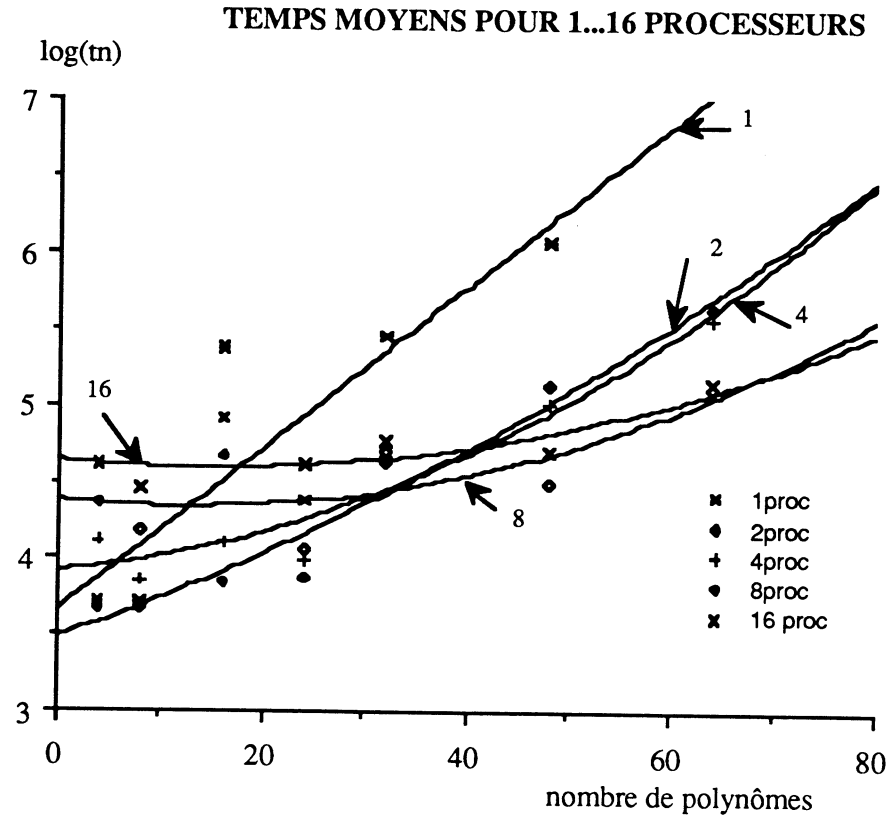
ii) Plus la dimension de l'anneau est faible et plus les écarts entre les temps d'exécution sont importants : l'anneau est d'autant plus sensible à une variation de

la taille des données que sa dimension est faible. Ainsi, pour une dimension égale à 4 les temps sont du même ordre de grandeur alors que pour une dimension égale à 0 on met 100 fois plus de temps pour obtenir une base sur 32 polynômes que sur 4.

iii) $\mathcal{A}(0)$ traite en des temps identiques des essais à 4 et 8 polynômes. $\mathcal{A}(1)$ traite en des temps équivalents des exemples jusqu'à 24 polynômes et présente pour 32 polynômes une variation de temps d'un rapport 10. C'est aussi le cas d'un anneau de dimension 3 mais à un degré moindre : le temps pour 32 polynômes n'étant que 2 fois et demi plus grand que pour 4 à 24 polynômes.
Ainsi tout se passe comme si $\mathcal{A}(1)$ pouvait traiter sans difficulté des exemples à une dizaine de polynômes, $\mathcal{A}(2)$ des exemples à une vingtaine de polynômes, $\mathcal{A}(3)$ des exemples à 25 polynômes et enfin $\mathcal{A}(4)$ des exemples à 35 polynômes.

iv) On met ainsi en évidence une sorte de 'capacité' d'un anneau de dimension donnée à traiter un problème de taille définie. Cette 'capacité' pourrait être définie comme le nombre de polynômes initiaux au delà duquel on observe une variation significative des temps d'éxécution pour une dimension fixée.
Il semblerait que cette capacité soit de l'ordre de quelques polynômes par processeurs (environ 3 ou 4 pour les essais effectués).
Cette observation, corroborée par l'étude de la figure suivante, ne fait que confirmer l'intuition selon laquelle il serait inutile d'utiliser un "Bazooka pour tuer une mouche".

•• Soit le graphe suivant :



Ce graphique représente **le logarithme décimal du temps moyen d'exécution d'exemples à n polynômes.** Chaque courbe est relative à un anneau de

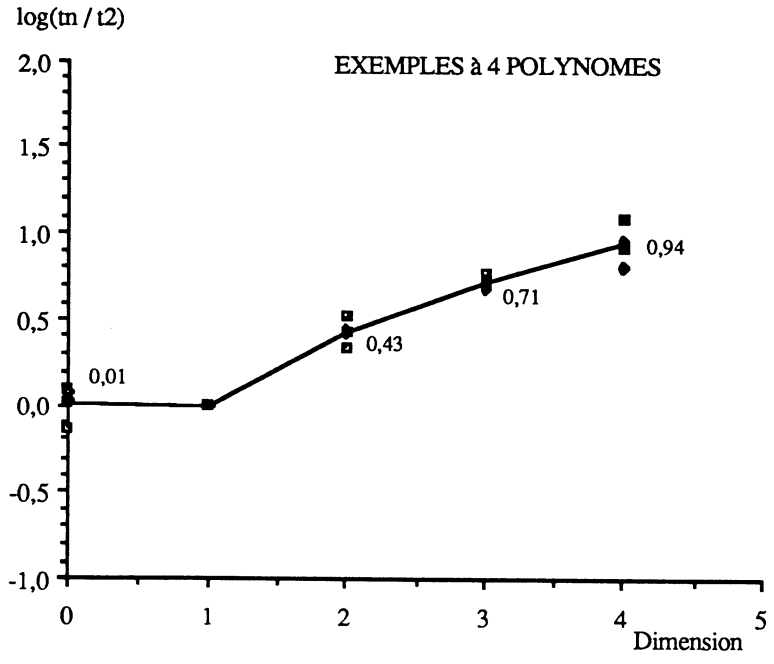
dimension donnée (0 à 4) et est obtenue au sens des moindres carrés des données expérimentales.

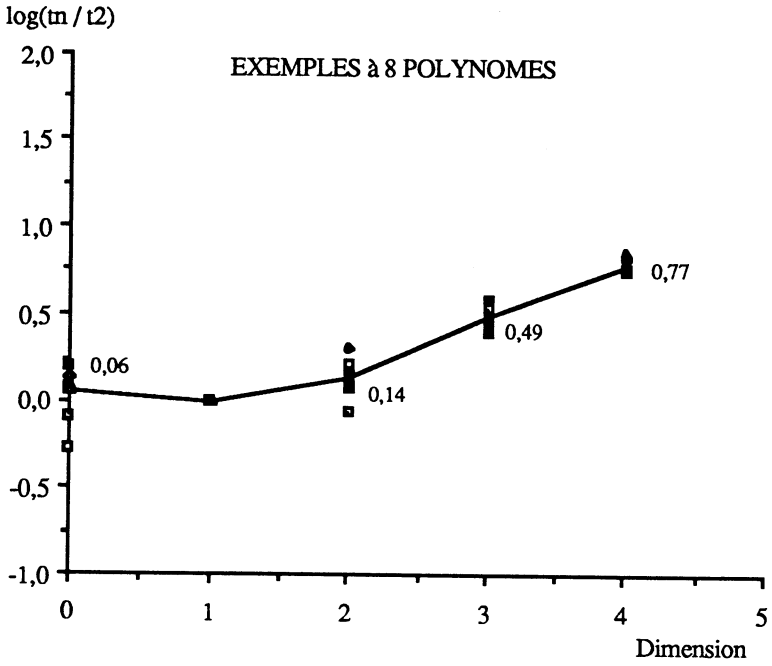
INTERPRETATION :

i) $\mathcal{A}(2)$ est toujours plus mauvais que $\mathcal{A}(1)$ et qui semble lui-être asympotiquement équivalent.

ii) un anneau de dimension donnée à un "handicap" (ordonnée à l'origine des courbes) plus important que les anneaux de dimensions inférieures mais qu'il est moins sensible à la taille des données que ces mêmes anneaux (courbes croissant moins rapidement).
Ainsi un anneau de dimension donnée peut devenir pour un nombre suffisamment grand de polynômes de départ plus performant que les anneaux de dimensions plus faibles.

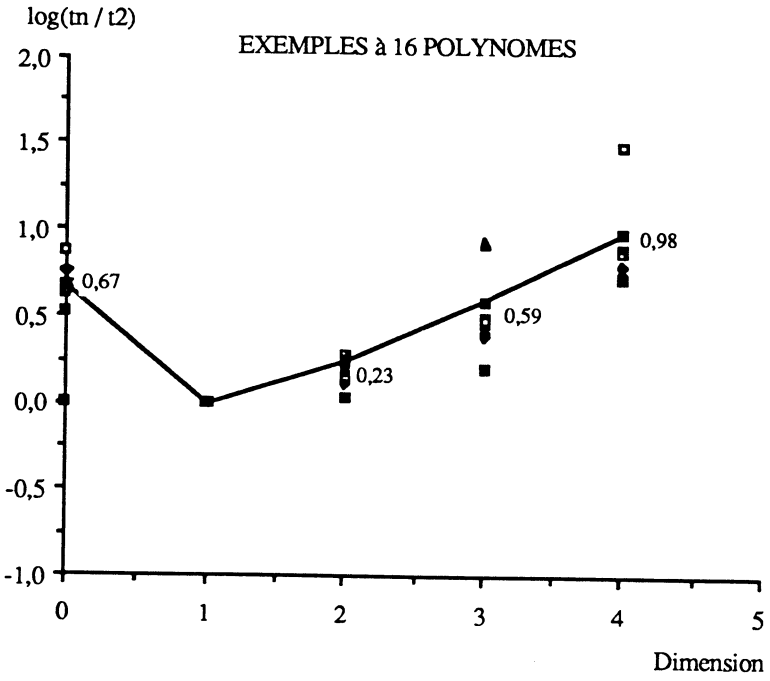
•• On représente en fonction de la dimension de l'anneau **la moyenne des logarithmes décimaux des rapports des temps mis pour exécuter un jeu d'exemples sur n processeurs au temps mis pour l'exécuter sur deux.**
L'interêt de cette représentation est de prendre comme référence l'anneau de dimension 1 qui est la plus petite structure sur laquelle on peut exécuter l'algorithme parallèle. On peut alors facilement comparer, d'une part algorithme séquentiel et algorithme parallèle, et d'autre part les exécutions sur des anneaux de dimensions différentes.

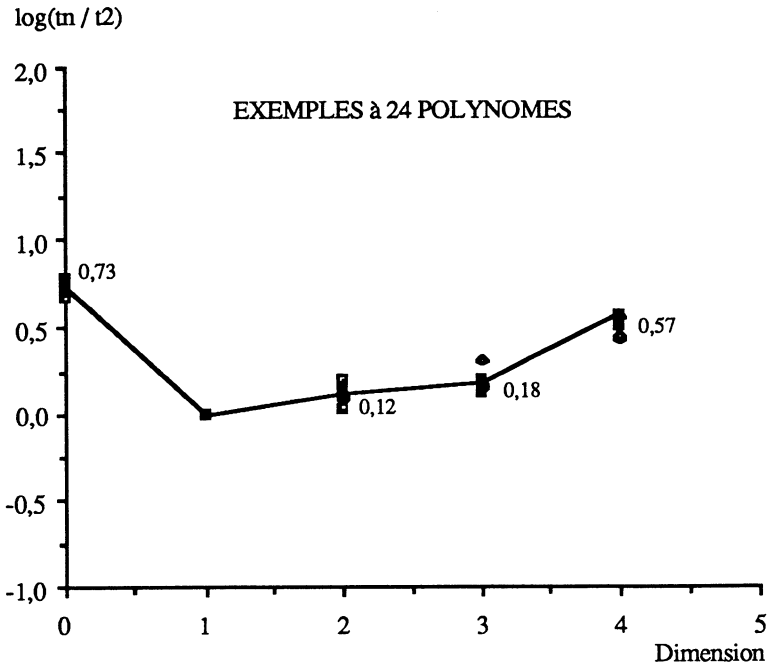




INTERPRETATION :

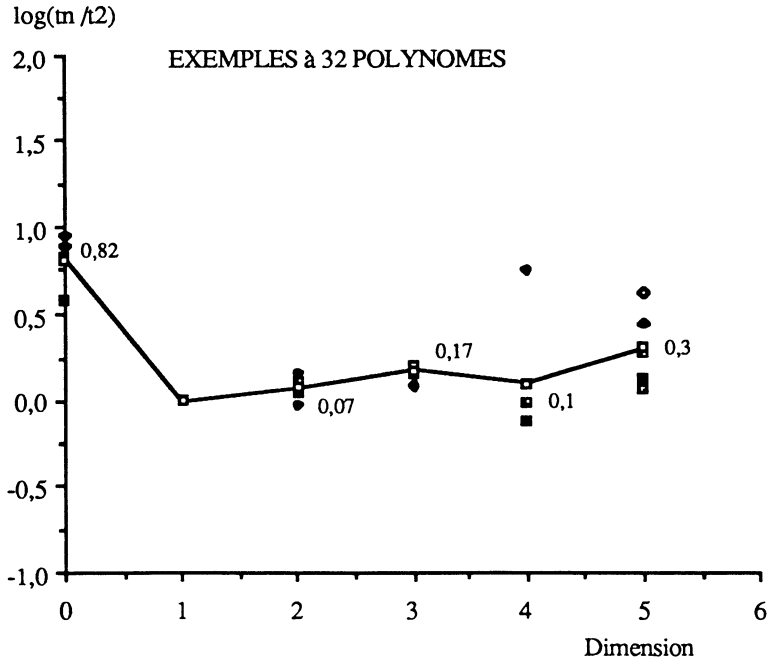
Pour 4 polynômes et 8 polynômes
L'algorithme séquentiel est le plus performant. A l'opposé, le temps moyen d'éxécution sur 16 processeurs est 8 fois plus important que celui sur 2 (7 fois plus que sur 1). On remarque que la variation est à peut près linéaire ce qui est normal car le temps total est égal au temps de calcul augmenté du temps de communication proportionnel au nombre de processeurs (on a souvent moins d'un polynôme par processeur ce qui indique que ces processeurs sont "paresseux").
Le comportement pour 8 polynômes est identique à celui observé pour 4 polynômes. On pourra noter une légère diminution des temps pour des dimensions d'anneaux supérieures à 2.

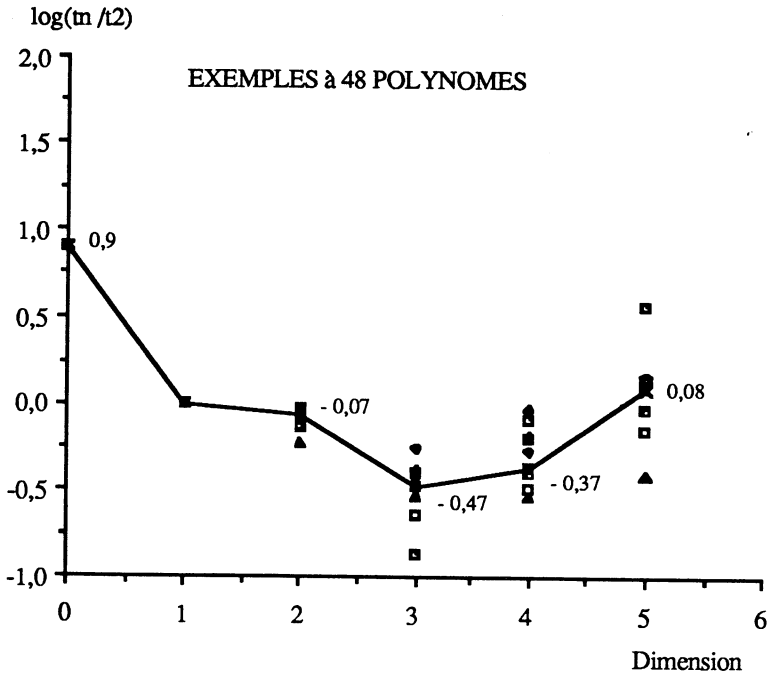


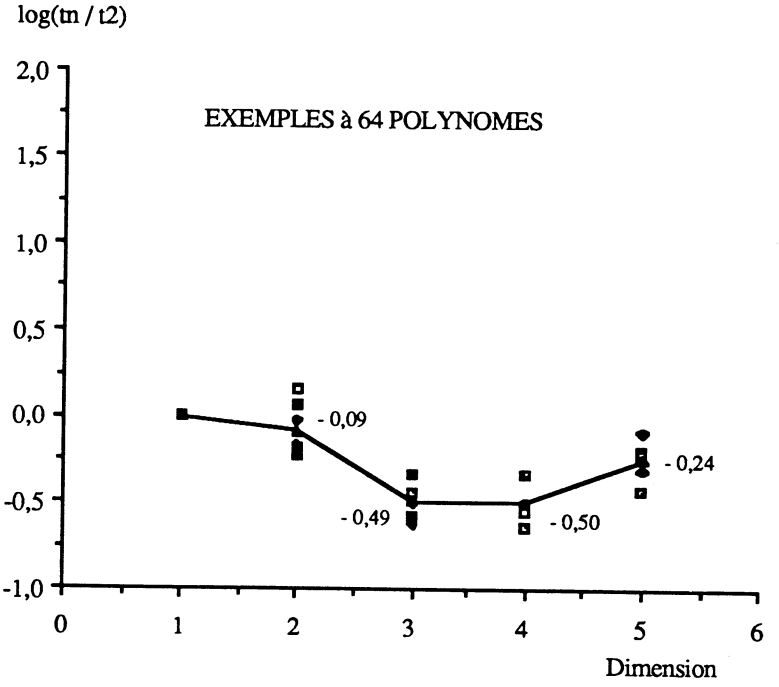


INTERPRETATION :

Pour 16 polynômes et 24 polynômes
L'algorithme séquentiel (dimension 0) devient moins performant que $\mathcal{A}(1)$ (il est équivalent $\mathcal{A}(3)$). Pour des dimensions supérieures à 2 les rapports continuent à décroître comme celà était déjà le cas pour 8 polynômes.
Les tendances évoquées pour 16 s'accentuent pour 24. $\mathcal{A}(4)$ est presque aussi performant que l'algorithme séquentiel. $\mathcal{A}(2)$ a un comportement un peu particulier et différent de celui des anneaux de dimensions supérieures : un exemple s'exécute toujours plus lentement sur 4 processeurs que sur 2. On donne par la suite une explication de cette "bizarrerie".





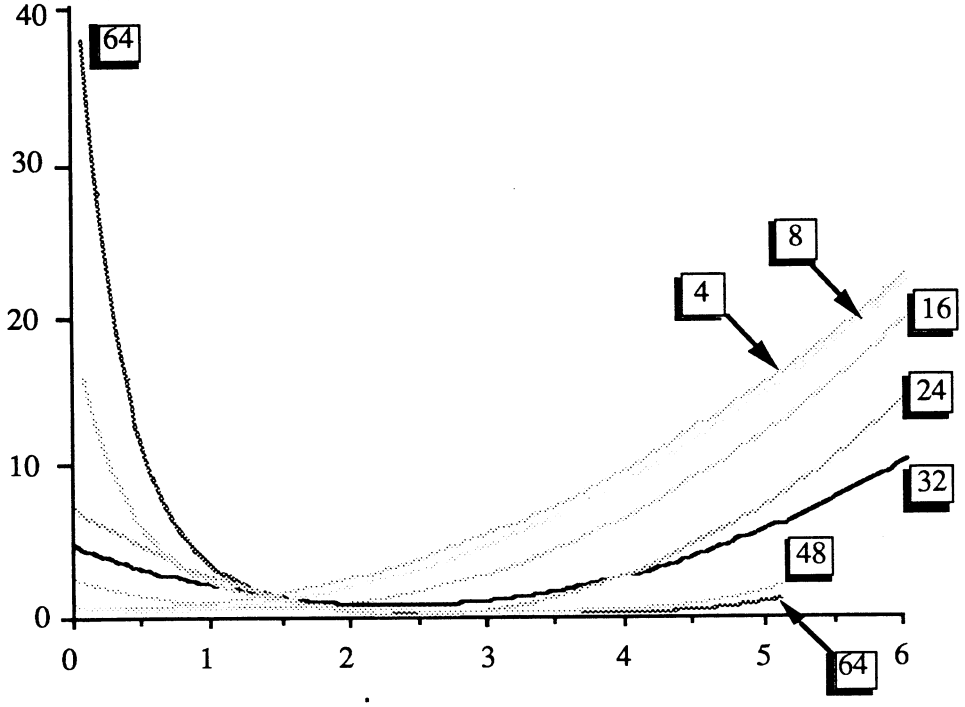


INTERPRETATION :

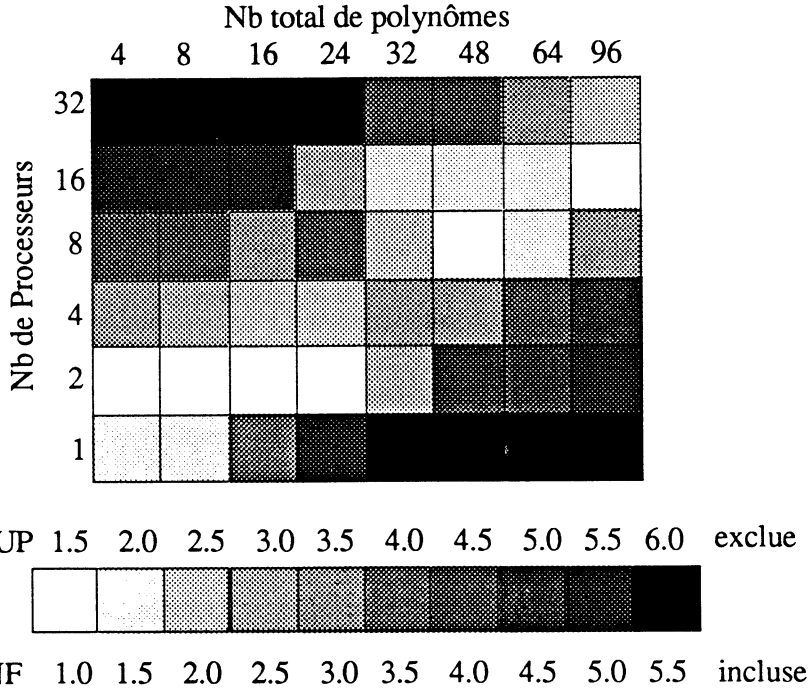
Pour 32, 48, 64 polynômes

Pour 32 polynômes l'algorithme parallèle se révèle très nettement plus performant que l'algorithme séquentiel. Les écarts entre les temps moyens pour des dimensions supérieures ou égales à 1 s'ammenuisent mais les tailles des exemples ne sont encore pas suffisantes pour permettre d'observer des différences de classement entre anneaux. En revanche pour 48 et 64 $\mathcal{A}(3)$ devient le plus performant. Si, à partir des temps obtenus de 4 à 48 polynômes, on extrapole le temps d'exécution mis par l'algorithme séquentiel pour 64 polynômes on obtient un temps d'environ 3 heures.

Pour résumer cette étude nous présentons le graphique suivant qui donne **la moyenne des rapports des temps mis pour exécuter les différents jeux d'exemples sur n processeurs au temps mis pour l'exécuter sur deux,** approximée en sens des moindres carrés. Il faut souligner que, de par l'approximation, ces courbes ne passent pas par le point (1,1) comme cela devrait être le cas. Cette approximation nous permet seulement de voir le comportement global de l'algorithme lorsqu'on augmente la dimension de l'anneau. Les courbes sont repérées par le nombre de polynômes des jeux d'essais



MOYENNE DES CLASSEMENTS DES PROCESSEURS EN FONCTION DU NOMBRE DE POLYNOMES



•• Etant donnée la grande dispersion relevée dans les temps d'exécution nous avons pensé représenter de manière qualitative les performances comparées des différents anneaux. Cette étude est résumée sur la figure ci dessus.

Remarques :

i) le comportement de l'anneau à deux processeurs $\mathcal{A}(1)$ est particulier. De par les procédures de communications certains calculs sont redondants.
En effet, au cours de la première phase de calcul on construit et on normalise les spolynômes des polynômes de référence F_i contenus dans chaque processeur i puis les familles F_i circulent sur l'anneau pour calculer les spolynômes associés aux couples de F_i x F_j ($i \neq j$). Cette circulation sur $\mathcal{A}(1)$ amène les processeurs 0 et 1 à calculer les mêmes spolynômes normalisés dans le noeud 0 par rapport à F_0 et dans le noeud 1 par rapport à F_1. De plus en calculant ces spolynômes on recalcule les spolynômes calculés dans la première phase. Globalement il y a redondance des données. Lorsque l'on normalise ensuite les spolynômes sur l'anneau ce sont en fait les mêmes spolynômes qui circulent du processeur 0 au processeur 1 et du processeur 1 au processeur 0.
A l'étape suivante le rôle des polynômes de référence est joué par ces derniers auquels on ajoute les polynômes précédemment calculés. Ces polynômes sont réduits entre eux avant de remplacer les polynômes de références. La redondance sur $\mathcal{A}(1)$ est telle que le nombre de normalisations "locales" est plus important et permet de réduire d'autant le nombre d'opérations à venir. Il semble alors que plus les normalisations peuvent se faire localement plus le gain de temps est important, ce que explique sans doute le comportement de $\mathcal{A}(1)$ souvent plus rapide que $\mathcal{A}(2)$.

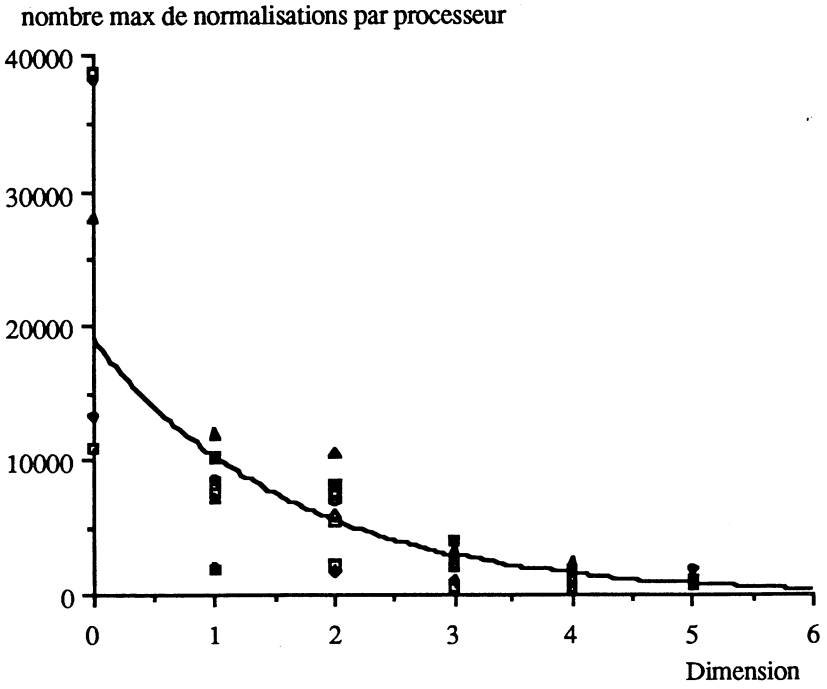
ii) on exploite alors $\mathcal{A}(1)$ pour construire un algorithme séquentiel plus performant que celui destiné à être parallélisé sur l'anneau (qui n'est pas le meilleur algorithme séquentiel !) cf II-6.

iii) et pourquoi, n'a-t-on pas utilisé la **notion d'efficacité** ? On ne peut parler d'efficacité que si l'on traite exactement le même problème sur 1 processeur et sur n. C'est à dire si le volume de calculs exécutés au cours de l'algorithme reste constant, que l'on travaille sur un ou sur plusieurs processeurs. Dans la plus part des problèmes déterministes ou en fait l'hypothèse même s'il y a effectivement une variation de ce volume [Villard88]. Dans le cas présent la variation de ce volume que l'on mesure par la variation du nombre d'opérations, est telle que l'on ne peut parler d'efficacité, puisqu'en fait on n'exécute pas le même algorithme lorsque l'on fait varier le nombre de processeurs.

••Variation du nombre d'opérations

Il y a une corrélation entre le nombre d'opérations effectuées par l'algorithme et son temps d'exécution. Nous avons déjà souligné que l'on ne maitrise pas le nombre de normalisations effectuées au cours du programme.

On mesure le maximum des nombres d'opérations effectuées par chaque processeur. Dans les exemples on a mesuré ce maximum dans le cas des normalisations faites dans chaque processeur, en moyenne on obtient :



II-6 Le programme et les données

Nous avons déjà soulevé le problème de la répartition des données. Nous avons fait quelques tests qui montre comment on pourrait, en effectuant un prétraitement des données afin d'améliorer l'algorithme parallèle.

On considère un exemple de 16 polynômes s'exécutant sur 16 proccesseurs :

* La répartition équilibrée obtenue avec l'algorithme REP (1 polynôme par processeurs) permet à l'algorithme de s'effectuer en :

38.06 s

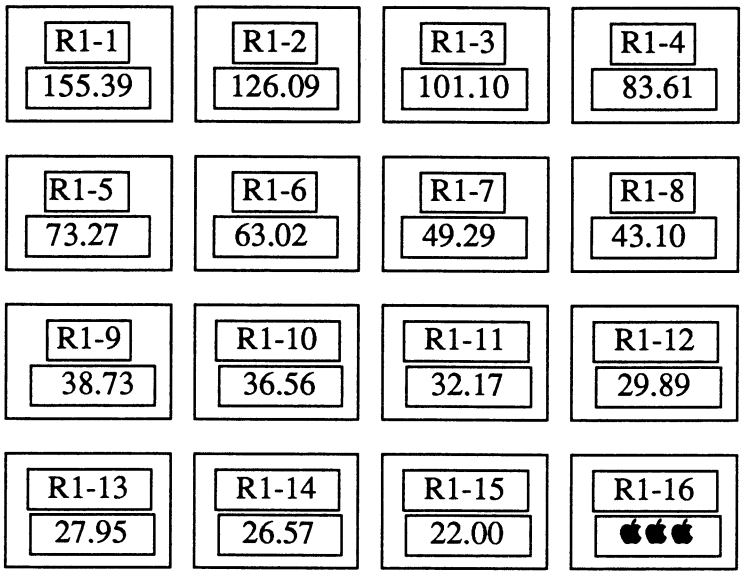
* On peut répartir les polynômes de la façon suivante :

♦ le processeur 0 contient les 16 polynômes ; le processeur 1 contient les 15 premiers, plus généralement le processeur i contient les 16-i premiers polynômes. On note cette répartition R1-1.

♦♦ à partir de R1-1 on enlève le dernier polynôme des processeurs 1 à 15, on obtient alors une nouvelle répartition que l'on note R1-2

♦♦♦ à l'étape k , c'est à dire pour la répartion notée R1-k, les processeurs numérotés de 16-k à 15 sont vides, les processeurs i, i variant de 1 à 15-k contiennent rsepectivement les (17-k-i) premiers polynômes et le processeur 0 les contient tous.

On a les temps d'exécution suivant (en secondes) :

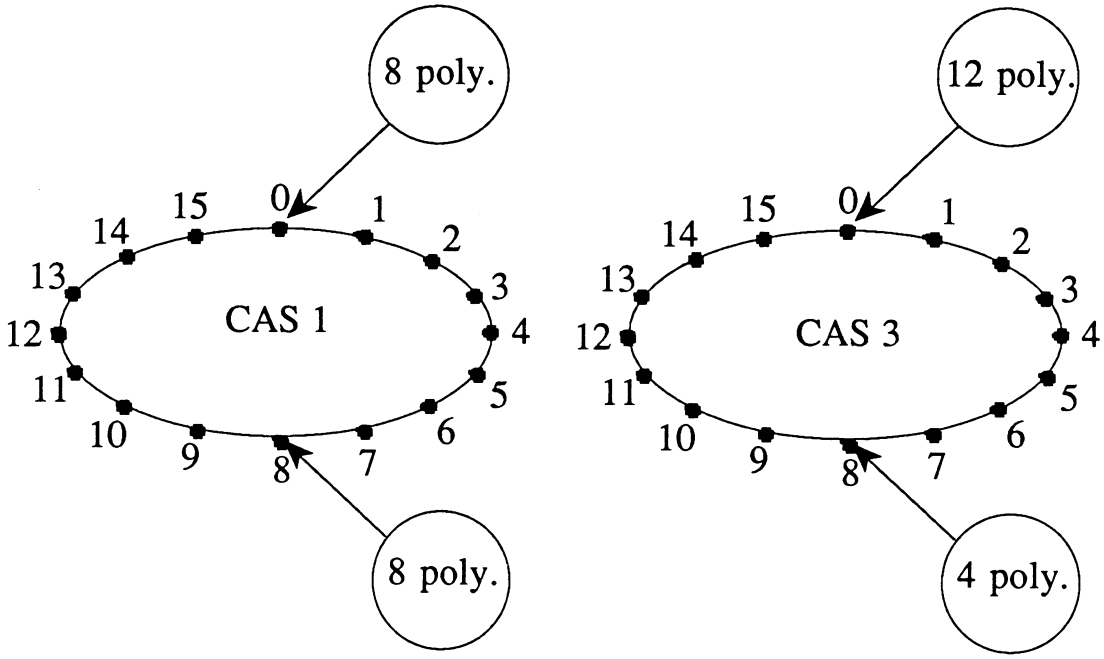


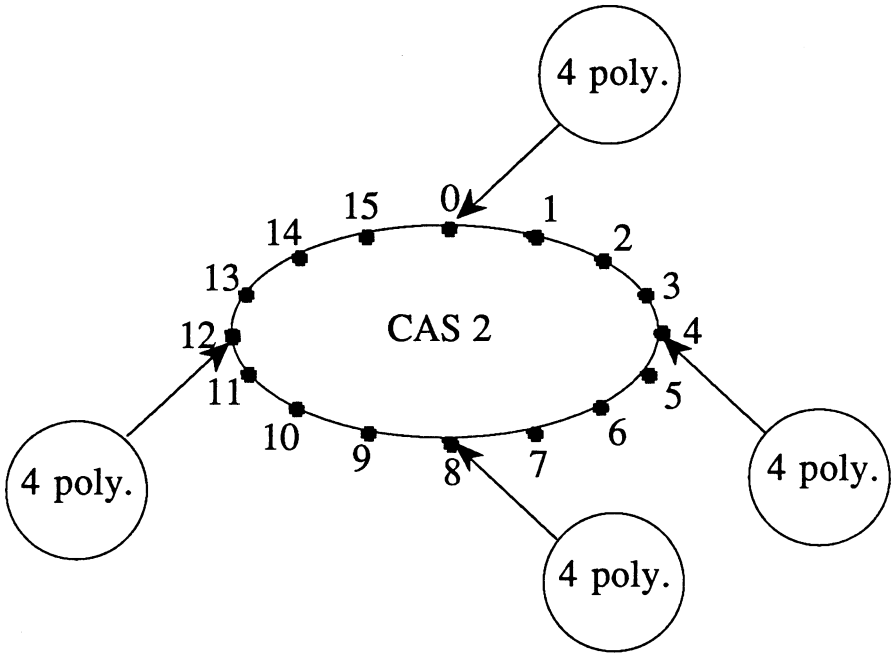
Remarques :

i) les répartitions non équilibrées peuvent être meilleures que celle obtenue par REP. Moins on a de polynômes dans les processeurs plus le temps d'exécution diminue.

ii) le temps obtenu avec la répartition R1-15 est le meilleur. Toutes les données sont dans le même processeur. Les processeurs 1 à 15 sont vides. Le nombre de spolynômes calculés en parallèle est moins important mais le nombre de réduction "locales" est plus important ce qui induit une diminution du nombre de calculs qui suivent.

* On peut également les répartir des différentes façons suivantes (cas1, cas2, cas3) :





Les temps respectivement obtenus pour les cas 1, 2 et 3 sont 17.99, 13.79 et 24.10 secondes.

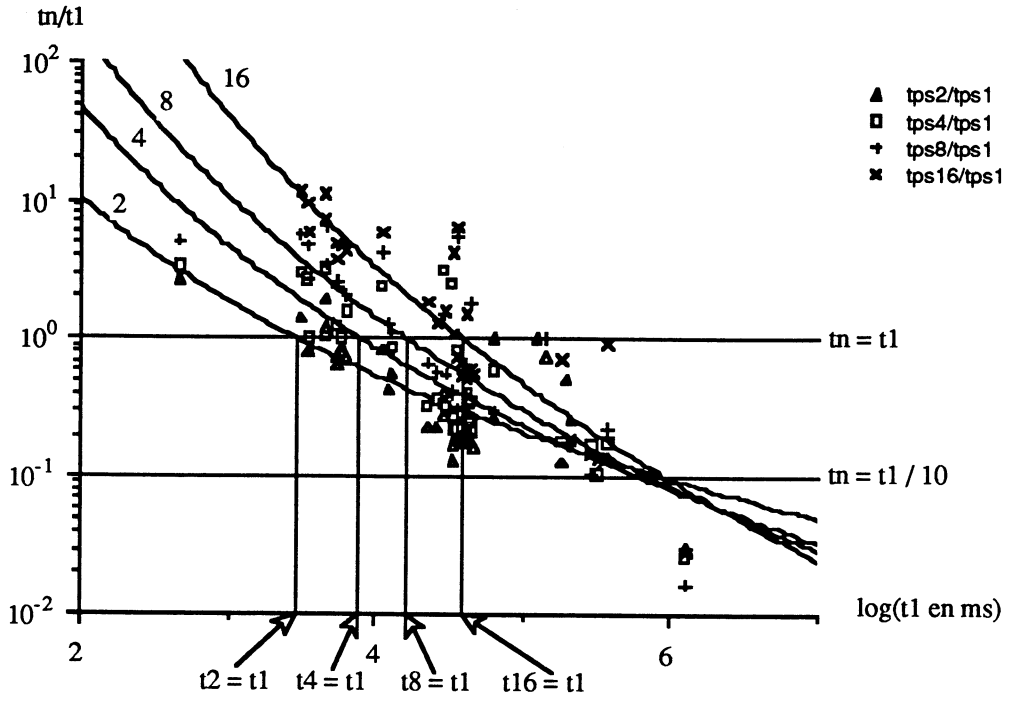
Remarques :

i) pour le deuxième cas l'algorithme s'exécute 36% plus rapidement que l'algorithme avec la répartition équidistribuée.
ii) le comportement de l'algorithme dans ces différents cas se rapproche de celui de l'algorithme effectué sur $\mathcal{A}(1)$. La différence de temps de la base associée à cet exemple sur $\mathcal{A}(1)$ et le cas 3 peut s'expliquée par une différence dans les temps de communication (qui ne sont pas négligeables dans le cas d'un "petit" exemple tel que celui-ci).

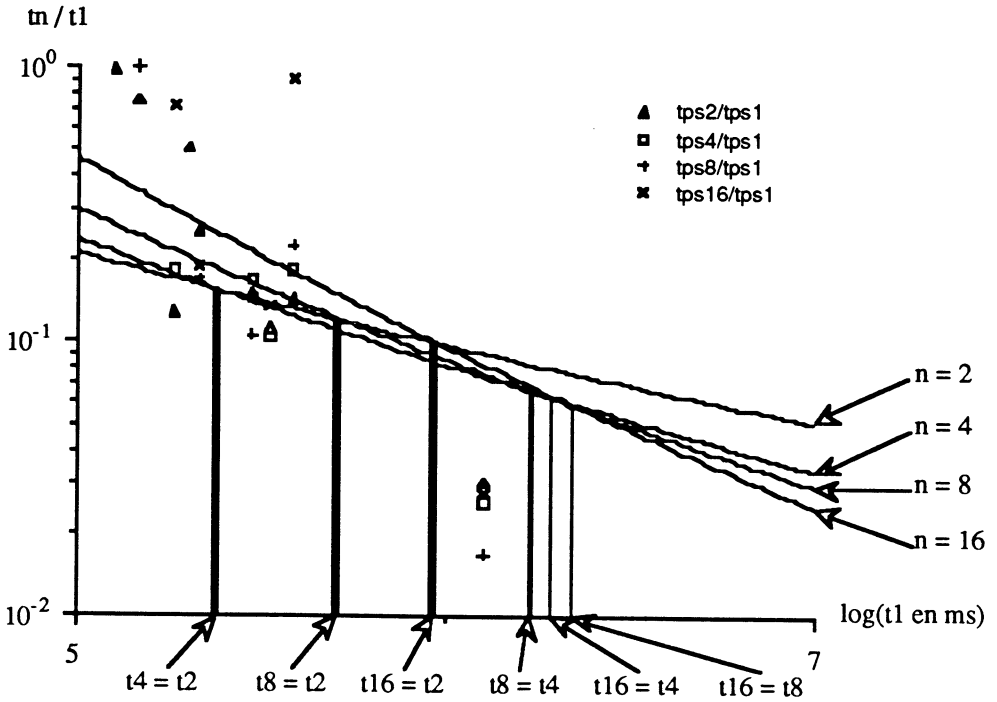
Dans le cas présent il semble, qu'au lieu de répartir régulierement les données, une bonne stratégie de répartition serait de répartir les polynômes de manière à se rapprocher de $\mathcal{A}(1)$.

II-7 Synthèse

Nous pouvons faire la synthèse de l'étude de la parallélisation d'un algorithme de type RBuch en représentant **le rapport du temps d'exécution de l'algorithme sur n processeurs au temps d'exécution de $\mathcal{A}(0)$ en fonction de ce dernier.** Cette représentation nous permet de voir dans quelles limites il peut être intéressant de paralléliser les algorithmes. On ne tient pas compte de la taille des données qui est un critère quelque peu limitatif mais uniquement du temps d'exécution de l'algorithme séquentiel.



Pour mettre en évidence les comportements relatifs des différents anneaux, nous pouvons faire un "zoom" de ce graphique lors le logarithme décimal de t1 est compris entre 5 et 7 :

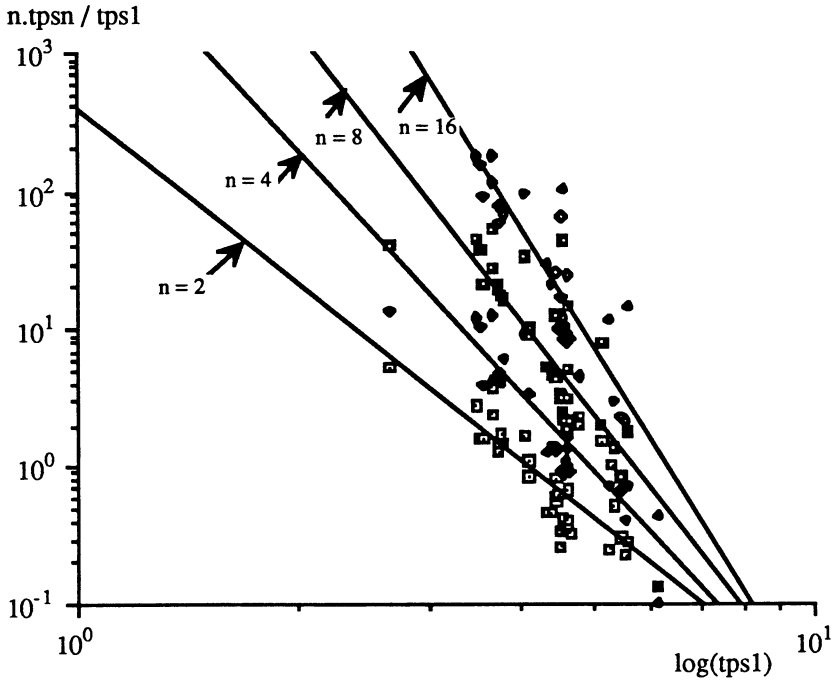


On peut alors estimer les temps pour lesquels le parallélisme devient interessant par le tableau suivant où l'on représente le temps à partir duquel le temps de l'algorithme sur n

processeurs devient k fois plus performant que l'algorithme séquentiel pour k =1, 10 ou 100 :

	k = 1	k = 10	k = 100
2 processeurs	2.9s	15 min	4972 h
4 processeurs	7.9s	10.5 min	118 h
8 processeurs	16.2s	12.4 min	14 h
16 processeurs	38.3s	15.5 min	16h

On peut représenter l'efficacité expérimentale avec le graphique suivant :



L'approximation du rapport n.tpsn sur tps1 (où n est le nombre de processeurs, tpsn le temps d'exécution sur n processeurs et tps1 celui sur 1) nous fournit des droites qui nous permettent de donner les efficacités expérimentales suivantes :
si le temps d'exécution sur un processeur est de la forme 10^p on a :

si on note e_n l'efficacité sur n : $e_n = p^{bn} / a_n$ avec

pour n = 2 $a_n = 4.10^2$ et bn = 4,2
pour n = 4 $a_n = 1.10^4$ et bn = 5,8
pour n = 8 $a_n = 2.10^5$ et bn = 6,9
pour n = 16 $a_n = 9.10^6$ et bn = 8,7

Conclusion :

°Le cadre de l'étude limite le nombre de variables, le nombre de monômes par polynômes pour rester homogène et pouvoir comparer les résultats. On peut cependant "prévoir" le comportement de l'algorithme avec des exemples de taille plus importante : il semble que si l'on augmente la taille des données, que ce soit en nombre de variables, nombre de monômes ou encore nombre de polynômes, ce soit sur les anneaux de dimensions les plus élevées que l'algorithme s'exécute en moins de temps. Le parallélisme est alors intéressant dans le cas des données de taille conséquente.

°°L'importance d'une étude plus approfondie des stratégies de répartition n'est pas à démontrer et il serait intéressant d'associer à l'algorithme un prétraitement des données qui fournirait une répartition et la dimension de l'anneau à choisir.

°°°L'algorithme séquentiel tourne environ 30 fois moins vite que $\mathcal{A}(1)$ dès que l'on traite des données de plus de 16 polynômes. L'algorithme séquentiel simulant le comportement de $\mathcal{A}(1)$ s'exécuterait environ deux fois moins vite que $\mathcal{A}(1)$ lui-même mais pourrait rester largement meilleur que l'algorithme séquentiel de départ dès que l'on a plus de 16 polynômes. Il serait également intéressant de créer un algorithme analogue lorsque l'on calcule des bases de Gröbner sur $\mathbb{Q}$. Le problème de gestion mémoire reste entier. Il est à souligner que lorsque les données sont réparties sur les processeurs le temps passé pour gérer les données est moins important que si l'on travaille avec un seul processeur.

°°°°Cette étude met en place la notion de "capacité" pour un anneau $\mathcal{A}(n)$ qui dénote l'importance jouée par le rapport taille des données / taille de l'anneau qui peut-être un paramètre à prendre en compte lors du préconditionnement des données cité plus haut.

°°°°°Il faut noter également un ralentissement des algorithmes dû à la nécessité de synchroniser les processeurs avant certaines procédures de communications. Cette synchronisation nécessaire provient des protocoles de communication associés au FPS 40.

III- LE CAS DE L'HYPERCUBE

III-1 Le cadre de l'étude

Dans ce qui suit on appelle $\mathcal{H}(n)$ l'algorithme parallèle sur un hypercube comprenant 2^n processeurs.

L'étude de cet algorithme parallèle a débuté en utilisant les mêmes jeux de données que pour $\mathcal{A}(n)$. Nous nous sommes vite aperçu que les temps d'exécution était bien plus faibles que dans le cas de l'anneau et qu'en conséquence l'étude de l'algorithme parallèle sur l'hypercube ne pourrait être menée avec de tels exemples. Nous avons donc décidé de générer aléatoirement des familles de polynômes booléens, en espérant que les temps obtenus soient importants afin de différencier (éventuellement) le comportement de l'algorithme en fonction du nombre de processeurs. On génère donc des familles de polynômes en précisant :

- le nombre de polynômes de la famille
- le nombre minimal et le nombre maximal de monômes par polynôme
- le nombre de variables.

Le nombre de variables a été choisi entre 2 et 20, le nombre de polynômes entre 10 et 96 et le nombre maximal de monômes par polynômes a été limité à 10, ceci pour obtenir des résultats à peu près homogènes.

Nous comparerons par la suite les deux algorithmes séquentiels correspondant respectivement au cas de l'anneau et de l'hypercube. Nous pouvons déjà dire que les temps de l'algorithme séquentiel $\mathcal{H}(0)$ sont faibles comparés à ceux de $\mathcal{A}(0)$ car la première opération consiste en une réduction des entrées ce qui, comme nous l'avons vu dans le cas de l'anneau entraîne un gain de temps important sur les algorithmes.
Nous verrons d'autres raisons directement liées à la structure de MGBuch (qui correspond à $\mathcal{H}(0)$).

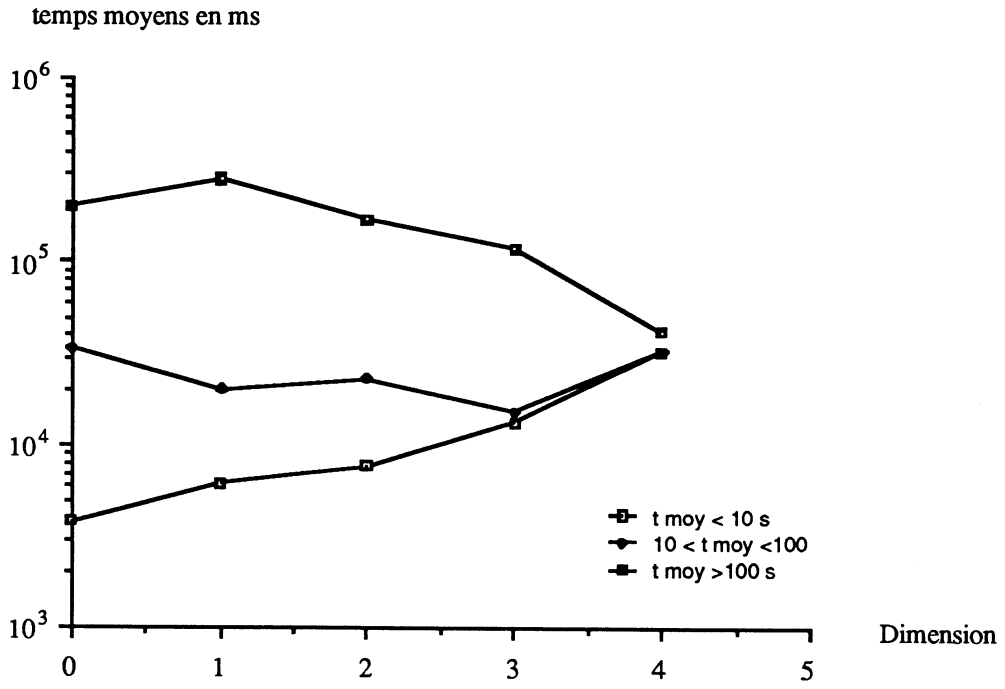
Si nous avons choisi MGBuch comme algorithme à paralléliser sur l'hypercube c'est justement pour ces performances en séquentiel. En effet la tâche indivisible de $\mathcal{H}(n)$ est le calcul complet d'une base de Gröbner réduite qui, à priori, doit être performant. Cette approche est totalement différente de la parallélisation sur l'anneau où l'algorithme de Buchberger est lui-même subdivisé en tâches indépendantes et où l'on ne calcule la base de Gröbner des polynômes donnés qu'une seule fois.

III-2 Le comportement global

Nous ne reviendrons pas sur la répartition des données entre les processeurs, bien que celà influence énormement le comportement global de l'algorithme. Nous avons réparti nos données comme dans le cas de l'anneau..

Nous n'avons pas décrit en détail l'algorithme sur deux processeurs : il s'agit simplement du calcul en parallèle de deux sous bases (avec MGBuch) puis du regroupement de ces deux sous bases dans un processeur afin de calculer la base finale à l'aide de MGBuch2.

Nous représentons alors **la moyenne des temps d'exécution de l'algorithme en fonction de la dimension de l'hypercube.** Nous avons pris plusieurs jeux exemples pour lequel les temps de $\mathcal{H}(0)$ étaient inférieurs à 10 secondes et nous avons alors calculés les moyennes des temps d'exécution de $\mathcal{H}(n)$. Puis nous avons faits de même pour des temps d'exécution de $\mathcal{H}(0)$ compris entre 10 et 100 secondes et pour des temps supérieurs à 100 secondes (les temps sont portés sur une échelle logarithmique).

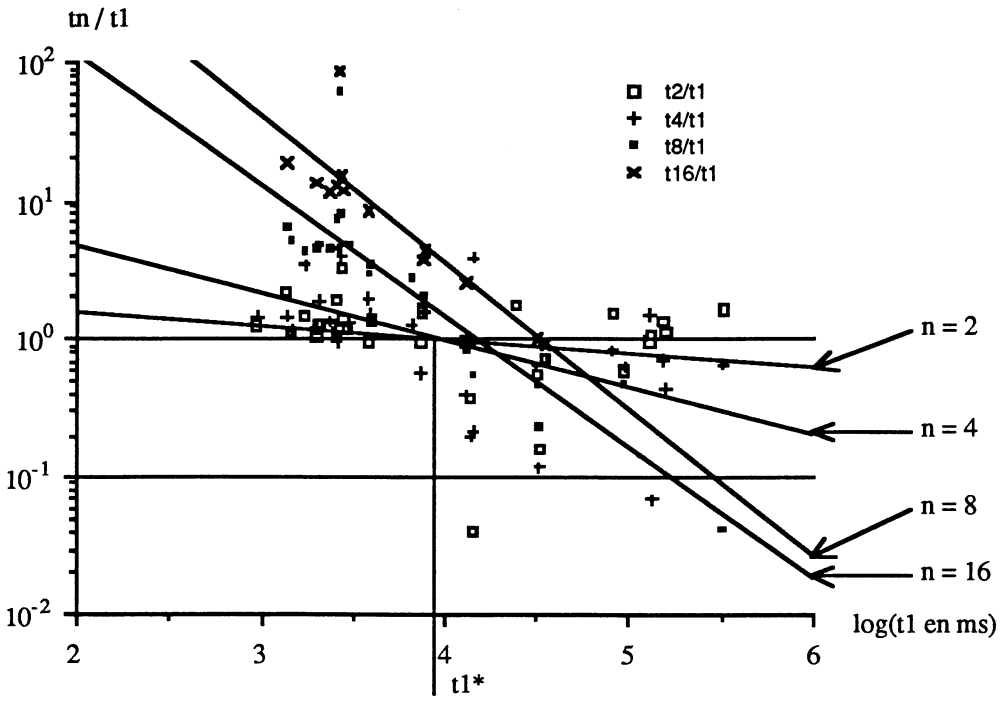


INTERPRETATION :

i) Un examen global de ce graphique montre qu'un hypercube de dimension élevée peut-être suivant l'exemple plus performant qu'un hypercube de dimension plus faible. Malheureusement ce comportement n'est pas systématique.

ii) Comme dans le cas de l'anneau on met en évidence la "capacité" d'un hypercube de dimension donnée à traiter un problème particulier : on remarque en effet que les temps d'exécution varient d'autant moins que la dimension de l'hypercube est importante (excepté pour le cas à deux processeurs).

Pour mettre en évidence ces divers phénomènes on représente **le temps sur n processeurs en fonction du temps sur un :**



INTERPRETATION :

i) On observe très grossièrement le même comportement qualitatif que sur l'anneau c'est à dire que pour des temps de calcul faibles on a le classement 1, 2, 4, 8, 16 (en moyenne) alors que pour des temps de calcul élevés on a le classement inverse.

ii) On remarque que le temps t1* pour lequel il devient intéressant d'utiliser $\mathcal{H}(n)$ avec n >0 est de l'ordre de 10^4 ms ce qui est comparable aux temps relevés sur $\mathcal{A}(n)$. Cependant contrairement à $\mathcal{A}(n)$, $\mathcal{H}(n)$ ne présente pas un comportement aussi net :

- d'une part la différence des temps d'exécution de $\mathcal{H}(0)$ et de $\mathcal{H}(1)$ est faible alors qu'elle est importante entre les temps d'exécution de $\mathcal{A}(0)$ et $\mathcal{A}(1)$ (cf III-4).

- d'autre part le graphique précédent ne fournit pas d'informations précises sur le comportement asympotique de $\mathcal{H}(1)$, la droite de régression n'est donnée qu'à titre indicatif car sa pente n'est pas significativement différente de zero.

Afin de mettre en évidence le comportement de $\mathcal{H}(n)$ nous proposons l'étude qualitative suivante :

lorsque l'on compare deux hypercubes de dimension n et n' avec n' supérieur à n on pourrait logiquement s'attendre à avoir systématiquement des performances meilleures sur $\mathcal{H}(n)$ que sur $\mathcal{H}(n')$ pour des petits temps et l'inverse pour des temps élevés.

Définissons la variable booléenne SUP(n', n) comme étant la variable qui vaut 1 si le temps sur $\mathcal{H}(n')$ est supérieur au temps sur $\mathcal{H}(n)$ et 0 dans le cas contraire et la variable $CUMUL_i(n', n)$ de la manière suivante :
- soient N essais E_i réalisés sur $\mathcal{H}(n)$ et $\mathcal{H}(n')$ et ordonnés par temps $T_i(n)$ croissant sur $\mathcal{H}(n)$,

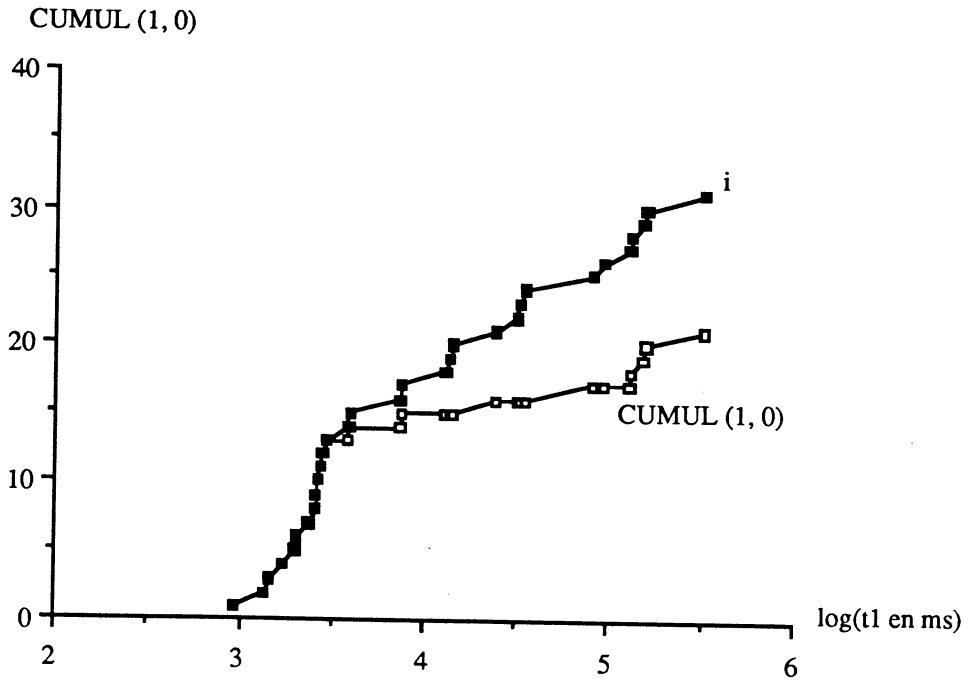
$$CUMUL_i(n', n) = \sum_{j=1}^{j=i} SUP_j(n', n) \quad ; \quad SUP_j(n', n) = \begin{cases} 1 \text{ si } T_j(n') > T_j(n) \\ 0 \text{ si } T_j(n') < T_j(n) \end{cases}$$

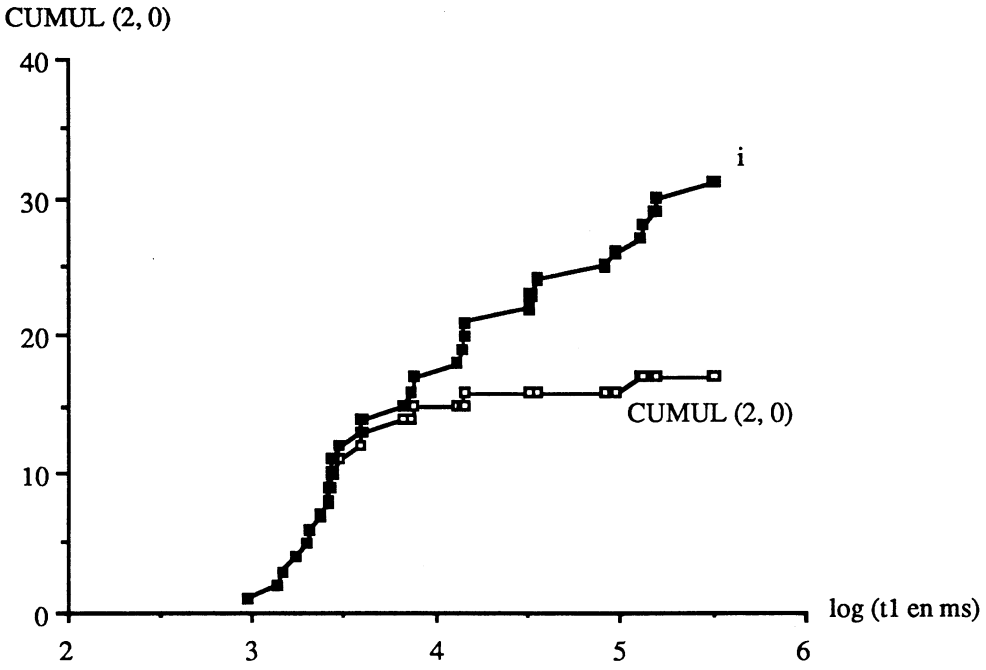
On devrait avoir entre $CUMUL_i(n',n)$ et i les tendances suivantes :
- tant que $T_i(n)$ est faible : $CUMUL_i(n',n) \cong i$
- dès que $T_i(n)$ devient assez grand : $CUMUL_i(n',n) < i$

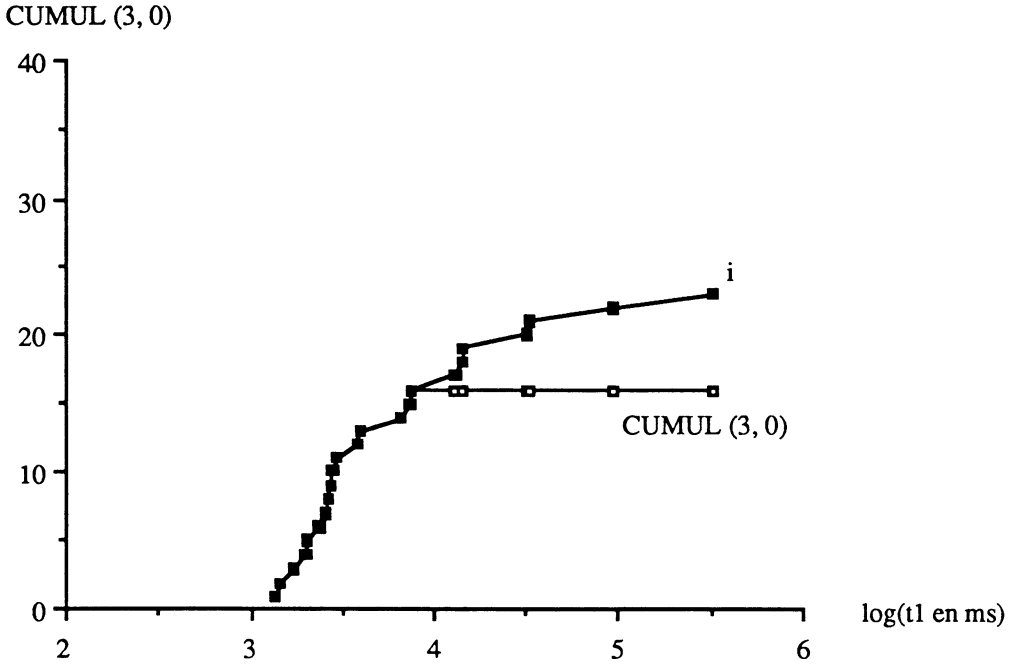
Ainsi, si l'on porte en abscisse les temps $T_i(n)$ et en ordonnée les deux courbes $CUMUL_i(n',n)$ et i, celles ci devraient se confondre aux faibles temps et se différencier de plus en plus lorsque les temps d'éxécution sur $\mathcal{H}(n)$ augmentent et tendre vers une asymptote horizontale.

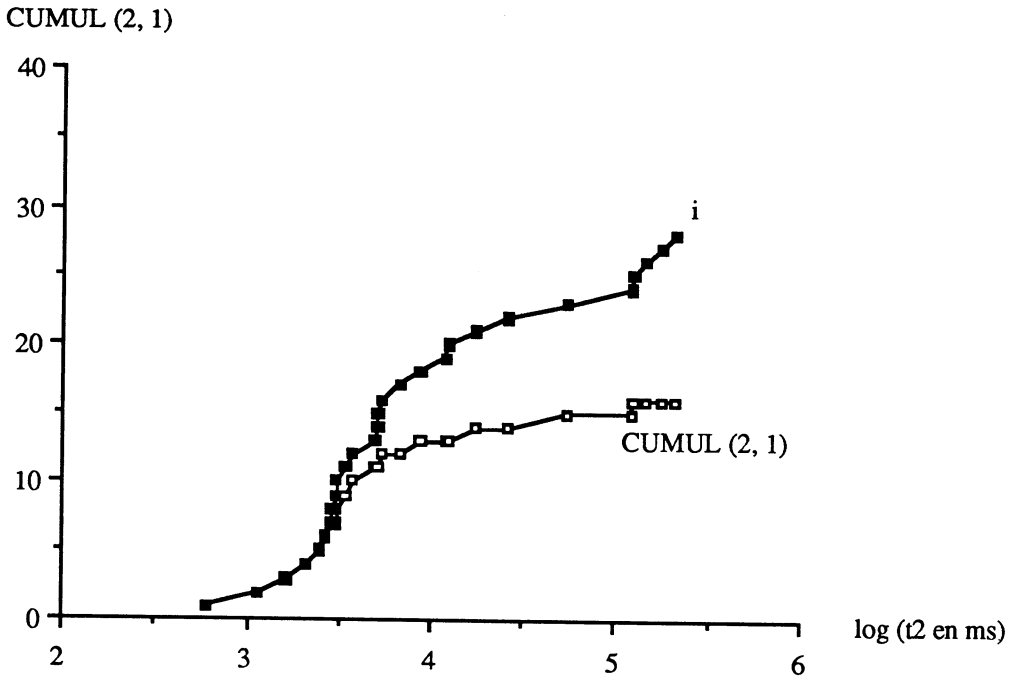
A l'opposé si $\mathcal{H}(n')$ est systématiquement plus mauvais que $\mathcal{H}(n)$ au delà d'un certain temps alors l'écart ($CUMUL_i(n',n)$ - i) est constant.

Ces graphiques ont été réalisés pour les couples (n',n) suivants : (1,0), (2,0), (3,0) et (2,1). Les résultats sont donnés ci dessous.




CUMUL (2, 0)
40
30
20
10
0
2
3
4
5
6
i
CUMUL (2, 0)
log (t1 en ms)


CUMUL (3, 0)
40
30
20
10
0
2
3
4
5
6
i
CUMUL (3, 0)
log(t1 en ms)



INTERPRETATION :

i) Sur les différents graphiques il est clair que tant que $T_i(n)$ est faible $CUMUL_i(n',n)$ est égal à i et qu'à partir d'une certaine valeur de $T_i(n)$ $CUMUL_i(n',n)$ est strictement inférieur à i (ce qui montre que le parallélisme apporte quelquechose).

ii) Pour $CUMUL_i(n', 0)$ avec n supérieur où égal à 2 une asymptote horizontale se dessine lorsque $T_i(n)$ augmente ce qui montre que l'algorithme $\mathcal{H}(n)$ tend à s'exécuter systématiquement plus rapidement que $\mathcal{H}(0)$.

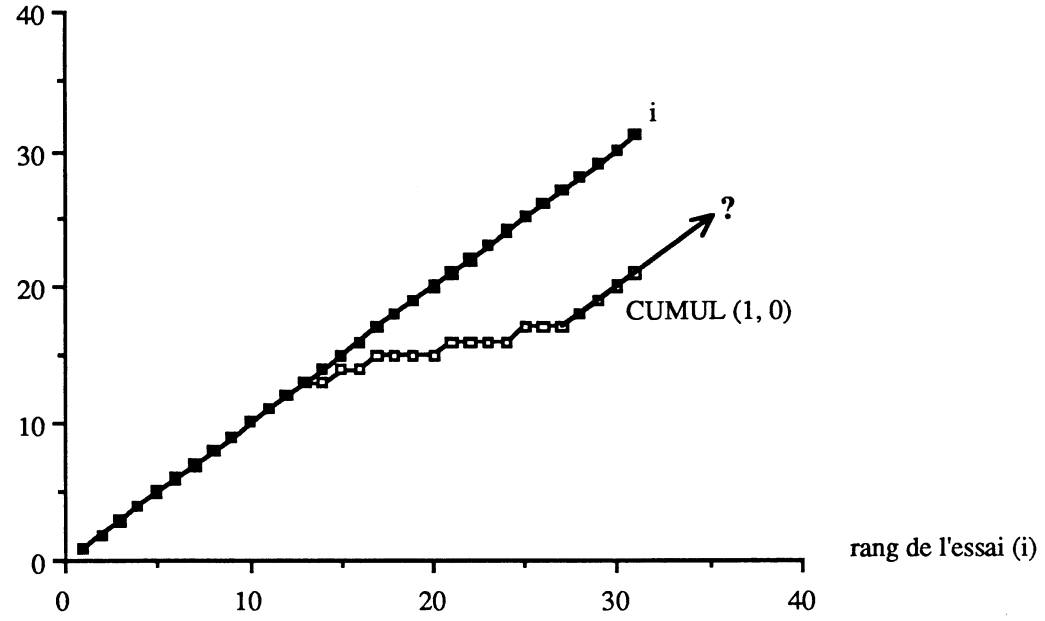
iii) La courbe $CUMUL_i(1, 0)$ ne semble pas présenter d'asymptote horizontale et paraît asymptotiquement parallèle à la courbe i.

iv) La courbe $CUMUL_i(2, 1)$ présente une asymptote horizontale que l'on peut interpréter par l'obtention systématique de meilleurs temps sur $\mathcal{H}(2)$ que sur $\mathcal{H}(1)$.

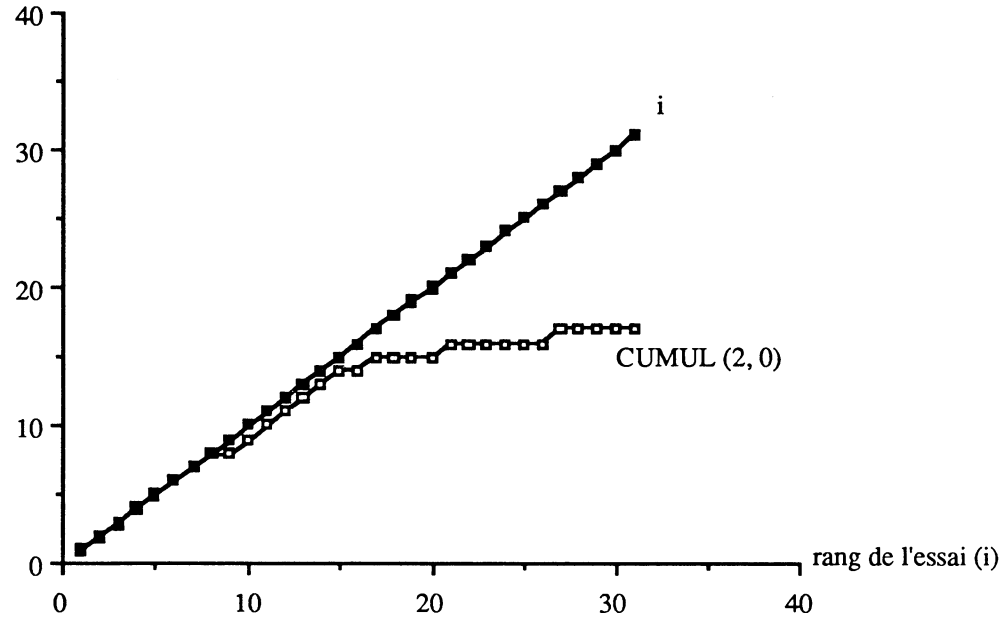
v) Il semble nécessaire de poursuivre cette étude avec des $T_i(n)$ plus importants (de l'ordre de l'heure).

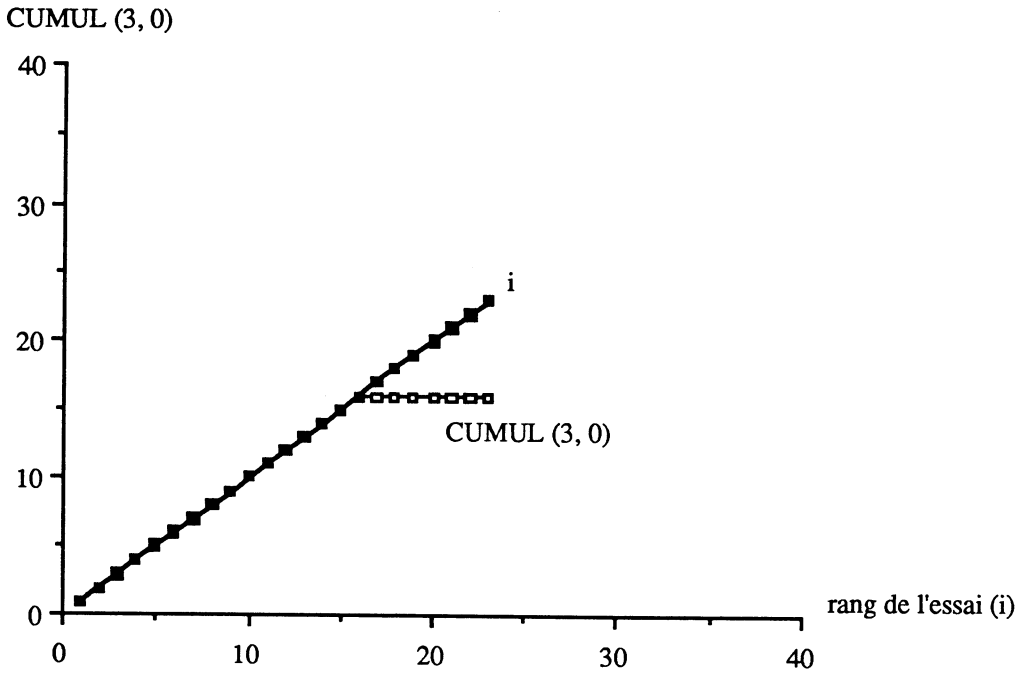
Pour mettre en évidence de manière encore plus qualitative ces divers phénomènes nous représentons les mêmes quantités $CUMUL_i(n', n)$ en fonction du numéro des exemples (ordonnés suivant les $T_i(n)$ croissants).

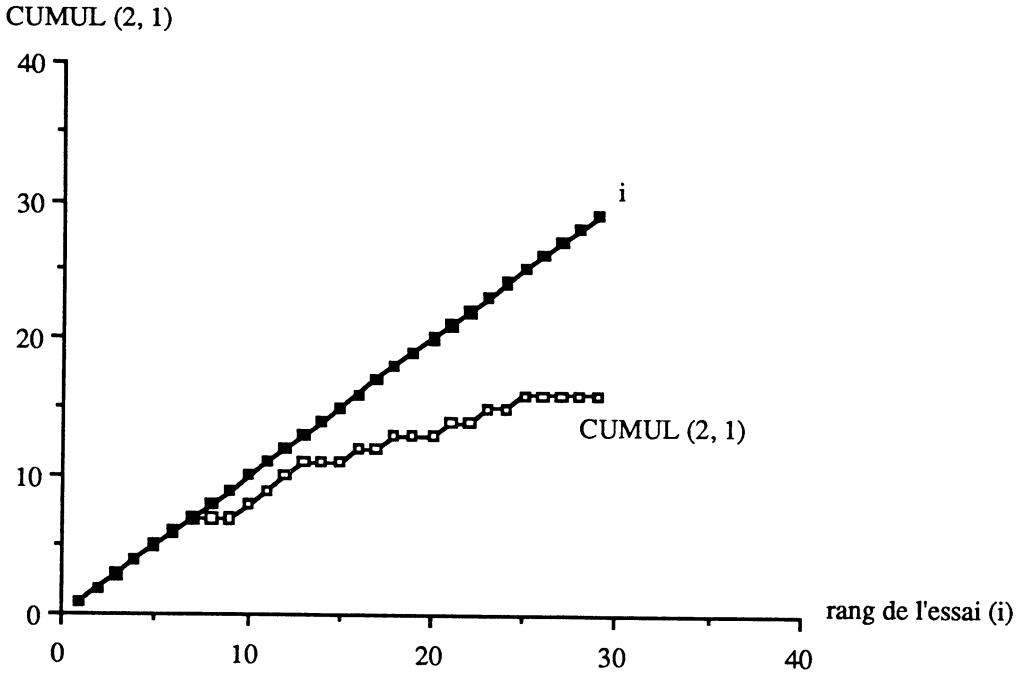
CUMUL (1, 0)



CUMUL (2, 0)







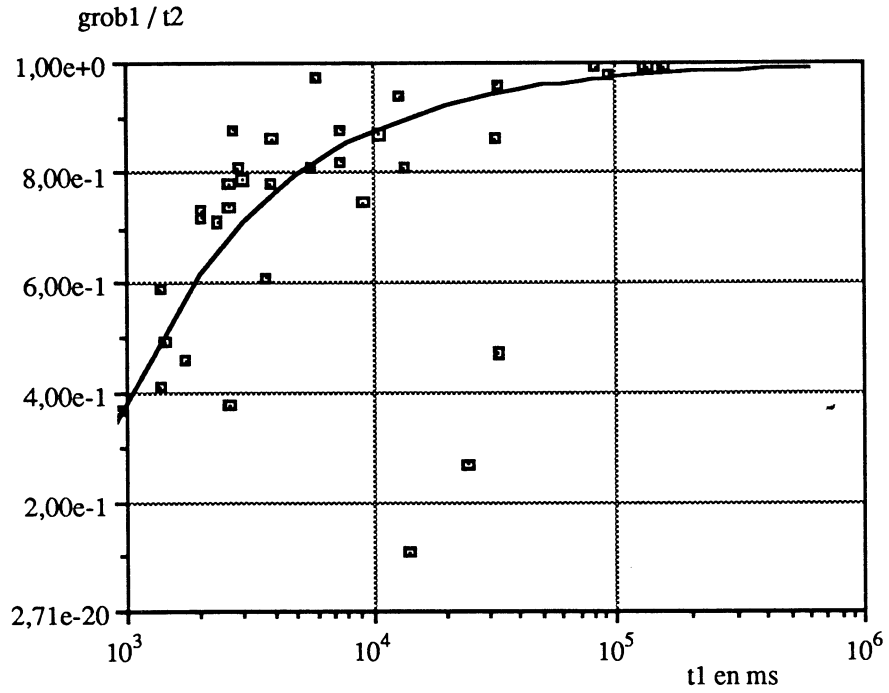
La tendance précédente se confirme. $\mathcal{H}(1)$ présente nettement un comportement différent des autres hypercubes : asymptotiquement il semblerait que sur deux processeurs l'algorithme ne soit pas meilleur que sur un. En revanche les hypercubes contenant plus de deux processeurs semblent de plus en plus performant lorsque leur dimension augmente **pour la gamme des essais testés.**

Pour tenter de mieux saisir le comportement des algorithmes $\mathcal{H}(n)$ nous regardons comment est réparti le temps d'exécution de l'algorithme entre les différents phases de calcul. Nous ne parlerons pas du temps de communication qui est très faible dans les exemples (de l'ordre de 10 à 500 ms).

III-3 De un à deux processeurs

Pour mettre en évidence l'influence du temps nécessaire au calcul de la première base locale à chaque processeur (base d'une demi famille d'entrée) on représente sur le graphique suivant **le rapport grobl / t2 en fonction de t1** où :

- grobl est le temps mis par le processeur le plus lent pour calculer cette base locale
- t2 est le temps d'exécution total de $\mathcal{H}(1)$
- t1 est le temps d'exécution totat de $\mathcal{H}(0)$



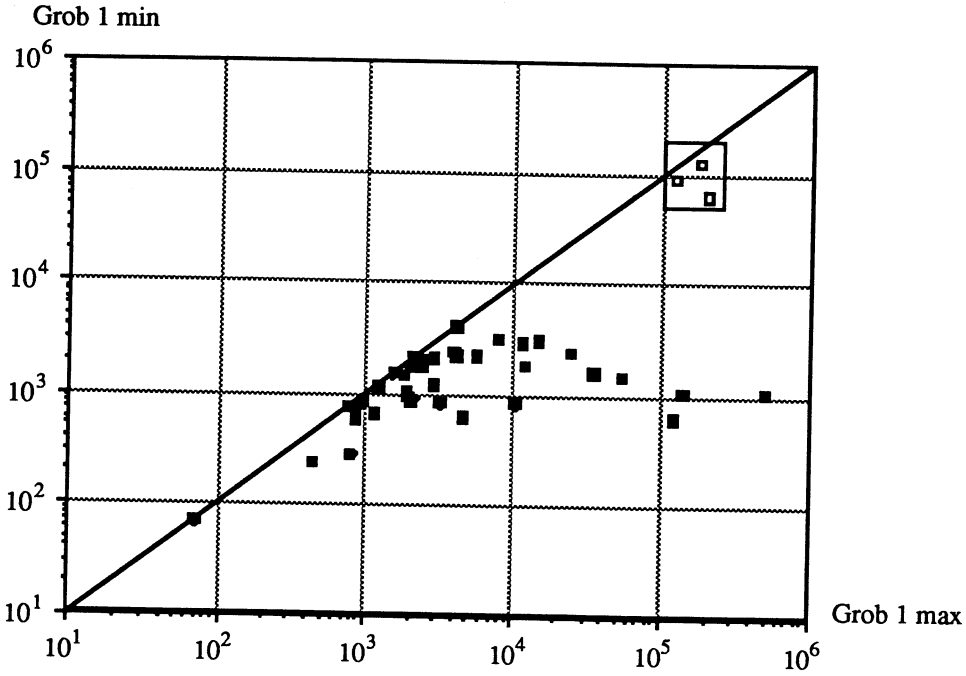
INTERPRETATION :

i) On remarque que plus le temps séquentiel augmente plus le temps apartit au calcul de la base de Gröbner associée aux polynômes donnés dans chaque processeur est important. Ce qui montre que si l'on veut réduire le temps d'éxécution de $\mathcal{H}(2)$ il est nécessaire d'avoir un algorithme de calcul de base de Gröbner performant.

ii) Le temps apartit au calcul de la base de Gröbner associée à deux sous-bases est faible par rapport au temps total.

iii) Nous avons porté le **maximum** des temps de calcul d'une base de Gröbner, puisque c'est le temps le plus long que conditionne l'algorithme : le processeur qui a terminé le premier doit attendre que l'autre est terminé pour procéder à la phase de communication suivante.

Nous nous intéressons à la différence entre les temps mis par les deux processeurs pour calculer la base de Gröbner associée à leur entrées. Nous représentons alors le plus court en fonction du plus long :



INTERPRETATION:

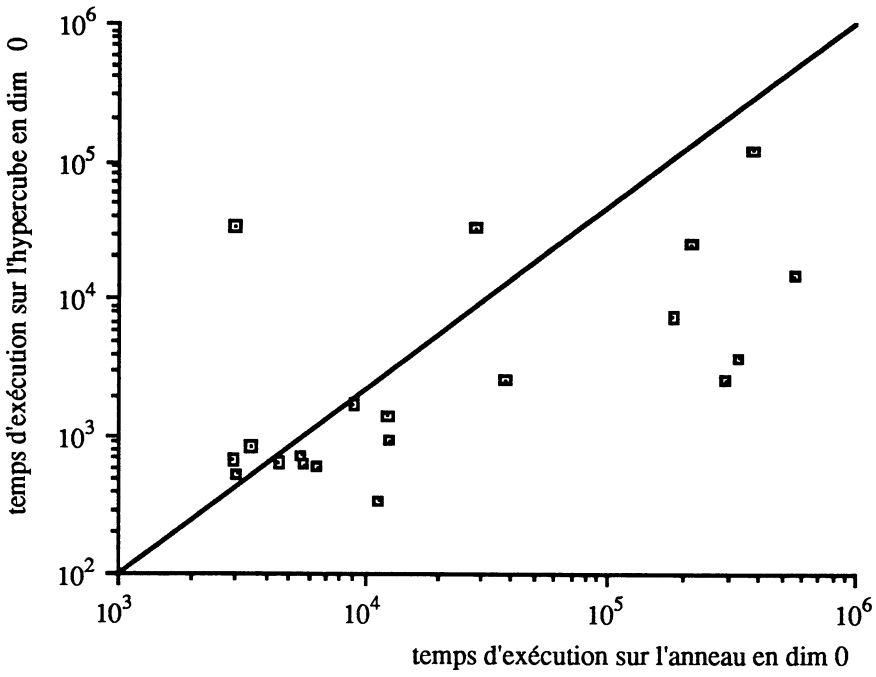
i) Mis à part les faibles temps et les trois points encadrés la différence entre les deux temps est notables. Ce qui montre, en particulier l'importance de la répartition des données : il serait agréable que les polynômes soient répartis de sorte que la différence de temps étudiée ici soit moindre. Ceci apporterait sans aucun doute énormement à la parallélisation. Malheureusement comme nous l'avons déjà signaler nous ne connaissons pas de borne suffisamment précise sur la complexité de l'algorithme MGBuch pour savoir à priori comment répartir les données pour minimiser cette différence.

ii) Les trois points encadrés sont particulier : ils ont été construits de sorte que l'on calcule dans chaque processeur la même base aux indices des variables près.

III-4 Entre l'anneau et l'hypercube

..Les algorithmes séquentiels :

Nous avons déjà souligné le fait que les algorithmes $\mathcal{H}(0)$ et $\mathcal{A}(0)$ sont différents.
Pour étudier les comportements relatifs de deux algorithmes nous portons les temps obtenus avec l'un en fonction de ceux obtenus avec l'autre :



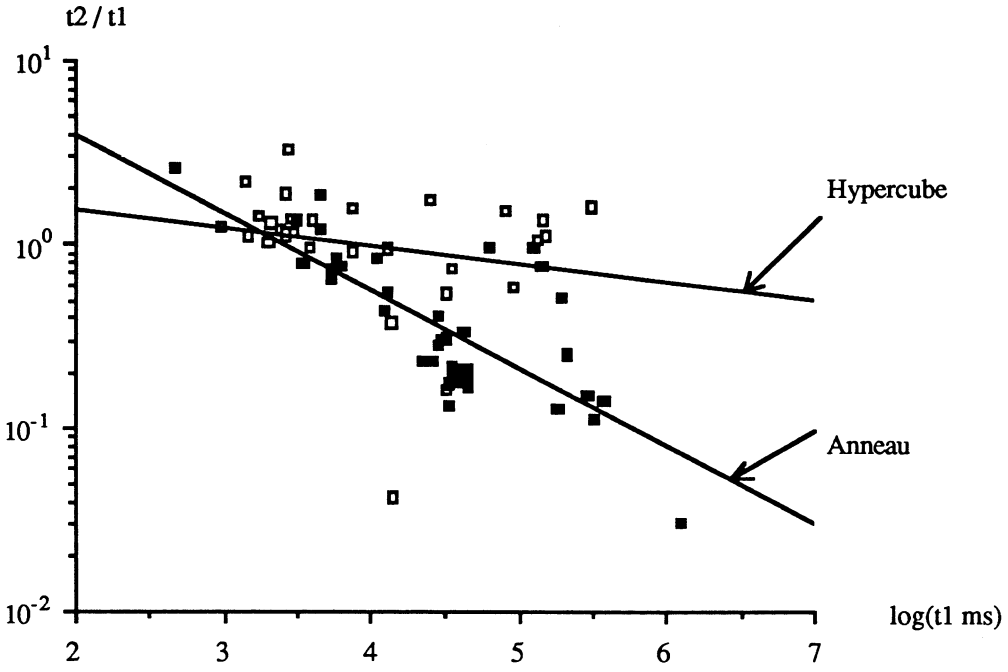
INTERPRETATION :

i) L'algorithme séquentiel MGbuch est celui qui correspond à la parallélisation faite sur l'hypercube. La parallélisation faite sur l'anneau correspond à un autre algorithme séquentiel : $\mathcal{A}(0)$. Ces deux algorithmes diffèrent par leur structure, notamment au niveau de la réduction comme nous l'avons déjà souligné : MGBuch active la procédure MRéd c'est à dire réduit les polynômes entre eux après chaque calcul d'un spolynôme alors que $\mathcal{A}(0)$ ne réduit qu'après avoir calculé et normalisé les spolynômes ; en réduisant au fur et à mesure la famille de polynômes, on évite ainsi des calculs supplémentaires. Sur de petits exemples le rapport des temps des deux algorithmes est environ de 2 à 3 , en revanche si l'on traite des données de taille plus importante ce rapport devient énorme.....

ii) La technique employée dans MGBuch est beaucoup plus performante. Ceci rejoint les résultats de Buchberger [Buchberger83a] qui tendent à prouver qu'il y a une diminution notable du temps de calcul lorsque l'on décide, après chaque calcul d'un nouveau polynôme p, de normaliser p par rapport aux polynômes existants mais également les polynômes existants par rapport à p.

Il est alors difficile de comparer directement les techniques de parallélisation puisque les algorithmes de départ sont différents. Nous pouvons tout de même comparer de façon qualitative ce qui se passe sur deux processeurs

..Les algorithmes parallèles :



INTERPRETATION :

i) Il est clair que plus les temps séquentiels augmentent plus on note une différence entre les rapports t2 / t1 de l'anneau et de l'hypercube.

ii) Le rapport t2 / t1 diminue moins vite (quand log(t1) augmente) dans le cas de l'hypercube que dans le cas de l'anneau, ce qui montre que, sur les exemples traités, $\mathcal{A}(1)$ devient rapidement plus performant que $\mathcal{H}(1)$.

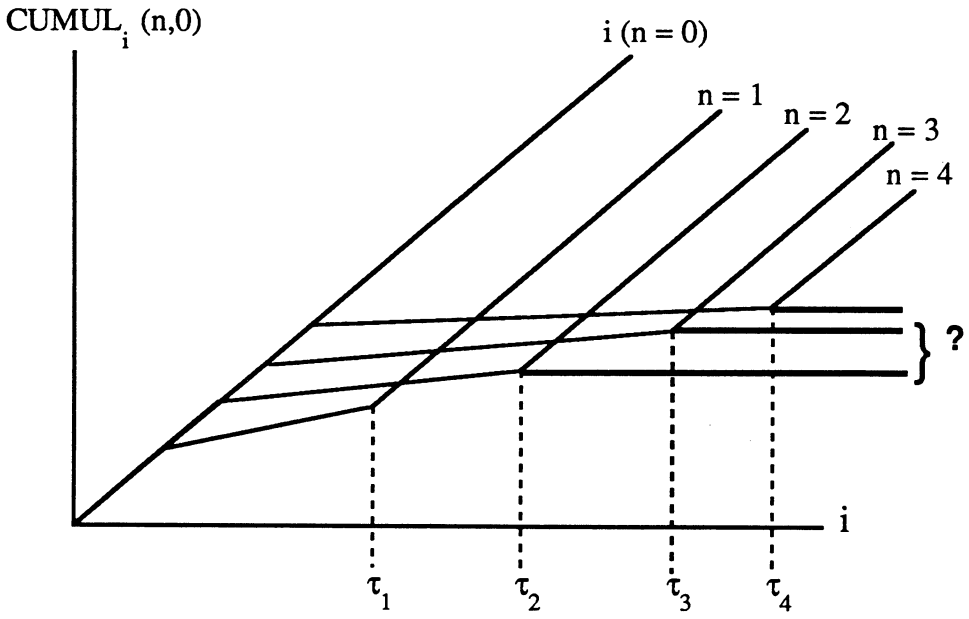
Conclusion :

°Le cadre de l'étude est restreint, comme dans le cas de l'anneau, même si nous avons utilisé un générateur aléatoire de polynômes et les conclusions que nous pouvons tirer ne sont valables que dans ce cadre.

°°Pour les mêmes raisons que pour l'anneau nous n'avons pas utilisé la notion d'efficacité : le volume des calculs exécutés au cours de l'algorithme n'est pas constant lorsque l'on modifie la dimension de l'hypercube.

°°°Pour des exemples dépassant notre cadre il est difficile de conclure. Que peut-il se passer ?

On représente la quantité $CUMUL_i(n, 0)$ en fonction de i de la façon suivante :



On pourrait avoir sur $\mathcal{H}(n)$ le même type de comportement que sur $\mathcal{H}(1)$, c'est à dire qu'à partir d'un certain temps τ_n le parallélisme n'apporterait plus rien et la courbe $CUMUL_i(n,0)$ deviendrait parallèle à i.

A l'opposé on pourrait espérer que l'apport du parallélisme ne soit pas nul et que l'on obtienne sur le graphique une asymptote horizontale pour $CUMUL_i(n,0)$.

Pour trancher entre ces deux hypothèses il serait nécessaire de tester les algorithmes sur des exemples dont le temps séquentiel soit très important (de l'ordre de l'heure ?).

°°°°Il faut encore souligner l'importance de la répartition des données qui conditionne de façon importante les temps d'exécution des algorithmes.

°°°°°Le problème de synchronisation est là encore inévitable et influence énormement la manière de paralléliser les algorithmes.

CONCLUSION :

Dans ce travail les bases de Gröbner ont d'abord été considérées comme un outil puis comme un objet à part entière sur lequel on peut agir.

En tant d'outil elles peuvent résoudre certains problèmes, en particulier des problèmes simples de logique et de preuve de circuits. Ce n'est qu'une parcelle de ce que l'on pourrait faire dans ce domaine. Un algorithme est toutefois proposé pour la preuve de circuits combinatoires. Il serait intéressant d'élargir ce domaine d'application dans le cas de la logique temporelle [Chazarain & Riscos88], [Kapur & Narendram85] et pouvoir ainsi traiter des circuits avec "rebouclage".

En tant qu'objet, elles peuvent être calculées par des algorithmes classiques (séquentiel). Ces algorithmes s'adaptent au cas des polynômes booléens qui permettent d'avoir des structures simples à manipuler et des opérations simples à exécuter. Ceci nous affranchit d'un certain nombre de problèmes inhérents au calcul des bases de Gröbner et nous permet de traiter la parallélisation de tels algorithmes (par essence séquentiels). Deux méthodes de parallélisation de granularités différentes ont été proposées : une granularité fine pour la parallélisation sur l'anneau et une granularité plus grosse pour la parallélisation sur l'hypercube.

Leurs performances et leurs comportements respectifs sont liées à la machine utilisée (en particulier à la synchronisation des taches). L'étude quantitative et qualitative du premier algorithme parallèle - dont les tâches indépendantes s'effectuent en parallèle et où les polynômes circulent le long d'un anneau - montre l'interêt de la parallélisation surtout pour des données de tailles importantes.

Le deuxième algorithme - basé sur un principe de "divide and conquer" - dont le nombre de communications dépend uniquement de la taille de l'hypercube considéré a un comportement beaucoup plus proche de l'algorithme séquentiel car il est construit à partir d'une module de base correspondant à un algorithme complet de calcul de base de Gröbner (exécuté sur des données de taille plus petites). Il serait intéressant de disposer de bornes théoriques dans le cas booléen pour pouvoir expliquer les comportements obversés des deux algorithmes et tenter de les étendre.

Au vu des résultats obtenus il semble qu'il soit plus intéressant (du moins pour des temps séquentiels importants) d'utiliser P1RGBuch plutôt que P2RGBuch.

Pour étudier l'efficacité de la parallélisation des bases de Gröbner il semble nécessaire de concevoir des méthodes de comparaisons spécifiques.
En effet nous avons longuement insisté sur le fait que les opérations réalisées n'étaient pas les mêmes en séquentiel et en parallèle et, même dans ce dernier cas, qu'elles variaient suivant le nombre de processeurs. Les comparaisons à l'aide des outils classiques ne présentent donc ici que peu d'utilité et c'est pourquoi nous avons essayé d'introduire de nouvelles méthodes d'analyse.

Il faut souligner, encore une fois, que les études ont été menées dans un cadre très particulier et que l'on doit rester très prudent quant à la généralisation de nos conclusions.

Quel que soit l'algorithme choisi nous avons pu noter l'importance de la répartition et nous pensons qu'il y aurait tout intéret à déterminer une stratégie performante en ce domaine.

Il nous paraît intéressant a posteriori de transposer P1RGBuch (cf chapitre3 partie programmation) sur un hypercube.

Finalement : il reste encore beaucoup à faire.

BIBLIOGRAPHIE

[Barrow84]
H.BARROW: "Verify a Program for Proving Correctess of Digital Hardware Designs" Artuifical Intelligence 1984

[Bryant86]
R.E.BRYANT : "Graph based algorithms for boolean function manipulation" IEEE Trans. on Computers Vol. C-35 8 p 677-691 1986.

[Buchberger70]
B.BUCHBERGER : "Ein algorithlisches Kriterium for die Lösbrarkeit eines algebraischen gleichungsystemsss" Aequationes Math 4 , 374-383, 1970.

[Buchberger79]
B.BUCHBERGER : "A criterion for detecting unnecessary reductions in the constructions of Gröbner bases" Proc EUROSAM 79, Marseille, juin 1979 (ed. by W. Ng), Springer LNCS 72,3-21.

[Buchberger83a]
B.BUCHBERGER : "Gröbner bases : an algorithmic method in polynomial ideal theory" Recent Trends in multidimensional systems theory chap 6 (ed. by N.k;Bose), Reidel 1985.

[Buchberger83b]
B.BUCHBERGER : "A note on the complexity of constructing Gröbner bases" Proc. Eurocal 83 , London, Mars 1983, (ed. by J. A. Van Hulzen), Springer LNCS 162, 137-145

[Chang &Lee70]
C.CHANG, R.CHAR-TUNG LEE : "Symbolic Logic and Mechanical Theorem Proving" Computer Science and Applied mathemarics. ed Werner Rheinbolt Academic Press73.

[Chazarain86]
J.CHAZARAIN : "The lady, the tiger and the Gröbner basis" Preprint n100 University of Nice. department of Mathematics march1986.

[Chazarain & Riscos88]
J.CHAZARAIN, A.RISCOS : "Multi-valued logic and Gröbner basis with application to modal logic" Preprint University of Nice. department of Mathematics 1988.

[Davenport85]
J. H.DAVENPORT : "Computer Algebra for Cylindrical Decomposition ": Bath Computer Science Technical Report 88-10

[Davenport & al86]
J.H DAVENPORT, Y. SIRET, E. TOURNIER : "Calcul Formel : Systèmes et Algorithmes de Manipulations Algébriques " Masson- collection eri 1986

[Davis &Putnam60]
M.DAVIS, H.PUTNAM : "A Computing Procedure for Quantification Theory"
J.A.C.M vol.7 number 3 mars1960 p201-215.

[Dunham & al59]
B.DUNHAM,R.FRIDSHAL,G.L.SWARD : "A Non-heuristic Program for Proving Elementary Logical Theorems"
Information Processing Unesco21960.

[Faugère & al89]
J.C. FAUGERE, P.GIANNI, D.LAZARD, T.MORA : "Efficient computation of zero-dimensional Gröbner bases by change of ordering"
LITP RR 89-52, juillet 1989.

[Flynn66]
M.J.FLYNN : "Very high-speed computing systems"
Proc. IEEE,54, p1901-1909, 1966

[Grätzer78]
G.GRATZER : "Universal Algebra"
Spinger_VerlagNew York Inc. Second édition 1979.

[Hariz88]
A.HARIZ, " Preuve formelle des circuits intégrés"
mémoire pour obtenir le titre d'inégnieur concepteur, TIM3 INPG, Juin 1988

[Hironaka75]
J.M.AROCA, H.HIRONAKA J.L.VICENTE : "The theory of maximal contact"
Memorias de Matematica del Instituto "Jorge Juan" n 29, 1975.

[Hullot80]
J.M.HULLOT : "Compilation de Formes Canoniques dans des théories équationnelles".
Thèse de docteur troisième cycle soutenue à l'université de Paris sude Centre d'Orsay, nov. 1980.

[Hsiang83]
J.HSIANG : "Topics in Automated Theorem Proving and Program Generation"
Thesis submitted for the degree of Doctor of Philosophy in Computer Science in the Graduate College of the University of Illinios at Urbana-Champaign, 1983.

[Guisti85]
M.GUISTI : "A note on the complexity of constructing standard bases"
Proc. vol 2 Eurocal 85 , Linz, Avril 1985, (ed. by B.F Caviness), Springer.

[Kapur & Narendram85]
D.KAPUR, P.NARENDRAM : "An Equational approach to Theorem Proving in First Predicate Calculus"
Proc. IJCAI, 1985, Los Angeles.

[Kuntzmann65]
J.KUNTZMANN :"Algèbre de Boole"
Dunod Paris 1965, Bibliothèque de l'ingénieur automaticien.

[Lascaux & Theodor86]
P.LASCAUX, R.THEODOR : "Analyse numérique matricielle appliquée à l'art de l'ingénieur" tome1, Masson 1986.

[Lazard83]
D.LAZARD : "Gröbner bases, Gaussian elimination and resolution of systems of algebraic equations"
Proc. Eurocal 83 , London, Mars 1983, (ed. by J. A. Van Hulzen), Springer LNCS 162, 137-145.

[Lejeune85]
M.LEJEUNE-JALABERT : "Effectivité de calculs polynomiaux"
Cours de DEA Université Grenoble I. Institut Fourier. Laboratoire de Mathematiques.

[Möller & Mora84]
H.M.MÖLLER, F.MORA : "Upper and lower for the degree of Gröbner bases"
Proc.Eurosam 84 , Cambridge, july 1984, (ed. by J. Fitch), Springer.

[Melenk & al88a]
H.MELENK, H.M.MÖLLER, W.NEUN : "On Gröbner Bases Computation on a Supercomputer Using REDUCE"
FB Mathematik und Informatik der Fernuniversität Hagen. Preprint SC 88-2.

[Melenk & al88b]
H.MELENK, H.M.MÖLLER, W.NEUN : "Symbolic solution of Large Stationnary Chemical Kinetics Problems"
FB Mathematik und Informatik der Fernuniversität Hagen. Preprint SC 88-7.

[Ponder88]
C. PONDER : " Parallelism Algorithms For Gröbner- Basis Reduction"
R.R Electrical Engineering and Computer Sciences University of California, Berkeley, CA. 94720 june 1988.

[Quine52]
W.V.QUINE : The problem of Simplifying Truth Functions"
American mathematical monthly oct 1952 p521-531

[Robinson65]
J.A.ROBINSON : "A Machine-Oriented Logic Based on the Resolution Principle"
J.A.C.M vol. 12, number1, january 1965.

[Roch & al 89]
J.L. ROCH, P. SENECHAUD, F. SIEBERT-ROCH, G. VILLARD : "Computer Algebra on MIMD Machine "
SIGSAM Bull., vol. 23, n° 1, Janv. 1989.

[Roch89]
J.L. ROCH : "Architecture du système PAC et son arithmetique rationnelle"
Thèse INPG , déc 1989.

[Saad86]
Y.SADD : "Communication complexity of the Gaussian elimination on a multiprocessor"
Lin. Alg.Appl. 77, 1986.

[Sakai & Sato88]
K.SAKAI, Y.SATO : " Boolean Groebner Bases"
ICOT, Institut for new generation computer technology March 1988.

[Sertafi73]
M.SERFATI : "Algèbres de Boole, introduction à la théorie algébrique des graphes orientés et aux sous ensembes flous"
C.D.U numero 25000. troisième semestre 1974.

[Stilllmann89]
J.STILLMANN : " The complexity of Computing the Gröbner Basis of an ideal Over a Boolean Polynomial ring"
R.R of State Univerty of New York at Albany March 1989.

[Stone36]
M.STONE : " The theory of Representation for Boolean Algebra", Trans.AMS 40, 1936 pp 37-137.

[Traverso & Donati89]
C.TRAVERSO, L.DONATI : "Experimenting the Gröbner Basis Algorithm With A1PI System"
Proc. ACM-SIGSAM 1989. ISSAC 89, Portland.

[Trinks78]
W.TRINKS : "On B.Buchberger's Method of Solving Algebraic Equations"
J. Number Theory 10/4 475-488, 1978.

[Villard88]
G.VILLARD : " Calcul formel et parallélisme: Résolution de sustèmes linéaires"
Thèse INPG, déc 88.

[Watt85]
S.WATT : "Bounded Parallelism in Computer Algebra"
Thesis presented to the University of Waterloo Ontario 1985

[Winkler84]
F.WINKLER : "On the complexity of Gröbner-bases algorithm over K[x,y,z]"
Proc.Eurosam 84 , Cambridge, july 1984, (ed. by J. Fitch), Springer.

[Xuong]
N.H.XUONG " Elements de combinatoire pour l'informatique : deuxième partie graphes théorie, algorithmes et applications"
Cours 1ière année INPG - ENSIMAG ; licence d'informatique à l'UJF.

AUTORISATION de SOUTENANCE

VU les dispositions de l'Arrêté du 23 novembre 1988 relatif aux Etudes doctorales

VU les rapports de présentation de

- Monsieur Yves ROBERT
- Monsieur James DAVENPORT

Mademoiselle SENECHAUD Pascale

est autorisé(e) à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR de L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, spécialité "Mathématiques Appliquées"

Fait à Grenoble, le 29 Janvier 1990

Pour le Président de l'I.N.P.-G
et par délégation,
le Vice-Président
P. VENNEREAU

Résumé :

Nous présentons les bases de Gröbner, leur utilisation et la parallélisation des algorithmes qui les calculent dans le cas de polynômes booléens.
Une première partie est consacrée à la présentation théorique des bases de Gröbner dans le cas général. Cette présentation se veut accessible à des non-specialistes. Une étude bibliographique de la complexité est faite.
Une deuxième partie concerne les applications des bases de Gröbner booléennes en calcul propositionnel et en preuve de circuits combinatoires. Nous proposons un algorithme de preuve formelle de circuits combinatoires hiérarchisés.
Dans la troisième partie nous adaptons l'algorithme séquentiel au cas booléen et nous étudions plus en détail la normalisation. Nous proposons deux méthodes de parallélisation à granularités différentes. Nous analysons et comparons plusieurs implantations parallèles et présentons des résultats expérimentaux. Les algorithmes sont généralisables au cas des polynômes à coefficients rationnels. Nous soulignons l'influence de la répartition des données sur le temps d'exécution. Nous présentons une méthode de répartition des polynômes basée sur la recherche de chemins de longueur donnée dans un graphe orienté. Cette répartition nous permet d'obtenir des résultats interprétables et de conclure sur les différents algorithmes.

Mots clés :

Algèbre de Boole, algorithmique parallèle, anneau de Boole, base de Grobner, calcul propositionnel, preuve de circuits.