Contribution à un environnement pour le calcul scientifique et la modélisation : strates et systèmes polynômiaux sur les corps finis

Pierre-Olivier Garreau

THESE

Spécialité : Mathématiques Appliquées

Présentée à

Université Joseph–Fourier – Grenoble I

(arrêtés ministériels du 5 juillet 1984 et du 30 mars 1992)

par

Pierre–Olivier Garreau

Contribution à un environnement pour le calcul scientifique et la modélisation : strates et systèmes polynomiaux sur les corps finis

Thèse soutenue devant la commission d'examen le :

30 septembre 1994

Composition du jury :

M. J. Della–Dora	Président
M. J.–P. Dedieu	Rapporteurs
M. G. Jacob	
M. P. Chenin	Examinateurs
M. B. Lacolle	
M. V. Quint	

Thèse préparée au sein du laboratoire LMC–IMAG

Le travail de recherche et de rédaction qui me permet de soutenir cette thèse n'aurait jamais pu être réalisé sans l'aide morale, scientifique et technique de certaines personnes. Parmi ces personnes je voudrais tout d'abord citer ma femme, Catherine, dont le soutien m'a été très précieux lors de ces années de thèse. C'est grâce à elle, et à mon fils Thomas, que j'ai pu disposer de la chaleur familiale qui m'a été bien souvent nécessaire pour poursuivre ce travail. Je tiens ensuite à remercier :

- Monsieur Jean Della Dora, Professeur à l'INPG, de l'honneur qu'il me fait de présider le jury.
- Messieurs Jean–Pierre Dedieu, Professeur à l'Université Paul Sabatier, et Gérard Jacob, Professeur à l'Université de Lille, d'avoir accepté d'être mes rapporteurs, je les en remercie vivement pour leurs remarques et corrections apportées sur cette thèse.
- Monsieur Patrick Chenin, Professeur à l'Université Joseph–Fourier, directeur de cette thèse, pour l'aide, les conseils et les orientations apportés sur ce travail tout au long de ces années. Qu'il trouve ici mes sentiments reconnaissants et sincères.
- Messieurs Bernard Lacolle, Professeur à l'Université Joseph–Fourier, et Vincent Quint, Directeur de Recherche à l'INRIA, d'avoir bien voulu participer au jury.
- Madame Irène Vatton et Monsieur Vincent Quint pour leur disponibilité et leur soutien logistique relatif au logiciel Grif développé à l'unité mixte Bull–Imag.
- Messieurs Luc Biard et Marcel Bouhier pour les nombreuses discussions passées relatives à ce travail de thèse.
- Messieurs Pierre–Jean Laurent et Jean–Marie Loiseaux pour la résolution de problèmes administratifs.

Un grand merci à tous les passagers du bureau 56, de la salle machine, et des autres personnes de la tour IRMA, pour leur amitié, pour leur compagnie et pour l'ambiance studieuse dans la bonne humeur qu'ils ont su maintenir tout au long de ces années.

Je remercie mes parents qui m'ont permis de faire de longues études, ainsi que mes soeurs, ma famille et belle–famille pour leur soutien et présence.

And thanks a lot to David Plowman for having given me your mathematics and computer virus.

Je remercie finalement tous ceux qui m'ont accompagné, aidé, d'une façon ou d'une autre, à un moment ou un autre au cours du long cheminement que représente cette thèse.

A ma femme Catherine pour son amour et sa patience,
à mon fils Thomas pour ses grands sourires

Résumé :

Cette thèse concerne le développement et la mise en œuvre d'un environnement pour le calcul scientifique et la modélisation. L'approche retenue est celle d'une décomposition stratifiée des problèmes scientifiques, ceci dans un double but :

- de marquer le cheminement progressif des étapes de description, allant de l'énoncé informel vers un langage cible en passant par des langages d'abstractions intermédiaires plus ou moins formalisés : langages naturel, mathématiques, algorithmiques, de programmation, etc;
- et d'obtenir une décomposition structurée, modulaire, pour aller du problème initial vers le programme.

Afin de disposer de méthodes pour un tel environnement, nous associons à tout énoncé en cours des conditions logiques dépendant d'un langage de description. Ceci dans le but de vérifier si la cohérence de la description du problème, mais aussi des flux de données intervenant dans un schéma de résolution est correct, ou non. Pour cela, il nous a paru nécessaire d'étudier des formulations logiques décrites par des systèmes polynômiaux sur les corps finis de la forme $\mathbf{Z}/p\mathbf{Z}$.

L'étude de ces systèmes nous conduisent à traiter le problème de l'élimination de quantificateurs sur un corps fini. Nous proposons un algorithme général, puis, dans le cas des systèmes polynômiaux plusieurs algorithmes sont proposés dont une généralisation de la méthode de Dixon–Biard. Enfin le problème de la déduction $P(x) \equiv 0\ [p] \Rightarrow Q(x) \equiv 0\ [p]$ est abordé. Ces algorithmes nous permettent de vérifier localement la cohérence d'un énoncé, mais aussi d'une décomposition de problèmes. Ceci rend envisageable la vérification globale d'un schéma de résolution de problème.

Ces aspects formels sont complétés par la réalisation d'un environnement d'édition de résolution de problèmes mathématiques, en lien avec des systèmes de calculs algébriques et numériques (Reduce, ESSL), sous le logiciel Grif, éditeur de documents structurés.

Mots clefs

environnement de programmation
modélisation
systèmes polynômiaux
géométrie algébrique
spécification
corps fini
résolution de problèmes

Abstract :

The study and the development of a scientific computing and modelling environment is the object of this thesis. The approach decided on is a stratified decomposition of the scientific problems. Our two aims in doing this are :

- to mark the progression of the phases of the description, from the informal towards the task language through intermediary abstraction languages more or less formalised : natural language, mathematics, algorithmics, computing, etc;
- and to get a structured and modular decomposition, to go from the original problem to the program.

In order to have methods for such an environment, we give a set of logical conditions depending on the description language to every decomposition. This enables us to check whether the validity of a problem description, but also the data flow which occurs in a resolution , is correct or not. With this aim in view, it was necessary to study logical formulations described by polynomial systems over finite fields of the form $\mathbf{Z}/p\mathbf{Z}$.

The study of these systems for such an environment, leads us to the quantifier elimination problem over a finite field. A general algorithm is given, then in the case of polynomial systems other algorithms are given, one of them is the generalisation of the Dixon–Biard method. Then we have the deduction problem $P(x) \equiv 0\ [p] \Rightarrow Q(x) \equiv 0\ [p]$. These algorithms allow us to check the validity of a problem locally, but also of a problem decomposition. Therefore, we can contemplate the global checking of a problem solving.

These formal aspects are completed by the realisation of an edition environment for solving mathematical problems, linked with algebraic and numerical systems (Reduce, ESSL), under Grif sofware, editor of structured documents.

Keywords

programming environment
modelling
polynomial systems
algebraic geometry
specification
finite field
problem solving

Introduction

Le travail présenté ici a pour thème le développement et la mise en œuvre d'un environnement pour le calcul scientifique et la modélisation. C'est un vaste domaine qui comporte de multiples aspects, tant informatiques que mathématiques, portant sur les systèmes de calcul numériques, les systèmes de calcul formel, les systèmes experts, les systèmes à base de connaissance, les langages de programmations, la génération automatique de programmes, les environnements de programmations, la visualisation graphique de courbes et surfaces, les éditeurs spécialisés pour les mathématiques, les interfaces hommes–machines, etc. La liste est longue et n'est pas exhaustive, mais nous pouvons constater que ce domaine relève essentiellement de techniques du génie informatique et des mathématiques appliquées.

Aussi sommes–nous plus particulièrement intéressés dans un premier temps à la conception d'un éditeur de formules mathématiques, GramMath, en lien avec des systèmes de calcul formels et numériques. Nous avons ainsi réalisé une telle interface sous le logiciel Grif, un environnement d'édition de documents structurés, développé par Quint et Vatton, [34], 35], [36] et [37]. Cette réalisation, présentée plus en détail en annexe B, nous permet d'évaluer par un système de calcul formel ou numérique les formules mathématiques éditées. Les résultats obtenus par ces systèmes sont à leur tour intégrés dans l'environnement d'édition, et peuvent être de nouveau manipulés par l'utilisateur. Ce traitement de texte et de formules mathématiques rejoint en ce sens les cahiers de brouillon interactifs du logiciel Camino Real [4] de Xerox, Palo Alto, ainsi que les blocs notes des systèmes de calculs formels comportant des aspects graphiques comme Mathematica [44], ou Maple. Cependant, ce type d'interface qui nous permet l'édition de feuilles de calculs scientifique, présente quelques limites pour des problèmes issus de la modélisation ou des problèmes de taille conséquente. Plus particulièrement lorsque nous avons à développer à la suite de ces calculs un programme, ou si nous voulons reprendre les calculs pour un problème similaire.

Dans un deuxième temps, dans le but de gérer une résolution de problème menée par un utilisateur potentiel, nous présentons une structure, les strates, au chapitre V. Ces strates nous permettent de décrire un schéma de résolution, de définir des problèmes et ce jusqu'à aboutir au niveau de programmation. Cependant pour assurer quelques vérifications sur les décompositions de problèmes en sous–problèmes, sur les schémas de résolutions, nous proposons d'associer aux langages de description employés par les conditions d'un problème, une logique qui s'intéresse aux états pris par les variables mises en jeu dans les flux de données. Ces états sont en nombre finis et la logique est multivaluée. Nous inspirons pour cela des travaux sur le langage Signal de l'équipe de Benveniste : Le Borgne [24], Le

Guernic et Benveniste [25], Le Guernic et Gautier [26], mais également de résultats présentés par Chazarain *et al* [11] sur des langages de type Pascal. Ces études associent à toute formule de la logique multi–valuée un polynôme sur un corps fini $\mathbf{Z}/p\mathbf{Z}$. Nous abordons et présentons dans la première partie de ce mémoire, les chapitres I à IV, plusieurs résultats à ce sujet. Ces résultats nous permettent d'étudier au sens logique la validité des descriptions des strates, et nous présentons en annexe C un tel exemple de vérification des strates.

Nous montrons qu'à toute formule logique m–valuée nous pouvons lui associer un polynôme sur un corps fini $\mathbf{Z}/p\mathbf{Z}$, avec $p \geq m$. Pour cela nous énonçons plusieurs lemmes sur les connecteurs logiques aux chapitres I et III, qui nous permettent de réécrire toute formule logique non quantifiée portant sur des congruences modulo p sous la forme d'un polynôme sur $\mathbf{Z}/p\mathbf{Z}$. Parmi les lemmes proposés aux chapitres I et III, nous énonçons et démontrons une proposition, la proposition III(3), qui nous permet de minimiser le degré employé pour le lemme du connecteur "et". Puis pour résoudre le problème d'une représentation unique pour toute formule logique, car lorsque p est supérieur à 2 nous n'avons plus l'isomorphisme de Stone, nous proposons d'autres lemmes mais qui sont bien plus coûteux d'un point de vue des calculs symboliques.
Dans certains cas nous pouvons réduire la caractéristique des corps finis pour des langages qui possèdent des états qui impliquent d'autres. Un exemple lié au cas de Signal est exposé au chapitre VI. Ceci nous amène à définir et proposer en plus des résolutions habituelles en une variable x, des résolutions en "x puissance d". Cet artifice nous permet à la fois de diminuer la caractéristique des corps finis mais aussi de réduire le degré pour ces résolutions.

Lorsque nous avons une formule logique multi–valuée quantifiée, nous nous intéressons au problème de l'élimination de ces quantificateurs et éventuellement par la recherche des solutions associées à ces variables quantifiées. Ce problème de l'élimination des quantificateurs est abordé aux chapitres II et III, et plusieurs théorèmes et algorithmes sont proposés.
De manière générale, le théorème II(1) nous permet d'éliminer des quantificateurs pour toute formule logique multi–valuée du premier ordre. La démonstration propose un fait un algorithme possible d'élimination. Nous avons étudié la complexité d'un tel algorithme et le résultat donné par la proposition II(2) montre qu'un tel algorithme est de complexité exponentielle en le nombre de variables. Puis pour le cas des corps $\mathbf{Z}/3\mathbf{Z}$, un autre algorithme nous permet non seulement de faire de l'élimination mais aussi de déterminer l'ensemble de toutes les solutions possibles. A la suite de ces propositions, nous montrons à travers plusieurs exemples et expérimentations que la réécriture d'une formule logique en introduisant de nouvelles variables nous permet de diminuer parfois les temps de calculs. Les résultats des expérimentations obtenus sur une machine RS/6000 nous permettent d'éliminer ainsi jusqu'à 12 variables quantifiées d'un polynôme qui comporte de plus une dizaine de paramètres.

Puis pour le cas de l'élimination de quantificateurs sur un système polynômial, nous énonçons un théorème III(1) qui porte sur la classe des algorithmes d'élimination incrémentales. Ce théorème nous permet d'encadrer la complexité de ces algorithmes entre deux bornes, l'une supérieure mais voisine de la borne exponentielle établie par l'algorithme précédent, et la borne inférieure, une borne linéaire en le nombre de variables à éliminer mais avec un facteur exponentiel en le nombre des paramètres. Les expérimentations menées sur divers exemples, nous montrent qu'en général les temps d'éliminations sont meilleurs que ceux obtenus au chapitre II.

Dans le but d'améliorer la complexité des algorithmes d'élimination, nous avons tenté d'adapter les méthodes issues de la théorie de l'élimination sur les corps infinis au cas des corps finis. Une étude sur les résultants à une et deux variables est abordée. Nous montronas ainsi que le résultant de Sylvester peut échouer tandis que le résultant de Caley non. Puis pour le cas de l'élimination de n variables nous généralisons sans démontrer, la méthode de Dixon–Biard que Biard démontre pour n=2 dans sa thèse [7]. Cependant cette méthode fournit l'équation implicite associée à un système. Aussi nous généralisons un algorithme d'inversion qui n'est plus formel mais numérique, qui nous permet de savoir si la réciproque est vérifiée ou non.

L'exposé est ainsi organisé : nous présentons tout d'abord les techniques d'élimination de variables d'un système polynômial sur un corps fini $\mathbf{Z}/p\mathbf{Z}$, puis dans les deux derniers chapitres, V et VI, nous présentons les strates. En annexe, nous donnons les programmes Reduce implémentant les algorithmes d'éliminations de variables sur un corps fini, et les programmes liés à la méthode de Dixon–Biard ainsi qu'une version de l'algorithme de Bareiss. La seconde annexe porte sur l'éditeur GramMath et sa composante calcul formel ou numérique. Et enfin dans l'annexe C nous présentons un exemple d'application des strates sur un problème issu de l'industrie.

Les chapitres sont constitués comme suit :

Chapitre I

L'objet de ce chapitre est de présenter les outils algébriques qui serviront par la suite. Ils concernent essentiellement le calcul dans un corps fini de type $\mathbf{Z}/p\mathbf{Z}$, où p désigne un nombre premier; et à la réécriture d'un ensemble de propositions logiques sur les équations de $\mathbf{Z}/p\mathbf{Z}$ en une seule équation.

Puis, nous énonçons des définitions relatives aux formules logiques. Elles nous faciliteront la formalisation des problèmes de décision et d'élimination. Quant aux lemmes sur les connecteurs logiques, ils nous permettront d'établir, au chapitre I, un théorème important sur les propositions des corps finis, mais aussi d'étudier l'existence de racines d'un polynôme de $\mathbf{Z}/p\mathbf{Z}$ et de les caractériser dans un cas particulier qui nous intéresse plus particulièrement, $\mathbf{Z}/3\mathbf{Z}$. Ceci fait l'objet de la section I.1.

Chapitre II

Dans ce chapitre, nous abordons le problème de l'élimination des quantificateurs d'une formule logique sur un corps $\mathbf{Z}/p\mathbf{Z}$. Nous proposons un algorithme d'élimination qui détermine l'équation de contrainte associée à une formule logique donnée. De plus, pour les corps $\mathbf{Z}/3\mathbf{Z}$, nous montrons que l'on peut déterminer les inconnues quantifiées par le quanteur existentiel. Ceci permet non seulement de fournir une condition pour qu'une formule, dans un corps fini, soit vraie, mais aussi de fournir la ou les solutions possibles. Cependant, les algorithmes proposés sont extrèmement coûteux, un coût exponentiel.

Chapitre III

Nous nous intéressons dans ce chapitre à l'élimination de variables d'un système polynômial. Lorsque nous éliminons des variables d'un système polynômial sur $\mathbf{Z}/p\mathbf{Z}$, nous proposons par le biais du théorème II(1), une méthodologie qui nous permet dans certaines situations, de réduire ce coût de façon non négligeable. En théorie, le coût reste exponentiel mais nous montrons dans le corollaire II(2), que ce coût est pour la plupart du temps bien inférieur à celui de l'élimination polynômiale d'un polynôme équivalent à un système polynômial. A la suite de ces démonstrations nous proposons plusieurs algorithmes à la section III.2 que nous expérimentons et appliquons sur des systèmes et sur des décompositions de systèmes à la section III.3.

Nous définissons la résolution de polynômes en « x puissance d », puis ce qu'est un système polynômial bien formé. Ces définitions vont nous permettre au chapitre VI de vérifier si une strate est cohérente en–elle même, c'est–à–dire si ses spécifications ne sont pas contradictoires ou inconsistantes par rapport à la logique associée au langage des spécifications.

Chapitre IV

Nous nous intéressons à un problème similaire de celui du chapitre précédent mais qui relève de la théorie de l'élimination. Ceci dans le but d'améliorer la complexité des algorithmes d'éliminations précédents, exponentielle, en une complexité qui serait au mieux polynômiale. Nous rappelons et exposons les principales méthodes d'élimination dans le cas des corps des réels et des complexes, puis nous tentons de les adapter au cas des corps finis $\mathbf{Z}/p\mathbf{Z}$.

Ce chapitre est composé de plusieurs parties qui traitent le cas de l'élimination d'une variable, puis de deux, et enfin de n variables en généralisant la méthode de Dixon et de Biard sans toutefois la démontrer. Nous concluons sur ces méthodes en les comparant avec celles développées aux chapitres précédents.

Chapitre V

Dans le but de définir un environnement de modélisation pour le calcul scientifique, nous définissons une structure que nous nommons *strate*. Elle nous permet de définir le problème initial, puis de suivre la progression de la résolution du problème mathématique jusqu'au niveau algorithmique ou de programmation. Les aspects des strates permettant la description d'un problème et ceux de la résolution de problème que nous avons jugés pertinents de formaliser sont les suivants :

- les descriptions de problèmes à l'aide de préconditions, conditions et post-conditions (V.1.1);
- les schémas de résolutions (V.1.2);
- les différents niveaux de description à travers les langages (V.3);
- les traductions de problèmes (V.3.2);
- les arguments permettant de passer ou de justifier un nouveau problème (V.1).

Afin de présenter ces différents aspects des strates, nous avons implémenté la structure des strates, ainsi que la présentation visuelle associée sous Grif.

Chapitre VI

Suite au chapitre précédent où nous avons introduit les strates, nous étudions les outils algébriques qui sont suceptibles de vérifier la cohérence des strates. A toute strate, nous associons une interprétation, défini au chapitre III, qui est une représentation logique sur un corps fini de la strate, une logique associée au langage employé pour décrire une strate. Ceci nous permet de vérifier la cohérence d'une strate ainsi que la cohérence des décompositions, ou des recompositions, des strates. L'interprétation des strates est à distinguer de la sémantique des strates, qui est le sens ou la représentation que nous attribuons à la formulation d'un problème mathématique ou de modélisation. Or, à toute formulation logique sur $\mathbf{Z}/p\mathbf{Z}$, nous avons vu que nous pouvions associer un polynôme sur $\mathbf{Z}/p\mathbf{Z}$, aussi pouvons–nous avoir une représentation algébrique des strates. Les questions de vérification de la cohérence d'une strate ou d'une décomposition en une autre strate, sont des questions qui sont formulées au niveau algébrique.

Puis, nous arbordons la vérification des décompositions d'une strate en une autre. Ceci revient à étudier au niveau algébrique un problème de déduction : quand avons–nous un polynôme qui peut–être déduit d'un système polynômial ? Une méthode, la méthode de Wu permet d'y répondre dans certains cas. Nous proposons une autre approche, par des méthodes, plus coûteuses, mais qui permettent de répondre à ce problème lorsque le corps est fini.

« Dans ses « Règles », Descartes projetait de développer une méthode universelle de résolution de problèmes. Voici une grossière esquisse du schéma que Descartes pensait pouvoir appliquer aux problèmes de toutes natures :

Ramener d'abord tout problème à un problème mathématique.

Ramener ensuite tout problème mathématique à un problème d'algèbre.

Ramener enfin tout problème d'algèbre à la résolution d'une seule équation. »

George Pólya, "La Découverte des Mathématiques", tome 1, 1962, à propos du "Discours de la Méthode" de René Descartes.

Table des matières

Chapitre I

Arithmétique des corps $\mathbf{Z}/p\mathbf{Z}$

L'objet de ce chapitre est de présenter les outils algébriques qui serviront aux chapitres suivants. Ils concernent essentiellement le calcul dans un corps fini de type $\mathbf{Z}/p\mathbf{Z}$, où p désigne un nombre premier; et à la réécriture d'un ensemble de propositions logiques sur les équations de $\mathbf{Z}/p\mathbf{Z}$ en une seule équation.

Nous redonnons pour la première partie de ce chapitre les démonstrations des propositions sur ces corps finis. De plus, nous nous sommes efforcés de fournir, dans la mesure du possible, des démonstrations qui ne fassent pas appels à des connaissances mathématiques approfondies dans les théories des nombres et des corps.

Suite à cette première partie, nous énonçons des définitions relatives aux formules logiques. Elles nous faciliteront la formalisation des problèmes de décision et d'élimination. Quant aux lemmes sur les connecteurs logiques, ils nous permettront d'établir, au chapitre suivant, un théorème important sur les propositions des corps finis, mais aussi d'étudier l'existence de racines d'un polynôme de $\mathbf{Z}/p\mathbf{Z}$ et de les caractériser dans un cas particulier qui nous intéresse plus particulièrement, $\mathbf{Z}/3\mathbf{Z}$. Ceci fait l'objet de la dernière section de ce chapitre.

I.1 Les classes résiduelles

Nous rappelons ci–dessous l'arithmétique des corps $\mathbf{Z}/p\mathbf{Z}$. Nous retrouvons les énoncés de ces propositions dans Davenport [17] ou Ribenboim [39].

I.1.1 Le calcul modulo p

Dans toutes les propositions énoncées ci–dessous, p désigne un nombre premier quelconque. Nous notons « α modulo p » par « α [p] », et « est congrue à » par le symbole $\equiv$ suivi si nécessaire de l'ordre de la congruence entre crochets.

Proposition I(1) :

Soient a et b deux entiers, nous avons :

$$(a+b)^p \equiv a^p + b^p \ [p] \tag{1}$$

Démonstration

Par la formule du binôme, nous développons :

$$(a+b)^p = \sum_{i=0}^{p} C_p^i \, a^{p-i} \, b^i$$

Par définition : $C_p^i = p\dfrac{(p-1)!}{i!\,(p-i)!}$

Comme p est premier, aucun facteur du dénominateur, $i!\,(p-i)!$, ne divise p sauf quand i vaut zéro ou p. Dans ce dernier cas : $C_p^0 = C_p^p = 1$. Par conséquent, pour tout i appartenant à $\{1, \ldots, p-1\}$, C_p^i est multiple de p. D'où la proposition. □

Proposition I(2) – petit théorème de Fermat :

Soit a un entier, nous avons :

$$a^p \equiv a \ [p] \tag{2}$$

Démonstration

Le théorème est vrai pour $a = 0$ et 1. En supposant le théorème vrai pour $(a-1)$, montrons qu'il est aussi vrai pour a. Nous avons $a = (a-1) + 1$; et nous pouvons appliquer la première proposition sur cette égalité :

$$a^p = ((a-1)+1)^p \equiv (a-1)^p + 1^p \ [p] \equiv (a-1) + 1 \ [p] \equiv a \ [p]$$

□

Proposition I(3) :

Soit x une indéterminée, nous avons :

$$x^p - x \equiv \prod_{i=0}^{p-1} (x-i) \ \ [p] \tag{3}$$

Démonstration

Suite au petit théorème de Fermat, tout entier x modulo p est racine de $x^p - x \ [p]$, d'où cette factorisation en facteurs linéaires. □

Proposition I(4) – résultat de Wilson :

Nous avons pour tout nombre p premier :

$$(p-1)! \equiv -1 \ [p] \tag{4}$$

Démonstration

Considérons le polynôme de la proposition précédente. Une simplification par x donne :

$$x^{p-1} - 1 \equiv \prod_{i=1}^{p-1} (x-i) \quad [p]$$

Les termes constants fournissent alors le résultat cherché. □

Proposition I(5) :

Pour tout entier k de $\{0, \dots, p-1\}$, nous avons :

$$C_{p-1}^{k} \equiv (-1)^{k} \, [p] \tag{5}$$

Démonstration

Nous avons vu que $C_p^k \equiv 0 \, [p]$ pour tout k différent de zéro et de p. La relation de construction du triangle de Pascal donne alors :

$$C_{p-1}^{k-1} + C_{p-1}^{k} = C_p^k \equiv 0 \, [p]$$

Qui peut se réécrire sous la forme :

$$C_{p-1}^{k} \equiv -C_{p-1}^{k-1} \, [p]$$

Cette relation de récurrence définit une suite géométrique de raison -1 et de premier terme 1, d'où la proposition pour tout k compris entre 0 et $(p-1)$ inclus. □

Proposition I(6) – Polynômes d'interpolation de Lagrange :

Soit Φ une table de valeurs dans $\mathbf{Z}/p\mathbf{Z}$ prises par les points de $\mathbf{Z}/p\mathbf{Z}^n$, nous pouvons exprimer une telle table par un polynôme P déterminé par la formule d'interpolation de Lagrange comme suit :

$$P(X_1, \dots, X_n) = (-1)^{np} \sum_{(i_1, \dots, i_n) \in \mathbf{Z}/p\mathbf{Z}^n} \Phi(i_1, \dots, i_n) \prod_{m=1}^{n} \prod_{k=0,\, k \neq i_m}^{p-1} (X_m - k) \tag{6}$$

Démonstration

A la différence des formules d'interpolations de Lagrange, nous n'avons pas de division pour le terme $X_m - k$. Aussi allons nous simplement montrer, ici, pour un m donné que :

$$\prod_{k=0,\, k \neq i_m}^{p-1} \frac{X_m - k}{i_m - k} = (-1)^{p} \prod_{k=0,\, k \neq i_m}^{p-1} (X_m - k)$$

Nous avons :

$$\prod_{k=0,\,k\neq i_m}^{p-1}(i_m-k)^{-1}=\prod_{0\le k<i_m}(i_m-k)^{-1}\prod_{i_m<k\le p-1}(i_m-k)^{-1}$$

$$=\prod_{1\le k\le i_m}k^{-1}\prod_{1\le k\le p-1-i_m}-k^{-1}=i_m!^{-1}(-1)^{p-1-i_m}(p-1-i_m)!^{-1}$$

Or, nous pouvons réécrire le produit des factorielles précédent, en introduisant un coefficient binomial :

$$=(-1)^{p-1-i_m}\frac{(p-1)!}{C_{p-1}^{i_m}}$$

Les propositions I(4) et I(5) nous permettent de déterminer la fraction précédente:

$$\equiv(-1)^{p-1-i_m}\frac{-1}{(-1)^{i_m}}[p]=(-1)^{p}$$

D'où la formule d'interpolation de Lagrange dans le cas d'un corps fini $\mathbf{Z}/p\mathbf{Z}$. □

I.1.2 Le théorème de Fermat–Euler

Afin de démontrer ce théorème, nous montrons deux propositions: le critère d'Euler, et son corollaire; qui permettent de caractériser certains corps $\mathbf{Z}/p\mathbf{Z}$ à l'aide des résidus quadratiques modulo p.

Définition I.1 :

Soit $p\neq 2$ un nombre premier et a un entier non multiple de p.

a s'appelle *résidu quadratique modulo p* si et seulement si il existe un entier x tel que $x^2\equiv a\,[p]$.

Nous employons la notation de Legendre :

$$\left(\frac{a}{p}\right)=\begin{cases}1, \text{ lorsque } a \text{ est un résidu quadratique modulo } p\\ -1, \text{ lorsque } a \text{ est un résidu non–quadratique modulo } p\end{cases}$$

Pour chaque nombre premier p impair, cette notation de Legendre définit une application de l'ensemble des entiers premiers à p dans l'ensemble $\{-1,1\}$.

Proposition I(7) – Critère d'Euler

Soient a un entier et p un nombre premier, alors :

$$\left(\frac{a}{p}\right)\equiv a^{\frac{p-1}{2}}\,[p] \qquad (7)$$

Démonstration

Soit r une racine primitive modulo p, c'est-à-dire r^k $[p]$ avec k variant de 1 à $(p-1)$ génère toutes les congruences possibles non nulles modulo p. Donc $\left(\frac{r}{p}\right) = -1$.

Si $a \equiv r^t\ [p]$ alors $\left(\frac{a}{p}\right) = \left(\frac{r^t}{p}\right)$. Comme r n'est pas multiple de p, et comme nous avons $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$, nous obtenons $\left(\frac{r^t}{p}\right) = \left(\frac{r}{p}\right)^t$, d'où $\left(\frac{a}{p}\right) = (-1)^t$

Suite au petit théorème de Fermat, comme r est non nul : $r^{p-1} \equiv 1\ [p]$. Étant donné que r est une racine primitive modulo p, nous avons $r^{\frac{p-1}{2}} \equiv -1\ [p]$. Ainsi :

$$\left(\frac{a}{p}\right) \equiv \left(r^{\frac{p-1}{2}}\right)^t = \left(r^t\right)^{\frac{p-1}{2}} = a^{\frac{p-1}{2}}\ [p]$$

□

Proposition I(8) :

Soit p un nombre premier impair, alors

$$\left(\frac{-1}{p}\right) = \begin{cases} 1, \text{ lorsque } p \equiv 1\ [4] \\ -1, \text{ lorsque } p \equiv -1\ [4] \end{cases} \qquad (8)$$

Démonstration

Nous avons $\left(\frac{-1}{p}\right) \equiv (-1)^{\frac{p-1}{2}}\ [p]$, or $\left(\frac{-1}{p}\right) = \pm 1$, donc $\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}}$

Enfin, $(-1)^{\frac{p-1}{2}} = 1$ lorsque $p \equiv 1\ [4]$ et à -1 lorsque $p \equiv -1\ [4]$.

□

Proposition I(9) – Théorème de Fermat-Euler, ou théorème de Noël de Fermat :

Soit p un nombre premier, alors :

$$p = 2 \text{ ou } p = 4k+1 \Leftrightarrow \exists\ a, b \in \mathbf{IN} / p = a^2 + b^2 \qquad (9)$$

Ce théorème admet plusieurs démonstrations dont en particulier une à l'aide de la notion d'idéaux non premiers et non maximaux $\mathbf{Z}p[i]$ de $\mathbf{Z}[i]$ dans Ribenboim [39], et une autre géométrique de Minkowski, [32], [33] et [41].

Démonstration

Le cas $p=2$ est trivial : $2 = 1^2 + 1^2$.

Considérons maintenant les nombres p premiers impairs.

La réciproque :

Nous savons que :

- le carré d'un entier pair vaut : $(2n)^2 = 4n^2 \equiv 0\ [4]$
- le carré d'un entier impair vaut : $(2n+1)^2 = 4n^2 + 4n + 1 \equiv 1\ [4]$

Ainsi, toute somme de deux entiers élevés au carré modulo 4 ne prend jamais la valeur 3. Par conséquent aucune somme de deux carrés ne peut exprimer les nombres de la forme $4k+3$, en particulier les nombres premiers p de cette forme.

L'implication :

Nous obtenons la congruence $a^2 b^{-2} + 1 \equiv 0\ [p]$, soit encore $a^2 b^{-2} \equiv -1\ [p]$.

Cette dernière congruence entraîne l'existence d'une racine carrée modulo p pour -1. Or la proposition I(8), nous indique alors que p serait un nombre premier de la forme $4k+1$. Ceci exclue les nombres premiers de la forme $4k+3$.

Démontrons maintenant que si p est un nombre premier de la forme $4k+1$, alors il peut toujours s'écrire comme une somme de deux entiers élevés au carré :

La proposition I(8) indique qu'il existe u tel que $u^2 \equiv -1\ [p]$. Soit L l'ensemble de toutes les paires (x, y) de $\mathbf{Z}^2$ tel que $y \equiv u\,x\ [p]$. Ceci forme un réseau, c'est–à dire un sous groupe additif généré par $e_1=(1, u)$ et $e_2=(-u, 1)$. Ces deux vecteurs forment un parallélogramme d'aire p qui est le domaine fondamental T associé à L, c'est–à dire qui consiste en tous les éléments de la forme $\Sigma\, a_i\, e_i$ avec $0 \le a_i < 1$.

Le théorème de Minkowski énonce que pour un réseau n–dimensionnel du plan avec un domaine fondamental T et un sous–ensemble X convexe symétrique borné, si : volume(X) $> 2^n$ volume(T), alors X contient un point non nul du réseau.

Prenons X un disque centré à l'origine de rayon R. Si $\pi R^2 > 2^2 p$ alors ce disque contient un point non nul de L. Choisissons par exemple $R^2 = 3p/2$. Dans ce cas il existe un couple (a, b) tel que :

$$0 \neq a^2 + b^2 \le R^2 = \frac{3}{2} p < 2p$$

Mais modulo p, nous avons :

$$a^2 + b^2 \equiv a^2 + u^2 a^2 \equiv 0\ [p]$$

Aussi $a^2 + b^2$, étant un multiple de p strictement compris entre 0 et $2p$, doit être égal à p.

Ce qui termine la démonstration du théorème de Fermat–Euler. □

I.2 Polynômes associés aux formules logiques

Avant de fournir les lemmes sur les formules logiques du calcul propositionnel classique dans les corps finis $\mathbf{Z}/p\mathbf{Z}$, nous donnons quelques définitions sur les formules logiques. Puis nous les précisons dans le cadre des corps finis.

I.2.1 Les formules logiques

Nous reprenons les définitions de Bürckert [9] et de Fraïssé [19] sur les formules connectives, logiques, sur les valeurs de vérité[2] et enfin sur les formes prénexes et les formules closes.

Définition I.2 :

> Une *formule connective* est formée de symboles, appelés *atomes*, notés a, b, c, ... et de *connections* : $\neg$ (« non »), $\vee$ (« ou »), $\wedge$ (« et »), $\Rightarrow$ (« implique », aussi dite « si... alors... »), $\Leftrightarrow$ (« si et seulement si »). Chaque connection est une fonction des *valeurs de vérité* « vrai » et « faux ».

Par exemple, la formule connective $a \Rightarrow b$ est vraie lorsque a est faux ou lorsque b est vrai. Cependant les connecteurs $\Rightarrow$ et $\Leftrightarrow$ sont superflus puisque l'implication $a \Rightarrow b$ peut se dire aussi $(\neg a) \vee b$.

Définition I.3 :

> Une *formule logique* est formée de symboles, appelés *individus*, notés x, y, z, ...; de *prédicats* notés α, β, ...; chacun étant affecté d'un entier nommé *arité*. Ajoutons pour constituer les formules logiques, le symbole d'identité = joignant deux individus; les connections connectant entre elles des formules atomiques pour former des formules composées; enfin les deux *quanteurs* $\forall$ (« pour tout » ou « quel que soit ») et $\exists$ (« il existe ») affectés chacun d'un individu[3].

Définition I.4 :

> La *valeur de vérité* (vrai ou faux) d'une formule logique dépend d'un ensemble E appelé *base*, dans lequel les individus prennent leurs valeurs; puis des *relations* sur E, attribuées à chaque prédicat; enfin des éléments de E attribués comme valeurs aux individus *libres*, c'est–à–dire non soumis au quanteur $\forall$ ni à $\exists$.

Par exemple, soit la formule $\forall y / \alpha(x, y)$, prenons pour base E l'ensemble des entiers naturels **IN** et pour α la relation binaire $\leq$ sur ces entiers, remplaçons x par 0 : la formule est alors vraie. Si nous remplaçons x par un entier non nul la formule devient fausse. Et si nous prenons pour base $\mathbf{Z}/p\mathbf{Z}$, la formule précédente est alors toujours fausse sauf si nous remplaçons x par y.

---

(2) Ces valeurs de vérités ont été introduites en toute généralité par Tarski (1936) d'après [19].

(3) Les logiciens indicent les individus qui suivent un quantificateur.

Définition I.5 :

Une formule logique est dite mise *sous forme prénexe* si tous les quantificateurs sont placés à gauche de Ψ, et dont la portée est Ψ. Elles sont du type suivant :

$$\Xi_1 x_1, \Xi_2 x_2, \ldots \Xi_n x_n / \Psi \qquad (10)$$

où les Ξ_i désignent soit le quanteur $\forall$, soit $\exists$; et où Ψ est une formule logique non quantifiée.

Proposition I(10) :

Toute formule logique peut être mise sous une forme prénexe équivalente à l'aide de règles syntaxiques générales qui n'ont rien de spécifique à la base E des valeurs de vérités.

Démonstration

En guise de démonstration, nous proposons un algorithme de réécriture qui procède en trois étapes :

1. Renommer les variables qui sont quantifiées plusieurs fois. Par exemple :
$$\forall x, (f(x) \text{ et } (\exists x, g(x)) \Leftrightarrow \forall x, (f(x) \text{ et } (\exists y, g(y))$$
2. Déplacer toutes les négations à l'intérieur en appliquant les lois de dualité sur les quantificateurs :
$$\text{non } (\forall x, f) \Leftrightarrow \exists x, \text{non } (f)$$
$$\text{non } (\exists x, f) \Leftrightarrow \forall x, \text{non } (f)$$
3. Déplacer tous les quantificateurs à l'extérieur. Nous supposons ici que g est indépendant de x :
$$(\forall x, f) \text{ et } g \Leftrightarrow \forall x, (f \text{ et } g)$$
$$(\forall x, f) \text{ ou } g \Leftrightarrow \forall x, (f \text{ ou } g)$$
$$(\exists x, f) \text{ et } g \Leftrightarrow \exists x, (f \text{ et } g)$$
$$(\exists x, f) \text{ ou } g \Leftrightarrow \exists x, (f \text{ ou } g)$$

Nous avons ainsi un algorithme qui réalise des réécritures de formules logiques sous forme prénexe, et ce indépendemment de la base E pour les valeurs de vérités. □

Définition I.6 :

Une formule logique est dite *close* lorsque tous ses individus sont quantifiés.
La formule de l'exemple précédent, $\forall y / \alpha(x, y)$, n'est pas close car l'individu x n'est pas quantifié.

Les définitions précédentes nous permettent de définir une grammaire BNF, Backus Naür Form, qui décrit les formules logiques de forme prénexes sur la base $\mathbf{Z}/p\mathbf{Z}$. Nous rappelons brièvement la syntaxe de telles grammaires :

- les variables grammaticales sont mises entre crochets < >,
- ::= suit la définition d'une variable grammaticale,
- la barre verticale | signifie une alternative pour la définition,
- les crochets [ ] signifient une option,
- les accolades { } signifient une répétition au moins une fois,
- et les guillemets " " encadrent toute chaîne de caractères constants.

<*formule logique prénexe*> ::= [<*quantificateurs*>] <*formule sans quanteur*>

<*quantificateurs*> ::= { <*quanteur*> <*individu*> } [<*liaison*>]

<*quanteur*> ::= "∀" | "∃"

<*liaison*> ::= " /" | " tel que "

<*formule sans quanteur*> ::= <*formule composée*> | <*formule atomique*>

<*formule composée*> ::= <*formule parenthésée*> | <*formule connective*> | <*négation*>

<*formule parenthésée*> ::=" (" <*formule sans quanteur*> ") "

<*formule connective*> ::= <*formule sans quanteur*> <*connecteur*> <*formule sans quanteur*>

<*connecteur*> ::=" ∨" | " ou " | " ∧ " | " et " | " ⇒" | " ⇔ "

<*négation*> ::= <*connecteur négation*> <*formule sans quanteur*>

<*connecteur négation*> ::= " ¬ " | " non "

<*formule atomique*> ::= <*prédicat*> | <*égalité*>

<*prédicat*> ::= <*nom prédicat*> " (" <*arguments*>") "

<*arguments*> ::= <*objet*> [" , " { <*objet*> }]

<*égalité*> ::= <*objet*> " ≡ " <*objet*> [" [" <*nombre premier*> "] "]

<*objet*> ::= <*individu*> | <*polynôme*>

La variable grammaticale <*individu*> désigne des symboles nommant les variables prenant leurs valeurs dans la base associée. Et la variable <*polynôme*> désigne une fonction polynôme ou une constante sur le corps fini $\mathbf{Z}/p\mathbf{Z}$. La seule particularité des formules logiques sur un corps fini est l'égalité = qui est remplacée par une congruence ≡ précisée éventuellement par un modulo.

I.2.2 Lemmes des connecteurs logiques

Dans les lemmes ci–dessous, p désigne un nombre premier. Nous établissons les lemmes qui nous permettent de réécrire une formule de la logique classique sur $\mathbf{Z}/p\mathbf{Z}$ en une expression dans l'anneau des polynômes de $\mathbf{Z}/p\mathbf{Z}$.

Lemme du connecteur « ou »

Soient deux entiers a et b, alors pour tout entiers r, s compris entre 1 et $p-1$, nous avons :

$$(a \equiv 0\ [p])\ \textbf{ou}\ (b \equiv 0\ [p]) \Leftrightarrow a^r b^s \equiv 0\ [p] \qquad (11)$$

Démonstration

Comme **Z/*p*Z** est un corps, il n'y a pas d'autres diviseurs de zéros que zéro lui–même. D'où le premier lemme sur le connecteur logique « ou ». □

Lorsque nous combinerons les lemmes des différents connecteurs logiques, nous choisirons le plus souvent $r = s = p-1$ ce qui aura pour effet de simplifier les calculs.

Lemme de la négation « non »

Soit un entier a,

$$\textbf{non}\ (a \equiv 0\ [p]) \Leftrightarrow 1 - a^{p-1} \equiv 0\ [p] \tag{12}$$

Démonstration

La négation de $a \equiv 0\ [p]$ est vraie pour toutes les valeurs de a modulo p différentes de 0. Suite à la proposition I(3), ces $(p-1)$ valeurs sont les seules racines du polynôme :

$$\prod_{k=1}^{p-1} (a-k) \equiv a^{p-1} - 1\ [p]$$

Ce qui montre ce lemme. □

Lemme du connecteur « implique »

Soient deux entiers a et b, et p un nombre premier, alors pour tout entier k compris entre 1 et $p-1$, nous avons :

$$((a \equiv 0\ [p]) \Rightarrow (b \equiv 0\ [p])) \Leftrightarrow (1 - a^{p-1})\, b^{k} \equiv 0\ [p] \tag{13}$$

Démonstration

L'implication $a \Rightarrow b$ peut se dire aussi $(\neg a) \vee b$. L'application des lemmes précédents des connecteurs « ou », en prenant r égal à 1, et « non » permet de conclure. □

Lemme du connecteur « et »

Soient deux entiers a et b, et p un nombre premier différent de 2,

$$(a \equiv 0\ [p])\ \textbf{et}\ (b \equiv 0\ [p]) \Leftrightarrow a^{p-1} + b^{p-1} \equiv 0\ [p] \tag{14}$$

Démonstration

La proposition I(3) nous permet d'écrire :

$$1 - a^{p-1} \equiv \prod_{k=1}^{p-1} (a-k) \equiv \begin{cases} 1, \text{ lorsque } a = 0 \\ 0, \text{ sinon} \end{cases}$$

Ce qui donne l'équivalence suivante :

$$a \equiv 0\ [p] \Leftrightarrow a^{p-1} \equiv 0\ [p]$$

Or dans **Z/p**Z, avec p différent de 2, la somme de $a^{p-1} + b^{p-1}\ [p]$ ne prend que trois valeurs possibles :

- 0 si et seulement si a et b sont nuls,
- 1 si et seulement si soit a, soit b, est nul mais pas les deux à la fois,
- 2 si et seulement si a et b sont tous deux non nuls.

Seul le dernier cas n'est pas vérifié dans **Z/2Z** : nous obtenons 0 au lieu de 2.

Nous montrons ainsi le lemme pour tout nombre premier impair p. □

Second lemme du connecteur « et »

Soient deux entiers a et b, et p un nombre premier de la forme $4k+3$,

$$(a \equiv 0\,[p]) \text{ et } (b \equiv 0\,[p]) \Leftrightarrow a^2 + b^2 \equiv 0\,[p] \qquad (15)$$

Démonstration

Le théorème de Fermat–Euler nous permet de garantir qu'il n'existe pas de sommes de carrés modulo p nulles lorsque p est de la forme $4k+3$. D'où ce second lemme. □

Nous donnons enfin un lemme pour le corps **Z/2Z** pour le connecteur « et » :

Lemme du connecteur « et » dans Z/2Z

Soient deux entiers a et b,

$$(a \equiv 0\,[2]) \text{ et } (b \equiv 0\,[2]) \Leftrightarrow a + b + ab \equiv 0\,[2] \qquad (16)$$

Démonstration

La formule (16) est aussi équivalente à $a + b \equiv ab\,[2]$. Or, si nous comparons les tables d'addition et de multiplication dans **Z/2Z**, nous avons :

$a+b\,[2]$		b	
		0	1
a	0	0	1
	1	1	0

$a\,b\,[2]$		b	
		0	1
a	0	0	0
	1	0	1

Seul le couple (0,0) prend la même valeur. Ce qui montre le lemme. □

Nous résumons les lemmes précédents dans la table de réécriture des connecteurs logiques ci–dessous :

Connecteurs	**Z/pZ**	$p \equiv 1\,[4]$	**Z/2Z**
A ou B	$A^r B^s,\ rs \neq 0$	—	A B
non A	$1 - A^{p-1}$	—	$1 - A$
$A \Rightarrow B$	$(1 - A^{p-1})\,B^k,\ k \neq 0$	—	$(1 - A)\,B$
A et B	$A^{p-1} + B^{p-1}$	$A^2 + B^2$	$A + B + AB$

Lorsque $p=2$, nous retrouvons l'isomorphisme de Stone, [3] et [11], qui pour la logique classique, transforme chaque formule connective en un polynôme de $\mathbf{Z}/2\mathbf{Z}$. La seule différence tient à la correspondance entre les valeurs de vérité qui est ici : 0 pour « vrai », et 1 pour « faux ».

I.3 Théorèmes d'existence de zéros dans les corps finis

I.3.1 Cas des fonctions de $\mathbf{Z}/p\mathbf{Z}[X]$

Proposition I(17) :

Soit p un nombre premier, et f une fonction de $\mathbf{Z}/p\mathbf{Z}[X]$,

$$\exists\, x / f(x) \equiv 0\ [p] \ \Leftrightarrow \prod_{j=0}^{p-1} f(j) \equiv 0\ [p] \tag{17}$$

Démonstration

La fonction f admet un zéro dans $\mathbf{Z}/p\mathbf{Z}$ si et seulement si $f(0) \equiv 0\ [p]$ ou $f(1) \equiv 0\ [p]$ ou ... $f(p-1) \equiv 0\ [p]$. Le lemme du connecteur « ou » appliqué $p-1$ fois avec $k=1$ termine la démonstration. □

Définition I.7 :

L'équation de la proposition I(17) associée à l'existence de racines d'une fonction f de $\mathbf{Z}/p\mathbf{Z}$, est appelée *équation de contrainte* ou *contrainte*.

I.3.2 Résolution dans $\mathbf{Z}/3\mathbf{Z}[X]$

Proposition I(18) :

Soit p un nombre premier, et trois entiers a, b, c de $\mathbf{Z}/3\mathbf{Z}$.

L'équation polynômiale de $\mathbf{Z}/3\mathbf{Z}$, $a x^2 + b x + c \equiv 0\ [3]$, est équivalente aux deux équations suivantes :

$$\begin{cases} c\,(a+b+c)\,(a-b+c) \equiv 0\ [3] \\ \\ x \equiv \Phi\,(1-a^2)(1-b^2)(1-c^2) - bc\,(1-a^2) + a\left(b \pm \sqrt{b^2-ac}\right)\ [3] \end{cases} \tag{18}$$

Démonstration

La proposition I(17) donne directement l'équation de contrainte pour les fonctions de $\mathbf{Z}/3\mathbf{Z}$. Cela correspond à la première équation.

Nous recherchons maintenant les zéros selon les différentes situations possibles :

1. Lorsque les coefficients a, b, c sont tous congrus à 0 modulo 3, nous avons x quelconque. Nous posons alors $x \equiv \Phi$, où Φ désigne une valeur quelconque de $\mathbf{Z}/3\mathbf{Z}$. Nous appelerons Φ paramètre libre(4).
2. Lorsque a est congru à 0 modulo 3 et b non congru à 0 modulko 3, nous avons $x \equiv -bc$ [3]
3. Et enfin, lorsque le coefficient a est non congru à 0 modulo 3, nous sommes dans le cas d'une résolution d'une équation quadratique. La solution est similaire à celle du corps des réels **IR** :

$$x = \frac{-b \pm \sqrt{b^2 - ac}}{2a} \equiv a\left(b \pm \sqrt{b^2 - ac}\right) \text{ [3]}$$

Nous remarquons que la condition sur le signe du discriminant, $\Delta = b^2 - ac$, est contenue dans la condition d'existence de racines x. En effet, si c est congru à 0, la condition est toujours vraie. Sinon, nous pouvons diviser par c et nous obtenons la contrainte : $(a + b + c)(a - b + c) \equiv 0$ [3] qui s'écrit aussi : $a^2 - b^2 - ac + 1$. Nous avons alors $\Delta = a^2 + ac + 1$. La table des valeurs ci–dessous montre que le discriminant Δ n'est jamais congru à -1 dans ce cas.

Δ [3]		a		
		–1	0	1
c	1	1	1	0
	–1	0	1	1

Les cas précédents s'écrivent respectivement sous forme logique comme suit :

1. $(a \equiv b \equiv c \equiv 0) \Rightarrow x \equiv \Phi$
2. $(a \equiv 0)$ et non $(b \equiv 0) \Rightarrow x \equiv -bc$
3. non $(a \equiv 0) \Rightarrow x \equiv a\left(b \pm \sqrt{b^2 - ac}\right)$

Par application des lemmes de la section I.2.2 sur ces formules connectives, en prenant $k = 2$ pour le lemme de la connection « ou », nous obtenons :

1. $x(1 - a^2)(1 - b^2)(1 - c^2) \equiv \Phi(1 - a^2)(1 - b^2)(1 - c^2)$
2. $x(1 - a^2)b^2 \equiv -bc(1 - a^2)$
3. $x a^2 \equiv a\left(b \pm \sqrt{b^2 - ac}\right)$

Ces trois équations correspondent à trois cas disjoints, aussi pouvons–nous les additionner. Le terme en facteur de x, après développement, vaut $1 + c^2(a^2 + b^2)^2$, ce qui ne s'annule jamais modulo 3, d'où la proposition. □

(4) Les paramètres libres sont aussi appelés « variables fantômes ».

I.3.3 Résolution "quadratique" dans $\mathbf{Z/3Z}[X]$

Nous serons amené à résoudre des équations du second degré non pas en x mais en x^2. Pour cela, nous avons la proposition suivante :

Proposition I(19) :

Dans **Z/3Z** la résolution en x^2 de l'équation $a x^2 + b x + c \equiv 0[3]$ est équivalente aux deux équations suivantes :

$$\begin{cases} ab^2(c-a) + ac + c^2 \equiv 0\ [3] \\ x^2 \equiv \Phi^2 (1-a^2)(1-b^2)(1-c^2) - ac\ [3] \end{cases} \tag{19}$$

Démonstration

Une solution en x^2 existe si et seulement si l'équation $a x^2 + b x + c$ prend l'une de ces formes :

$$\begin{cases} a x^2 + c \equiv 0 \text{ , avec le coefficient } a \text{ non congru à 0;} \\ b x \equiv 0 \text{ , avec le coefficient } b \text{ non congru à 0.} \\ \text{ou bien, } a \equiv b \equiv c \equiv 0 \end{cases}$$

En appliquant la proposition I(16) sur la première congruence, nous obtenons comme équation de contrainte : $c(a+c) \equiv 0$. Les autres cas donnent une contrainte toujours vraie, c'est-à-dire $0 \equiv 0$. Pour qu'une solution en x^2 existe, nous devons avoir :

$$\begin{cases} b \equiv 0 \text{ et } c(a+c) \equiv 0 \\ a \equiv 0 \text{ et } c \equiv 0 \\ \text{ou bien, } a \equiv b \equiv c \equiv 0 \end{cases}$$

L'application des lemmes sur les connecteurs logiques donne :

$$(b^2 + c^2 (a+c)^2)(b^2 + c^2)(a^2 + c^2) \equiv 0$$

Ce qui est équivalent après développement à la contrainte de la proposition :

$$ab^2(c-a) + ac + c^2 \equiv 0$$

Pour déterminer la solution en x^2, nous considérons chaque cas :

$$\begin{cases} a x^2 + c \equiv 0 \Leftrightarrow x^2 \equiv -ac \\ b x \equiv 0 \Leftrightarrow x^2 \equiv 0 \\ \text{ou bien, } x^2 \equiv \Phi^2 (1-a^2)(1-b^2)(1-c^2), \text{ avec } \Phi \text{ paramètre libre de } \mathbf{Z/3Z} \end{cases}$$

Ces différents cas sont disjoints, aussi pouvons–nous les ajouter, ce qui donne la solution en x^2 de la proposition.□

Chapitre II

Élimination des quantificateurs dans les corps finis

Nous abordons le problème de l'élimination des quantificateurs d'une formule logique du premier ordre sur un corps $\mathbf{Z}/p\mathbf{Z}$. Nous proposons un algorithme d'élimination qui détermine l'équation de contrainte associée à une formule logique donnée. De plus, pour les corps $\mathbf{Z}/3\mathbf{Z}$, nous montrons que l'on peut déterminer les inconnues quantifiées par le quanteur existentiel. Ceci permet non seulement de fournir une condition pour qu'une formule, dans un corps fini, soit vraie, mais aussi de fournir la ou les solutions possibles.

II.1 Introduction

Il s'agit donc de trouver un algorithme qui, étant donné une formule quantifiée, fournisse une formule équivalente sans quantificateur. Dans le cadre de la géométrie et de l'algèbre élémentaire, Tarski [42] a montré que pour toute formule il existe une formule équivalente sans quanteur $\exists$ ou $\forall$, et a proposé un algorithme d'élimination assez limité [16]. Par la suite, plusieurs méthodes ont été développées plus particulièrement la décomposition cylindrique algébrique de Collins, [16], [31], [2], [38], et ce problème n'a pas de solution satisfaisante car les algorithmes sont a fortiori de complexité doublement exponentielle d'après Davenport et Heintz [17]. De plus l'ordre d'élimination des variables quantifiées, ou leur nombre, peuvent conduire dans certains cas à un échec de la méthode de décomposition cylindrique algébrique comme le montrent Davenport [18] pour le problème de déménagement de piano, et Arnon [6] pour la recherche d'une condition pour qu'un polynôme du quatrième degré soit positif.

Il s'avère que dans le cadre des corps finis $\mathbf{Z}/p\mathbf{Z}$, nous montrons que toute formule logique du premier ordre, quantifiée, peut être réduite à une formule sans quantificateur. De plus, les algorithmes que nous proposons sont de complexité simplement exponentielle. Pour le corps $\mathbf{Z}/3\mathbf{Z}$, nous pouvons non seulement fournir une condition sur l'existence de solutions pour une formule quantifiée, mais aussi calculer l'ensemble de toutes les solutions possibles.

II.2 Élimination des quantificateurs dans les corps finis

Nous énonçons et démontrons le théorème suivant :

Théorème II(1) :

Pour toute formule logique du premier ordre Φ sur un corps fini $\mathbf{Z}/p\mathbf{Z}$, il existe une expression algébrique Ψ de $\mathbf{Z}/p\mathbf{Z}$, sans quantificateur existentiel ou universel qui lui est équivalente :

$$\Xi_1 x_1 \in \mathbf{Z}/p\mathbf{Z} \ldots \Xi_n x_n \in \mathbf{Z}/p\mathbf{Z} \; / \; \Phi(x_1, \ldots, x_n, y_1, \ldots, y_m) \Leftrightarrow \Psi(y_1, \ldots, y_m) \equiv 0 \; [p] \tag{1}$$

où Ξ_j désignent soit le quanteur $\forall$, soit $\exists$; les variables x_j sont les variables quantifiées; et les variables y_k sont des paramètres.

Démonstration

Nous avons vu au chapitre I que toute formule de $\mathbf{Z}/p\mathbf{Z}$ peut être mise sous une forme prénexe équivalente à l'aide de règles syntaxiques générales, qui n'ont rien de spécifique à l'arithmétique des corps finis.

Une méthode incrémentale consiste à éliminer les quantificateurs un par un . Il suffit d'éliminer le quantificateur de la formule $\Xi x_n \; / \; \Phi(x_1, x_2, \ldots, x_n)$ pour obtenir une formule équivalente Φ_{n-1} sans quantificateurs et portant sur x_1, x_2, ... , x_{n-1}, puis de recommencer l'élimination sur les variables restantes. On obtient finalement une formule Φ_0 sans quantificateurs.

Reste à éliminer le quantificateur Ξ qui est soit existentiel, soit universel, de la formule suivante : « $\Xi x / \Phi(x)$ » où Φ est mis sous forme disjonctive ou conjonctive l'aide des connecteurs logiques. Or, nous avons vu, au moyen des lemmes établis sur les connecteurs logiques, que toute proposition portant sur des égalités du corps $\mathbf{Z}/p\mathbf{Z}$ est équivalente à une expression de $\mathbf{Z}/p\mathbf{Z}$.

Ainsi le problème initial revient à traiter l'une de ces éliminations :

- Soit éliminer le quantificateur existentiel de « $\exists x_k / \Phi_k(x_k) \equiv 0 \; [p]$ » où Φ_k peut comporter des paramètres.

Nous avons établi une proposition sur l'existence de zéros dans les corps finis, la proposition I(17), qui nous permet d'écrire l'équivalence suivante :

$$\exists x_k / \Phi_k(x_k) \equiv 0\,[p] \;\Leftrightarrow\; \prod_{j=0}^{p-1} \Phi_k(j) \equiv 0\,[p]$$

Si nous ne nous étions pas limités aux logiques du premier ordre, cette dernière équivalence serait fausse. En effet, si Φ_k admet un paramètre y, une fonction polynomiale, le produit précédent pourrait s'annuler sans que x_k existe. Ceci du fait que l'anneau des fonctions polynomiales n'est pas intègre.

- Soit éliminer le quantificateur universel de « $\forall x_k / \Phi_k(x_k) \equiv 0\,[p]$ » où Φ_k peut comporter des paramètres.

  Nous pouvons, sur un corps fini, réécrire l'expression précédente à l'aide de la double négation comme suit :

  $$\forall x_k / \Phi_k(x_k) \equiv 0\,[p] \;\Leftrightarrow\; \text{non}\,(\,\exists x_k / \text{non}\,(\,\Phi_k(x_k) \equiv 0\,[p]\,)\,)$$

  Soit encore en appliquant le lemme de la négation :

  $$\Leftrightarrow\; \text{non}\,(\exists x_k / 1 - \Phi_k^{p-1}(x_k) \equiv 0\,[p]\,)$$

  L'élimination du quantificateur existentiel, vue auparavant, nous donne :

  $$\Leftrightarrow\; \text{non} \left(\prod_{j=0}^{p-1} \left(1 - \Phi_k^{p-1}(x_k) \right) \equiv 0\,[p] \right)$$

  En appliquant de nouveau le lemme de la négation, nous obtenons :

  $$\Leftrightarrow\; 1 - \left(\prod_{j=0}^{p-1} \left(1 - \Phi_k^{p-1}(x_k) \right) \right)^{p-1} \equiv 0\,[p]$$

  En remarquant que $\left(1 - a^{p-1} \right)^{p-1} \equiv 1 - a^{p-1}\,[p]$, nous avons finalement :

  $$\Leftrightarrow\; 1 - \prod_{j=0}^{p-1} \left(1 - \Phi_k^{p-1}(x_k) \right) \equiv 0\,[p]$$

Chacune de ces élimations de quantificateurs nous donne une expression algébrique qui définit un nouveau Φ_{k-1}. De proche en proche nous pouvons ainsi éliminer toutes les variables quantifiées, ce qui termine la démonstration. □

Ce théorème ne mentionne pas une correspondance biunivoque entre une formule logique sur $\mathbf{Z}/p\mathbf{Z}$ et une expression algébrique de $\mathbf{Z}/p\mathbf{Z}$. Nous avons vu au chapitre I que, lorsque $p = 2$, nous avons l'isomorphisme de Stone. Dans ce cas, nous avons une et une seule expression algébrique correspondant à une formule logique sur

$\mathbf{Z}/2\mathbf{Z}$. Cependant, pour les autres nombres premiers p, nous n'avons plus isomorphisme entre l'ensemble des formules logiques sur $\mathbf{Z}/p\mathbf{Z}$ et l'ensemble des expressions algébriques de $\mathbf{Z}/p\mathbf{Z}$.

Ceci vient du fait que les lemmes de réécriture d'une formule logique en une expression algébrique, construisent un polynôme parmi une famille de polynômes qui admettent tous les mêmes zéros. Nous pouvons même préciser que ce sont les lemmes des connecteurs logiques « ou » et « et » qui génèrent une telle famille de polynômes. En effet, une formule connective sur $\mathbf{Z}/p\mathbf{Z}$ comme : (P et Q et R), peut admettre trois expressions algébriques différentes de la forme : $(P^{p-1}+Q^{p-1})^{p-1}+R^{p-1}$, les deux autres expressions s'obtenant par permutation circulaire de P, Q et R. Le lemme du connecteur logique « ou » peut admettre, du simple fait du choix de la puissance, $(p-1)^2$ expressions différentes : $P^k Q^r$, avec k et r non nuls.

Cependant, bien que nous n'ayons plus l'isomorphisme de Stone pour p premier différent de 2, nous pouvons représenter toute formule construite avec polynômes et des congruences modulo p par un polynôme ne prenant sur $\mathbf{Z}/p\mathbf{Z}$ que les valeurs 0 ou 1. Ainsi par exemple :

a) $A \equiv 0\,[p] \Leftrightarrow A^{p-1} \equiv 0\,[p]$

b) **non** $(A \equiv 0\,[p]) \Leftrightarrow 1 - A^{p-1} \equiv 0\,[p]$

c) $(A \equiv 0\,[p])$ **ou** $(B \equiv 0\,[p]) \Leftrightarrow (AB)^{p-1} \equiv 0\,[p]$

d) $(A \equiv 0\,[p])$ **et** $(B \equiv 0\,[p]) \Leftrightarrow A^{p-1} + B^{p-1} - (AB)^{p-1} \equiv 0\,[p]$

Cette dernière équivalence est simplement la traduction de l'égalité dans les ensembles :

$$\text{Card}(A \cap B) = \text{Card}(A \cup B) - \text{Card}(A) - \text{Card}(B)$$

Égalité vraie aussi dans les algèbres de Boole puisque A^{p-1} ne prend que la valeur 0 ou 1. Dans ce cas nous retrouvons la formule classique établie en I(16).

II.3 Algorithmes d'élimination des quanteurs dans les corps finis

II.3.1 Algorithme dans le cas général

La démonstration du théorème II(1), nous fournit un premier algorithme d'élimination des quantificateurs. Pour qu'un tel algorithme de calcul symbolique puisse manipuler des objets définis dans les corps finis $\mathbf{Z}/p\mathbf{Z}$, il suffit de définir pour le système de calcul formel les règles suivantes :

1. Soit une règle récursive :

   Pour tout x, remplacer x^p par x,

   soit une règle non récursive :

Pour tout x remplacer x^n par $\begin{cases} x^{p-1} \text{ si } n \equiv 0\ [p-1] \\ x^{n \text{ modulo } (p-1)}, \text{ sinon} \end{cases}$

2. Pour tout entier n, remplacer n par n modulo p

La première règle est une traduction du petit théorème de Fermat. La seconde règle ne s'applique pas aux entiers présents dans les puissances.

Nous supposerons que l'algorithme accepte en entrée une formule logique Φ déja mise sous forme prénexe et un ensemble de variables quantifiées soit par $\exists$, soit par $\forall$, à éliminer. L'algorithme retourne une équation de contrainte Ψ.

Algorithme d'élimination des quantificateurs dans Z/pZ :

Entrées : Φ, variables

Sorties : Ψ

$\Psi \leftarrow$ *Polynôme associé à*: Φ

Tant que (variables $\neq \varnothing$) **faire**
x $\leftarrow$ *PrendreUneVariableDans* : variables
Enlever : x *des* : variables

$\prod \leftarrow 1$
Pour j **allant de** 0 **à** p–1
faire $\prod \leftarrow \prod *$ (*Substituer* : x *par* : j *dans* :Ψ)

Si x quantifié par $\forall$
alors $\Psi \leftarrow 1 - \prod^{**}(p-1)$
sinon $\Psi \leftarrow \prod$

fin Tant que

Retourner Ψ

Il est clair que cet algorithme se termine, puisqu'à chaque itération, le nombre des variables diminue d'une variable et s'arrête lorsqu'il n'y en a plus. Bien qu'un tel algorithme soit aisé à mettre en œuvre à l'aide d'un système de calcul formel, il est coûteux en pratique car la complexité est exponentielle :

- en manipulations symboliques, et
- en mémoire.

Nous énonçons puis montrons les propositions suivantes relative à la complexité de l'algorithme précédent :

Proposition II(2) :

L'algorithme précédent d'élimination des quantificateurs existentiels et universels dans les corps finis $\mathbf{Z}/p\mathbf{Z}$ nécessite :

$$\begin{cases} O(p^{2(m+n)}\operatorname{Log} p) & \text{opérations symboliques,} \\ O(p^{m+n}) & \text{mémorisation de monômes en } m+n \text{ variables.} \end{cases} \tag{2}$$

où m est le nombre des paramètres, et n le nombre de variables quantifiées.

Démonstration

Nous reprenons les notations de la démonstration du théorème II(1) :

Soit la formule logique Φ, mise sous forme prénexe et uniquement avec les quanteurs universels $\forall$, ce qui correspond à un calcul plus coûteux que pour les quanteurs existentiels.

Soient les n variables x_j quantifiées par $\forall$, et m variables y_k non quantifiées.

La réécriture de Φ à l'aide des lemmes des connecteurs logiques nous donne une expression polynomiale Ψ en $m+n$ variables. Le degré maximum en chacune des variables est $p-1$, cela donne donc pour Ψ un développement en au plus p^{m+n} monômes.

Nous notons par k le nombre de variables restante à éliminer lors d'une étape de l'algorithme. L'élimination d'un quantificateur de Φ au moyen de la dernière expression établie dans la démonstration de II(1) conduit à calculer p polynômes en $m+k-1$ variables, puis à effectuer leur produit pour déterminer le polynôme Π de l'algorithme, et enfin à élever à la puissance $p-1$. Au lieu d'effectuer p produits, on peut les réduire à $\log_2 p$ en employant une méthode binaire pour la multiplication ainsi que pour l'élévation à la puissance $p-1$.

Un produit de deux polynômes de p^{m+k-1} monômes donne $p^{2(m+k-1)}$ monômes. Après réduction des degrés par la règle associée au petit théorème de Fermat, nous nous retrouvons de nouveau avec au plus p^{m+k-1} monômes. La multiplication a donc pour complexité $O(p^{2(m+k)}\operatorname{Log} p)$. Ceci nous renseigne aussi sur la taille mémoire nécessaire aux calculs qui est en $O(p^{m+k})$ juste avant d'effectuer les produits. Puis, le calcul de la puissance nécessite les mêmes opérations, ce qui nous donne la même complexité. Ces complexités s'additionnent pour finalement obtenir, lors d'une élimination de variable, une complexité en $O(p^{2(m+k)}\operatorname{Log} p)$.

On effectue n éliminations, ce qui revient à parcourir k de n à 1. Le cumul des complexités de chacune des étapes donne finalement une complexité sur le nombre des opérations symboliques en $O(p^{2(m+n)} \operatorname{Log} p)$, et une place mémoire initiale en $O(p^{m+n})$ qui va en diminuant jusqu'à $O(p^{m})$.
Ce qui montre la proposition. □

L'élimination de quantificateur qui nous intéresse plus particulièrement est celle des quanteurs existentiels, que nous illustrons par les exemples qui suivent.

II.3.2 Exemples dans Z/17Z et Z/19Z

Le théorème de Fermat dans Z/pZ

Considérons par exemple le théorème de Fermat dans les corps **Z/*p*Z**. Formulé à l'aide des connecteurs et quantificateurs logiques sur **Z/*p*Z**, nous avons :

$$\exists x \exists y \exists z / (x^n + y^n \equiv z^n\ [p]) \text{ et non } (x\,y\,z \equiv 0\ [p]) \text{ et non } (n\ [p] \in \{0, 1\})$$

La première étape de notre algorithme d'élimination est de réécrire cette formule en une congruence modulo p. Pour $p = 17$, nous obtiendrons alors le développement de:

$$\exists x \exists y \exists z / \{ (x^n + y^n - z^n)^{16} + (1 - (x\,y\,z)^{16})^{16} \}^{16} + \{ (1 - (n(n-1))^{16} \}^{16} \equiv 0\ [17]$$

Et pour $p = 19$, le développement de :

$$\exists x \exists y \exists z / \{ (x^n + y^n - z^n)^2 + (1 - (x\,y\,z)^{18})^2 \}^2 + \{ (1 - (n(n-1))^{18} \}^2 \equiv 0\ [19]$$

Puis, en supposant que P représente l'un des deux polynômes précédent, l'algorithme va calculer le produit suivant :

$$\prod_{x=0}^{p-1} \prod_{y=0}^{p-1} \prod_{z=0}^{p-1} P(x, y, z, n)\ [p]$$

Ce qui nous donne une équation de contrainte fonction de n qui :

- pour $p = 17$, est vérifiée pour :

  $$n \in \{ 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15 \}$$

- et, pour $p = 19$, est vérifiée pour :

  $$n \in \{ 2, 3, 4, 5, 7, 8, 10, 11, 13, 14, 15, 16, 17 \}$$

Si nous avions donné une valeur particulière de n, par exemple $n = 6$ ou 12, l'équation de contrainte obtenue serait alors, dans les deux cas, toujours fausse puisque nous aurions alors : $c \equiv 0\ [p]$, avec c une constante non nulle et $p = 17$ ou 19.

Intersections de courbes discrètes

Nous recherchons la condition pour que deux courbes discrètes s'intersectent. Par exemple, soient un "cercle discret" d'équation « $x^2+y^2 \equiv R^2\ [p]$ » et la courbe discrète « $x^5+y^5 \equiv S^5\ [p]$ », qui admet des solutions à la fois pour $p = 17$ et 19 d'après l'exemple précédent. Nous avons la formulation logique suivante :

$$\exists x \exists y / (x^2+y^2 \equiv R^2\ [p]) \text{ et } (x^5+y^5 \equiv S^5\ [p])$$

L'algorithme, en appliquant les lemmes sur le connecteur logique « et », va réécrire cette formule logique. Pour **Z**/17**Z**, avant développement des puissances, nous avons :

$$\exists x \exists y / (x^2+y^2-R^2)^{16}+(x^5+y^5-S^5)^{16} \equiv 0\ [17]$$

L'algorithme nous fournit alors la contrainte[1] suivante :

$$R^2 S^2 \left(9R^{14}S^{14}+15R^{12}+11R^{10}S^2+3R^8S^4+13R^6S^6+6R^4S^8+10R^2S^{10}+S^{12}\right) = 0\ [17]$$

Puis pour **Z**/19**Z**, avant développement des puissances, nous avons :

$$\exists x \exists y / (x^2+y^2-R^2)^2+(x^5+y^5-S^5)^2 \equiv 0\ [19]$$

L'algorithme nous fournit alors la contrainte[1] :

$$\begin{aligned}&17R^{18}S^{16}+6R^{18}S^{10}+6R^{18}S^4+17R^{16}S^{18}+6R^{16}S^6+11R^{16}+5R^{14}S^{14}+5R^{14}S^8\\&+2R^{14}S^2+18R^{12}S^{16}+12R^{12}S^{10}+5R^{12}S^4+18R^{10}S^{18}+R^{10}S^{12}+12R^{10}S^6+11R^{10}\\&+17R^8S^{14}+16R^8S^8+18R^8S^2+3R^6S^{16}+15R^6S^{10}+14R^6S^4+4R^4S^{18}+17R^4S^{12}\\&+9R^4S^6+8R^4+R^2S^{14}+15R^2S^8+16R^2S^2+S^{16}+17S^4 = 0\ [19]\end{aligned}$$

Nous avons par exemple les conditions triviales pour **Z**/17**Z** : $R = 0$ ou $S = 0$, tandis que dans **Z**/19**Z**, nous obtenons : $R = 0$ et $S = 0$. Comme précédemment, nous pouvons rechercher les racines en R et S de ces contraintes.

Cependant, lorsque nous énumérons les solutions en (R,S), puis fabriquons par réduction de la formule logique en polynôme, nous obtenons :

$$\begin{aligned}&R^2S^{18}+7R^4S^{16}+17R^{18}S^4+2R^{10}S^{10}+4R^{12}S^8+8R^{14}S^6+10R^{16}S^4+15R^{18}S^2\\&+2S^2+10R^2 = 0\ [19]\end{aligned}$$

Ce polynôme comporte 10 monômes au lieu des 31 du polynôme obtenu par notre algorithme d'élimination. En revanche, le polynôme résultant d'une solution mathématique rigoureuse est :

$$S^8+10R^2S^6+18R^6+9R^8 \equiv 0\ [19]$$

Ces trois polynômes admettent le même ensemble de zéros, 145 zéros, et ils permettent d'illustrer que l'algorithme d'élimination nous retourne un polynôme parmi la famille des polynômes ayant un même ensemble de zéros. Ce polynôme

---

(1) Ces résultats ont été directement insérés dans ce document Grif au moyen de l'interface Grif/Reduce, [21], exposée dans l'annexe B, après avoir exécuté une session Reduce pour l'élimination des variables.

n'est pas en général un polynôme mis sous une forme réduite, soit par le nombre de mônomes, soit par le degré total ou en chacune des variables.

II.3.3 Exemple détaillé dans Z/3Z

Considérons par exemple le problème suivant : sous quelles conditions un "cercle discret" centré à l'origine de rayon unité, et une "droite discrète" du plan $\mathbf{Z}/3\mathbf{Z}^2$ s'intersectent–ils ? La formulation logique de ce problème est de rechercher tous les couples (x,y) tels qu'ils appartiennent à la fois au cercle discret et à une droite discrète, soit encore :

$$\exists x \exists y / (x^2 + y^2 \equiv 1 \ [3]) \text{ et } (y \equiv a x + b [3])$$

L'application des lemmes sur les connecteurs logiques nous permettent d'obtenir l'équivalence suivante :

$$\exists x \exists y / (x^2 + y^2 - 1)^2 + (y - a x - b)^2 \equiv 0 \ [3]$$

L'algorithme d'élimination des quantificateurs consiste alors à déterminer l'équation de contrainte suivante :

$$\prod_{x=0}^{2} \prod_{y=0}^{2} ((x^2 + y^2 - 1)^2 + (y - a x - b)^2) \equiv 0 \ [3]$$

Le développement de ce produit est :

$$(b^2+1)\,(b^2+b+1)\,(b^2-b+1)\,(a^2-ab+b^2)\,(a^2-ab+b^2+a+b-1)\,(a^2-ab+b^2-a-b-1)$$

$$(a^2+ab+b^2)\,(a^2+ab+b^2-a+b-1)\,(a^2+ab+b^2+a-b-1) \equiv 0 \ [3]$$

Or une étude attentive de ces facteurs nous permet de repérer des facteurs qui ne s'annulent jamais. Ces facteurs, que nous appelons "facteurs parasites", sont :

$$(b^2+1)\ (a^2-ab+b^2+a+b-1)\,(a^2+ab+b^2-a+b-1)$$

De plus, les autres facteurs ont des racines qui sont contenues dans un autre facteur, ou bien admettent des racines multiples. Ils forment une autre classe de facteurs parasites. Ainsi, avons–nous :

- $(a^2-ab+b^2-a-b-1)$ qui a ses racines contenues dans (a^2-ab+b^2)
- $(a^2+ab+b^2+a-b-1)$ qui a ses racines contenues dans (a^2+ab+b^2)
- (a^2-ab+b^2) qui admet des racines doubles car c'est aussi $(a+b)^2$
- (a^2+ab+b^2) qui admet des racines doubles car c'est aussi $(a-b)^2$

Finalement, la condition d'existence de solutions à notre problème d'intersection, est réduite à :

$$(b^2+b+1)\,(b^2-b+1)\,(a+b)\,(a-b) \equiv 0 \ [3]$$

Ce qui donne après développement :

$$a^2\,(b^2-1) \equiv 0 \ [3]$$

Or, malgré la présence de facteurs parasites, l'algorithme d'élimination de quantificateurs nous retourne la même équation de contrainte. La condition d'existence de solutions à notre problème est alors :

$$\exists x \exists y / (x^2 + y^2 \equiv 1\ [3]) \text{ et } (y \equiv a x + b\ [3]) \Leftrightarrow ((a \equiv 0\ [3]) \text{ ou non } (b \equiv 0\ [3]))$$

II.3.4 Extension de l'algorithme d'élimination

L'algorithme d'élimination précédent permet de fournir une formule sans quantificateur équivalente à une formule initiale. Cependant, dans le cas d'une élimination de variables quantifiées, il donne seulement une condition d'existence des solutions sans déterminer les solutions des variables quantifiées de la formule initiale.

Dans le cas de **Z/3Z** nous proposons un algorithme, qui non seulement élimine les quantificateurs, mais détermine aussi quelles sont les valeurs ou expressions que prennent les variables quantifiées grâce aux propositions I(18) et I(19). Nous modifions l'algorithme précédent de manière à inclure le calcul des solutions.

A l'algorithme précédent, nous rajoutons la variable *solutions* qui contiendra comme résultat un ensemble d'équations, le membre gauche étant la variable quantifiée et le membre droit la ou les solutions associées. Afin de préciser si nous voulons une résolution en x ou en x^2 pour une variable donnée lors d'une étape de l'algorithme, l'ensemble variables contient des variables sous les formes suivantes : x ou x^2. La procédure *Déterminer:* • *dans:* • décide alors en fonction de la forme de la variable quelle résolution effectuer. Lorsqu'il s'agit d'une résolution en x, nous introduisons l'opérateur signe $\pm$ suivi d'un numéro afin de le différencier d'un autre opérateur signe. La seule règle particulière de cet opérateur, est qu'il est remplacé par 1 lorsqu'il est élevé au carré. La procédure *Substituer:* • *par:* • *dans:* •, substitue x ou x^2 dans les équations des solutions. Le seul cas particulier à traiter porte sur les substitutions en x^2 dans une équation qui comporte un facteur en x. Nous pouvons considérer ce cas particulier soit comme une erreur dans le choix de la stratégie de résolution pour une variable donnée, soit comme un cas nécessitant le calcul de la racine puis sa substitution. Cependant le second choix, donne, éventuellement, une nouvelle équation de contrainte qui s'ajoute aux autres.

Pour déterminer les solutions des variables quantifiées, nous appliquons la méthodologie suivante: lors de la k–ième étape de l'algorithme, nous avons déterminé k équations correspondant chacune à une solution, et nous avons une équation de contrainte qui porte au plus sur $n-k$ variables quantifiées. Nous prenons une variable parmi les sur $n-k$ variables quantifiées restantes. La procédure *Déterminer:* • *dans:* • fournit alors une nouvelle solution que nous substituons à l'ensemble des k équations précédentes, puis qui s'ajoute à cet ensemble d'équations. La suite de l'algorithme calcule l'équation de contrainte comme précédemment.

Nous avons alors l'algorithme suivant, la variable x est une variable symbolique qui aura pour contenu soit un nom de variable, soit un nom de variable élevé au carré.

Extension de l'algorithme d'élimination dans le cas Z/3Z :

Entrées : Φ, variables
Sorties : Ψ, solutions

$\Psi \leftarrow$ *Traduire* : Φ
solutions $\leftarrow \varnothing$

Tant que (variables $\neq \varnothing$) **faire**
x $\leftarrow$ *PrendreUneVariableDans*: variables
Enlever: x *des*: variables

uneSolution $\leftarrow$ *Déterminer*: x *dans*: Ψ
Pour chaque équation **dans** solutions
faire équation $\leftarrow$ *Substituer*: x *par*: uneSolution *dans*: équation
solutions $\leftarrow$ *AjouterEquation*: (x = uneSolution) *dans*: solutions

$\prod \leftarrow 1$
Pour j **allant de** 0 **à** p–1
faire $\prod \leftarrow \prod *$ (*Substituer*: x *par*: j *dans*:Ψ)

$\Psi \leftarrow \prod$
fin Tant que

Retourner Ψ **et** solutions

Pour les mêmes raisons que pour le premier algorithme, cet algorithme se termine du simple fait qu'à chaque itération le nombre des variables diminue jusqu'à être nul. Nous avons la proposition suivante pour la complexité :

Proposition II(3) :

L'algorithme précédent d'élimination des quantificateurs dans le corps fini **Z/3Z** et de détermination des solutions associées aux variables quantifiées nécessite :

$$\begin{cases} O(3^{2(m+n)}\,\mathrm{Log}\,3 + n^2) & \text{opérations symboliques} \\ O(n3^m) & \text{mémorisation de monômes en } m \text{ variables.} \end{cases} \quad (3)$$

où m est le nombre des paramètres, et n le nombre de variables quantifiées.

Démonstration

A l'algorithme d'élimination précédent, nous avons ajouté de nouvelles instructions, qui sont en $O(n^2)$ opérations, et en $O(n3^m)$ pour la place mémoire, due au

calcul des n solutions qui sont fonction des m paramètres. Nous négligeons la création des paramètres libres. Ces complexités s'ajoutent à celles établies à la proposition II(2). D'où les résultats. □

II.3.5 Suite de l'exemple détaillé de Z/3Z

Résolution en x et y

Nous reprenons le problème précédent: quelles sont les intersections entre un "cercle discret" centré à l'origine de rayon unité, et une "droite discrète" du plan $\mathbf{Z}/3\mathbf{Z}^2$? Nous décrivons les diverses étapes de l'algorithme précédent qui doit nous proposer au bout du compte des solutions en x et y de :

$$\exists x \exists y / (x^2 + y^2 - 1)^2 + (y - a\,x - b)^2 \equiv 0 \ [3]$$

La première itération cherche une solution pour x. En appliquant la proposition I(18), l'algorithme détermine alors deux équations, l'équation de contrainte et une solution pour x qui sont respectivement[(1)] :

$$2y^2a^2b^2 + 2y^2a^2 + y^2 + yb + a^2b^2 + a^2 + b^2 = 0$$

$$x = \pm^{(1)}\left(y^2a^2b^2 + 2y^2a^2 + 2y^2b^2 + y^2 + 2a^2b^2 + a^2 + b^2 + 2\right) + ab\left(y^2 + 2\right)$$

L'opérateur signe plus ou moins, ±, est suivi d'un numéro, ceci afin de pouvoir repérer s'il s'agit d'un même signe ±, ou d'un autre différent pour la suite des calculs.

La seconde itération détermine une solution pour y à partir de l'équation de contrainte précédente. Il en obtient une nouvelle qui est :

$$a^2\left(2b^2 + 1\right) = 0$$

et détermine aussi une solution pour y :

$$y = \pm^{(1)}2ab^2 + b\left(a^2 + 1\right)$$

Il ne reste plus qu'à substituer cette solution y dans les autres solutions calculées précédemment, soit ici seulement x. Ce qui nous donne finalement pour x :

$$x = \pm^{(1)}\left(a^2b^2 + a^2 + b^2 + 2\right) + ab$$

Nous avons deux solutions pour (x , y). Si nous n'avions pas différencié les signes ±, nous aurions pu penser que l'algorithme d'élimination propose 4 solutions différentes. Un autre exemple, donné en II.4.2, utilise trois signes ± dont deux identiques.

Résolution en x^2 et y^2

Nous pouvons également rechercher des solutions en x^2 et en y^2 au problème précédent. La première itération cherche une solution pour x en appliquant la proposition I(19). L'algorithme obtient alors deux

équations, l'équation de contrainte et l'équation de la solution pour x qui sont respectivement[1] :

$$y^2 a^2 b^2 + y^2 a^2 + 2y^2 + 2y a^2 b + 2yb + a^2 b^2 + a^2 + 2b^2 = 0$$

$$x^2 = y^2 b^2 + y^2 - y a^2 b + 2yb - a^2 b^2 - a^2 + b^2 + 1$$

La seconde itération détermine une solution pour y à partir de l'équation de contrainte précédente. Il en obtient une nouvelle qui est :

$$a^2 + b^2 = 0$$

et détermine aussi une solution pour y :

$$y^2 = 2b^2\left(a^2 + 1\right)$$

Il ne reste plus qu'à substituer cette solution y dans les autres solutions calculées précédemment, soit ici seulement x. Ce qui nous donne finalement :

$$x^2 = 2a^2 + 2b^2 + 1$$

Nous pouvons ici réécrire ces solutions x^2 et y^2 en remplacant b^2 par $-a^2$. Dans ce cas, nous obtenons $x^2 = 1$ et $y^2 = 0$. En général ce genre de substitution n'est pas possible à partir de l'équation de contrainte.

Résolutions en x^2 et y, ou en x et y^2

Nous pouvons également effectuer des résolutions mixtes. Ainsi, la recherche de solutions en x^2 et y, ou en x et y^2 nous donne respectivement les résultats suivants[1] :

$$\left\{ \begin{array}{l} a^2\left(b^2+1\right) = 0 \\ x^2 = a^2 b^2 + 2a^2 + 2b^2 + 1 \\ y = b \end{array} \right\}$$

et

$$\left\{ \begin{array}{l} a^2 b^2 + a^2 + b^2 = 0 \\ x = \pm\left(2a^2 b^2 + a^2 + b^2 + 2\right) + 2ab \\ y^2 = b^2\left(a^2+2\right) \end{array} \right\}$$

II.4 Expérimentations

Nous avons implémenté sur le système de calcul formel Reduce les algorithmes précédents, et les expérimentations suivantes ont été réalisées sur une machine IBM RS/6000.

II.4.1 Temps de calculs expérimentaux

Dans cette section nous étudions les temps d'exécutions des algorithmes précédents dans le corps **Z**/3**Z**. Les conclusions de cette étude nous amèneront à étudier de nouveaux problèmes.

II.4.1.1 Polynômes "denses"

Nous envisageons dans un premier temps le cas où tous les coefficients sont présents, soit pour un total de n variables 3^n monômes, et ce sous plusieurs formes différentes :

- coefficients sous forme de paramètres, tous différents,
- coefficients sous la forme d'un seul paramètre,
- coefficient sous forme de constantes.

Il s'avère que toutes les expérimentations donnent les mêmes temps de calculs, quels que soient la nature des coefficients, et leur nombre. Ces expérimentations nous permettent d'étudier les temps de calculs dans des situtations extrêmes. Nous n'avons pas pu dépasser 3 variables, les temps de calculs devenant excessifs.

Nombre de variables à éliminer	Temps *cpu*	
	Algorithme de calcul de la contrainte	Algorithme de calcul des solutions
1	70 ms	360 ms
2	2 130 ms	47 680 ms
3	64 677 ms	—

Fig. 2.1 : Temps expérimentaux dans le pire des cas

Ces données, bien que nous en ayons peu, nous confirment que la complexité des algorithmes précédents est exponentielle. Pour le premier algorithme, le rapport entre deux temps successifs est de 30, et pour le second sans doute d'au moins 130.

II.4.1.2 Polynômes "creux"

Dans la pratique, nous avons rarement des manipulations de polynômes "denses". Les expérimentations suivantes ont été réalisées avec des polynômes où une très grande partie des coefficients sont absents. Par exemple, dans **Z**/3**Z**, un polynôme dense en 13 variables comporte au plus $3^{13} = 1\,594\,323$ monômes, et pour un même nombre de variables, avec un polynôme issu d'un problème particulier, nous avons seulement 5 751 monômes, ce qui représente un polynôme "creux", par analogie au calcul matriciel. Cette–fois ci le nombre maximum de variables que nous pouvons éliminer et calculer leurs solutions se situe aux alentours de 10–12 selon les cas. La figure suivante donne les temps *cpu* moyens obtenus avec les algorithmes précédents pour déterminer les solutions.

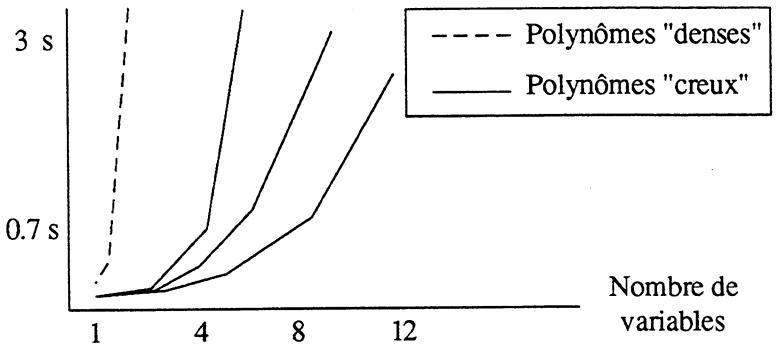


Fig. 2.2 : Temps cpu moyens de détermination des solutions dans **Z***/3***Z**

Une autre façon de mesurer la complexité de ces algorithmes, est de donner les temps intermédiaires entre chaque élimination de variable. Ainsi, pour un polynôme de degré total 6, en 5 variables et 23 monômes, nous avons eu les temps moyens intermédiaires suivants quel que soit l'ordre d'élimination des variables :

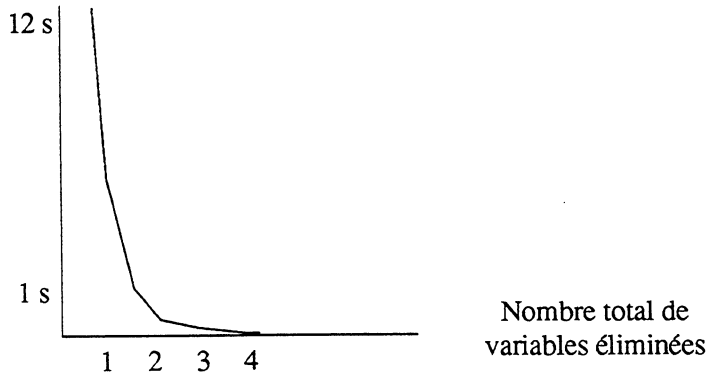


Fig. 2.3 : Temps cpu intermédiaires

Cette décroissance exponentielle des temps de calculs, s'explique par le fait que les équations de contraintes comportent de moins en moins de variables, le nombre des monômes diminue alors de façon exponentielle. Ce qu'il y a de plus de coûteux est l'élimination de la première variable. Les autres se font en moyenne aux alentours d'une dixième de seconde en temps *cpu*.

II.4.2 Influence de l'ordre

A la section précédente, nous avons mentionné un choix sur l'ordre d'élimination des variables. La question que nous nous posons est : étant donné une formule logique sur **Z***/p***Z**, obtenons–nous les mêmes solutions quel que soit l'ordre choisi pour l'élimination ?

A la suite du théorème II(1), nous avons remarqué que nous n'avons pas forcément unicité des expressions algébriques. Aussi formulons–nous l'hypothèse que nous effectuons l'élimination sur une même expression algébrique pour une formule logique donnée.

Dans ce cas, nous avons un contre–exemple avec l'exemple donné à la section II.3 :

$$\exists x \exists y / (x^2 + y^2 - 1)^2 + (y - a\,x - b)^2 \equiv 0 \text{ [3]}$$

où a et b ne sont pas quantifiés.

En appliquant l'algorithme de détermination des solutions, nous obtenons des résultats[(1)] différents selon que nous éliminons la variable x puis y :

$$\left\{ \begin{array}{l} x = \pm^{(1)} \left(a^2 b^2 + a^2 + b^2 + 2 \right) + a\,b \\ y = \pm^{(1)} 2\,a\,b^2 + b\left(a^2 + 1 \right) \end{array} \right\}$$

ou bien d'abord y puis x :

$$\left\{ \begin{array}{l} y = \pm^{(1)} 2\,a^2 b^2 + b\left(a^2 + 1 \right) \\ x = \pm^{(2)} \left(2\,a^2 b^2 + a^2 + b^2 + 2 \right) + \pm^{(1)} 2\,a\,b^2 + a\,b \end{array} \right\}$$

Le premier groupe de solution indique, par le nombres de signes ± différents, que l'intersection d'un cercle discret et d'une droite discrète admet au plus 2 solutions, et le second groupe au plus 4. Or, nous ne pouvons pas avoir plus de 2 intersections entre un "cercle discret" et une "droite discrète" de $\mathbf{Z}/3\mathbf{Z}^2$ comme on peut le constater sur la figure suivante :

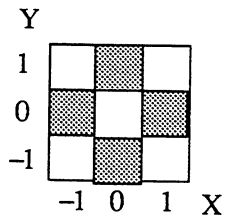


Fig. 2.4 : "Cercle discret" dans $\mathbf{Z}/3\mathbf{Z}^2$

Nous avons ainsi l'ordre d'élimination des variables qui peut influencer les résultats, et même fournir des solutions plus compliquées que d'autres.

Chapitre III

Élimination des quantificateurs dans les systèmes polynomiaux sur les corps finis

Nous nous intéressons dans ce chapitre à l'élimination de variables d'un système polynomial. Nous avons vu au chapitre précédent qu'il est extrêmement coûteux d'éliminer des variables – un coût exponentiel – d'un polynôme sur $\mathbf{Z}/p\mathbf{Z}$. Cependant, lorsque nous éliminons des variables d'un système polynomial sur $\mathbf{Z}/p\mathbf{Z}$, nous proposons par le biais du théorème III(1) qui suit, une méthodologie qui nous permet dans certaines situations, de réduire ce coût de façon non négligeable. En théorie, le coût reste exponentiel mais nous montrons dans le corollaire III(2) que ce coût est pour la plupart du temps bien inférieur à celui de l'élimination polynomiale d'un polynôme équivalent à un système polynomial.

Puis nous proposons plusieurs algorithmes à la section III.2 que nous expérimentons et appliquons sur des systèmes et sur des décompositions de systèmes à la section III.3. Nous observons des irrégularités dans les temps *cpu* de nos expérimentations que nous interprétons grâce au théorème de Fermat–Euler. Puis, en cherchant à améliorer les calculs lié à l'utilisation du lemme du connecteur logique « et », nous proposons et démontrons la proposition III(3) qui rend le calcul le moins coûteux possible.

A la section III.4 nous étudions quelques propriétés des lemmes sur les connecteurs logiques, ceci dans le but de pouvoir comparer les polynômes obtenus lors d'élimination sur des systèmes différents. Ceci nous amène à proposer de nouveaux lemmes sur les connecteurs logiques mais qui sont très coûteux. Nous illustrons ces propriétés sur une comparaison de deux systèmes polynomiaux.

Enfin, dans le but de réduire les degrés et de diminuer la complexité des calculs, nous définissons la résolution de polynômes en « x puissance d », puis ce qu'est un système

polynomial bien formé à la section III.5. Ces définitions seront reprises au dernier chapitre sur les strates.

III.1 Éliminations de variables d'un système polynomial

Nous énonçons le théorème suivant qui nous permettra par la suite de mettre en œuvre un algorithme d'élimination de variables pour des systèmes polynmiaux sur $\mathbf{Z}/p\mathbf{Z}$.

Théorème III(1) :

Soit un ensemble de r polynômes F_k sur $\mathbf{Z}/p\mathbf{Z}$ portant chacun sur $m_k + n_k$ variables prises parmi $m + n$ variables;

Quel que soit la méthode incrémentale d'élimination employée, l'élimination de n variables du système polynomial admet une complexité C encadrée comme suit :

$$\Omega\left(n p^{2(m+1)}(1+\mathrm{Log}_2 q)\right) = C(m,n,r) = O\left(p^{2(m+n)}(1+r\,\mathrm{Log}_2 q)\right) \quad (1)$$

où q compris entre 2 et $p-1$ représente la puissance employée par le lemme du connecteur logique "et".

Démonstration

Supposons que nous disposions d'un algorithme, qui à l'étape k, sélectionne une variable x parmi les v_k variables restantes, puis sélectionne les r_k polynômes comportant la variable x parmi les s_k équations du système. Une fois ces sélections effectuées, la méthode consiste à déterminer un polynôme P_k équivalent au système des r_k polynômes. Puis, une méthode d'élimination de la variable x dans le polynôme P_k pourra être appliquée. Ceci nous donnera une équation de contrainte qui s'ajoute au système privé des r_k polynômes. De cette manière, le nouveau système ne comporte plus la variable x sélectionnée, tout en étant équivalent au système précédent.

Au chapitre II, nous avons vu que le coût d'une élimination de n variables parmi v d'un polynôme P sur $\mathbf{Z}/p\mathbf{Z}$ est en $O(p^{2v})$ pour les opérations et pour la place mémoire. De même, nous avons vu que le produit de q polynômes ayant tous les monômes présents pour v variables admet un coût en $O(p^{2v}\,\mathrm{Log}_2 q)$ pour les opérations et en $O(p^{2v})$ pour la place mémoire.

Le coût de l'élimination d'une variable lors de l'étape k de tout algorithme, est fonction du calcul d'un polynôme P_k et de l'élimination d'une variable. Le calcul du polynôme nécessite l'application du lemme "et", c'est–à–dire un calcul de la forme $A^q + B^q$, de manière répétée, $r_k - 1$ fois exactement. D'où un coût pour ce calcul, en supposant que tous les monômes sont présents pour les v_k variables restantes, en $O(r_k p^{2v_k} \mathrm{Log}_2 q)$. Puis, en supposant de nouveau que tous les monômes sont présents dans le polynôme P_k pour les v_k variables, le coût de l'élimination d'une variable sera en $O(p^{2v_k})$. Soit un coût total lors de la k–ième étape en :

$$O(p^{2v_k}(r_k \mathrm{Log}_2 q + 1))$$

Le seules hypothèses que nous ayons sur r_k et v_k sont :

$$1 \le r_k \le s_k \text{, où } \begin{cases} s_1 = r \\ s_{k+1} = s_k - r_k + 1 \end{cases}$$

et

$$1 \le v_k \le m + n + 1 - k$$

Ces hypothèses sont justifiées par la méthode générale employée : lors d'un étape, nous choisissons r_k polynômes parmi s_k, puis nous réduisons ces polynômes après élimination à un seul polynôme. Au système polynomial, nous ôtons r_k équations pour ensuite en rajouter une seule, l'équation de contrainte. D'où la relation de récurrence sur les s_k. Toujours lors d'une étape, nous supposons qu'il nous restera au moins une variable à éliminer parmi v_k et au plus une variable à éliminer parmi le total des variables initiales privé des variables déjà éliminées, soit $k - 1$.
Finalement, le coût total de tout algorithme est la sommation suivante :

$$O\left(\sum_{k=1}^{n} p^{2v_k}(r_k \mathrm{Log}_2 q + 1)\right)$$

Afin d'obtenir l'encadrement proposé, il nous reste à considérer les situations extrêmes, le pire des cas et le meilleur des cas :

Le pire des cas

Dans le pire des cas, lors de chaque étape d'élimination d'une variable, nous aurons toujours r polynômes qui contiennent la variable à éliminer, et de plus ces polynômes porteront sur toutes les variables en cours, soit pour la k–ième étape, $m + n + 1 - k$ variables. Le calcul de la complexité dans cette situation nous donne :

$$\sum_{k=1}^{n} p^{2(m+n+1-k)}(r\,\mathrm{Log}_2\, q+1) \approx p^{2(m+n)}(r\,\mathrm{Log}_2\, q+1)$$

Le meilleur des cas

Dans le meilleur des cas, lors de chaque étape d'élimination d'une variable, nous aurons toujours un seul polynôme qui contient la variable à éliminer, et de plus ce polynôme portera au moins sur la variable à éliminer avec éventuellement les m autres variables, soit pour la k–ième étape, $1 \le v_k \le m+1$.

En retenant $v_k = m+1$, le calcul de la complexité dans cette situation nous donne :

$$\sum_{k=1}^{n} p^{2(m+1)}(\mathrm{Log}_2\, q+1) = n\, p^{2(m+1)}(\mathrm{Log}_2\, q+1)$$

Les deux situations extrêmes précédentes permettent d'encadrer la complexité d'un algorithme incrémental d'élimination. □

Suite à ce théorème, en faisant la comparaison avec les algorithmes d'éliminations du chapitre précédent, nous avons le corollaire suivant :

Corollaire III(2) :

Soit un entier n strictement supérieur à 1;

Soit un système (S) de r polynômes F_k sur $\mathbf{Z}/p\mathbf{Z}$ portant chacun sur $m_k + n_k$ variables prises parmi $m + n$ variables;

Soit un polynôme F sur $\mathbf{Z}/p\mathbf{Z}$ équivalent au système (S) précédent.

Une méthode d'élimination de n variables du polynôme F admet une complexité encadrée par :

$$np^{2(m+1)}(1+\mathrm{Log}_2\, q) \le p^{2(m+n)} \le p^{2(m+n)}(1+r\mathrm{Log}_2\, q) \qquad (2)$$

où q compris entre 2 et $p-1$ représente la puissance employée par le lemme du connecteur logique "et".

Démonstration

Lorsque q vaut 2, l'encadrement est immédiat, et est vérifié pour toutes les valeurs entière de n, y compris 1.

Pour les valeurs de q supérieures à 2, l'inégalité de droite est toujours vérifiée car :

$$1 + r\mathrm{Log}_2\, q > 1$$

Mais pour la borne inférieure, si n vaut 1 l'inégalité de gauche ne sera pas vérifiée car :

$$p^{2(m+1)}(1+\mathrm{Log}_2 q) > p^{2(m+1)}$$

En revanche, pour les valeurs de n supérieures à 1, nous aurons :

$$\begin{cases} 1+\mathrm{Log}_2 q \le p, \text{ car } 1 < q < p \\ np^{2(m+1)}(1+\mathrm{Log}_2 q) \le np^{2m+3} \le p^{2(m+n)} \end{cases}$$

La dernière inégalité est toujours vérifiée, ce qui termine la démonstration. □

Ce corollaire montre que la complexité d'un algorithme d'élimination incrémental de n variables d'un système polynomial sur $\mathbf{Z}/p\mathbf{Z}$ peut, dans certaines situations, être meilleur qu'une élimination d'un polynôme équivalent à un système polynomial. Lorsque q vaut 2, ce sera toujours plus intéressant de procéder à l'élimination directement sur le système. Et, lorsque q est supérieur à 2, la complexité d'élimination de variables sur un système a de fortes chances d'être meilleure que celle de l'élimination de variables d'un polynôme, car la borne supérieure est plus proche de la complexité d'élimination de variables d'un polynôme, que l'est la borne inférieure. Cependant, dans la pratique, l'estimation du coût de l'élimination, nous fournit un ordre de grandeur assez précis sur le coût en nombre d'opérations et en mémoire.

III.2 Algorithmes d'élimination pour les systèmes

Nous présentons dans la section III.2.1 un premier algorithme directement inspiré de la classe des algorithmes incrémentaux évoqués dans la démonstration du théorème III(1). Puis nous proposons à la section suivante, deux algorithmes qui affinent le choix des variables à éliminer en fonction des variables présentes dans chaque polynôme du système en cours. Ils utilisent tous deux des matrices booléennes d'incidence sur les variables d'un système polynomial, mais la différence réside en ce que le premier algorithme détermine une fois pour toutes l'ordre d'élimination des variables, et le second procède à une mise à jour à chaque étape de l'ensemble des variables restantes à éliminer.

III.2.1 Un algorithme général

Nous supposerons que l'algorithme accepte en entrée un système polynomial (S), et une liste de variables quantifiées à éliminer selon l'ordre d'apparition des variables dans cette liste. L'algorithme retourne un ensemble de solutions Ψ, si nous pouvons les déterminer, ainsi qu'un système d'équations de contraintes C. Nous reprenons les règles de calcul pour les corps finis, règles décrites en II.3.1.

Algorithme général d'élimination des quantificateurs
pour un système de polynômes sur Z/pZ :

Entrées : S, variables
Sorties : Ψ, C

Ψ ← ∅
C ← S

Tant que (variables ≠ ∅) **faire**
x ← *PrendreUneVariableDans* : variables
Enlever : x *des* : variables

S' ← *SélectionnerLesPolynômesDe* : C *comportant* : x
Enlever : S' *de* : C

P ← *AppliquerLemmeDu_ET_sur* : S'
Contrainte ← *CalculerEquationDeContraintePour*: P *selon*: x
Solution ← *Résoudre* : P *en* : x

Ajouter : (x = Solution) *à* : Ψ
Substituer : x *par* : Solution *dansSystème* : C

Si Contrainte *différentDe* : (0 = 0)
alors *Ajouter* : Contrainte *à* : C

fin Tant que

Retourner Ψ **et** C

Il est clair que cet algorithme se termine, puisqu'à chaque itération, le nombre des variables diminue d'une variable et s'arrête lorsqu'il n'y en a plus.

Le coût de cet algorithme est conditionné par l'application du lemme du connecteur "et" et par la détermination d'une équation de contrainte et d'une solution. Le lemme précédent réduit un sous–système S' de polynômes contenant une même variable en un seul polynôme de manière à pouvoir appliquer ensuite un des algorithmes d'éliminations décrits en II. Ces algorithmes retournent une équation de contrainte, et une solution lorsque p vaut 3. Nous verrons qu'il est possible de déterminer les solutions pour tout corps fini $\mathbf{Z}/p\mathbf{Z}$.

Nous pouvons d'autre part éviter le calcul des solutions des variables à éliminer en ne conservant que le polynôme P obtenu par application d'un lemme du connecteur "et".

III.2.2 Algorithmes déterminant l'ordre d'élimination

Afin de réduire le coût de l'élimination des variables, nous pouvons déterminer un ordre d'élimination qui soit le plus judicieux possible. Pour ce faire, deux variantes sont possibles:

- une première méthode est de déterminer une fois pour toutes l'ordre d'élimination,
- un second méthode consiste à déterminer au fur et à mesure des itérations, quelles sont les éliminations les moins coûteuses.

Nous supposerons pour mettre en œuvre de tels algorithmes, disposer d'une procédure qui détermine en fonction d'un ensemble de variables, et d'un système polynomial, quelle est l'ordre d'élimination le moins coûteux. Nous appellerons cette procédure "*DéterminerOrdreSelon* : S *pour*: variables". Il suffit pour cela de déterminer la matrice booléenne d'incidence des variables dans le système polynomial, puis de rechercher quelle est la variable qui nécessite le minimum d'opérations pour l'élimination. Nous pouvons pour prévoir le coût imposer deux contraintes : d'une part avoir un nombre aussi petit que possible de polynômes faisant intervenir une variable donnée, et d'autre part que cet ensemble de polynômes porte sur un nombre minimum de variables différentes. Nous ne sommes pas assurés que ces heuristiques nous donneront le meilleur choix à chaque étape, mais elles garantissent au moins que les temps de calculs ne seront pas prohibitifs à la fois en nombre d'opérations et en place mémoire.

Premier algorithme d'élimination des quantificateurs
pour un système de polynômes sur Z/pZ
avec détermination de l'ordre d'élimination

Entrées : S, variables
Sorties : Ψ, C

---

variables ← *DéterminerOrdreSelon* : S *pour* : variables

Déterminer : Ψ *et* : C *par_un_algorithme_d'élimination* : S, variables

Retourner Ψ et C

Comme il se peut que nous ayons en cours de calculs des polynômes qui comportent moins de variables que prévu, un second algorithme serait de déterminer à chaque itération l'ordre d'élimination des variables. Nous avons repris la même procédure pour déterminer l'ordre d'élimination des variables, ce qui ne serait pas très efficace puisque seul le choix de la première variable nous intéresse dans ce cas.

Second algorithme d'élimination des quantificateurs
pour un système de polynômes sur Z/pZ
avec détermination de l'ordre d'élimination

Entrées : S, variables
Sorties : Ψ, C

$\Psi \leftarrow \varnothing$
$C \leftarrow S$

Tant que (variables $\neq \varnothing$) **faire**
variables $\leftarrow$ *DéterminerOrdreSelon* : C *pour* : variables
(même corps de boucle que pour l'algorithme général)
fin Tant que

Retourner Ψ **et** C

D'un point de vue théorique, la complexité du calcul de la détermination de l'ordre est polynomial en le nombre de variables à éliminer et en le nombre de polynômes. Aussi nous ne faisons qu'augmenter les temps de calculs par rapport à l'algorithme général. Nous n'avons aucune garantie – d'un point de vue théorique – que ces algorithmes sont meilleurs que l'algorithme général puisqu'ils donneront dans le pire et le meilleur des cas les mêmes complexités algorithmiques en négligeant la complexité due aux calculs de l'ordre d'élimination des variables. D'un point de vue pratique, nous avons implémenté l'algorithme général[1] en fournissant tous les ordres d'éliminations possibles afin de déceler des variations notables des temps d'exécutions. Les variations de temps obtenues n'étaient pas significatifs et ne semblaient pas correspondre aux heuristiques précédentes, aussi avons–nous mis de côté ces variantes algorithmiques.

III.3 Expérimentations sur des systèmes équivalents

Au chapitre VI, nous aurons à décider si deux systèmes polynomiaux sur $\mathbf{Z}/p\mathbf{Z}$ sont équivalents ou non, ou de façon moins restrictive, si l'un des système peut s'en déduire de l'autre. Ici, nous faisons une étude préliminaire sur des systèmes équivalents afin de vérifier si l'introduction de nouvelles variables dans un système polynomial augmente ou non le coût de l'élimination par rapport au système initial.

Nous avons choisi deux exemples simples afin d'illustrer la mise en œuvre de l'algorithme général – implémenté sous Reduce sur une machine RS/6000 – pour résoudre des systèmes obtenus selon différentes décompositions possibles. Ces variantes nous permettent d'illustrer la proposition III(1).

---

(1) Sur une machine RS/6000 dans le système de calcul formel Reduce 3.5

Nous concluons ces expérimentations en observant que les irrégularités observées sont dues à l'application, ou non, du théorème de Fermat–Euler à travers le lemme du connecteur logique "et". Enfin, pour terminer nous énonçons en III.3.3, suite à une remarque sur des irrégularités observées sur les temps d'exécutions, une proposition qui nous permet, pour un corps $\mathbf{Z}/p\mathbf{Z}$ donné, de calculer le lemme "et" de la façon la plus efficace possible.

A travers les exemple qui suivant, nous avons toujours des temps de calculs beaucoup plus rapides que ceux obtenus au chapitre précédent, ce qui illustre bien la proposition III(2).

III.3.1 Réécritures de systèmes

Un premier exemple

Soit la formule F sous forme prénexe suivante qui représente l'intersection de courbes discrètes de $\mathbf{Z}/p\mathbf{Z}^2$:

$$\exists x \exists y / (x^2 + y^2 = \mathrm{R}^2) \text{ et } (x^5 + y^5 = \mathrm{S}^5)$$

Cette expression comporte deux quantificateurs et quatre variables en tout. La condition d'existence d'une intersection dépend des paramètres R et S.

Une première décomposition consiste à traiter les cas où R s'annule et où R ne s'annule pas. Ceci nous donne la formule suivante équivalente à la précédente :

$$(\exists x \exists y / (x^2 + y^2 = 0) \text{ et } (x^5 + y^5 = \mathrm{S}^5))$$
$$\text{ou } (\exists a \exists b / (a^2 + b^2 = 1) \text{ et } (a^5 + b^5 = c^5))$$

Nous obtenons ainsi un nouveau problème d'élimination qui porte sur quatre quantificateurs au lieu de deux. Mais comme ces quantificateurs sont disjoints, ils forment deux problèmes d'élimination qui ne concernent que trois variables chacun.

La figure suivante schématise cette décomposition.

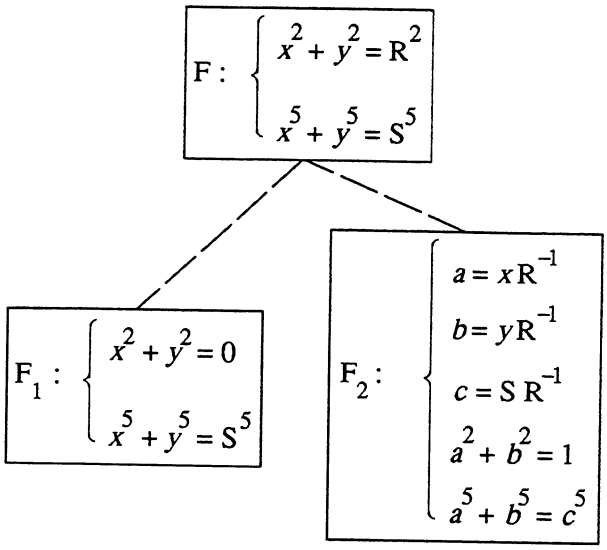


Fig. 3.1 : Décomposition de F en F_1 et F_2

Au chapitre II, nous avons établi qu'en théorie un algorithme d'élimination est en $O(p^{2n})$ opérations, où p nombre premier désigne la caractéristique du corps fini $\mathbf{Z}/p\mathbf{Z}$, et où n est le nombre total de variables intervenant dans une expression. Ainsi, en ayant trois variables pour F_1 et F_2 au lieu de quatre pour F, nous devrions avoir un gain de temps facteur de p^2.
Sur une machine IBM RS/6000, les programmes Reduce d'élimination nous donnent les temps *cpu* suivants pour quelques valeurs de p supérieures à 5.

Élimination des variables	Temps *cpu* moyen	
	pour F	pour F_1 et F_2
Z/7Z	2.3 s	0.3 s
Z/11Z	22 s	4 s
Z/13Z	5 min 45	10 s
Z/17Z	30 min 48	37 s
Z/19Z	4 min 29	62 s

Fig. 3.2 : Temps cpu selon le corps $\mathbf{Z}/p\mathbf{Z}$

En théorie, l'élimination de x et y du système F a un coût proportionnel à p^8 car nous avons en tout quatre variables dans F. En revanche, l'élimination de x et y dans le système F_1 a un coût proportionnel à p^6 de même que l'élimination de a et

b dans le système F_2. Cependant, le tableau de Fig. 3.2 semble indiquer que les écarts entre les différentes éliminations pour un même corps, sont proportionnels à p au lieu de p^2. Cela peut s'expliquer par des coûts supplémentaires dus aux applications du lemme "et" que nous avons négligés.

Si nous continuons à décomposer de la même façon en considérant les cas où S s'annule, et où S ne s'annule pas, nous obtenons de cette manière les systèmes $F_{1,1}, F_{1,2}, F_{2,1}, F_{2,2}$. comme montre la figure suivante :

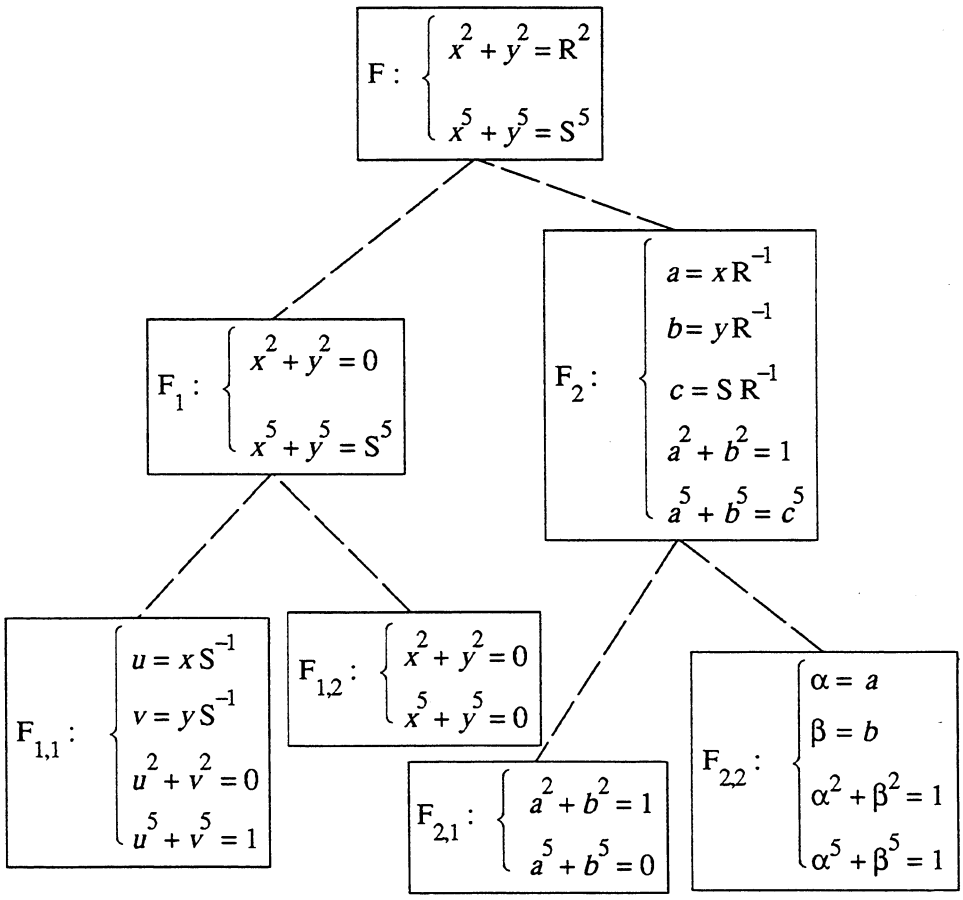


Fig. 3.3 : Décomposition de F_1 en $F_{1,1}, F_{1,2}$ puis de F_2 en $F_{2,1}, F_{2,2}$

Pour le système $F_{2,2}$ de la Fig. 3.3, nous n'avons pas développé toutes les valeurs prises par c^5 car celles–ci dépendent du corps $\mathbf{Z}/p\mathbf{Z}$. Les expérimentations qui suivent en tiennent compte.

En choisissant quelques valeurs de p supérieures à 5, nous obtenons les temps *cpu* suivants (le tiret indique un temps *cpu* que nous n'avons pas pu obtenir) :

Élimination des variables	Temps *cpu* moyen		
	F	F_1, F_2	$F_{1,1}, F_{1,2}, F_{2,1}, F_{2,2}$
Z/7Z	2.3 s	0.3 s	0.3 s
Z/11Z	22 s	4 s	0.8 s
Z/13Z	5 min 45	10 s	2.3 s
Z/17Z	30 min 48	37.5 s	4.5 s
Z/19Z	4 min 29	1 min 2	1.9 s
Z/23Z	12 min 32	—	2.9 s

Fig. 3.4 : Comparaison des temps cpu selon la décomposition

Cette fois-ci la décomposition est nettement avantageuse, et nous permet de prendre quelques valeurs de p assez élevées pour résoudre le même problème dans d'autres corps **Z**/p**Z**. Les nombres premiers qui n'apparaissent pas fournissent soit des temps *cpu* trop élevés, soit ils n'ont pas été retenus.

Z/p**Z**	29	31	37	41	43	47	53	59	61	83	107	167
Temps *cpu*	20 s	4.5 s	33 s	27 s	24 s	10 s	2 m' 1 s	15 s	1 m' 27 s	29 s	47 s	1 m' 52 s

Fig. 3.5 : Temps cpu pour F'' pour quelques grandes valeurs de p

En théorie, l'élimination de x et y du système F a un coût proportionnel à p^2 car nous n'avons jamais plus de 2 variables à éliminer. En pratique, un calcul d'interpolation en puissance sur ces données nous donnent $p^{1.48}$. La présence de "pics" dans les temps d'exécution, par exemple pour le corps **Z**/17**Z**, est expliquée à la section suivante III.3.3.

Exemple dans Z/3Z

Soit le problème de la recherche de l'existence d'intersections entre un cercle discret et une droite discrète de **Z**/3**Z**. Une première décomposition consiste à considérer les différentes droites possibles.

Ceci nous permet de distinguer quatre cas qui ne sont pas forcément disjoints.

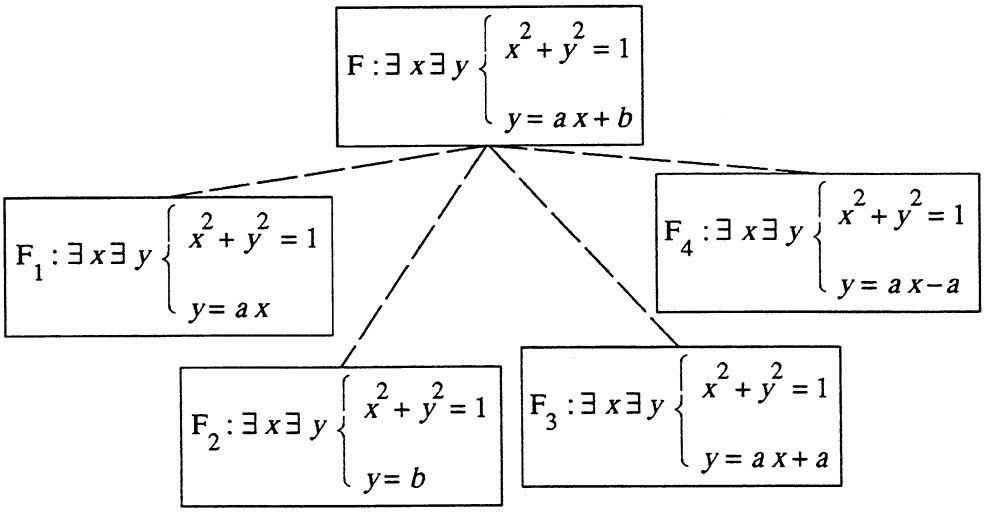


Fig. 3.6 : Décomposition de F en F_1, F_2, F_3, F_4

L'élimination de x et y de F nécessite un temps de calcul de 130 ms, tandis que la somme des temps de calculs de chaque système de la décomposition de F nous donne un temps total de 150 ms. Bien que les échelles de temps soient faibles, nous avons ici un exemple de décomposition qui ne nous donne pas de gain de temps, et qui ne se généralise pas pour d'autres valeurs supérieures de p.

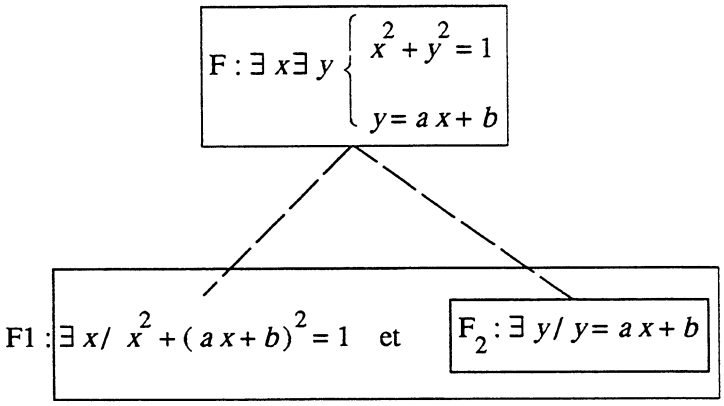


Fig. 3.7 : Décomposition de F en F_1, F_2

Par contre, si nous décomposons F en substituant y dans la première équation, nous pouvons former deux problèmes d'éliminations. Ces éliminations nous donnent un temps total de 20 ms. Cette fois–ci, bien que nous ayons de nouveau des échelles de temps très petites, nous avons un gain de temps non négligeable.

Nous pouvons vérifier que cet écart croît en prenant différentes valeurs de p.

Élimination des variables	Temps *cpu* moyen	
	F	F_1, F_2
Z/3Z	0.15 s	0.02 s
Z/5Z	0.8 s	0.04 s
Z/7Z	1.8 s	0.07 s
Z/11Z	28 s	0.19 s
Z/13Z	50 s	0.34 s

Fig. 3.8 : Temps cpu selon les corps $\mathbf{Z}/p\mathbf{Z}$

III.3.2 Introduction de nouvelles variables

Le second point qui nous intéresse plus particulièrement, est de décomposer un problème en introduisant de nouvelles variables de manière à réduire les dépendances entre variables. Ces réductions nous permettent par là–même de réduire le nombre de variables à éliminer.

Exemple

Savoir s'il existe un couple x, y satisfaisant $\begin{cases} x^2 + y^2 = R^2 \\ x^5 + y^5 = S^5 \end{cases}$ revient à éliminer deux variables parmi quatre, et a pour complexité $O(p^4)$. Cependant, nous pouvons introduire de nouvelles variables de manière à réduire cette complexité.

En effet, en faisant la différence de la première équation élevée à la puissance cinquième et de la seconde équation élevée au carré nous obtenons :

$$5R^2 x^2 y^2 (1 + x^2 y^2) = R^{10} - S^{10}$$

En introduisant les variables suivantes : $X = x^2$ et $Y = y^2$, nous avons le système suivant :

$$\begin{cases} XY = \upsilon^2 \\ X + Y = R^2 \\ 5R^2 XY(1 + XY) = R^{10} - S^{10} \end{cases}$$

où la première équation dénote l'existence d'une racine dans $\mathbf{Z}/p\mathbf{Z}$ pour le produit XY.

Finalement, en introduisant $\tau = XY$, et une équation pour la somme et le produit pour X et Y, nous obtenons :

$$\begin{cases} \tau = \upsilon^2 \\ t^2 - R^2 t + \tau = 0 \\ 5R^2\tau(1+\tau) = R^{10} - S^{10} \end{cases}$$

Le problème initial consiste donc à chercher l'existence de trois variables, ce qui est davantage par rapport au problème initial, mais les éliminations porteront seulement sur une seule variable à la fois :

$$\exists \tau / (5R^2\tau(1+\tau) = R^{10} - S^{10} \text{ et } (\exists \upsilon / \tau = \upsilon^2) \text{ et } (\exists t / t^2 - R^2 t + \tau = 0))$$

Nous aurons donc le schéma de décomposition suivant, où les problèmes d'élimination sont encadrés :

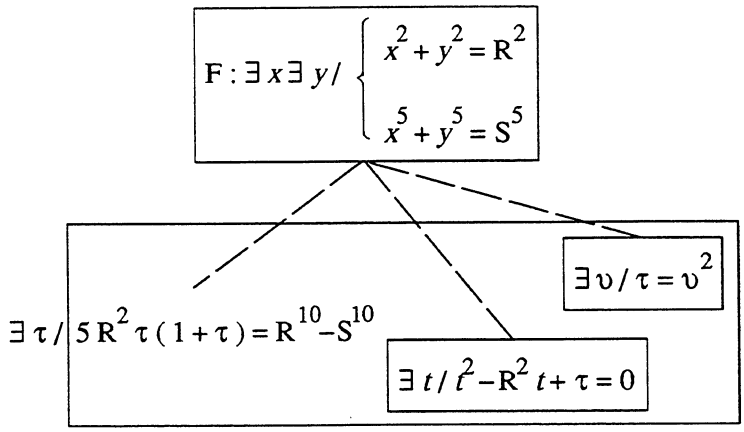


Fig. 3.9 : Une décomposition de F

L'élimination de t aura un coût en $O(p^3)$, l'élimination de υ aura un coût en $O(p^2)$, et l'élimination de τ aura un coût en $O(p^3)$.

En théorie, le gain est un facteur de p.

Élimination des variables	Temps *cpu* moyen	
	F	F'
Z/7Z	2.3 s	42 s
Z/11Z	22 s	1 min 43
Z/13Z	5 min 43	—
Z/17Z	30 min 48	—
Z/19Z	4 min 29	14 min 36

Fig. 3.10 : Temps cpu selon le corps Z/pZ

En fait, il s'avère que nous n'avons pas obtenu de gain sur les temps d'exécution en décomposant ainsi. Nous nous trouvons dans une situation où l'élimination appliquée à la décomposition du système F est plus coûteux que l'élimination des variables de F.

D'autre part, les temps de calculs demeurent extrêmement coûteux. Ce coût est dû, après l'élimination des variables t et υ, au calcul de la nouvelle expression en τ à l'aide d'un lemme du connecteur "et", puis à l'élimination de τ après avoir déterminé l'expression.

Nous pouvons continuer à décomposer en remarquant que nous avons à déterminer des racines d'une équation du second degré en τ. D'où :

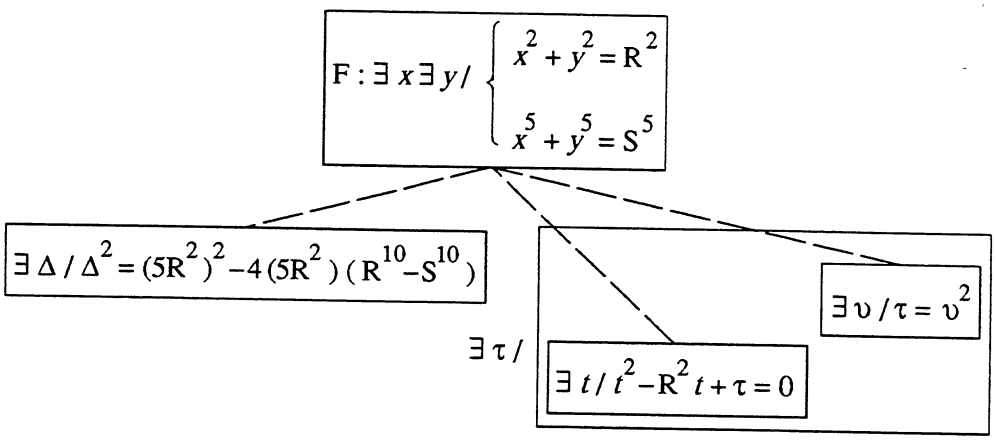


Fig. 3.11 : Une seconde décomposition F''

Cette décomposition F'' nous fournit les temps *cpu* suivants :

Élimination des variables	Temps *cpu* moyen		
	F	F'	F''
Z/7Z	2.3 s	42 s	0.2 s
Z/11Z	22 s	1 min 43	0.4 s
Z/13Z	5 min 43	—	2.9 s
Z/17Z	30 min 48	—	9.6 s
Z/19Z	4 min 29	14 min 36	0.9 s
Z/23Z	12 min 32	—	1.6 s

Fig. 3.12 : Comparaison des temps cpu pour F, F' et F''

Cette fois-ci la décomposition est nettement avantageuse, et nous permet de prendre quelques valeurs de p assez élevées pour résoudre le même problème dans des corps **Z**/p**Z**. Les nombres premiers qui n'apparaissent pas fournissent, par estimation, des temps *cpu* supérieurs à 7 min.

Z/p**Z**	29	31	37	47	59	83	107	167
Temps *cpu*	2 min 10	3.3 s	7 min 23	11 s	20 s	52 s	1 min 40	6 min 5

Fig. 3.13 : Temps cpu pour F'' pour quelques grandes valeurs de p

III.3.3 Interprétations des irrégularités

La présence de "pics" dans les temps de calculs trouvent leur cause dans le théorème de Fermat-Euler. En effet, l'algorithme d'élimination pour les système polynomiaux, applique le lemme du connecteur "et" en fonction de la caractéristique du corps fini. Ainsi, nous aurons soit le calcul de $A^2 + B^2$, soit $A^{p-1} + B^{p-1}$, selon que p est de la forme $4k+3$ ou non.

La table suivante montre quels sont les temps d'exécutions si nous appliquons ou non le théorème de Fermat–Euler pour un même problème :

Élimination des variables	Temps *cpu* moyen	
	En appliquant Fermat–Euler	Sans appliquer Fermat–Euler
Z/7Z	2.3 s	8.1 s
Z/11Z	22 s	1 min 20
Z/13Z	Ne s'applique pas	5 min 47
Z/17Z	Ne s'applique pas	½ h
Z/19Z	4 min 29	1 h
Z/23Z	12 min 32	3 h ¼

Fig. 3.14 : Écarts de temps du au théorème de Fermat–Euler

Lorsque cela est possible, il est préférable d'utiliser un corps $\mathbf{Z}/p\mathbf{Z}$ tel que p est congru à 3 modulo 4. Cependant, lorsque ce n'est pas le cas, nous pouvons déterminer quelles sont les plus petites puissances n telles que :

$$x^n + y^n \equiv 0\,[p] \Leftrightarrow x \equiv 0\,[p] \text{ et } y \equiv 0\,[p] \qquad (3)$$

Nous avons ainsi les résultats suivants :

$\mathbf{Z}/p\mathbf{Z}$	$p \equiv 5\,[8]$	17	41	73	89	97	113	137	193
n	4	16	8	8	8	32	16	8	64

Fig. 3.15 : Quelques résultats obtenus par Reduce

Ces résultats nous conduisent à énoncer la proposition suivante :

Proposition III(3) :

Soit p un nombre premier impair tel que :

$$p \equiv n+1\,[2n] \text{ , avec } n = 2^k, \text{ où } k \in \mathrm{IN}^*$$

alors, l'entier n est la plus petite puissance telle que nous ayons pour tout couple (x, y) d'entiers relatifs l'équivalence suivante :

$$x^n + y^n \equiv 0\,[p] \Leftrightarrow x \equiv 0\,[p] \text{ et } y \equiv 0\,[p]$$

Nous avons vérifié cette proposition, avec l'aide d'un programme Reduce optimisé, pour tous les nombres premiers impairs inférieurs à 977. Puis, nous avons pu fournir une démonstration dont nous proposons une version simplifiée.

Démonstration

Le couple (0,0) est une solution évidente. Si l'un des éléments du couple (x,y) est non nul alors l'autre est non nul lui aussi. Supposons qu'il existe un couple (x,y) tel que le produit xy est non congru à 0 modulo p, et qui vérifie l'équation (3). Dans ce cas notons z comme étant le produit xy^{-1}, ce qui nous permet d'écrire :

$$z^n \equiv -1\,[p]$$

Écrivons le nombre premier p impair sous la forme suivante $p = 2^k(2m+1)+1$ avec $k \geq 1$.

Nous notons que le groupe multiplicatif cyclique $\mathbf{Z}/p\mathbf{Z}^*$ est isomorphe au groupe additif $\mathbf{Z}/(p-1)\mathbf{Z}$, lui même isomorphe au produit des groupes additifs $\mathbf{Z}/2^k\mathbf{Z}$ et $\mathbf{Z}/(2m+1)\mathbf{Z}$. L'isomorphisme ϕ composé est :

$$\phi(\alpha^u) = (u \bmod 2^k,\ u \bmod (2m+1))$$

avec α générateur du groupe $\mathbf{Z}/p\mathbf{Z}^*$.

En particulier, suite à la démonstration du critère d'Euler du chapitre I, nous avons :

$$\phi(-1) = \phi\left(\alpha^{\frac{p-1}{2}}\right) = (2^{k-1}, 0)$$

Pour l'exposant $n = 2^{k'}(2m'+1)$, il existe $z \in \mathbf{Z}/p\mathbf{Z}^*$ si et seulement si $\phi(z^n) = \phi(-1)$, c'est–à–dire s'il existe un couple (a,b) de $\mathbf{Z}/2^k\mathbf{Z} \times \mathbf{Z}/(2m+1)\mathbf{Z}$ tel que :

a) $2^{k'}(2m'+1)\,a \equiv 2^{k-1}\,[2^k]$, et

b) $2^{k'}(2m'+1)\,b \equiv 0\,[2m+1]$.

Si $k' < k$, alors en notant β l'inverse de $2m'+1$ dans $\mathbf{Z}/2^k\mathbf{Z}$, le nombre $a = \beta\, 2^{k-k'-1}$ est l'unique solution de a), et b=0 est solution de b).

Si $k' \geq k$, alors $2^{k'}(2m'+1)\,a \equiv 0\,[2^k]$, et l'équation a) n'a pas de solution.

Aussi, 2^k est bien la plus petite puissance, d'où la proposition. □

Cette proposition signifie en particulier que pour les corps finis de caractéristique p un nombre de Fermat, c'est–à–dire un nombre premier de la forme $2^{2^k}+1$, le lemme "et" ne peut pas employer de puissance inférieure à $p-1$. Voici les cinq premiers nombres de Fermat : 3, 5, 17, 257 et 65537.

Si, d'autre part, nous ne pouvons pas éviter un corps, tel que $\mathbf{Z}/17\mathbf{Z}$, où le lemme "et" nous oblige à prendre une puissance assez élevée, nous pouvons choisir un corps fini de

caractéristique supérieure, tel qu'il admette un lemme "et" de coût nettement inférieur. Ainsi vaut–il mieux éviter le corps **Z**/17**Z** pour prendre **Z**/19**Z**, où nous évitons des calculs de puissance 16 pour de simples élévations au carré.

III.4 Propriétés sur des lemmes des connecteurs logiques

III.4.1 Compositions des lemmes

Lorsque nous composons plus d'une fois un même lemme sur les connecteurs logiques du chapitre I, nous observons qu'ils sont associatifs, mais pour que certains d'entre eux, les expressions algébriques ne sont pas identiquement égales. Par exemple, la double négation ne nous redonne pas l'équation initiale :

Propriété III(4) :

Soit A une expression algébrique sur le corps **Z**/*p***Z**, alors

$$\textbf{non}\ (\textbf{non}\ (\ A \equiv 0\ [p]\ )) \Leftrightarrow A^{p-1} \equiv 0\ [p] \qquad (4)$$

Démonstration

Le lemme de la négation est donné par l'équivalence suivante :

$$\textbf{non}\ (\ A \equiv 0\ [p]) \Leftrightarrow 1 - A^{p-1} \equiv 0\ [p]$$

Une seconde application de la négation nous donne :

$$\textbf{non}\ (\ \textbf{non}\ (\ A \equiv 0\ [p]\ )) \Leftrightarrow \textbf{non}\ (\ 1 - A^{p-1}\ ) \equiv 0\ [p]$$

$$\Leftrightarrow 1 - \left(\ 1 - A^{p-1}\right)^{p-1} \equiv 0\ [p]$$

En développant et en appliquant la proposition I(5), nous obtenons :

$$\sum_{k=0}^{p-1} \left((-1)^k A^{(p-1)^k} \right) - 1 \equiv 0\ [p]$$

Or pour tout k non nul : $A^{(p-1)^k} \equiv A^{p-1}\ [p]$. Ce qui nous permet d'avoir :

$$1 - 1 + A^{p-1} \equiv 0\ [p]$$

D'où la propriété. □

Propriété III(5) :

Soient A, B, C trois expressions algébriques sur le corps **Z**/*p***Z**, alors le polynôme associé à une composition de deux connecteurs logiques « ou » est symétrique lorsque nous avons les mêmes puissances car :

$$\left(\ A \equiv 0\ [p]\ \textbf{ou}\ B \equiv 0\ [p]\ \right)\ \textbf{ou}\ C \equiv 0\ [p] \Leftrightarrow (ABC)^k \equiv 0\ [p] \qquad (5)$$

avec k un entier compris entre 1 et $p-1$.

Démonstration

Nous avons vu auparavant le lemme du connecteur « ou » :

$$A \equiv 0\,[p] \textbf{ ou } B \equiv 0\,[p] \Leftrightarrow \forall\, k \in \{1, \ldots, p-1\},\ A^k B^k \equiv 0\,[p]$$

Il est clair qu'une seconde composition « ou », pour tout k non nul, nous donnera un polynôme symétrique, d'où la propriété. □

Propriété III(6) :

Soient A, B, D trois expressions algébriques sur le corps $\mathbf{Z}/p\mathbf{Z}$;

Quelque soit la puissance q employée par le lemme du connecteur logique « et »,

$$\left(A \equiv 0\,[p] \textbf{ et } B \equiv 0\,[p] \right) \Leftrightarrow A^q + B^q \equiv 0\,[p] \qquad (6)$$

le polynôme associé à une composition de deux « et » n'est pas symétrique.

Démonstration

Composons successivement le lemme du connecteur « et » sur le membre droit de l'équivalence (6) :

$$\left(A \equiv 0\,[p] \textbf{ et } B \equiv 0\,[p] \right) \textbf{ et } D \equiv 0\,[p] \Leftrightarrow \left(A^q + B^q \right)^q + D^q \equiv 0\,[p]$$

Un développement par la formule du binôme de Newton, nous donne :

$$\Leftrightarrow \sum_{k=0}^{q} C_q^k A^{qk} B^{q(q-k)} + D^q \equiv 0\,[p]$$

Pour que nous ayons égalité avec la formule logique :

$$A \equiv 0\,[p] \textbf{ et } \left(B \equiv 0\,[p] \textbf{ et } D \equiv 0\,[p] \right);$$

Soit encore :

$$A^q + \left(B^q + D^q \right)^q \equiv 0\,[p]\ ;$$

Nous devrions avoir les coefficients binomiaux nuls modulo p, c'est–à–dire, avoir q égal à p – voir démonstration de la proposition I(1) – ce qui est exclu, d'où la propriété. □

III.4.2 Lemmes des connecteurs symétriques

Pour rendre les lemmes sur le connecteur logique « et » symétrique, nous proposons une nouvelle proposition sur ce lemme. Il nous servira à comparer des polynômes issus de formulations logiques, afin de savoir s'ils représentent la même formule logique ou non, de savoir également si deux systèmes polynomiaux sont équivalents ou non.

Lemme du connecteur « et » symétrique :

Soient A, B, deux expressions algébriques sur le corps $\mathbf{Z}/p\mathbf{Z}$, alors le polynôme de (7) obtenu par des compositions successives du connecteur logique « et » est symétrique,

$$A \equiv 0\,[p] \text{ et } B \equiv 0\,[p] \Leftrightarrow \left(A^q + B^q\right)^{p-1} \equiv 0\,[p] \qquad (7)$$

où q représente une puissance comprise entre 2 et $p-1$, telle que nous ayons l'équivalence précédente toujours vérifiée.

Démonstration

Nous avons montré plusieurs propositions qui permettent de déterminer une puissance q: les lemmes du connecteur « et » du chapitre I, et la proposition III(3).

Montrons que la composition du lemme précédent fournit toujours un polynôme symétrique.

Nous rappelons que nous avons les relations suivantes :

$$\begin{cases} \alpha^{(p-1)^k} \equiv \alpha^{p-1}\,[p], \text{ avec } k \neq 0 \\ \\ \beta^0 \equiv 1\,[p], \text{ avec } \beta \text{ quelconque} \end{cases}$$

Considérons la composition suivante :

$$\left(A \equiv 0\,[p] \text{ et } B \equiv 0\,[p]\right) \text{ et } D \equiv 0\,[p] \Leftrightarrow \left(\left(\left(A^q + B^q\right)^{p-1}\right)^q + D^q\right)^{p-1} \equiv 0\,[p]$$

En simplifiant, nous avons :

$$\Leftrightarrow \left(\left(A^q + B^q\right)^{p-1} + D^q\right)^{p-1} \equiv 0\,[p]$$

Puis un développement, nous donne :

$$\Leftrightarrow \sum_{k=0}^{p-1} C_{p-1}^k \left(\left(A^q + B^q\right)^{p-1}\right)^k \left(D^q\right)^{p-1-k} \equiv 0\,[p]$$

De nouveau nous simplifions, et développons afin d'obtenir :

$$\Leftrightarrow D^{q^{p-1}} + \left(A^q + B^q\right)^{p-1} \left[\sum_{k=1}^{p-1} (-1)^k D^{q^{p-1-k}}\right] \equiv 0\,[p]$$

Le calcul de la somme précédente revient à évaluer :

$$\sum_{k=0}^{p-1} x^k = \begin{cases} \left(x^p - 1\right)(x-1)^{-1}, \text{ si } x \text{ est non congru à 1 modulo } p \\ \\ 0 \text{ sinon} \end{cases}$$

Ce qui s'écrit aussi pour tout x, comme étant :

$$\sum_{k=0}^{p-1} x^k \equiv x^{p-1} \; [p]$$

Par conséquent, nous avons pour la somme précédente :

$$D^{q^{p-1}} \sum_{k=1}^{p-1} (-1)^k D^{q^{-k}} \equiv D^{q^{p-1}} \left(\left(-D^{q^{-1}} \right)^{p-1} - 1 \right) \equiv 1 - D^{q^{p-1}} \; [p]$$

D'où :

$$\left(\left(A^q + B^q \right)^{p-1} + D^q \right)^{p-1} \equiv D^{p-1} + \left(A^q + B^q \right)^{p-1} \left(1 - D^{p-1} \right) [p]$$

En recommençant de nouveau ces calculs sur le terme $\left(A^q + B^q \right)^{p-1}$:

$$\equiv D^{p-1} + \left(B^{p-1} + A^{p-1} \left(1 - B^{p-1} \right) \right) \left(1 - D^{p-1} \right) [p]$$

Un développement nous fournit finalement un polynôme symétrique en A, B et D :

$$\equiv A^{p-1} + B^{p-1} + D^{p-1} - (AB)^{p-1} - (AD)^{p-1} - (BD)^{p-1} + (ABD)^{p-1} \; [p]$$

Ce qui démontre la proposition. □

III.4.3 Coût d'un développement multinomial

Nous avons souvent des développements multinomiaux de polynômes, aussi une question est de savoir s'il vaut mieux employer un lemme d'un connecteur logique symétrique ou non. Nous montrons dans la proposition qui suit, que l'emploi du lemme du connecteur logique « et » symétrique coûte très cher. En effet, nous avons la proposition suivante :

Proposition III(8) :

Soient m monômes u_i distincts, soit n un entier compris entre 2 et $p-1$, alors le développement multinomial sur $\mathbf{Z}/p\mathbf{Z}$ de :

$$\left(u_1 + u_2 + \ldots + u_m \right)^n \tag{8}$$

comporte au plus C_{m+n-1}^{m-1} monômes.

Démonstration

Le développement de (8) est le suivant :

$$\sum_{0 \le l \le j \le \ldots \le k \le n} C_n^k C_k^i \ldots C_i^j C_j^l u_1^{n-k} u_2^{k-i} \ldots u_m^{j-l}$$

Nous montrons sur $\mathbf{Z}/p\mathbf{Z}$ à propos de ce développement :

i qu'aucun coefficient binomial n'est congru à 0,

ii que le petit théorème de Fermat peut réduire le nombre total des monômes obtenus,

iii que le nombre total de monômes ne peut pas excéder C_{m+n-1}^{m-1}.

i Par hypothèse, n est inférieur à p un nombre premier, aussi p ne divise aucun facteur de $n!$. Lorsque k vaut zéro, C_n^0 vaut 1. Par conséquent, le coefficient binomial C_n^k n'est pas multiple de p pour toutes les valeurs prises par k.

Pour tous les autres coefficients binomiaux, nous avons le même résultat, ce qui montre que le produit $C_n^k C_k^i \ldots C_i^j C_j^l$ n'est jamais congru à zéro modulo p.

ii Le développement peut comporter un nombre total des monômes inférieur dans le cas où nous sommes en présence d'un monôme u_i qui contient un facteur de la forme x^d, d compris entre 2 et $p-1$. En effet, nous pouvons avoir la situation suivante :

$$x^{d^a} \equiv x^{d^b} [p],$$

lorsque a est différent de b et lorsque $a\ d \equiv b\ d [p-1]$, suite au petit théorème de Fermat. Ce qui aura pour effet de réduire éventuellement deux monômes en un seul.

iii Soit l'hypothèse de récurrence sur m que le développement multinomial comporte C_{m+n-1}^{m-1} monômes. Lorsque m vaut 1, nous avons un seul monôme, soit encore C_n^0 ; de même, lorsque m vaut 2 nous avons au plus $n+1$ monômes, soit encore C_{n+1}^1. Cette hypothèse est vraie pour m égal à 1 et 2. Supposons–la vrai à l'ordre m, et montrons que cela entraîne qu'elle est vraie au rang $m+1$. Notons U les m premiers termes, nous avons au rang $m+1$:

$$\left(U + u_{m+1}\right)^n = \sum_{k=0}^{n} C_n^k U^k u_{m+1}^{n-k}$$

Le terme U^k comporte C_{m+k-1}^{m-1} monômes, de même que $U^k u_{m+1}^{n-k}$. En tout, nous aurons donc au plus $\sum_{k=0}^{n} C_{m+k-1}^{m-1}$ monômes, soit aussi C_{m+n}^{m}. Donc l'hypothèse est vérifiée au rang $m+1$.

Par conséquent, pour tout m, nous aurons au plus C_{m+n-1}^{m-1} monômes.

Ce qui termine la démonstration. □

A partir de la proposition précédente, nous comparons les différents lemmes du connecteur logique « et ». Nous supposerons pour cela que A et B sont deux polynômes sur $\mathbf{Z}/p\mathbf{Z}$ ayant respectivement α et β monômes.

Le lemme du connecteur «et», lorsque composé successivement ne donne pas un polynôme symétrique, détermine le polynôme suivant :

$$A^q + B^q$$

Ceci fournit un polynôme ayant au plus $C^{\alpha-1}_{\alpha+q-1} + C^{\beta-1}_{\beta+q-1}$ monômes. Notons cette borne ω.

Le lemme du connecteur « et », lorsque composé successivement donne un polynôme symétrique, détermine le polynôme suivant :

$$\left(A^q + B^q\right)^{p-1}$$

Nous avons au plus ω monômes à développer selon la formule multinomiale, soit au plus $C^{\omega-1}_{\omega+p-2}$ monômes. Ce dernier développement comporte environ ω^{p-1} monômes de plus. Ce qui est considérable, même pour de petites valeurs de p. Aussi, est–il préférable, de façon générale, d'utiliser un lemme d'un connecteur logique, qui met en jeu un minimum de calcul de puissances, et que ces dernières soient les plus petites possibles. Or, nous avons montré que les plus petites puissances étaient données par la proposition III(3).

Le constat précédent est lié à la façon dont un système de calcul formel effectue les calculs sur $\mathbf{Z}/p\mathbf{Z}$. Nous pouvons définir les règles élémentaires de calculs comme en II.3.1, mais par la suite, toutes les expressions contenant des élévations de puissances imbriquées seront calculées les unes après les autres, alors que ces calculs peuvent ne pas être nécessaires. Par exemple pour $A^{q^{p-1}}$, il est inutile de calculer A^q avant d'élever le résultat obtenu à la puissance $p-1$. Il suffit de calculer directement A^{p-1}. Aussi, pouvons–nous définir de nouvelles règles qui évitent de tels calculs en réduisant les puissances avant de développer. De plus, les calculs de la forme $(A^q + B^r)^{p-1}$ se simplifient en un polynôme qui ne fait plus intervenir les puissances q et r.

Cependant les règles précédentes, en supposant qu'elles puissent être programmées dans un langage de calcul symbolique, ne suffisent pas, car en plus des problèmes de calculs, de ceux liés à la représentation interne des polynômes comme le montre Davenport dans [17], nous avons le problème suivant pour des polynômes définis sur $\mathbf{Z}/p\mathbf{Z}$. Soient r et s supérieurs à 1, calculons :

$$x\left(1-x^{\frac{p-1}{2}}\right)^r\left(1+x^{\frac{p-1}{2}}\right)^s$$

Si nous développons chacune de ces puissances, nous aurons au plus $r+1$ et $s+1$ monômes. Puis le produit de ces puissances donneront au plus $(r+1)(s+1)$ monômes. Et enfin, le produit par x nous donnera zéro modulo p. Si nous avions développé – au lieu de simplifier – le produit précédent comme suit :

$$\left[x\left(1-x^{\frac{p-1}{2}}\right)\left(1+x^{\frac{p-1}{2}}\right)\right]\left(1-x^{\frac{p-1}{2}}\right)^{r-1}\left(1+x^{\frac{p-1}{2}}\right)^{s-1}$$

Le produit des facteurs ne comportant pas de calculs de puissance nous évite à ce moment–là une évaluation des puissances, soit un coût bien inférieur au calcul précédent. Mais dans l'hypothèse où ce premier produit est non nul, nous aurons à déterminer $4\,rs$ monômes. Le coût d'une telle stratégie peut devenir plus ou moins cher selon les valeurs de r et s, qu'un simple développement. Aussi n'est–il pas aisé de définir quelles sont les meilleures règles de calculs qui évitent de calculer des polynômes intermédiaires trop importants du fait de développements multinomiaux.

III.4.4 Comparaison de deux systèmes polynomiaux

Soient deux systèmes polynomiaux sur $\mathbf{Z}/p\mathbf{Z}$, (S_1) et (S_2), sur lesquels nous voulons savoir s'ils sont équivalents ou non. Nous pouvons réduire chacun de ces systèmes à un polynôme équivalent, P_1 et P_2 respectivement, puis les comparer terme à terme. Or, selon le choix du lemme du connecteur logique « et » employé, nous pourrons ou non faire ce genre de comparaison. Voici un exemple qui illustre cette situation :

Exemple

Considérons les deux systèmes suivants sur $\mathbf{Z}/3\mathbf{Z}$:

$$(S_1)\ \begin{cases} x_1 = 0 \\ x_2 = 0 \\ x_3 = 0 \end{cases} \qquad (S_2)\ \begin{cases} x_1 = y_1 \\ y_1 = 0 \\ x_2 = 0 \\ x_3 = 0 \end{cases}$$

- Si nous choisissons un lemme du connecteur « et » pour lequel le polynôme associé n'est pas symétrique pour des compositions successives, nous aurons pour le système (S_1), respectivement (S_2), le polynôme P_1 équivalent, respectivement P_2 :

$$P_1(x_1, x_2, x_3) = x_1^2 + (x_2^2 + x_3^2)^2$$

$$P_2(x_1, x_2, x_3, y_1) = ((x_1^2 - y_1^2)^2 + y_1^2)^2 + (x_2^2 + x_3^2)^2$$

L'élimination de la variable y_1 du polynôme P_2 – sans application du lemme « et » dans l'algorithme d'élimination – nous fournit le polynôme suivant :

$$P(x_1, x_2, x_3) = (x_1^2 + x_2^2)^2 + x_3^2$$

Or, ce polynôme P n'est pas identiquement égal au polynôme P_1. Pourtant, ces derniers représentent deux systèmes équivalents. Leur point commun est qu'ils admettent le même ensemble de zéros du fait qu'ils ont été définis à partir d'un même système polynomial mais en appliquant les lemmes «et» dans un ordre différent.

- En revanche, si nous choisissons un lemme du connecteur « et » pour lequel le polynôme associé est symétrique pour des compositions successives, nous aurons pour le système (S_1), respectivement (S_2), le polynôme P_1 équivalent, respectivement P_2, d'après la proposition III(7) :

$$P_1(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2 - x_1^2 x_2^2 - x_1^2 x_3^2 - x_2^2 x_3^2 + x_1^2 x_2^2 x_3^2$$

$$P_2(x_1, x_2, x_3, y_1) = x_1^2 + x_2^2 + x_3^2 + y_1^2 - x_1^2 x_2^2 - x_1^2 x_3^2 - x_1^2 y_1^2 - x_2^2 x_3^2 - x_2^2 y_1^2 - x_3^2 y_1^2 + x_1^2 x_2^2 x_3^2 + x_1^2 x_2^2 y_1^2 + x_1^2 x_3^2 y_1^2 + x_2^2 x_3^2 y_1^2 - x_1^2 x_2^2 x_3^2 y_1^2$$

L'élimination de la variable y_1 du polynôme P_2 nous fournira un polynôme identiquement égal au polynôme P_1 mais après un temps de calcul plus long, 170 ms au lieu de 40 ms[(2)]. Ces temps sont très faibles, mais si ces lemmes sont employés de façon répétée, sur des polynômes comportant plusieurs centaines de monômes, nous aurons des écarts de temps bien plus élevés.

III.5 Systèmes de $\mathbf{Z}/p\mathbf{Z}$ bien formés

Nous voulons traiter les cas particuliers où si l'équation algébrique s'écrit $P(x) \equiv 0\ [p]$ alors l'équation suivante : $x^k P(x) \equiv 0\ [p]$, pour un k convenable, peut–être considérée comme une équation en la variable x^d, d'où une réduction de degré ainsi que de la complexité des calculs. Un second intérêt, est de pouvoir rechercher l'existence de solutions, solutions qui sont des racines d–ième, sans les calculer de manière explicite. Ceci fait l'objet des sections qui suivent.

III.5.1 Résolution en " x puissance d "

Nous avons vu au chapitre I ce qu'était une résolution en x, puis en x^2 d'une équation de degré 2 dans $\mathbf{Z}/3\mathbf{Z}$. De manière plus générale, nous introduisons maintenant ce qu'est une " résolution en x^d " sur $\mathbf{Z}/p\mathbf{Z}$:

---

(2) Temps *cpu* obtenus sur un IBM RS6000

(3) Pour les corps finis, un noyau est l'ensemble des x tels que $x^d = 1$

Définition III.1 :

Nous appelons résolution en x^d d'un polynôme P(x) de $\mathbf{Z}/p\mathbf{Z}$, la recherche de racines x^d de P telles qu'on puisse exprimer P(x), comme étant $x^u Q(x^d)$, u compris entre 0 et $d-1$, et Q(t) polynôme de $\mathbf{Z}/p\mathbf{Z}$. Nous aurons à ce moment–là :

- si $u > 0$, alors $x^d = 0$ est une racine de P,
- si t_0 est racine de Q(t), et si x^d peut prendre la valeur t_0, alors $x^d = t_0$ est une racine de P.

Remarque :

Lorsque $d = 1$, nous retrouvons la définition habituelle de la résolution en x d'un polynôme.

Exemples

Dans $\mathbf{Z}/5\mathbf{Z}$, la résolution en x^d du polynôme $P(x) = b_1x^4+b_2x^3+b_3x^2+b_4x+b_5=0$ sera la recherche de racines des polynômes suivants selon la valeur de d :

1. $d = 2$
   a) $b_1x^4+b_3x^2+b_5 = 0$
   b) $b_2x^3+b_4x = 0$
2. $d = 3$
   a) $b_1x^4+b_4x = 0$
   b) $b_2x^3+b_5 = 0$
   c) $b_3x^2 = 0$
3. $d = 4$
   a) $b_1x^4+b_5 = 0$
   b) $b_2x^3 = 0$
   c) $b_3x^2 = 0$
   d) $b_4x = 0$

Or, dans $\mathbf{Z}/p\mathbf{Z}$ nous pouvons toujours exprimer les racines d'un polynôme à l'aide des radicaux d'après la théorie de Galois ([22] et [41]). Par ailleurs, des méthodes de factorisation de polynômes ayant leurs cœfficients sur $\mathbf{Z}/p\mathbf{Z}[X_1, \ldots X_n]$ ont été développées récemment, Viry [43], et peuvent s'adapter au cas des polynômes définis sur $\mathbf{Z}/p\mathbf{Z}[X_1, \ldots X_n]$.

En outre, à chacune de ces résolutions en x^d nous pouvons associer les conditions d'existence des racines. Pour cela nous procédons à l'aide des lemmes sur les connecteurs logiques

pour regrouper les différentes conditions associées aux différents polynômes possibles pour une résolution en x^d. Une condition associée à un polynôme est simplement le produit des différentes valeurs prises par x^d.

Exemple 1

La résolution en x^5 dans **Z/7Z** de $P(x) = b_1x^6+b_2x^5+b_3x^4+b_4x^3+b_5x^2+b_6x+b_7 = 0$ ne sera possible que si P est de la forme de l'un des polynômes suivants:

a) $b_1x^6+b_6x = 0$

b) $b_2x^5+b_7 = 0$

c) $b_3x^4 = 0$

d) $b_4x^3 = 0$

e) $b_5x^2 = 0$

En sachant que x^5 ne peut jamais prendre la valeur 4 dans **Z/7Z**, les conditions associées à chaque polynôme sont respectivement:

a) $b_2=b_3=b_4=b_5=b_7=0$

b) $b_1=b_3=b_4=b_5=b_6=0$ et

$$b_7\,(b_2+b_7)\,(2\,b_2+b_7)\,(3\,b_2+b_7)\,(5\,b_2+b_7)\,(6\,b_2+b_7) = 0$$

c) $b_1=b_2=b_4=b_5=b_6=b_7=0$

d) $b_1=b_2=b_3=b_5=b_6=b_7=0$

e) $b_1=b_2=b_3=b_4=b_5=b_7=0$

Exemple 2

La résolution en x^6 dans **Z/7Z** de $P(x) = b_1x^6+b_2x^5+b_3x^4+b_4x^3+b_5x^2+b_6x+b_7 = 0$ ne sera possible que si P est de la forme de l'un des polynômes suivants:

a) $b_1x^6+b_7 = 0$

b) $b_2x^5 = 0$

c) $b_3x^4 = 0$

d) $b_4x^3 = 0$

e) $b_5x^2 = 0$

f) $b_6x = 0$

En sachant que x^6 ne prend que 0 ou 1 comme valeur dans **Z/7Z**, les conditions associées à chaque polynôme sont respectivement:

a) $b_2=b_3=b_4=b_5=b_6 \quad =0$ et $b_7\,(b_1+b_7) = 0$

b) $b_1= \quad b_3=b_4=b_5=b_6=b_7=0$

c) $b_1=b_2= \quad b_4=b_5=b_6=b_7=0$

d) $b_1=b_2=b_3= \quad b_5=b_6=b_7=0$

e) $b_1=b_2=b_3=b_4= \quad b_6=b_7=0$

f) $b_1=b_2=b_3=b_4=b_5= \quad b_7=0$

La proposition suivante nous donne une condition nécessaire pour qu'un polynôme en x admette une résolution en x^d :

Proposition III(9) :

Soit un P polynôme sur le corps $\mathbf{Z}/p\mathbf{Z}$. Notons m le nombre des termes de P contenant la variable x.

Une condition nécessaire pour que nous ayons une résolution en x^d sur P est que :

$$m < \frac{p}{d} + 1 \qquad (9)$$

Démonstration

D'après la définition III.1, si nous avons une résolution en x^d pour P, alors il existe un polynôme Q sur le corps $\mathbf{Z}/p\mathbf{Z}$ tel que $P(x) \equiv x^u\, Q(x^d)$.

Aussi avons–nous : $Q(x^d) = \sum_{j=0}^{N} a_j x^{d^j}$

Cette somme comporte au plus N+1 termes en x, or nous avons $d.N < p$, d'où la proposition. □

III.5.2 Systèmes polynomiaux bien formés

Définition III.2 :

Nous dirons qu'un polynôme P de $\mathbf{Z}/p\mathbf{Z}[x_1, \ldots, x_m, a_1, \ldots, a_q]$ est général en les paramètres $\{a_j\}_{j=1\ldots q}$ pour $\left\{ a_j^{d_j} \right\}_{j=1\ldots q}$ et $\left\{ x_k^{d_k} \right\}_{k=1\ldots m}$, si et seulement si la condition d'existence de racine de P soit est toujours vérifiée, c'est–à–dire est identiquement nulle, soit appartient à $\mathbf{Z}/p\mathbf{Z}[a_1, \ldots, a_q]$ et possède plus d'un zéro.

Exemple

Dans $\mathbf{Z}/3\mathbf{Z}$, le polynôme $x^2 + a^2\,(x + x^2)$ est général en a pour x, mais n'est pas général en a pour x^2.

En effet, lorsqu'on résoud l'équation $P(x, a) = 0$ en x, nous avons une équation de contrainte identiquement nulle sur l'existence de racines en x. Mais lorsqu'on résoud l'équation $P(x, a) = 0$ en x^2, nous avons cette fois–ci comme équation de contrainte, $a = 0$. Cette condition n'est pas identiquement nulle et ne fait pas intervenir plus d'un paramètre. Aussi, $P(x, a)$ n'est pas général en a pour x^2.

Remarque

Il se peut que nous obtenions une équation de contrainte en plusieurs variables qui ne soit jamais satisfaite, ou qui n'admette qu'un seul zéro. Ce qui a pour effet, soit de rejeter tous les jeux possibles de paramètres, ou bien de n'accepter seulement que tel ou tel paramètre ait une valeur fixe. Le premier cas rend le polynôme invalide car il ne s'annulera jamais quelque soient les paramètres. Et le second cas fixe un paramètre à une valeur constante, ce qui signifie que ce n'est plus un paramètre mais une constante. En revanche, s'il existe plusieurs n–uplets possibles de paramètres annulant l'équation de contrainte, alors le polynôme reste général en les paramètres, et nous aurons une équation de contrainte sur les paramètres.

Nous avons la même définition pour un système polynomial.

Définition III.3 :

Nous dirons qu'un système polynomial $\{P_i\}_{i=1...r}$ de $\mathbf{Z}/p\mathbf{Z}[x_1, \ldots, x_m, a_1, \ldots, a_q]$ est général en les paramètres $\{a_j\}_{j=1...q}$ en $\left\{ a_j^{d_j} \right\}_{j=1...q}$ et en $\left\{ x_k^{d_k} \right\}_{k=1...m}$, si et seulement si la condition d'existence de racine de P est toujours vérifiée, c'est–à–dire est identiquement nulle, ou si elle appartient à $\mathbf{Z}/p\mathbf{Z}[a_1, \ldots, a_q]$ et admet plus d'un zéro.

Lorsqu'un système polynomial vérifie la définition précédente, nous dirons qu'il est tout simplement bien formé pour indiquer qu'il existe un ensemble de variables choisi comme paramètre, et pour les variables restantes un choix sur le type de résolution.

Chapitre IV

Théorie de l'élimination dans les corps finis

Nous nous intéressons à un problème similaire de celui du chapitre précédent mais qui relève de la théorie de l'élimination. Ceci dans le but d'améliorer la complexité des algorithmes d'éliminations précédents, exponentielle, en une complexité qui serait au mieux polynomial. Nous rappelons et exposons les principales méthodes d'élimination dans le cas des corps des réels et des complexes, puis nous tentons de les adapter au cas des corps finis $\mathbf{Z}/p\mathbf{Z}$.

Ce chapitre est composé de plusieurs parties qui traitent le cas de l'élimination d'une variable à la section IV.2, puis de deux en IV.3, et enfin de n variables, section IV.4, en généralisant la méthode de Dixon et de Biard sans toutefois la démontrer. Nous appelons le résultant obtenu par cette méthode, résultant de Dixon–Biard, qui n'est pas toujours le résultant théorique. Pour la recherche de l'équation implicite, nous faisons quelques comparaisons, en IV.4.3, avec les algorithmes développés aux chapitres précédents. Cependant, nous perdons par les méthodes du calcul du résultant, l'équivalence entre un système polynomial comportant des variables à éliminer, et l'équation implicite obtenue. Aussi, pour avoir la réciproque, nous traitons le problème de l'inversion en IV.4.4, en généralisant une méthode proposée par Biard en [7]. Nous faisons juste une comparaison avec un problème d'inversion traité par Arnon au moyen de la décomposition cylindrique algébrique dans [5].

Notre conclusion est contrastée. En effet, le calcul du résultant de Dixon–Biard échoue pour les systèmes polynômiaux paramétriques généralisés, ceci essentiellement à cause de calculs de déterminants. En revanche, si nous avons seulement un système paramétrique, le calcul du résultant est en général beaucoup plus rapide, d'autant plus que sur un corps fini les degrés et les cœfficients sont limités. Mais, nous ne pouvons connaître l'équivalence que si nous fournissons un jeu de valeurs vérifiant l'équation implicite, grâce à l'algorithme d'inversion. De plus, nous n'avons aucune équation de contrainte sur les variables éliminées,

aussi nous ne connaissons pas quels sont les paramètres admissibles. Ces limitations par rapport aux algorithmes d'éliminations sur $\mathbf{Z}/p\mathbf{Z}$ rendent ainsi l'utilisation du calcul du résultant plus contraignant et moins général.

IV.1 Introduction

Au chapitre II, nous avons vu comment nous pouvions éliminer les quantificateurs d'une expression logique sur $\mathbf{Z}/p\mathbf{Z}$ de la forme :

$$\exists\, x_1 \in \mathbf{Z}/p\mathbf{Z} \ldots \exists\, x_n \in \mathbf{Z}/p\mathbf{Z} \;/\; \Phi(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

Si nous développons cette formule logique à l'aide des connecteurs logiques « et », nous obtenons un système de r équations :

$$\exists\, x_1 \in \mathbf{Z}/p\mathbf{Z} \ldots \exists\, x_n \in \mathbf{Z}/p\mathbf{Z} \;/ \begin{cases} P_1(x_1, \ldots, x_n, y_1, \ldots, y_m) \equiv 0\ [p] \\ \ldots \\ P_r(x_1, \ldots, x_n, y_1, \ldots, y_m) \equiv 0\ [p] \end{cases} \quad (1)$$

Nous notons par (M) l'ensemble[1] des x qui vérifient un tel système.

Éliminer les n variables quantifiées $x_1, \ldots, x_n$ revient donc à résoudre le système (1) des r équations. Ce qui revient à obtenir une première représentation de ce système, la représentation paramétrique.

Représentation paramétrique

Nous devons donc trouver une représentation de (M) de la forme suivante :

$$x \in (\mathrm{M}) \Leftrightarrow \begin{cases} x_1 \equiv Q_1(\Phi_1, \ldots, \Phi_s, y_1, \ldots, y_m)\ [p] \\ \ldots \\ x_n \equiv Q_n(\Phi_1, \ldots, \Phi_s, y_1, \ldots, y_m)\ [p] \end{cases} \quad (2)$$

où les $Q_1, \ldots, Q_n$ sont des polynômes, et les $\Phi_1, \ldots, \Phi_s$ des paramètres libres.

Une telle représentation de (M) est dite *paramétrique*. Elle est dite *minimale* lorsque le nombre de paramètres libres est minimal. Et, ces paramétrisations minimales ne sont pas forcément uniques.

Mais nous pouvons aussi déterminer sous quelles conditions les solutions x existent, et ce indépendamment des paramètres $\Phi_1, \ldots, \Phi_m$. Ce qui revient à déterminer l'équation implicite du système (2).

---

(1) Les algébristes nomment ces ensembles définis sur un corps *variétés algébriques.*

Représentation implicite

Une autre représentation de (M) est d'obtenir l'équation suivante non identiquement nulle telle que :

$$x \in (M) \Rightarrow f(x_1, \ldots, x_n, y_1, \ldots, y_m) \equiv 0 \,[p] \qquad (3)$$

Une telle équation est dite *équation implicite* de (M). L'équation implicite contient l'ensemble (M), et si nous substituons les inconnues x par leur représentation paramétrique, nous avons alors :

$$\forall \Phi_1 \ldots \forall \Phi_s \; , \; f(\ldots, Q_k(\Phi_1, \ldots, \Phi_s, y_1, \ldots, y_m), \ldots) \equiv 0 \,[p]$$

La question qui nous intéresse est d'obtenir à partir du système (1) ou du système (2) une représentation implicite. Ce problème, problème d'implicitisation, peut s'aborder de deux manières :

1. La première technique revient à éliminer les paramètres de libres du système paramétrique (2). Ceci est un problème d'élimination de variables qui est équivalent à un problème d'élimination de quantificateurs. Nous disposons d'un algorithme d'élimination de quantificateurs dans un corps fini, mais de complexité exponentielle selon le nombre de variables à éliminer.
2. La deuxième technique consiste à rechercher des zéros non triviaux communs à toutes les équations polynomiales du système (1). La recherche de ce facteur commun relève de la théorie de l'élimination que nous abordons dans ce chapitre.

En vue d'obtenir un algorithme qui soit de complexité meilleure que ceux d'élimination des quantificateurs, c'est–à–dire de complexité au mieux polynomiale, nous étudions les méthodes relevant de la théorie de l'élimination dans un corps **IK**, qui représente le corps des réels **IR** ou des complexes **C**. Puis, nous adaptons lorsque cela est possible au cas des corps finis $\mathbf{Z}/p\mathbf{Z}$.

IV.1.1 Le problème de l'élimination

La technique de l'élimination avait autrefois pour but la résolution des équations algébriques à plusieurs inconnues. L'objectif était de remplacer un système d'équations par un autre équivalent, dont l'une des équations ne contenant plus qu'une inconnue, permettait d'en trouver les valeurs. Ce problème de l'élimination s'est élargi : étant données des fonctions $f_1, \ldots, f_r$ de n variables $x_1, \ldots, x_n$ à coefficient dans un corps **IK**, il s'agit de trouver une fonction R des coefficients des f_i qui puisse se mettre sous la forme

$$R = \lambda_1 f_1 + \ldots + \lambda_r f_r$$

où les λ_i désignent des polynômes fonction des variables $x_1, \ldots, x_n$ et des coefficients de $f_1, \ldots, f_r$, qui soit de degré minimum par rapport à ces coefficients. Cette fonction R existe toujours et s'appelle l'*éliminant* ou le *résultant* de $f_1, \ldots, f_r$.

Plusieurs méthodes permettent l'élimination d'une ou de plusieurs variables. Elles reposent toutes sur l'algèbre linéaire, et font notamment intervenir la notion de déterminant et d'indépendance linéaire.

IV.1.2 Méthode générale dans Z/pZ

Une méthode générale consiste à considérer les équations de **Z/pZ** comme étant des équations d'un corps non fini. Le calcul du résultant de ces équations peut alors employer les méthodes usuelles d'élimination d'une ou de plusieurs variables, puis projeter dans le corps fini les résultats ainsi obtenus.

Cette méthode se schématise comme suit :

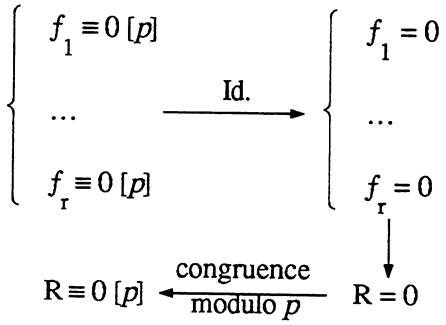


Fig. 4.1 : Méthode du résultant pour les corps finis

Cependant, cette méthode peut échouer dans certains cas et nous fournir un résultant identiquement nul. Par exemple, si le facteur commun obtenu est $x^p - x$, dans le corps fini **Z/pZ** ce facteur est identiquement nul du fait du petit théorème de Fermat. Aussi, présentons-nous les techniques permettant de déterminer le résultant dans un corps infini **IK**, puis nous les adaptons au cas des corps finis.

IV.2 Méthodes à une variable

IV.2.1 Définition du résultant

Nous rappelons brièvement la définition du résultant de deux équations :

Définition :

Soient f et g deux polynômes de degré m et n respectivement :

$$f(x) = \sum_{i=0}^{m} a_i x^i, \text{ et } g(x) = \sum_{j=0}^{n} b_j x^j$$

Soient, dans une extension de **IK**, les racines $\alpha_1, \ldots, \alpha_m$ de f ; et $\beta_1, \ldots, \beta_n$ de g.

On a l'identité : $\prod_{i=1}^{m}\prod_{j=1}^{n}(\alpha_i-\beta_j)=\frac{(-1)^{mn}}{b_n^m}\prod_{i=1}^{m}g(\alpha_i)=\frac{(-1)^{mn}}{a_m^n}\prod_{j=1}^{n}f(\beta_j)$

Une preuve algorithmique est donnée par Duval dans Davenport [17] p. 217.

De cette identité, on définit le résultant en x de f et g :

$$\mathrm{Res}_x(f,g)=a_m^n b_n^m\prod_{i=1}^{m}\prod_{j=1}^{n}(\alpha_i-\beta_j)$$

$$=(-1)^{mn}a_m^n\prod_{i=1}^{m}g(\alpha_i)=(-1)^{mn}b_n^m\prod_{j=1}^{n}f(\beta_j)$$

Proposition IV (1) :

La condition nécessaire et suffisante pour que le système $f(x)=0$ et $g(x)=0$ admette des racines communes, ou un facteur commun, est que la résultante soit nulle. Soit encore :

$$\mathrm{Res}_x(f,g)=0 \Leftrightarrow \begin{cases} f(x)=0 \\ g(x)=0 \end{cases} \qquad (4)$$

IV.2.2 Une méthode générale

Voici une méthode générale décrite par Laurent [23], qui contient comme cas particulier d'autres méthodes, en particulier celle de Cayley que nous abordons en IV.2.3.

Soit $\{\Theta_i\}_{i=1..m}$ une famille libre de m polynômes de degrés strictement inférieurs à m, c'est-à-dire une base des polynômes de degré au plus $m-1$ sur le corps **IK**. On effectue m divisions euclidiennes des polynômes $\Theta_i f$ par g. Ce qui donne :

$$\begin{cases} \Theta_1(x)f(x)=q_1 g(x)+r_1 \\ \qquad\qquad \dots \\ \Theta_m(x)f(x)=q_m g(x)+r_m \end{cases}$$

Les restes sont tous de degré strictement inférieur à m, nous aurons donc :

$$\begin{cases} \Theta_1(x)f(x)-q_1 g(x)=c_{1,m}x^{m-1}+\dots+c_{1,2}x+c_{1,1} \\ \qquad\qquad \dots \\ \Theta_m(x)f(x)-q_m g(x)=c_{m,m}x^{m-1}+\dots+c_{m,2}x+c_{m,1} \end{cases}$$

Posons successivement pour chacune de ces équations $x=\beta_1, \ldots, x=\beta_m$. Ceci donne m^2 équations du type :

$$\Theta_i(\beta_j) f(\beta_j) = c_{1,m}\beta_j^{m-1} + \ldots + c_{1,2}\beta_j + c_{1,1}$$

Ces équations s'écrivent sous forme matricielle comme suit :

$$\begin{bmatrix} \cdots \\ \cdots \Theta_i(\beta_j) \cdots \\ \cdots \end{bmatrix} \begin{bmatrix} & & 0 \\ & f(\beta_j) & \\ 0 & & \end{bmatrix} = \begin{bmatrix} \cdots \\ \cdots c_{i,k} \cdots \\ \cdots \end{bmatrix} \begin{bmatrix} \cdots \\ \cdots \beta_j^i \cdots \\ \cdots \end{bmatrix}$$

Le déterminant donne :

$$\begin{vmatrix} \cdots \\ \cdots \Theta_i(\beta_j) \cdots \\ \cdots \end{vmatrix} \prod_{j=1}^{m} f(\beta_j) = \begin{vmatrix} \cdots \\ \cdots c_{i,k} \cdots \\ \cdots \end{vmatrix} \begin{vmatrix} \cdots \\ \cdots \beta_j^i \cdots \\ \cdots \end{vmatrix}$$

A partir du produit des $f(\beta_j)$ nous retrouvons, à un facteur près, le résultant de f et g. Un choix judicieux des fonctions Θ_i peut simplifier les calculs. En effet, si on choisit comme famille de polynômes la base canonique des polynômes de degré inférieur à m, on aura après simplification :

$$\prod_{j=1}^{m} f(\beta_j) = \begin{vmatrix} \cdots \\ \cdots c_{i,k} \cdots \\ \cdots \end{vmatrix}$$

Cette méthode présente l'avantage de n'exiger qu'une seule division, celle de f par g.

IV.2.3 Résultant de Cayley

Soient f et g deux polynômes de degré m et n en x respectivement :

$$f(x) = \sum_{i=0}^{m} a_i x^i, \text{ et } g(x) = \sum_{j=0}^{n} b_j x^j$$

En nous inspirant de la méthode de Cauchy dans Laurent [23], posons le rapport suivant égal à un polynôme en y de coefficient $X_i(x)$:

$$\frac{f(x)g(y) - g(x)f(y)}{x-y} = \sum_{i=0}^{N-1} X_i(x) y^i$$

où $N = \max(n, m)$.

Le coefficient $X_i(x)$ est la combinaison linéaire de deux polynômes $f_i(x)$ et $g_i(x)$ qui sont respectivement coefficients de y^i dans le quotient de $f(y)-f(x)$ et de $g(x)-g(y)$ par $y-x$.

On a donc :

$$X_i(x) = g(x)\, f_i(x) - f(x)\, g_i(x)$$

Par conséquent $X_i(x)$ est le reste de la division de $g(x) f_i(x)$ par $f(x)$ et $g_i(x)$ est le quotient.

Si l'on pose $h_{i,j} = \begin{vmatrix} a_i & a_j \\ b_i & b_j \end{vmatrix}$ le déterminant des coefficients de f et g nous aurons :

$$X_0(x) = g(x) f_0(x) - f(x) g_0(x) = h_{0,N-1} x^{N-2} + \ldots + h_{0,2} x + h_{0,1}$$

$$X_1(x) = (h_{0,N} + h_{1,N-1}) x^{N-2} + \ldots + (h_{0,3} + h_{1,2}) x + (h_{0,2} + h_{1,1})$$

$$X_2(x) = (h_{0,N+1} + h_{1,N} + h_{2,N-1}) x^{N-2} + \ldots + (h_{0,4} + h_{1,3} + h_{2,2}) x + (h_{0,3} + h_{1,2} + h_{2,1})$$

...

$$X_{N-1}(x) = (h_{0,2N-3} + h_{1,2N-4} + \ldots + h_{N-2,N-1}) x^{N-2} + \ldots + (h_{0,N-1} + h_{1,N-2} + \ldots + h_{N-2,1})$$

où $h_{i,j}$ s'annule pour $i = j$ et pour i ou j strictement supérieurs à N – 1.

Le résultant de Cayley peut se définir comme étant le déterminant de la matrice de taille $(N-1)^2$ suivante :

$$M = \begin{bmatrix} h_{0,1} & h_{0,2} & \cdots & h_{0,N-1} \\ h_{0,2} + h_{1,1} & h_{0,3} + h_{1,2} & & h_{0,N} + h_{1,N-1} \\ \cdots & \cdots & \cdots & \cdots \\ h_{0,N-1} + h_{1,N-2} + \ldots + h_{N-2,1} & \cdots & & h_{0,2N-3} + h_{1,2N-4} + \ldots + h_{N-2,N-1} \end{bmatrix}$$

On voit que le résultant de Cayley est fonction des $h_{i,j} = a_i b_j - a_j b_i$.

IV.2.4 Résultant de Cayley dans Z/3Z

Les fonctions du corps Z/3Z qui nous intéressent plus particulièrement sont définies comme suit :

$$\begin{cases} f(x, y, u) = P(x, y) - \varphi(u) = P(x, y) - (a u^2 + bu + c) \\ g(x, y, u) = Q(x, y) - \phi(u) = Q(x, y) - (a' u^2 + b' u + c') \end{cases}$$

En tenant compte des $h_{i,j}$ qui s'annulent, nous obtenons la matrice carrée d'ordre 2 suivante:

$$M = \begin{bmatrix} h_{0,1} & h_{0,2} \\ h_{0,2} + h_{1,1} & h_{0,3} + h_{1,2} \end{bmatrix} = \begin{bmatrix} h_{0,1} & h_{0,2} \\ h_{0,2} & h_{1,2} \end{bmatrix}$$

En calculant explicitement les coefficients, nous obtenons :

$$M = \begin{bmatrix} \begin{vmatrix} b & c-P(x,y) \\ b' & c'-Q(x,y) \end{vmatrix} & \begin{vmatrix} a & c-P(x,y) \\ a' & c'-Q(x,y) \end{vmatrix} \\ \begin{vmatrix} a & c-P(x,y) \\ a' & c'-Q(x,y) \end{vmatrix} & \begin{vmatrix} a & b \\ a' & b' \end{vmatrix} \end{bmatrix}$$

Ainsi tout système sur **Z**/3**Z** de la forme :

$$\begin{cases} P(x,y) = \varphi(u) \\ Q(x,y) = \phi(u) \end{cases}$$

admet des solutions si et seulement si le déterminant de la matrice M précédente s'annule.

IV.2.5 Méthode des coefficients indéterminés

Nous avons vu dans l'introduction, IV.1.1, que le résultant pouvait se mettre sous la forme $R = Res_x(f, g) = \lambda f - \mu g$. Nous montrons la proposition suivante qui nous permet de développer la méthode des coefficients indéterminés.

Proposition IV (2) :

Les polynômes λ et μ qui vérifient :

$$Res_x(f, g) = \lambda f - \mu g$$

sont de degrés en la variable x strictement inférieur à g et f respectivement, et sont uniques à une constante près.

Démonstration

En effet, supposons qu'il existe un autre couple (λ', μ') de polynômes qui vérifient $R = \lambda' f - \mu' g$, alors $(\lambda - \lambda') f - (\mu - \mu') g = 0$.

Dans le cas où f et g n'ont pas de facteur commun, $(\lambda - \lambda')$ s'annulera toutes les fois que g s'annule, ce qui est contradictoire puisque le degré de $(\lambda - \lambda')$ est inférieur à celui de g.

Lorsque f et g sont non premiers entre eux, le résultant R s'annule par définition. Dans ce cas, les polynômes λ et μ existent encore mais ne ils ne sont plus déterminés qu'à un facteur près. □

Suite à cette proposition, nous pouvons employer la méthode des coefficients indéterminés pour calculer le résultant et les polynômes λ et μ.

Il suffit de poser :

$$\lambda = \sum_{i=0}^{n-1} p_i x^i \text{ et } \mu = \sum_{j=0}^{m-1} q_j x^j$$

En reportant dans l'équation $R = \lambda f - \mu g$ avec :

$$f(x) = \sum_{i=0}^{m} a_i x^i, \text{ et } g(x) = \sum_{j=0}^{n} b_j x^j$$

et en identifiant les termes de même degré, nous avons :

$$\begin{cases} a_m p_{n-1} - b_n q_{m-1} = 0 \\ \quad \cdots \\ \sum_{j=0}^{k} (a_j p_{k-j} - b_j q_{k-j}) = 0 \\ \quad \cdots \\ a_0 p_1 - b_0 q_1 + a_1 p_0 - b_1 q_0 = 0 \\ a_0 p_0 - b_0 q_0 = R \end{cases}$$

Ceci forme $m + n$ équations à $m + n$ indéterminées . Une fois les inconnues déterminées en fonction de R, nous pouvons calculer ce R à un facteur constant près. Or, nous remarquons que si nous avons f constante, alors ce facteur est $a_0^n b_n^m$.

Pour les corps infinis, une autre méthode consiste à différencier $m+ n+ 1$ fois l'équation $R = \lambda f - \mu g$, et donner à chaque fois une valeur arbitraire à x.

IV.2.6 Méthode des coefficients indéterminés dans Z/3Z

Soit le système suivant :

$$\begin{cases} f(x) = x^3 - x^2 - x + 1 \\ g(x) = x^3 - 3x^2 + 3x - 1 \end{cases}$$

Nous cherchons l'existence d'un facteur commun aux deux fonctions précédentes, puis s'il existe nous désirons le déterminer. La méthode des coefficients indéterminés va nous permettre d'y répondre :

Les polynômes λ et μ sont respectivement de degrés inférieurs à g et f, soit ici 2. Nous avons à résoudre l'équation suivante :

$$(ax^2 + bx + c) f(x) - (\alpha x^2 + \beta x + \gamma) g(x) = R$$

Le calcul des inconnues précédentes nous fournit un système à 6 inconnues qui est :

$$\begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ -1 & 1 & 0 & 3 & -1 & 0 \\ -1 & -1 & 1 & -3 & 3 & -1 \\ 1 & -1 & -1 & 1 & -3 & 3 \\ 0 & 1 & -1 & 0 & 1 & -3 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ R \end{bmatrix}$$

Après triangularisation du système précédent, nous obtenons l'équation suivante :

$$\begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 2 & -1 & 0 \\ 0 & 0 & 1 & -2 & 2 & -1 \\ 0 & 0 & 0 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ R \end{bmatrix}$$

Ce qui donne R = 0. Par conséquent une ou plusieurs racines communes à f et g existent. Pour déterminer le facteur commun, il suffit de calculer l'équation : $\lambda f - \mu g = 0$.

Nous obtenons en exprimant les coefficients en fonction de α et γ :

$$\lambda(x) f(x) - \mu(x) g(x) = 2 (x-1)^2 (x^2 (\alpha - \gamma) + \alpha x + \gamma) = 0$$

Or cette équation doit être vérifiée pour tout coefficient α et γ, ce qui entraîne que le facteur commun aux polynômes f et g est le polynôme $(x-1)^2$.

Cependant, le calcul des racines du second polynôme permet d'avoir une nouvelle factorisation de R :

$$\lambda(x) f(x) - \mu(x) g(x) = 2 (x-1)^2 (x+1) ((\alpha - \gamma) x + \gamma) = 0$$

Aussi, quels que soient les valeurs de α et γ, le facteur commun constant de R est le polynôme $(x-1)^2 (x+1)$. Or, -1 n'est pas racine de g, aussi avons–nous un facteur qui perturbe la solution aussi l'appellerons–nous "facteur parasite".

IV.2.7 Résultant de Sylvester

Cette méthode découle directement de la méthode des coefficients indéterminés. En effet, la matrice des coefficients des polynômes λ et μ permet, sans calculer leurs coefficients, de déterminer le résultant en utilisant le déterminant.

Sylvester simplifie les calculs en considérant le système des $m+n$ polynômes suivants :

$$\begin{cases} f(x) = a_m x^m + \ldots + a_1 x + a_0 \\ x f(x) = a_m x^{m+1} + \ldots + a_1 x^2 + a_0 x \\ \ldots \\ x^{n-1} f(x) = a_m x^{m+n+1} + \ldots + a_1 x^n + a_0 x^{n-1} \\ g(x) = b_n x^n + \ldots + b_1 x + b_0 \\ \ldots \\ x^{m-1} g(x) = b_n x^{n+m+1} + \ldots + b_1 x^m + b_0 x^{m-1} \end{cases}$$

Le déterminant des coefficients des seconds membres donne le résultant de Sylvester :

$$\mathrm{Res}_x(f,g) = \begin{vmatrix} a_m & \cdots & a_1 & a_0 & & 0 \\ & \ddots & & & \ddots & \\ 0 & & a_m & \cdots & \cdots & a_0 \\ b_n & \cdots & \cdots & b_0 & & 0 \\ & \ddots & & & \ddots & \\ 0 & & b_n & \cdots & \cdots & b_0 \end{vmatrix}$$

Le calcul de ce résultant, à cause de la structure spéciale du déterminant, se ramène à effectuer l'algorithme d'Euclide comme élimination de Gauss. De là le lien entre résultant et calcul du pgcd. La plupart des systèmes de calculs formels actuels s'inspirent de cette méthode de calcul de résultant.

IV.2.8 Résultant de Sylvester dans $\mathbf{Z}/p\mathbf{Z}$

Soit le système suivant des $m+n$ polynômes sur $\mathbf{Z}/p\mathbf{Z}$ où $m < p$ et $n < p$.

$$\begin{cases} f(x) = a_m x^m + \ldots + a_1 x + a_0 \\ x f(x) = a_m x^{m+1} + \ldots + a_1 x^2 + a_0 x \\ \ldots \\ x^{n-1} f(x) = a_m x^{m+n+1} + \ldots + a_1 x^n + a_0 x^{n-1} \\ g(x) = b_n x^n + \ldots + b_1 x + b_0 \\ \ldots \\ x^{m-1} g(x) = b_n x^{n+m+1} + \ldots + b_1 x^m + b_0 x^{m-1} \end{cases}$$

Si nous avons $m+n-1<p$ alors nus pouvons calculer le déterminant des coefficients des seconds membres. Nous retrouvons le résultant de Sylvester.

Cependant lorsque $m+n-1\geq p$, nous obtenons un système d'équations polynomiales de degré au plus $p-1$. Par conséquent des dépendances linéaires sont présentes, dans ce cas le résultant de Sylvester sera toujours identiquement nul. Aussi, un façon d'adapter la méthode de Sylvester aux corps finis $\mathbf{Z}/p\mathbf{Z}$ est d'extraire un système d'équations linéairement indépendantes du système à l'aide d'une triangularisation du système, ce qui nous permet d'évaluer ensuite un déterminant sur le second membre des $p-1$ équations.

IV.3 Méthodes à deux variables

IV.3.1 Éliminations successives

Soient trois polynômes f, g et h fonction des variables x et y, et le système suivant :

$$\begin{cases} f(x,y)=0 \\ g(x,y)=0 \\ h(x,y)=0 \end{cases}$$

Nous pouvons éliminer x de f et de g en formant la résultante en x de f et de g; puis de même former la résultante en x de g et de h. Ceci nous donne deux polynômes en y :

$$\begin{cases} \mathrm{Res}_x(f,g)=0 \\ \mathrm{Res}_x(g,h)=0 \end{cases}$$

Enfin, on élimine y de ces deux résultantes, ce qui donne finalement :

$$\mathrm{Res}_y(\mathrm{Res}_x(f,g),\mathrm{Res}_x(g,h))=0$$

Cette équation n'est pas le résultant du système des équations précédentes mais contient en facteur le résultant recherché, et nous pouvons en déduire le véritable résultant. Posons :

$$\mathrm{Res}_x(f,g)=\lambda f+\mu g, \text{ et } \mathrm{Res}_x(g,h)=\nu g+\rho h$$

où λ, μ, ν, ρ, sont des polynômes en x que l'on sait former, par exemple par la méthode des coefficients indéterminés.

Aux équations initiales, l'élimination successive substitue les équations suivantes :

$$\begin{cases} f=0 \\ \mathrm{Res}_x(f,g)=\lambda f+\mu g=0 \\ \mathrm{Res}_x(g,h)=\nu g+\rho h=0 \end{cases}$$

Le résultant en y de ce système est :

$$\mathrm{Res}_y(f, \mu, g)\,\mathrm{Res}_y(f, g, \rho)\,\mathrm{Res}_y(f, g, h)$$

On a en effet les relations suivantes qui sont aussi celles du pgcd :

$$\mathrm{Res}_y(\lambda f, \mu g, h) = \mathrm{Res}_y(\lambda, g, h)\,\mathrm{Res}_y(f, \mu, h)\,\mathrm{Res}_y(f, g, h)$$

$$\mathrm{Res}_y(f, \lambda f + \mu g, h) = \mathrm{Res}_y(f, \mu, h)\,\mathrm{Res}_y(f, g, h)$$

D'où finalement :

$$\mathrm{Res}_y(f, \lambda f + \mu g, \nu g + \rho h) = \mathrm{Res}_y(f, \mu, h)\,\mathrm{Res}_y(f, g, \rho)\,\mathrm{Res}_y(f, g, h)$$

Par rapport au système initial, le facteur introduit est :

$$\mathrm{Res}_y(f, \mu, h)\,\mathrm{Res}_y(f, g, \rho)$$

Pour l'élimination successive, nous pouvons choisir deux fonctions parmi trois pour éliminer x, soit trois résultants possibles. Mais nous pouvons aussi procéder de manière duale en éliminant y avant x, ce qui donne de nouveau trois autres résultants possibles. Pour trois équations, l'élimination successive a donc six variantes, et chacune de ces variantes contient en facteur le résultant cherché. Nous pouvons étendre ce résultat pour le cas de n variables.

IV.3.2 Méthode de Dixon

La méthode de Dixon [1] généralise celle de Cayley [23]. Nous reprenons la présentation de Biard [7]. Formons le déterminant suivant, où x représente un triplet (x_1, x_2, x_3) :

$$\Delta(x, u, v, u', v') = \begin{vmatrix} f(x,u,v) & f(x,u',v) & f(x,u',v') \\ g(x,u,v) & g(x,u',v) & g(x,u',v') \\ h(x,u,v) & h(x,u',v) & h(x,u',v') \end{vmatrix}$$

Il est clair que ce déterminant s'annule si l'on pose $u = u'$, ou $v = v'$. Aussi se factorise-t-il par $(u-u')$ et $(v-v')$. Posons :

$$\delta(x, y, u, v, u', v') = \frac{\Delta(x, u, v, u', v')}{(u-u')(v-v')}$$

Ceci forme un polynôme symétrique que nous pouvons écrire sous forme matricielle :

$$\delta(x, u, v, u', v') = [\,1, u', v', u'v', \ldots, u^{n-1} v'^{2m-1}\,]\; C \begin{bmatrix} u^{2n-1} v^{m-1} \\ \cdots \\ uv \\ u \\ v \\ 1 \end{bmatrix}$$

où C est une matrice carrée d'ordre $2nm$.

On remarque que si les équations f, g et h ont une racine commune $(u', v') = (\alpha, \beta)$, alors le polynôme $\delta(x, u, v, \alpha, \beta)$ est identiquement nul en u et v. Ce qui signifie que pour tout u et v, nous avons :

$$[\,1, \alpha, \beta, \alpha\beta, \ldots, \alpha^{n-1}\beta^{2m-1}\,]\; C \begin{bmatrix} u^{2n-1} v^{m-1} \\ \cdots \\ uv \\ u \\ v \\ 1 \end{bmatrix} = 0$$

Ce qui équivaut à ce que le déterminant de C s'annule, c'est le résultant de Dixon. Ce résultant n'est pas le résultant théorique mais il le contient en facteur. Dans certains cas, les deux coïncident.

IV.4 Méthode à *n* variables

Dans le cas général, il n'y a pas de formulation qui exprime pour tout n le résultant d'un système. Et la formulation la plus générale du résultant l'exprime comme un rapport de deux déterminants, Macaulay [27]. Si tous les polynômes ont les mêmes degrés, une autre formulation améliorée est donnée par Macaulay [28]. Cependant, dans la plupart des cas, ces deux déterminants s'annulent, et demeurent assez complexes et coûteux d'un point de vue de calculs du fait de la manipulations de ces systèmes de déterminants.

Quand le résultant peut s'exprimer comme déterminant d'une matrice, plusieurs méthodes existent pour les cas particuliers correspondant à n = 3, 4, 5, 6, qui sont citées dans Abhyankar [1].

Une approche plus générale, qui combine l'utilisation des bases de Gröbner, des interpolations sur plusieurs variables, est présentée à travers divers algorithmes par Manocha dans [30]. Ces méthodes concernent essentiellement le calcul d'un résultant d'un système sans paramètres, et sont programmées en C++ afin d'éviter des temps d'exécutions et d'optimiser la gestion mémoire d'un système de calcul formel basé en général sur des langages interprétés comme le Lisp.

En revanche, dans la section qui suit, nous présentons une méthode qui généralise le résultant de Dixon en se basant sur les travaux de Biard [7]. Pour éviter toute confusion avec la méthode de Dixon, nous nommons ce résultant, résultant de Dixon–Biard. Nous ne sommes pas du tout assurés d'avoir le résultant théorique à cause de facteurs "parasistes", mais cette méthode présente l'avantage d'avoir une complexité polynomiale par rapport aux algorithmes d'éliminations présentés auparavant, d'être relativement simple à mettre en œuvre et peu coûteuse en mémoire dans les cas des corps finis du simple fait de la limitation des degrés. De plus, pour le problème de la réciproque, section IV.4.4, nous avons un algorithme qui nous permet de décider pour un jeu de données vérifiant l'équation implicite si la réciproque est vraie ou non.

IV.4.1 Généralisation de la méthode de Dixon

La méthode que nous exposons sans la démontrer, est l'extension du résultant de Dixon au cas de l'élimination de n variables pour un système de $n+1$ variables. Comme nous nous sommes inspirés de celle de Biard [7] pour le cas $n=3$, nous la nommons Dixon–Biard. Pour simplifier les notations, nous ne faisons pas apparaître les variables $x_0, x_1, \dots, x_n$, mais elles sont présentes dans toutes les expressions qui suivent. En particulier, les fonctions f_i sont de la forme $f_i(x_0, \dots, x_n, u_1, \dots, u_n)$ mais sont notées comme suit $f_i(u_1, \dots, u_n)$.

Nous formons le déterminant $\Delta(u_1, u_2, \dots, u_n, v_1, v_2, \dots, v_n)$ suivant :

$$\begin{vmatrix} f_0(u_1, \dots, u_{n-1}, u_n) & f_0(u_1, \dots, u_{n-1}, v_n) & f_0(u_1, \dots, u_{n-2}, v_{n-1}, v_n) & \cdots & f_0(v_1, \dots, v_{n-2}, v_{n-1}, v_n) \\ f_1(u_1, \dots, u_{n-1}, u_n) & f_1(u_1, \dots, u_{n-1}, v_n) & f_1(u_1, \dots, u_{n-2}, v_{n-1}, v_n) & \cdots & f_1(v_1, \dots, v_{n-2}, v_{n-1}, v_n) \\ \cdots & \cdots & \cdots & & \cdots \\ f_n(u_1, \dots, u_{n-1}, u_n) & f_n(u_1, \dots, u_{n-1}, v_n) & f_n(u_1, \dots, u_{n-2}, v_{n-1}, v_n) & \cdots & f_n(v_1, \dots, v_{n-2}, v_{n-1}, v_n) \end{vmatrix}.$$

Il est clair que ce déterminant s'annule pour tous les $u_i = v_i$. Aussi, Δ se factorise par le produit des $(u_i - v_i)$.

Posons : $\delta(u_1, u_2, \dots, u_n, v_1, v_2, \dots, v_n) = \dfrac{\Delta(u_1, u_2, \dots, u_n, v_1, v_2, \dots, v_n)}{\prod_{i=1}^{n} (x_i - v_i)}$

Ceci forme un polynôme symétrique, que nous pouvons écrire sous forme matricielle :

$$\delta = [\,1, v_1, \ldots, v_n, v_1v_2, v_1v_3, \ldots, \prod_{i=1}^{n} v_i^{m_i}\,]\,C \begin{bmatrix} \prod_{i=1}^{n} u_i^{m_i} \\ \ldots \\ u_1u_3 \\ u_1u_2 \\ u_n \\ \ldots \\ u_1 \\ 1 \end{bmatrix}$$

où C est une matrice carrée.

Nous remarquons, comme pour la méthode de Cayley et de Dixon, que si les équations f_i ont une racine commune, alors le polynôme δ est identiquement nul pour tous les u_i. Ce qui signifie que nous avons :

$$C \begin{bmatrix} \prod_{i=1}^{n} u_i^{m_i} \\ \ldots \\ u_1u_3 \\ u_1u_2 \\ u_n \\ \ldots \\ u_1 \\ 1 \end{bmatrix} = 0$$

Ce qui équivaut à ce que le déterminant de C soit nul. Ce déterminant est le résultant de Dixon–Biard généralisé, et est fonction des variables $x_0, x_1, \ldots, x_n$.

IV.4.2 Recherche de l'équation implicite

En nous inspirant toujours de Biard [7] qui propose et démontre à partir du résultant de Dixon une méthode de recherche de l'équation implicite associée à un système, nous proposons à la suite de IV.4.1 une méthode similaire. Nous l'avons implémentée pour les corps finis $\mathbf{Z}/p\mathbf{Z}$ et infinis **IR**, et expérimentée afin de pouvoir comparer les temps de calculs, ainsi que les résultats, obtenus avec les algorithmes d'élimination dans les corps finis des chapitres précédents.

L'équation implicite d'un système paramétrique est donnée par la méthode de Dixon–Biard généralisé précédente. Cependant il se peut que le déterminant de la matrice C soit

identiquement nul, dans ce cas, une méthode est de calculer le déterminant d'une sous–matrice T de rang maximum de C, c'est–à–dire une mineure de C.

La méthode du résultant de Dixon–Biard généralisée pour des systèmes de $n+1$ polynômes et ayant n paramètres est résumée dans la figure ci–dessous. Nous avons simplifié quelques notations. Ainsi U désigne l'ensemble des paramètres u_i, X l'ensemble des x_i, et $V_{(S)}$, V_f désignent les variétés engendrées par le système (S) et l'équation implicite $f(X) = 0$ respectivement. De plus nous précisons que la mineure T dépend des X en la notant T_X, et le vecteur des monômes en u_i par $\begin{bmatrix} U \\ 1 \end{bmatrix}_{(n,m)}$.

$$\left[X \in V_{(S)} \quad \Leftrightarrow \exists\, U \in \mathbf{Z}/p\mathbf{Z}^n / \begin{cases} P_1(X, U) \equiv 0\ [p] \\ \dots \\ P_{n+1}(X, U) \equiv 0\ [p] \end{cases} \right]$$

$$\Downarrow$$

$$\left[\exists\, U \in \mathbf{Z}/p\mathbf{Z}^n / \frac{\Delta}{\prod_{i=1}^{n} u_i - \bar{u}_i} = \delta \equiv 0\ [p] \quad \Leftrightarrow \quad \exists\, U \in \mathbf{Z}/p\mathbf{Z}^n / T_X \begin{bmatrix} U \\ 1 \end{bmatrix}_{(n,m)} \equiv 0\ [p] \right]$$

$$\Downarrow$$

$$\left[\det T_X = f(X) = 0 \quad \Leftrightarrow \quad X \in V_f \right]$$

Fig. 4.2 : Résultant de Dixon

L'algorithme qui généralise la méthode de Dixon au cas de n variables est le suivant :

Méthode de Dixon généralisée :

Entrées : S, variables, paramètres
Sorties : équation

Définir les paramètres conjuguées à partir des paramètres

Construire la matrice M *à partir de* S *et des* paramètres *et des paramètres conjugués*

$\Delta \leftarrow$ det M

$\delta \leftarrow$ *Simplifier* Δ

Extraire la matrice C de δ

T ← *Appliquer Bareiss modifié sur* : C

équation ← *Dernier pivot non nul de* : C

Retourner équation

La méthode de recherche de la mineure T de C est effectuée à partir d'un algorithme de Bareiss modifié. Nous donnons en annexe A les programmes Reduce qui appliquent la méthode de l'algorithme précédent, ainsi que l'algorithme de Bareiss.

IV.4.3 Expérimentations dans Z/3Z

La complexité de la méthode Dixon généralisé semble être polynomial, car nous effectuons successivement :

- un calcul du déterminant Δ, puis le calcul de δ,
- une extraction de la matrice C à partir de δ,
- une détermination du rang et d'un déterminant d'une mineure de C par un algorithme de type Bareiss,

Or il s'avère que la recherche du rang d'une matrice, lorsque nous sommes dans un corps fini, est différente de celle des corps infinis comme **IR**. En effet, soit par exemple le corps fini $\mathbf{Z}/p\mathbf{Z}$ et la recherche du rang de la matrice M suivante:

$$M = \begin{bmatrix} a^{p-1} & 0 & 0 \\ 0 & b^{p-1} & 0 \\ 0 & 0 & a^{p-1} - b^{p-1} \end{bmatrix}$$

Si nous observons la structure de cette matrice, nous pourrions conclure, comme M est diagonale, que les colonnes sont linéairement indépendantes d'où un rang égal à 3, et par conséquent que cette matrice est inversible. Or, le calcul du déterminant, le produit des élements diagonaux, nous donne dans $\mathbf{Z}/p\mathbf{Z}$:

$$a^{p-1} b^{p-1} \left(a^{p-1} - b^{p-1} \right) \equiv (a\,b)^{p-1} - (a\,b)^{p-1} \equiv 0\ [p]$$

Aussi ne pouvons–nous pas appliquer les méthodes classiques de recherche du rang. Elles nous donneront seulement une majoration du rang. Nous n'avons pas rencontré de situation de ce type, aussi peut–être que la méthode de Dixon–Biard ne la génère–t–elle pas.

IV.4.3.1 Des demi–droites

Un exemple classique qui illustre bien les limites des méthodes de recherche d'équation implicite, quelque soit la caractéristique du corps, est celui des demi–droites. En effet, dans $\mathbf{Z}/p\mathbf{Z}$ considérons la paramétrisation suivante :

$$\begin{cases} x_1 \equiv u_1^2 \,[p] \\ x_2 \equiv u_1^2 \,[p] \end{cases} \tag{5}$$

Nous aurons par un calcul du résultant, l'équation d'une droite entière au lieu d'une demi–droite :

$$\begin{cases} x_1 \equiv u_1^2 \,[p] \\ x_2 \equiv u_1^2 \,[p] \end{cases} \Rightarrow x_1 \equiv x_2 \,[p] \tag{6}$$

C'est dû au problème des paramétrisations minimales. En revanche, les algorithmes d'éliminations pour les corps finis nous donnent l'équation implicite d'une demi–droite tout en ayant l'équivalence (résultat pour $p = 3$) :

$$\begin{cases} x_1 \equiv u_1^2 \,[p] \\ x_2 \equiv u_1^2 \,[p] \end{cases} \Leftrightarrow x_1^2 x_2^2 - x_1^2 x_2 - x_1 x_2^2 - x_1 - x_2 \equiv 0 \,[p] \tag{7}$$

Cette dernière équation implicite correspond en fait à la formulation logique suivante qui à l'équation implicite de (6) exclue les points ayant une coordonnée négative :

$$\begin{cases} x_1 \equiv u_1^2 \,[p] \\ x_2 \equiv u_1^2 \,[p] \end{cases} \Leftrightarrow (x_1 - x_2 \equiv 0\,[p]) \textbf{ et non } ((x_1 \equiv -1\,[p]) \textbf{ ou } (x_2 \equiv -1\,[p])) \tag{8}$$

IV.4.3.2 Les hypersphères

Bien que nous risquons de ne pas avoir d'équivalence par une méthode de calcul du résultant, nous avons réalisé quelques expérimentations afin de comparer les temps d'exécutions, mais aussi les résultats obtenus, avec les algorithmes d'éliminations sur les corps finis. Nous commençons par un exemple simple d'élimination de variables. Pour cela nous considérons les équations paramétriques d'une hypersphère de dimension d et de rayon R, en supposant que les paramètres permettent le calcul de x_d^2, système défini comme suit:

$$\begin{cases} x_1 = u_1 \\ \cdots \\ x_{d-1} = u_d \\ x_d^2 = R^2 - \sum_{k=1}^{d-1} u_k^2 \end{cases} \tag{9}$$

Nous pouvons bien appliquer la méthode de Dixon généralisée car nous avons d équations et $(d-1)$ variables à éliminer.

Nous effectuons une comparaison avec l'algorithme d'élimination décrit au chapitre III.

Z/3Z d	Dixon généralisé	Élimination (chap. III)
3	0.9 s	1.9 s
4	1.6 s	7.3 s
5	2.6 s	35 s
6	4.6 s	3 min 23
7	10.7 s	23 min 34
8	17 s	—
9	33 s	—
10	1 min 8	—
12	2 min 47	—

Fig. 4.3 : Comparaison des temps selon le nombre de paramètres à éliminer

La méthode de Dixon généralisée nous donne bien l'équation d'une sphère pour toutes les dimensions, et les temps de calculs sont extrêmement plus rapides que ceux des algorithmes d'éliminations vus auparavant. Ceci s'explique par le fait que la structure du déterminant Δ est très creuse.

IV.4.3.3 Exemple ne donnant pas l'équivalence

Nous considérons maintenant le système suivant :

$$\begin{cases} x_1^2 + x_2^2 = u_1 \\ \cdots \\ x_{d-1}^2 + x_d^2 = u_{d-1} \\ x_d^2 = R^2 - \sum_{k=1}^{d-1} u_k^2 \end{cases} \qquad (10)$$

Nous pouvons bien appliquer la méthode de Dixon–Biard généralisée car nous avons d équations et $(d-1)$ variables à éliminer.

Nous comparons avec l'algorithme d'élimination décrit au chapitre III.

Z/3Z *d*	Dixon–Biard généralisé	Élimination (chap. III)
3	1 s	1.4 s
4	3 s	5.2 s
5	10 s	22 s
6	42 s	1 min 56
7	3 min 32	9 min 47
8	21 min 48	—

Fig. 4.4 : Comparaison des temps selon le nombre de paramètres à éliminer

Malheureusement pour cet exemple, la méthode de Dixon généralisée échoue, car quelque soit le nombre de variables à éliminer, nous obtenons toujours la même équation implicite:

$$x_1^2 \equiv R^2 \ [3]$$

Les équations implicites fournies par les algorithmes d'éliminations sont toutes de la forme:

$$x_1^2 \equiv R^2 + \sum_{k=2}^{d} x_k^2 - \sum_{k=1}^{d-1} x_k^2 x_{k+1}^2 \ [3]$$

De plus, ces derniers nous fournissent une équation de contrainte sur les paramètres u_i.

IV.4.3.4 Comparaison avec les exemples du chapitre III

En reprenant les exemples proposés en III.3.1, nous appliquons un calcul du résultant de Dixon–Biard sur ces derniers :

1. Dans **Z/3Z**, pour l'exemple de l'intersection des droites avec le cercle discret, nous avons deux équations et deux variables à éliminer. Il nous manque une équation, il suffit pour cela de rajouter une troisième équation en appliquant le lemme du "et" sur les précédentes. Nous pouvons appliquer cette technique aussi bien pour réduire ou diminuer le nombre d'équation, afin d'utiliser le résultant de Dixon–Biard qui nécessite $n+1$ équations pour n paramètres.

Le système est formé comme suit:

$$\begin{cases} x^2 + y^2 = 1 \\ y = a\,x + b \\ \left(x^2 + y^2 - 1\right)^2 + (\,y - a\,x + b\,)^2 = 0 \end{cases}$$

Dans ce cas nous obtenons le même résultat, avec un temps similaire de 0.1 s.

2. Dans $\mathbf{Z}/7\mathbf{Z}$, pour appliquer la méthode de Dixon–Biard sur le système suivant :

$$\begin{cases} x^2 + y^2 = \mathrm{R}^2 \\ x^5 + y^5 = \mathrm{S}^5 \end{cases}$$

nous rajoutons l'équation suivante :

$$\left(x^2 + y^2 - \mathrm{R}^2\right)^2 + \left(x^5 + y^5 - \mathrm{S}^5\right)^2 = 0$$

Or, cette fois–ci, bien que nous ayons deux paramètres à éliminer parmi trois équations, et des degrés limités à au plus 6 en une variable, la méthode de Dixon–Biard nécessite plus de 6 heures de temps *cpu*. Ce temps de calcul est essentiellement du au calcul du déterminant de la matrice C. En revanche, déterminer la matrice C ne prend que quelques secondes.

Il s'avère que lorsque nous appliquons le résultant de Dixon–Biard sur des corps finis, $\mathbf{Z}/p\mathbf{Z}$, de caractéristique supérieure à 3, nous obtenons des temps de calculs extrêmement longs, surtout à cause du calcul du déterminant de la matrice C. Les cas particuliers pour lesquels nous obtenons des temps raisonnables, c'est–à–dire de l'ordre de quelques minutes, sont les systèmes paramétriques de la forme suivante :

$$\begin{cases} x_1 \equiv \varphi_1(y_1, \ldots, y_n)\ [p] \\ \qquad \ldots \\ x_m \equiv \varphi_n(y_1, \ldots, y_n)\ [p] \end{cases}$$

En effet, la matrice ne contiendra que des cœfficients qui sont affines en les variables x_i et des cœfficients compris entre 0 et $p - 1$. Aussi le calcul du déterminant d'une telle matrice est peu onéreux sur $\mathbf{Z}/p\mathbf{Z}$.

IV.4.4 Problème de l'inversion

Dans cette section nous abordons le problème de l'inversion. Ce problème a été traité et expérimenté dans un cadre numérique dans la thèse de Biard [7] pour $n = 2$. Le problème est formulé comme suit :

Étant donné un système polynomial (S) comportant des paramètres, $u_1, \ldots u_n$ que nous notons U, et $f(x_1, \ldots x_m) \equiv 0\ [p]$ l'équation implicite associée obtenue par un calcul du résultant, celui de Dixon dans notre cas, nous nous intéressons au problème réciproque. C'est–à–dire, lorsqu'un m–uplet X_0 vérifie l'équation implicite, avons–nous existence de paramètres U pour que le système (S) soit lui aussi vérifié ? C'est–à–dire, avons–nous X_0 appartenant à la variété $V_{(S)}$ engendrée par (S) ? Quels critères pouvons–nous fournir sur ces paramètres ?

Nous rappelons que la méthode du calcul du résultant de Dixon–Biard est donné à la figure Fig. 4.2. Le problème de l'inversion est schématisé par la figure suivante :

$$\left[X_0 \in V_f \quad \Leftrightarrow \det T_{X_0} = f(X_0) = 0 \right]$$

$$¿ \Downarrow_{(*)} ?$$

$$\left[\exists\, U \in \mathbf{Z}/p\mathbf{Z}^n \,/\, T_{X_0}\begin{bmatrix} U \\ 1 \end{bmatrix}_{(n,m)} \equiv 0\ [p] \quad \Leftrightarrow \exists\, U \in \mathbf{Z}/p\mathbf{Z}^n \,/\, \frac{\Delta}{\prod_{i=1}^{n} u_i - \bar{u}_i} = \delta \equiv 0\ [p] \right]$$

$$¿ \Downarrow_{(**)} ?$$

$$\left[\exists\, U \in \mathbf{Z}/p\mathbf{Z}^n \,/\, \begin{cases} P_1(X_0, U) \equiv 0\ [p] \\ \ldots \\ P_{n+1}(X_0, U) \equiv 0\ [p] \end{cases} \Leftrightarrow \quad X_0 \in V_{(S)} \right]$$

Fig. 4.5 : Problème de l'inversion à partir du résultant de Dixon–Biard

Or, pour que la première implication, celle notée $\Downarrow_{(*)}$, se réalise, nous devons avoir existence de paramètres U tels que le système $T_{X_0}\begin{bmatrix} U \\ 1 \end{bmatrix}_{(n,m)}$, où les inconnues sont les U, soit vérifié. Pour déterminer ces paramètres U, nous pouvons employer une méthode classique de triangulation d'une matrice, par exemple le pivot de Gauss, en prenant soin de ne pas permuter les lignes. Ceci afin de préserver l'ordre des monômes en U du vecteur. Nous obtenons alors une matrice triangulaire, notons–la $T^{\angle}_{X_0}$ et supposons qu'elle soit d'ordre N.

La figure ci–dessous montre comment cette matrice est constituée :

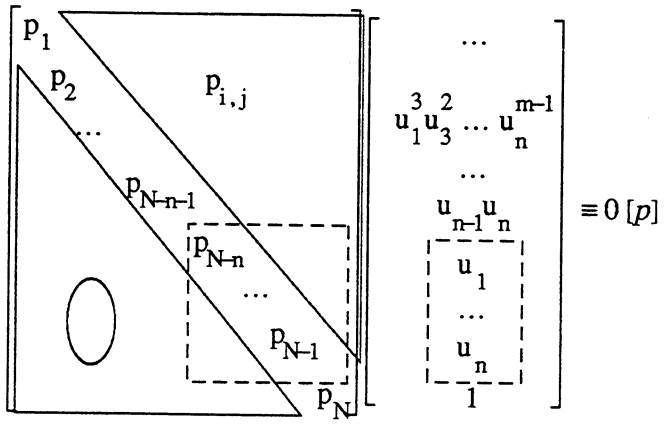


Fig. 4.6 : Matrice T_X triangularisée

Nous pouvons être dans l'une des trois situations suivantes :

1. Si le pivot p_N est non nul, alors aucune valeur des paramètres U ne pourra rendre nul le système de la Fig. 4.6.

2. Si le rang de $T^{\angle}_{X_0}$ est N – 1, avec le pivot p_N nul, nous pouvons déterminer les paramètres U_0 à partir d'un système linéaire extrait de $T^{\angle}_{X_0}$, système que nous avons encadré en pointillé dans la figure Fig. 4.6. Puis, nous vérifions si ces paramètres U_0 annulent le reste des équations. Si c'est le cas, alors la première implication $\Downarrow_{(*)}$ est vérifiée, et il ne nous reste plus qu'à vérifier la seconde implication, celle notée $\Downarrow_{(**)}$ dans Fig. 4.5. Ceci fait l'objet du paragraphe suivant.

3. Si le rang de $T^{\angle}_{X_0}$ est inférieur à N – 1, avec le pivot p_N nul, nous avons un système non linéaire à résoudre. Dans certains cas, il se peut que nous puissions extraire comme précédemment un sous–système qui se ramène à une résolution linéaire. Mais, en toute généralité nous pouvons aussi employer un algorithme d'élimination vu au chapitre précédent sur le système formé par la matrice $T^{\angle}_{X_0}$. L'équation de contrainte, si elle est identiquement nulle, nous permet d'affirmer qu'il existe des paramètres U_0 tels que la première implication $\Downarrow_{(*)}$ soit vérifiée, et pour le cas des corps **Z/3Z**, fournir un ensemble de solutions pour les paramètres U_0. Le paragraphe qui vient traite le cas de la seconde implication lorsqu'il existe un jeu de paramètres qui vérifient la première implication de la figure Fig. 4.6.

Supposons que nous ayons un jeu de paramètres U_0 tels que le système $T_{X_0}\begin{bmatrix} U_0 \\ 1 \end{bmatrix}_{(n,m)}$ soit identiquement nul, il nous reste plus qu'à vérifier la seconde implication, celle notée $\Downarrow_{(**)}$ dans la figure Fig. 4.5. Pour cela, il suffit d'évaluer les polynômes $P_i(X_0, U_0)$, pour i allant de 1 à $n+1$, du système (S) et vérifier qu'ils sont bien tous congrus à zéro modulo p. Cette évaluation, si elle est vérifiée, nous permet d'affirmer alors que le m–uplet X_0 appartient aussi à la variété engendrée par (S).

IV.4.4.1 Comparaison avec la décomposition cylindrique algébrique

Dans [5], Arnon propose le problème d'implicitisation suivant :

$$\begin{cases} x = f(u) = -505\,u^3 + 864\,u^2 - 570\,u + 343 \\ y = g(u) = 211\,u^3 - 276\,u^2 - 90\,u + 345 \end{cases} \tag{11}$$

La recherche de l'équation implicite de la courbe paramétrique cubique par une méthode de décomposition cylindrique algébrique de Collin modifiée, a nécessité 89 minutes de calculs et 855 cellules, ceci en 1988. A partir de ces données, Arnon a pu déterminer l'équation implicite de cette courbe paramétrique :

$$\begin{aligned} F(x, y) = {} & 128787625\,y^3 + 161430825\,xy^2 - 123690261954\,y^2 \\ & + 67449315\,x^2 y - 118396536972\,xy + 43730624273196\,y \\ & + 9393931\,x^3 - 28185530898\,x^2 + 20125457356836\,x \\ & - 4937175920971288 = 0 \end{aligned} \tag{12}$$

Or, l'algorithme du résultant de Dixon–Biard généralisé que nous avons programmé peut également éliminer un seul paramètre, dans ce cas c'est le résultant de Cayley. Aussi, avons–nous repris l'exemple précédent afin de comparer les temps de calculs, en tenant compte du fait que les ordinateurs sont beaucoup plus rapides aujourd'hui. L'algorithme nous donne en ½ seconde la matrice C suivante :

$$\begin{bmatrix} 211x + 505y - 246598 & 12(-23x - 72y + 32729) & 30(-3x + 19y - 5526) \\ 165720 & 211x + 505y - 481678 & 12(-23x - 72y + 32729) \\ -42942 & 165720 & 211x + 505y - 246598 \end{bmatrix} \tag{13}$$

Le calcul du déterminant de cette matrice nous donne ensuite la fonction implicite F précédente. L'écart de temps ne se justifie pas seulement par l'évolution des machines, car les calculs réalisés pour déterminer cette matrice C sont élémentaires.

En revanche nous n'avons aucune information sur les points, ou ensembles de points qui ne vérifient pas le problème de l'inversion. Ainsi, Arnon précise que la décomposition cylindrique algébrique lui a fourni le point suivant :

$$\left(x = \frac{4842167022180059}{45767461033}, y = \frac{-2166473912400475}{45767461033} \right) \tag{14}$$

Pour vérifier si ce point ne vérifie pas le problème de l'inversion, il suffit dans notre cas de rechercher le rang de C après avoir remplacé x et y par les valeurs de (14). Nous obtenons la matrice suivante de rang 3 :

$$\begin{bmatrix} \frac{-72372084082494024}{45767461033} & \frac{535395362192706864}{45767461033} & \frac{-1670685162064641840}{45767461033} \\ 165720 & \frac{-72372084082729104}{45767461033} & \frac{535395362192706864}{45767461033} \\ -42942 & 165720 & \frac{-72372084082494024}{45767461033} \end{bmatrix} \tag{15}$$

Aussi ce point, pour des valeurs réelles du paramètre u, n'admet pas de paramétrisation. Cependant, cela signifie qu'il existe un paramètre s complexe tel que $x = f(s)$ et $y = g(s)$.

De plus, la décomposition cylindrique algébrique permet de connaître quels sont les composants de la fonction implicite. Pour l'exemple précédent, il y a un point isolé qui correspond à (14) et une autre composante de dimension 1.

Chapitre V

Les strates – structure et représentation

Dans le but de définir un environnement de modélisation pour le calcul scientifique, nous définissons une structure que nous nommons *strate*. Elle nous permet de définir le problème initial, de le décomposer, d'avoir différents niveaux d'abstractions, de suivre la progression de la résolution du problème formulé dans divers langages, naturel, mathématique jusqu'au niveau algorithmique ou de programmation. Cela nous permet de suivre toutes les étapes de la résolution, de vérifier les décompositions de problèmes et de la mettre aisément à jour. De plus, lorsqu'un problème similaire est abordé nous pouvons reprendre des résolutions qui ont déjà été réalisées. Cette structure de strate ne correspond pas à une feuille de notes où sont intégrées du texte, des formules, des graphiques, comme le proposent divers systèmes de calcul formel tels que Camino Real [5], Mathematica [44] ou Maple. En effet, ces documents ne possèdent aucune structure logique justifiant tel calcul, telle algorithme. En revanche, les strates peuvent permettre de vérifier s'il n'y a pas des incohérences dans la formulation d'un calcul, d'un problème. En outre, nous proposons des algorithmes de projections qui permettent, suite à une modification, de regénérer un programme sans que nous ayons besoin de le réécrire.

Les aspects des strates permettant la description d'un problème et ceux de la résolution de problème que nous avons jugés pertinents de formaliser sont les suivants :

- les descriptions de problèmes à l'aide de préconditions, conditions et postconditions (V.1.1);
- les schémas de résolutions (V.1.3);
- les différents niveaux de description à travers les langages (V.3);
- les traductions de problèmes (V.3.2);
- les arguments permettant de passer ou de justifier un nouveau problème (V.1.6).

Afin de présenter ces différents aspects des strates, nous avons implémenté la structure des strates, ainsi que la présentation graphique associée, à partir des langages du logiciel Grif, [34] et [36]. Ces langages permettent de définir une structure au moyen d'une grammaire formelle, ainsi qu'une ou plusieurs présentations associées par des règles de présentations au moyen de positionnement de boîtes, de création d'éléments graphiques, etc. Les algorithmes de projections de strates, ont quant à eux, été programmés dans le système orienté–objet Smalltalk–80 où nous avons testé et développé ces notions et définitions de strates.

Dans le but de vérifier la cohérence des strates, la cohérence d'un schéma de résolution, puis la cohérence d'une décomposition, et enfin la cohérence de l'ensemble de toutes les décompositions, nous associons aux strates une représentation logique et algébrique dans un corps fini. Nous traitons cet aspect au chapitre suivant, en proposant des méthodes de vérifications basées sur les outils algébriques développés aux premiers chapitres.

V.1 Les strates

Avant d'introduire la structure des strates, nous nous intéressons à la description en général d'un problème de nature scientifique, puis au principe général de résolution de tels problèmes.

Pour d'éviter toute confusion entre un problème au sens général et un problème auquel nous associons une structure qui précise ses caractéristiques et qui le situe parmi d'autres, nous nommons ces derniers *strates*. Ce terme est introduit lorsque nous abordons les schémas de résolutions à la sous–section V.1.3.

V.1.1 Description de problèmes issus de la modélisation

La description de problème de modélisation forment un ensemble de conditions formalisant le problème de façon mathématique. Nous classons les variables de tout problème en trois catégories : les données, les paramètres et les résultats. Une autre catégorie, les variables locales c'est à dire interne à un problème, est introduite en V.1.5. Les données et les paramètres forment les entrées, et les résultats les sorties du problème. La différence entre les données et les paramètres tient au fait que les données sont des quantités connues et fixes *a priori*, tandis que les paramètres sont des variables dont nous pouvons fixer à volonté la valeur numérique.

Puis nous distinguons trois sortes de conditions selon qu'elles portent sur les données, les paramètres et/ou les résultats du problème, qui sont :

1. **les préconditions :** conditions ne portant que sur les données et/ou les paramètres;
2. **les conditions :** conditions entre les données et/ou les paramètres, et les résultats;
3. **les postconditions :** conditions ne portant que sur les résultats.

Les préconditions servent à formuler des hypothèses, des contraintes, ou des propriétés sur les données et paramètres fournis au problème. De même, les postconditions formulent des conditions que doivent vérifier les résultats du problème. Les conditions quant à elles, expriment une ou plusieurs relations entre les données, éventuellement les paramètres, et les résultats.

V.1.2 La résolution d'un problème scientifique

Si nous désignons par E l'ensemble des données et des paramètres du problème, et par S l'ensemble des résultats à déterminer, l'ensemble des conditions du problème se résume comme suit :

$$\begin{cases} \text{préconditions (E)} \\ \text{conditions (E, S)} \\ \text{postconditions (S)} \end{cases} \quad (1)$$

Résoudre le problème, revient à déterminer une fonction F qui calcule les résultats S en fonction des données E. Les résultats ainsi obtenus par cette fonction, doivent aussi vérifier les postconditions. Or, comme S est déterminé à partir de E, les postconditions dépendent maintenant des données E.

Le but d'une résolution est alors d'obtenir F, de manière à ce que :

$$\begin{cases} \text{préconditions (E)} \\ S \leftarrow F(E) \\ \text{postconditions (} F(E) \text{)} \end{cases} \quad (2)$$

L'union des préconditions et des postconditions ne concernent au bout du compte que des données E. En général, la fonction F obtenue permet d'avoir les postconditions de (2) toujours vraies, mais il se peut que cela ne soit pas le cas.

La mise en place d'une méthodologie pour déterminer la fonction de calcul F est réalisée à l'aide des schémas de résolution. Ils permettent de décomposer, de réécrire les conditions du problème initial en sous–problèmes, ou en d'autres problèmes – pas forcément équivalents.

V.1.3 Strates et schémas de résolutions

Dans le but d'obtenir une fonction F déterminant les résultats en fonction des données, nous introduisons les schémas de résolution. Ces schémas permettent la description d'une ou plusieurs stratégies de résolution de problème. Ces schémas sont constitués d'un ensemble de strates correspondant chacune à un sous–problème, et ces strates sont agencées entre–elles de manière à se transmettre les résultats d'une strate à une autre. Et ceci, afin de pouvoir, au bout du compte, obtenir au moyen de ce schéma une fonction de calcul F associée au problème à résoudre qui puisse déterminer les résultats du problème.

Afin de pouvoir retenir de quel problème sont issues les strates dans ces schémas de résolution, nous associons une référence vers le problème en question. Nous donnons en V.1.4 la structure complète d'une strate.
Ces schémas de résolution constituent un graphe orienté dont les nœuds sont les strates, et dont les arcs orientés définissent l'ordre de précédence des transferts des résultats d'une strate vers les données de la suivante. Nous évitons les problèmes de renommage de variables en donnant les mêmes noms de variables d'un élément du schéma à un autre.

Nous représentons ces schémas orientés par des graphes orientés au moyen des opérateurs suivants :

- les séquences : ce sont une dépendance séquentielle d'un sous–schéma de strates, c'est–à–dire un sous–graphe orienté, vers un autre sous–schéma de strates;
- les choix : ce sont un choix de plusieurs stratégies ou méthodes de résolution afin d'obtenir les mêmes résultats;
- les parallélismes : ce sont une indépendance en termes de calcul de résultats d'un sous–schéma par rapport à un autre sous–schéma de strates;
- les tests, les boucles : ce sont des opérateurs de contrôle qui permettent la définition d'alternatives ou d'itérations. L'élément du schéma qui produit une valeur de booléenne dans les différents tests, définit une strate particulière qui retourne un seul résultat, une variable de type booléen.

Nous décrivons plus en détail la structure de ces schémas de résolution en V.2.

V.1.4 Structure générale d'une strate

Une strate comporte plusieurs parties principales qui sont donc :

1. un titre, qui sert à nommer la strate;
2. un attribut qui précise quel est le langage employé dans la strate;
3. une option qui précise la provenance du problème d'une autre strate que nous appelons *strate père.*
4. les données, les paramètres et les résultats, qui sont sous forme de liste de variables;
5. un commentaire, qui permet d'annoter tout ce qui est informel et relatif au problème en cours;
6. un ou plusieurs arguments qui justifient d'une manière ou d'une autre le descriptif, ou l'écriture de la strate à partir de la strate père;
7. en option, une ou plusieurs équations de lien entre la strate et la strate père;
8. un descriptif, dans le langage de la strate, qui formalise à l'aide de conditions les équations, les inéquations du problème, les différentes contraintes ou hypothèses portées sur les variables du problème : données, paramètres, résultats ou éventuellement les variables locales au descriptif;

9. un schéma de résolution défini soit comme une stratégie de résolution, soit comme un algorithme.

V.1.5 Variables locales et paramètres

Les conditions du descriptif de la strate peuvent comporter des variables autres que les données et les résultats. Nous dirons que ces variables sont locales au problème. Le descriptif comporte alors une zone rassemblant les variables locales.

Les schémas de résolutions que nous verrons à la section V.2, que ce soit un diagramme de résolution, ou un algorithme, emploient toutes les variables de la strate, les données, les paramètres, et les variables locales, afin de fournir au bout du compte les résultats. Ces variables locales qui interviennent dans le graphe orienté du schéma de résolution peuvent avoir plusieurs statuts selon qu'elles soient en entrée ou en sortie d'un nœud du graphe. En effet, si elles font partie de entrées d'un nœud, c'est–à–dire d'une strate, elle deviennent des entrées de la strate correspondante, et si elles sont en sortie alors elles deviennent les résultats de la strate correspondante. Il faut noter que d'un nœud à un autre, leur statut peut changer. Ainsi, elles peuvent faire partie des résultats d'une strate, avant d'être une donnée ou un paramètre pour la ou les strates suivantes.

Cependant lorsque nous sommes dans la situation particulière où une variable locale n'est déterminée par aucune strate, bien qu'elle intervienne dans les descriptifs des strates du schéma de résolution, nous pouvons la considérer temporairement comme un paramètre. Ce statut permet soit de considérer cette variable comme une constante du problème de la strate, à initialiser ultérieurement, soit de préciser que le problème comporte une nouvelle variable qui est à déterminer éventuellement par une des strates du schéma de résolution.

Par exemple, soit le schéma de résolution suivant, Fig. 5.1, qui comporte les nœuds suivants, A, B, C et D, qui renvoient chacun vers une strate différente, et trois variables locales X, Y, Z intervenant dans le descriptif de la strate et qui sont transférés d'un nœud vers un autre.

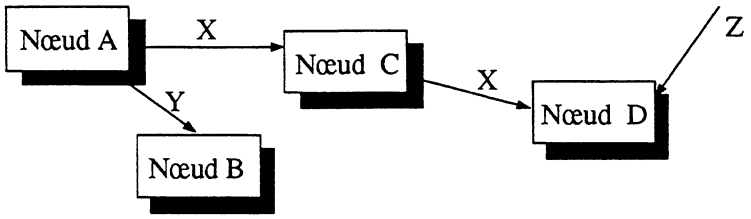


Fig. 5.1 : Variables locales dans un schéma de résolution

Le nœud A semble déterminer les variables X et Y. Elles figureront donc parmi les résultats de la strate associée au nœud A. Les strates des nœuds B et C, auront en revanche parmi leurs données les variables X ou Y. Le nœud C, de plus, transmet sa donnée X, modifiée ou inchangée, au nœud D. La variable X figure donc comme un résultat de la strate associée à C. Nous pouvons, dans le cas où la strate C ne modifie pas X, considérer cette variable comme

un paramètre de la strate. Par contre le nœud D aura comme entrée X, et le paramètre Z, car cette dernière variable n'est pas déterminée par une autre strate de ce schéma de résolution.

V.1.6 Argumentation

Pour une strate donnée, nous pouvons préciser l'argument qui nous a permis de l'obtenir. Ils peuvent être de plusieurs ordres, ainsi nous avons recensé les arguments suivants :

a) les argumentations informelles, du fait de l'expérience ou de l'habitude du spécialiste;

b) les justifications renvoyant à une propriété, proposition, théorème, connus;

c) et enfin, les argumentations formelles, c'est–à–dire celles qui renvoient à une ou plusieurs strates qui les définissent de façon formelle.

Il se peut que nous ayons d'autres formes d'argumentations, mais notre but n'est pas de vérifier la formulation mathématique ou algorithmique des problèmes de façon automatique en nous aidant en particulier des argumentations . Les argumentations nous servent seulement, de façon informelle, de notes permettant de préciser ou de justifier par quel moyen telle ou telle strate a été obtenue.

V.2 Les schémas de résolution

V.2.1 La décomposition d'une strate

Lorsque nous avons à décomposer un problème en sous–problèmes, nous précisons l'agencement des strates correspondant aux sous–problèmes grâce au schéma de résolution de la strate à décomposer. La figure ci–dessous illustre un exemple de décomposition :

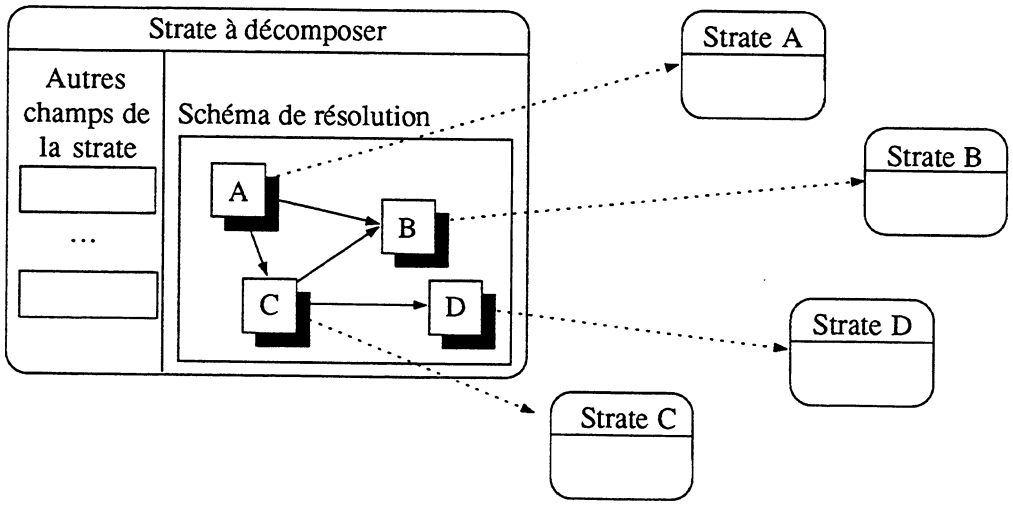


Fig. 5.2 : Exemple de décomposition de strate

Puis nous pouvons recommencer à décomposer chacune des strates associées aux sous–problèmes jusqu'à ce que nous obtenions un problème élémentaire, ou une résolution algorithmique décrite dans un langage cible. Or, nous pouvons décrire quelles sont les décompositions élémentaires en définissant quelques opérateurs pour construire ou modifier un schéma de résolution. Nous verrons en VI.5.3 que ces opérateurs nous permettent de faciliter la vérification de la cohérence d'un schéma lorsque nous le mettons à jour. Nous pouvons également avoir également une représentation moins stricte pour les schémas de résolutions, qui est basée sur les graphes, en ayant seulement une structure de contrôle pour les tests et les boucles, représentées par un cycle dans un graphe. Ceci présente l'avantage de pouvoir décrire des agencements de sous–problèmes selon un graphe non planaire, comme celui la figure Fig. 5.2.

Nous proposons deux sortes de présentations pour les schémas de résolutions : une vue graphique et une vue textuelle. Elles représentent toutes deux l'ensemble des opérateurs décrits en V.1.3 qui permettent de construire un agencement de strates. Nous donnons ci–dessous la grammaire de ces schémas de résolution en illustrant chacun des opérateurs par un exemple ayant une vue graphique et textuelle.

V.2.2 Grammaire des schémas de résolutions

La grammaire de ces schémas de résolutions en notation Backus Naur Form a pour racine <Schéma> qui propose une séquence d'objets de nature différente : les éléments du schéma ou bien des opérateurs de construction :

<Schéma> ::= { <objet_Schéma> }

<objet_Schéma> ::= <Élément> | <Choix> | <Parallèle> | <Test> |
<Répéter> | <Tant_que> | <Appel_Sous_schéma>

Les éléments du schémas ont un contenu qui supporte un commentaire, *<Label>*, et en option une référence à une autre strate, *<Renvoi_Strate>* :

<Élément> ::= <Contenu>

<Contenu> ::= *<Label>* "[" *<Renvoi_Strate>* "]"

Par exemple, une séquence de trois éléments aura pour vue textuelle:

Déterminer les équations *[Strate initialisations]*
Résoudre les équations *[Strate résolution]*
Vérifier le résultat *[Strate vérification]*

Fig. 5.3 : Exemple de séquence

Et la même séquence aura la vue graphique suivante :

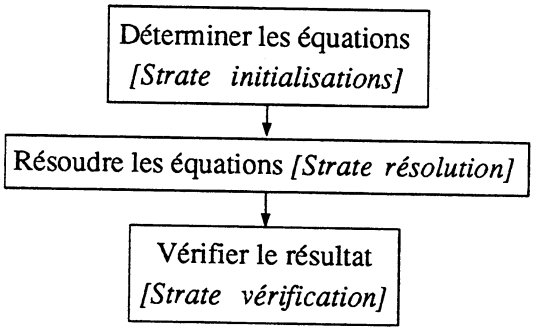


Fig. 5.4 : Une séquence de trois éléments

V.2.2.1 Les constructions "choix"

L'opérateur de choix construit une liste comportant au moins deux sous–schémas :

<Choix> ::= <objet_Choix> <objet_Choix> { <objet_Choix> }

<objet_Choix> ::= <Schéma>

Par exemple, un choix de méthodes aura pour vue textuelle :

Choix (i) :
| Appliquer la méthode de Gauss *[Gauss]*
Choix (ii) :
| Employer l'algorithme de Bareiss *[Bareiss]*
Choix (iii) :
| Faire une méthode numérique itérative *[Méthodes itératives]*

Fig. 5.5 : Exemple de choix

Et le même exemple aura la vue graphique suivante :

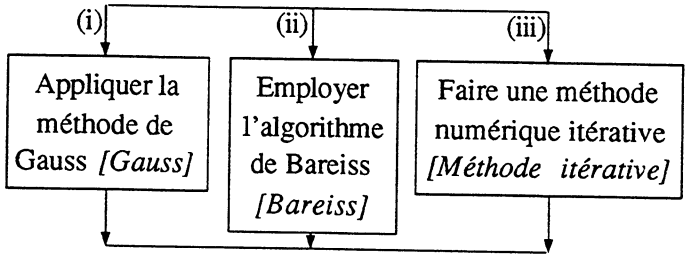


Fig. 5.6 : Exemple de construction choix

V.2.2.2 Les constructions "parallèles"

L'opérateur parallèle construit lui aussi une liste d'au moins deux sous–schémas :

<Parallèle> ::= <objet_Parallèle> <objet_Parallèle> { <objet_Parallèle> }

<objet_Parallèle> ::= <Schéma>

Par exemple, les deux éléments mis en parallèle auront pour vue textuelle :

parallèle 1 :
| Déterminer le vecteur b *[Vecteur b]*
parallèle 2 :
| Calculer la matrice A *[Matrice A]*

Fig. 5.7 : Exemple de construction parallèle

Et le même exemple aura la vue graphique suivante (différente de la construction choix du fait que les différents objets parallèles ne sont pas numérotés) :

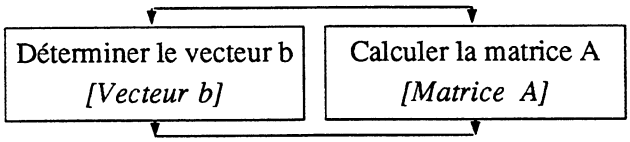


Fig. 5.8 : Exemple de construction parallèle

V.2.2.3 Les tests

L'opérateur test permet de construire les trois sortes de branchements conditionnels : "*si ... alors ... sinon*", "*si .. alors ...*", et le "*si ... sinon ...*".

<Test> ::= <objet_Test> <objet_Cas_Vrai> <objet_Cas_Faux> |
<objet_Test> <objet_Cas_Vrai> |
<objet_Test> <objet_Cas_Faux>

<objet_Test> ::= <Contenu>

<objet_Cas_Vrai> ::= <Schéma>

<objet_Cas_Faux> ::= <Schéma>

Par exemple, nous aurons pour vue textuelle :

si Matrice semi–définie positive ? *[Strate SDD]*
alors
| Méthode de gradient conjugué *[Strate GradConj]*
sinon
| Méthode Quasi–Newton *[Strate Quasi–Newton]*

Fig. 5.9 : Exemple de construction test

Et cet opérateur aura la vue graphique suivante :

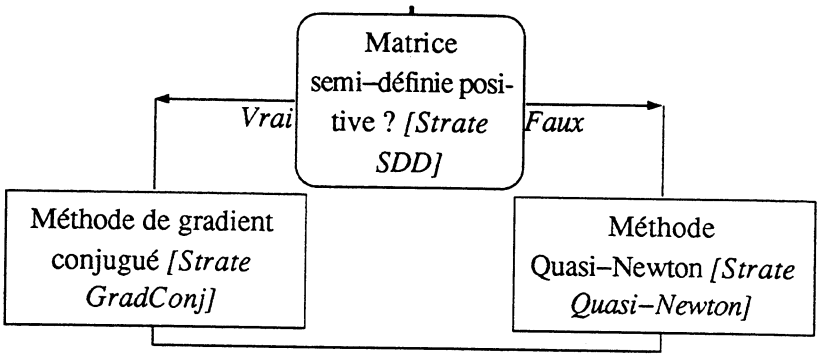


Fig. 5.10 : Exemple de construction test

Nous autorisons aussi la construction de tests n'ayant qu'une seule alternative du type "*si ... alors ...*" ou "*si ... sinon ...*", comme par exemple :

si Matrice semi–définie positive ? *[Strate SDD]*
alors
| Méthode de gradient conjugué *[Strate GradConj]*

Fig. 5.11 : Exemple de construction test

Le même exemple aura la vue graphique :

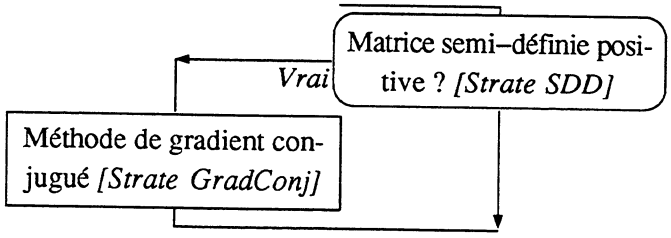


Fig. 5.12 : Exemple de construction test

V.2.2.4 Les boucles "répéter"

La boucle "*répéter ... jusqu'à ...*" des schémas de résolution est définie comme suit :

<Répéter> ::= <objet_Répéter> <objet_Jusqu_à>
<objet_Répéter> ::= <Schéma>
<objet_Jusqu_à> ::= <Contenu>

Par exemple, nous auons pour vue textuelle :

répéter
| Calculer U_{n+1} *[Strate Suite Un]*
jusqu'à Test d'arrêt *[Strate test]*

Fig. 5.13 : Exemple de construction répéter

Et nous aurons la vue graphique suivante :

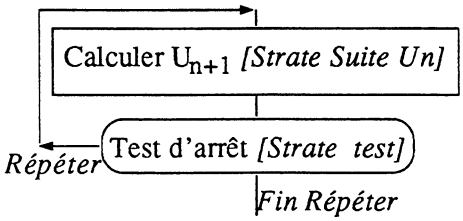


Fig. 5.14 : Exemple de construction répéter

V.2.2.5 Les boucles "tant que"

La boucle "*tant .. que ...*"des schémas de résolutions est définie par :

<Tant_que> ::= <objet_Tant_que> <objet_Faire>
<objet_Tant_que> ::= <Contenu>
<objet_Faire> ::= <Schéma>

Par exemple, nous aurons pour vue textuelle :

tant que Non Test d'arrêt *[Strate test]*
faire
| Calculer U_{n+1} *[Strate Suite Un]*

Fig. 5.15 : Exemple de boucle tant que

Et nous aurons la vue graphique suivante :

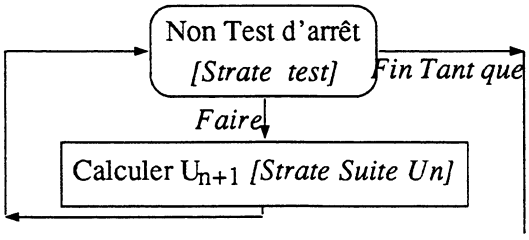


Fig. 5.16 : Exemple de boucle tant que

V.2.2.6 Appels de sous–schémas et récursivité

La grammaire des schémas de résolution permet de définir des schémas annexes :

<Appel_Sous_Schéma> ::= *<Référence Nom_sous_Schéma>*

<Sous_Schéma> ::= *<Nom_sous_Schéma>* <Schéma>

Par exemple, nous aurons pour vue textuelle :

Un schéma annexe

———

Un schéma annexe

| Un élément de schéma *[Strate X]*

Fig. 5.17 : Exemple de sous–schéma

Et nous aurons la vue graphique suivante :

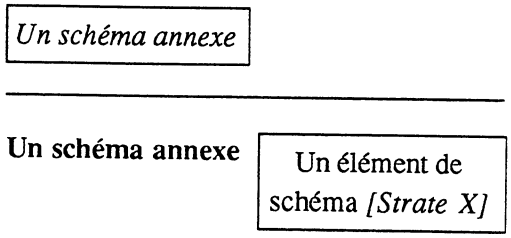


Fig. 5.18 : Exemple de sous–schéma

L'intérêt de définir des appels de sous–schémas et les sous–schémas associés est de pouvoir définir la récursivité.

Nous pouvons ainsi mettre en œuvre une stratégie "*divide and conquer*" comme suit :

Initialiser les données *[Initialisations]*
Divide & Conquer
Afficher le résultat *[Affichage]*

Divide & Conquer
| **si** Taille du problème importante
| **alors**
| | Diviser le problème en problèmes de taille inférieure
| | **répéter**
| | | *Divide & Conquer*
| | **jusqu'à** ce que tous les sous–problèmes soient résolus
| | Regrouper tous les problèmes résolus
| **sinon**
| | Appliquer une méthode de résolution directe

Fig. 5.19 : Stratégie "Divide and conquer"

V.3 Les langages des strates

V.3.1 Les différents langages d'une résolution

Les strates peuvent également tenir compte des différents langages qui interviennent lors de la résolution d'un problème. Nous formulons la propriété suivante pour une strate : toutes ses conditions, préconditions, postconditions, définition des types sur les variables, sont définies dans un seul langage. Les commentaires, les argumentations sont des zones informelles où nous pouvons annoter, indiquer ce que nous voulons, pourvu que cela ait un rapport avec la résolution du problème.

Ceci nous permet d'avoir par exemple des niveaux linguistiques de moins en moins abstraits, de plus en plus proches d'un langage qui puisse être interprété ou compilé par une machine.

Par exemple, la figure ci–dessous montre une résolution de problème qui emploie un langage naturel, puis mathématique, puis algorithmique ou de programmation.

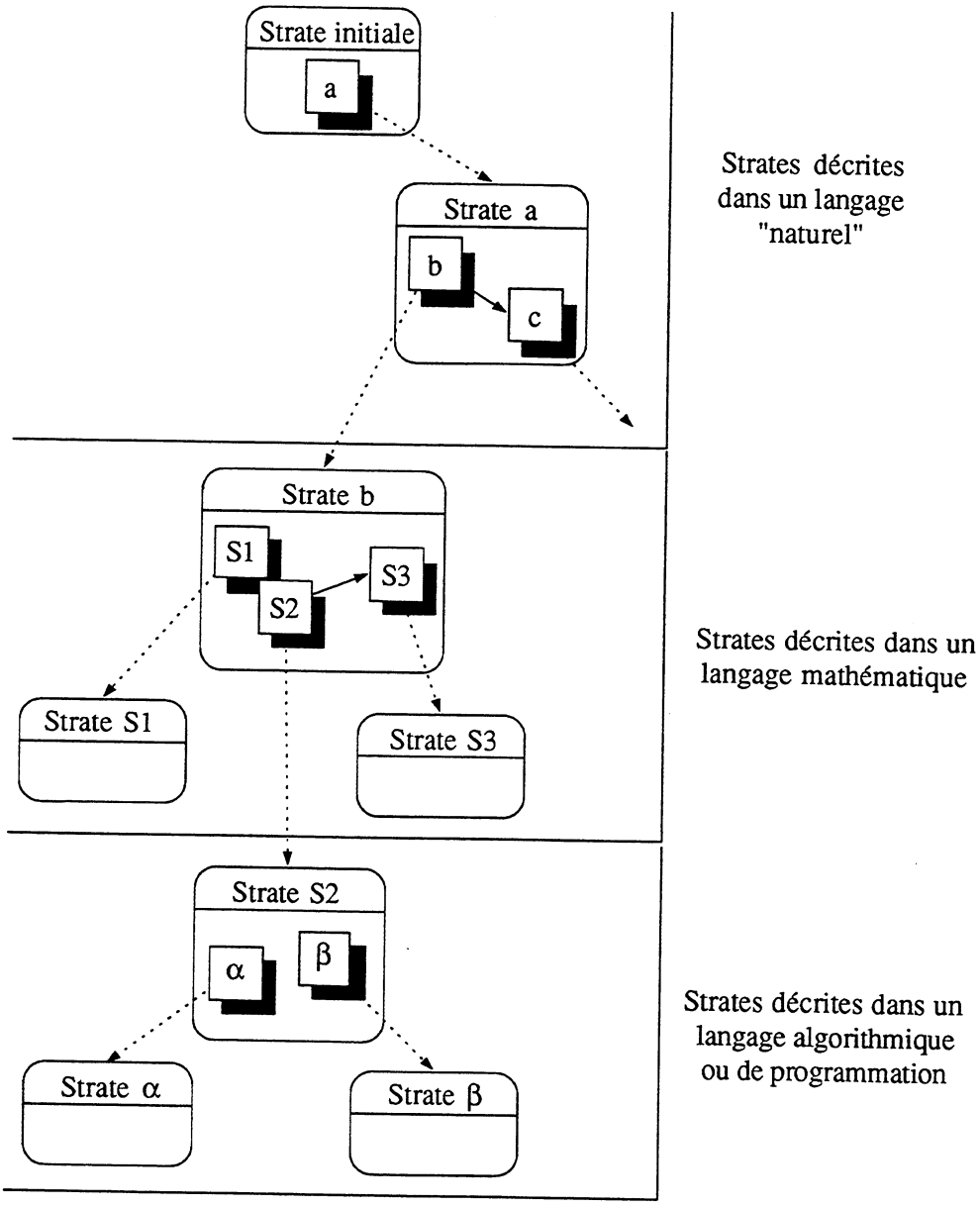


Fig. 5.20 : Exemple de différents niveaux linguistiques

Cependant, il se peut que nous ayons des allers–retours entre des formulations de problèmes dans divers langages. Ceci peut se produire, et la structure des strates nous permet d'en tenir compte.

Par exemple :

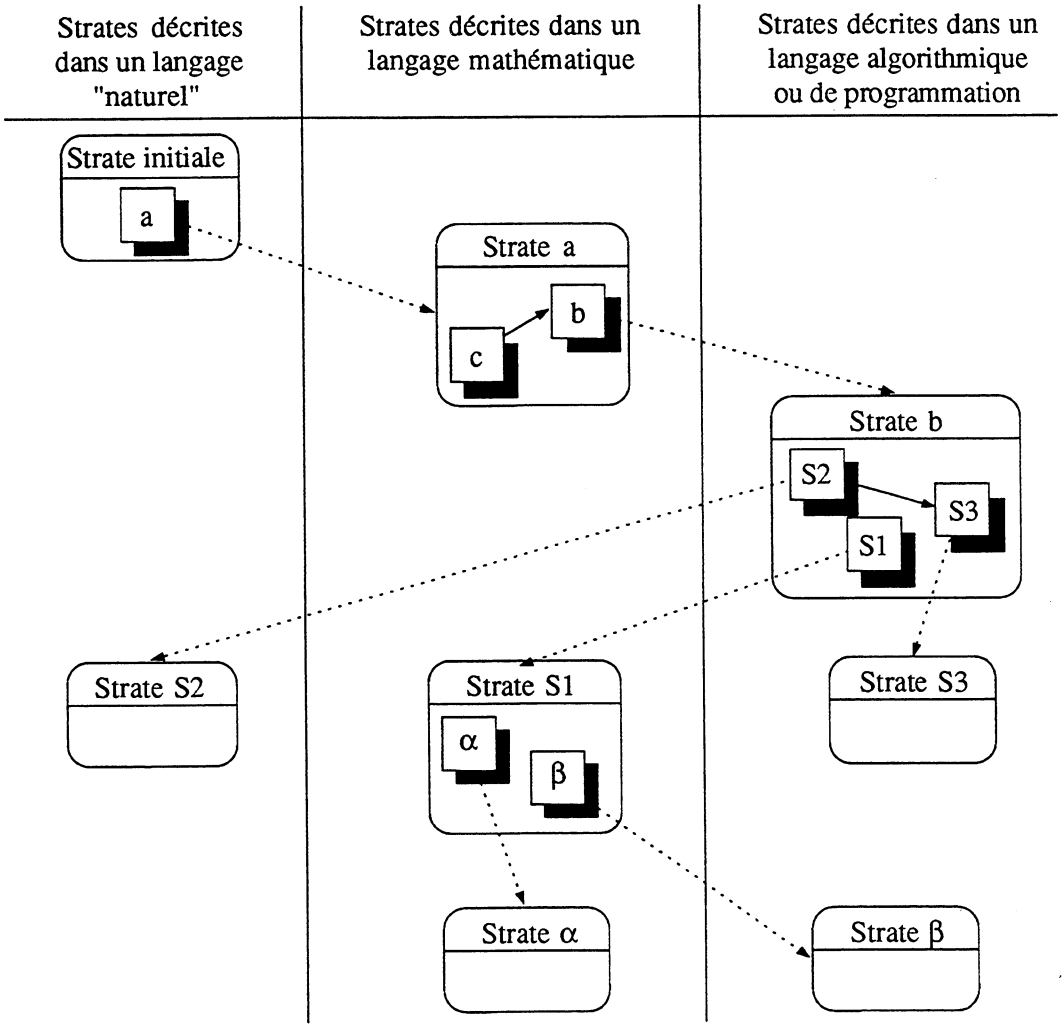


Fig. 5.21 : Exemple de niveaux linguistiques imbriqués

V.3.2 Les traductions entre strates

Lorsque nous passons d'un langage à un autre, nous proposons une strate qui réalise la traduction complète de la strate père. Ceci, afin de pouvoir effectuer un passage correct de variables, lors de la projection de l'ensemble des strates. Ce passage a lieu dans les deux sens: la strate père transmet ses données à la strate fils, données qui permettront le calcul, ultérieurement, des résultats de la strate fils. Aussi faut–il traduire ces résultats pour les transmettre à la strate père. Lorsque les variables sont du même type, ou portent sur un même domaine, les strates faisant partie du même langage, dans ce cas, les transferts des données, des résultats est direct.

Aussi, nous définissons pour les strates de traductions des équations de liens qui nous permettent de faire les correspondances nécessaires.

V.4 La projection des strates

Nous nous intéressons à la projection de l'ensemble des strates en un programme, plus précisément de l'ensemble des strates décrites uniquement dans une famille de langages cibles en un schéma algorithmique. Cette famille de langages cibles est dans le cas qui nous intéresse, celle des langages qu'une machine peut interpréter ou compiler. Ce sont par exemple des langages de programmation du type Pascal, C, Fortran, voire même des langages symboliques comme le Lisp, Reduce, Maple, etc. Mais, nous pouvons donner une autre définition pour ces langages selon la finalité que nous nous sommes fixée. Ainsi, ce langage pourrait très bien être un langage de haut niveau, non interprétable par une machine, ceci dans le cadre d'une modélisation où nous avons besoin d'avoir par exemple une formulation d'un modèle mathématique, logique, à partir d'un phénomène physique ou d'un problème du monde réel.

V.4.1 Vue d'ensemble d'une résolution de problème

Selon le choix d'un ou de plusieurs langages cibles, la méthode de projection des strates reste inchangée. Supposons que nous ayons une arborescence, définie à partir des schémas de résolutions, complète. Les strates qui ne possèdent pas de schémas de résolutions sont nommées strates feuilles.

Par exemple, la figure ci–dessous montre un arbre de résolution, pour lequel les strates feuilles sont : S1, α, β, S3, III, Δ, Λ et Γ.

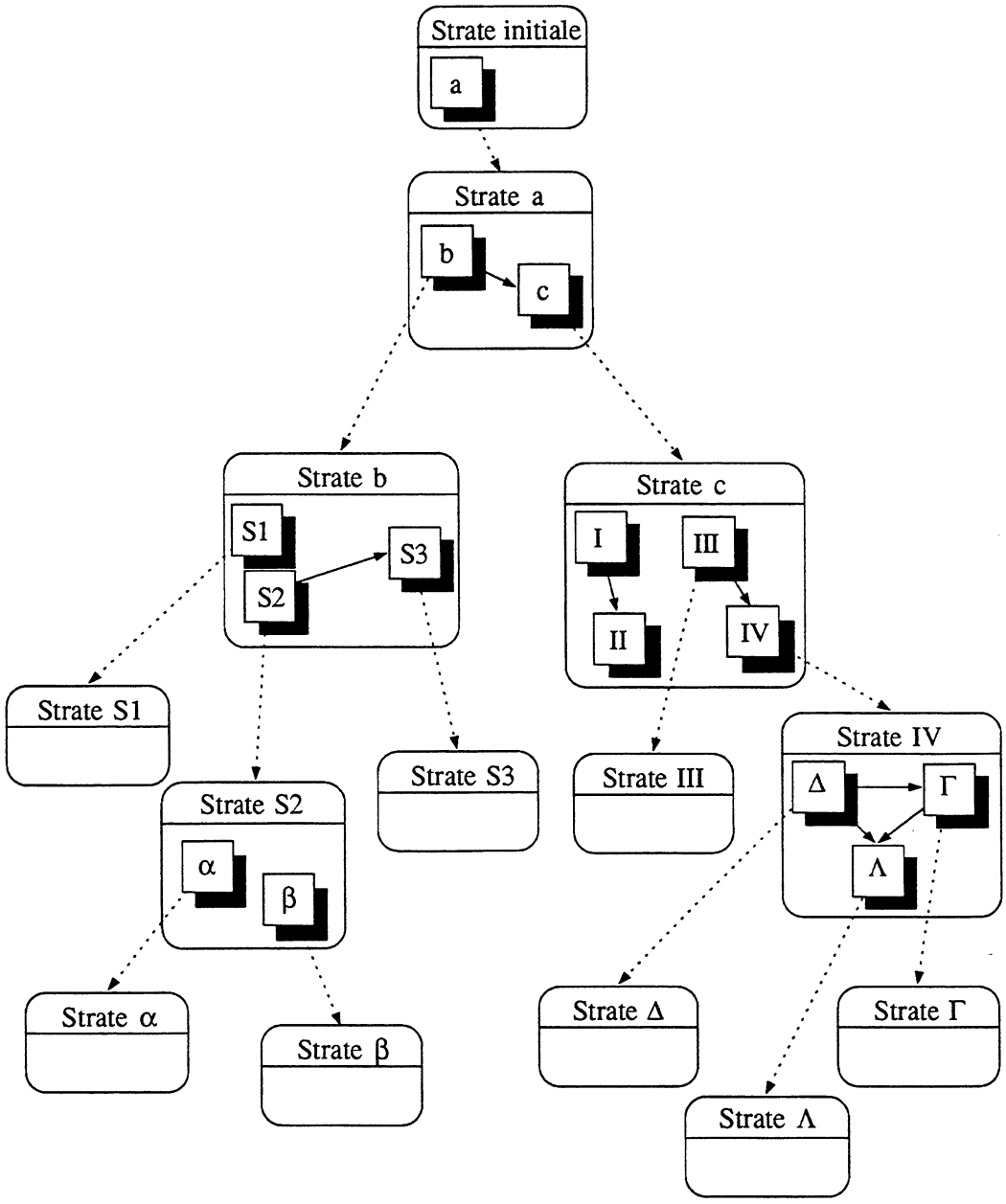


Fig. 5.22 : Exemple de résolution de problème

Le problème est, étant donné un ensemble de strates construites à partir de schémas de résolution, quel est le schéma de résolution ne concernant que les strates feuilles ? Si les langages cibles sont situés au niveau d'un langage de programmation, alors ce schéma nous permettra d'en déduire un algorithme. En revanche, si les langages cibles sont situés à un niveau mathématique, logique, alors ce schéma nous donnera résolution du problème initial.

Cependant, avant de proposer un algorithme permettant de déterminer un schéma ne concernant que les strates feuilles, il nous faut étudier le problème qui apparaît sur les liaisons entre strates. Nous l'abordons à la section qui suit, puis à la section V.4.3, nous proposons un algorithme qui détermine la projection d'un ensemble de strates.

V.4.2 Projection des liens

Nous considérons un exemple de projection simple afin d'illustrer le problème des liens entre les strates. Soit une strate A admettant comme schéma de résolution une séquence de deux éléments. Chaque élément renvoie vers une strate, les strates B et C. Ces dernières admettent de nouveau une décomposition par un schéma de résolution. La figure Fig. 5.23 montre une telle situation :

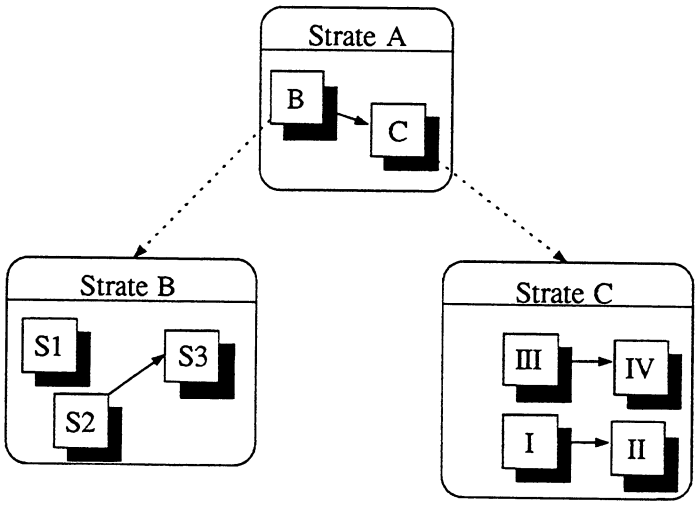


Fig. 5.23 : Exemple de strates à projeter

La projection de ces strates va consister à remplacer les éléments du schéma de la strate père A par les schémas des strates B et C. Nous aurons ainsi directement une séquence entre les strates B et C :

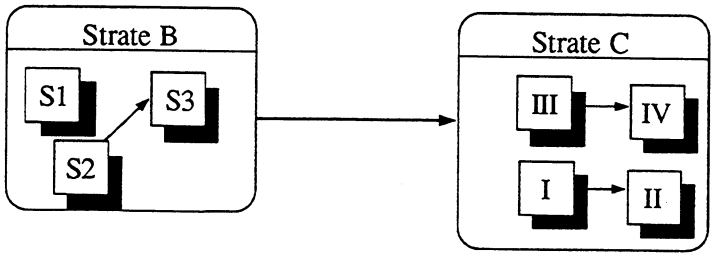


Fig. 5.24 : Projection de Fig. 5.23

Or, nous n'avons pas défini comment se relient les éléments des schémas des strates B et C entre eux.

Pour résoudre ceci, nous devons configurer les liens entre les différents éléments des deux schémas de résolutions afin de n'en former qu'un seul.

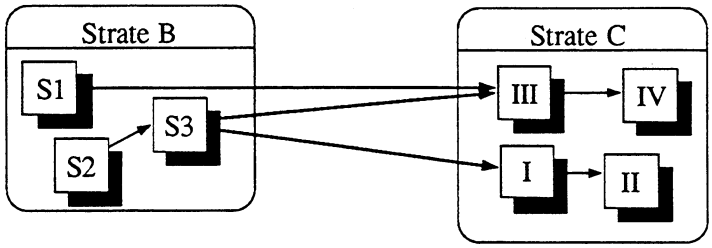


Fig. 5.25 : Configuration des liens entre les schémas de résolutions de Fig. 5.24

Nous avons une première méthode, en V.4.3, qui configure systématiquement les éléments en fonction des données et des résultats de chaque strate associée à un élément du schéma. Il se peut que nous ayons des ambiguïtés lorsque les deux schémas de résolution contiennent des constructions "choix". Une autre méthode envisagée est de définir ou de redéfinir les liens de façon interactive. Ceci permet à l'utilisateur de modifier un schéma de résolution. Mais, définir de cette manière les liens entre deux schémas devient fastidieux si nous avons des schémas comportant de nombreux éléments, ou si les strates à projeter sont en nombre important.

V.4.3 Algorithmes de configuration des liens

Nous proposons un algorithme de base qui configure de façon systématique les liens entre une séquence de deux strates en fonction des données et des résultats des strates des deux schémas de résolution des strates 1 et 2..

Algorithme de configuration de liens entre une séquence
de deux strates d'un schéma de résolution :
Entrées : Strate1, Strate2
Sorties : Projection

Schéma1 ← *PrendreSchémaDans* : Strate1
Schéma2 ← *PrendreSchémaDans* : Strate2
Projection ←*MettreEnParallèle* : Schéma1 *et* : Schéma2

Pour chaque strateSchéma1 **de** Schéma1 **faire**
Pour chaque strateSchéma2 **de** Schéma2 **faire**
Si (résultats **de** strateSchéma1) ∩ (données **de** strateSchéma2) ≠ ∅
alors
RajouterLienDans : Projection
entre : strateSchéma1 *et*: strateSchéma2
fin Si
fin Pour chaque

```
        fin Pour chaque

    Retourner Projection
```

Nous généralisons la configuration de liens pour toutes les strates d'un même schéma de résolution d'une strate père dans l'algorithme suivant :

```
Algorithme de configuration de liens entre les strates
du schéma de résolution d'une strate père :
Entrées : StratePère
Sorties : Projection

Schéma ← PrendreSchémaDans : StratePère

Pour chaque élément1 de Schéma faire
Pour chaque élément2 de Schéma faire
    Si élément2 ∈ Successeurs(élément1)
        alors
        Schéma1 ← PrendreSchémaDans : élément1
        Schéma2 ← PrendreSchémaDans : élément2
        Pour chaque Strate1 de Schéma1 faire
        Pour chaque Strate2 de Schéma2 faire
            Ajouter : Strate2 à : Projection
            ( Mettre à jour la projection en appliquant l'algorithme précé-
            dent de configuration de liens entre deux strates )
            fin Pour chaque
            fin Pour chaque
        fin Si
    fin Pour chaque
    fin Pour chaque

Retourner Projection
```

Et, enfin, l'algorithme de projection de toutes les strates effectue un parcours récursif de l'arborescence des strates pères. Pour cela il suffit de tester si une strate contient un schéma de résolution ou non, et de s'arrêter dès que nous avons parcouru toutes les strates. Le résultat nous donne la projection des strates feuilles sous forme d'un schéma de résolution.

Algorithme récursif de projection des strates
Entrées : StrateInitiale
Sorties : ProjectionGlobale

Schéma ← *PrendreSchémaDans* : StrateInitiale
Pour chaque Strate **de** Schéma **faire**
Si Strate *contient un schéma de résolution*
alors
Projection ← (obtenu par un appel récursif en passant comme argument Strate pour la state père)
Schéma ← *Remplacer* : Strate *par* : Projection
fin Si
ProjectionGlobale ← (Appliquer l'algorithme précédent de projection en passant comme strate père la StrateInitiale)
fin Pour chaque

Retourner ProjectionGlobale

Une autre variante serait d'effectuer des projections de manière descendante, ce qui nous conduirait au même résultat. Ces algorithmes de projection ont été implémenté sur une machine Tektronix dans un langage orienté–objet, Smalltalk–80, ainsi qu'une interface graphique associée permettant la manipulation de graphes orientés. Ils ont servi d'application pour la modélisation de carrières dans le cadre du projet européen Petrus [10], mais surtout initialement, pour servir de maquette pour tester et aboutir à ces définitions de strates.

V.5 Présentation visuelle des strates et exemples

Suite aux expérimentations menées sur une maquette réalisée dans un langage orienté–objet, Smalltalk–80, qui nous ont conduit à élaborer une représentation adéquate et définitive des strates, nous avons développé sous le logiciel Grif, éditeur de documents structurés, un éditeur interactif de strates. Ces dernières peuvent être insérées dans d'autres types de documents structurés de Grif, comme un article, un chapitre, etc. La section qui suit présente l'aspect des strates telles qu'elles apparaissent lorsqu'elles ne sont pas définies par l'utilisateur. Puis nous présentons un exemple à la section V.5.3.

V.5.1 Présentation graphique des strates

Pour les strates, nous avons deux vues, la vue intégrale, et une vue de type "tables des matières", que nous appellerons "table des strates" qui nous permet d'avoir un aperçu de l'ensemble des strates. Voici la présentation brute d'une vue vue intégrale d'une strate telle qu'elle apparaît sous le logiciel Grif – nous avons supprimé la strate numéro 2 qui est identique à la première :

1) [redacted]

Langage = [redacted]

Données : [redacted]

Résultats : [redacted]

Commentaire :

[redacted]

Descriptif :

Variables locales :

[redacted]

Préconditions :

[redacted]

Conditions :

[redacted]

Postconditions :

[redacted]

Schéma de résolution :

[redacted]

L'utilisateur peut rajouter de nouveaux champs qui précisent quelle est la strate père, quels sont les paramètres, quelle est l'argumentation qui nous a permis d'obtenir cette strate, quelles sont les équations de liens s'il s'agit d'une strate qui en traduit une autre. Ces différentes options s'insèrent dans la strate et ont la présentation suivante :

2) [redacted]

Langage = [redacted]

Strate père : [redacted]

Données : [redacted]

Résultats : [redacted]

Paramètres : [redacted]

Commentaire :

[redacted]

Arguments :

[redacted]

Equations de liens :

[redacted]

Descriptif :

Variables locales :

Préconditions :

Conditions :

Postconditions :

Schéma de résolution :

D'autre part, pour faciliter la distinction des différents niveaux de langages, nous avons regroupé les strates de même langage, lorsque cela est possible, entre elles. Ainsi, seule la strate initiale ou une strate de traduction précise le langage employé, les autres sous–entendent qu'elles sont décrites dans le même langage. Nous avons également la possibilité de supprimer des zones, ainsi en détruisant les descriptifs et les commentaires, nous avons la présentation suivante pour les strates de même langage qui suivent une strate initiale ou de traduction :

3)

Langage =

Données :

Résultats :

Schéma de résolution :

3.1)

Données :

Résultats :

Descriptif :

Conditions :

Schéma de résolution :

3.2) ▒

Données : ▒

Résultats : ▒

Commentaire :

▒

Schéma de résolution :

V.5.2 Objets de base des strates

Nous avons également défini des objets de base pour décrire les strates. Ainsi, Grif dispose d'une structure élémentaire qui est du texte, et un éditeur syntaxique de formules mathématiques, "Math". Nous avons développé à la suite de recherches menées par Bouhier [8], sur le problème de la génération de la sémantique d'une formule de mathématiques, un autre éditeur de formules, "GramMath", mais cette fois–ci, orienté par la sémantique des formules. Cet éditeur, décrit en [21] et dans l'annexe B, présente l'avantage d'éditer des formules qui peuvent être évaluées par un système de calcul formel ou numérique, puis ensuite de récupérer sous un document Grif, les résultats graphiques. Nous donnons en annexe B les possibilités de cet éditeur ainsi que des exemples d'évaluations.

Les derniers objets de base concernent les variables pour définir les données, résultats, paramètres, mais également les variables locales. Ils concernent aussi les types pour définir dans le descriptif une forme particulière de condition qui porte sur le type, le domaine d'une ou de plusieurs variables.

V.5.3 Exemple de strates

Afin d'illustrer les strates, nous avons choisi un exemple sur l'intégration d'une fonction réelle à une variable. Un autre exemple est présenté en annexe <AnnexeC>.

1) Strate initiale

Langage = *Mathématiques*

Données : f, a, b

Résultats : I

Commentaire :

Étant donné une fonction réelle f à une variable définie sur l'intervalle [a,b], nous désirons intégrer cette fonction sur cet intervalle.

Descriptif :

Variables locales :

x

Préconditions :

$f : IR \rightarrow IR$

$a, b \in IR / a \leq b$

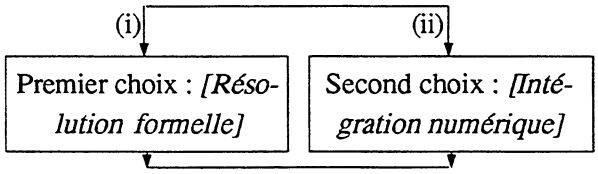
Conditions :

$$I = \int_a^b f(x)\,dx$$

Postconditions :

$I \in IR$

Schéma de résolution :



1.1) Intégration numérique

Strate père : *[Strate initiale]*

Données : f, a, b

Résultats : I

Commentaire :

Le résultat S obtenu sera une approximation de la valeur I recherchée. Nous pouvons dans certains cas obtenir S = I. Dans tous les cas, nous définirons la valeur de I comme étant celle déterminée par S.

Descriptif :

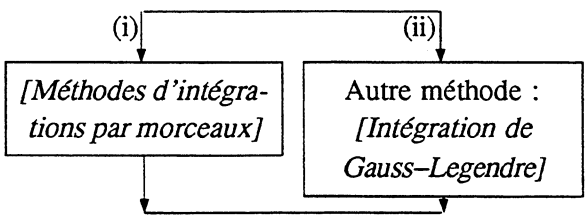
Variables locales :

S

Conditions :

$I \leftarrow S$

Schéma de résolution :



1.2) Intégration par morceaux

Données : f, a, b

Résultats : S

Paramètres : N

Commentaire :

Le paramètre N désigne le nombre d'intervalles sur lesquels nous intégrerons par morceaux.

Arguments :

Relation de Chasles généralisé appliqué aux intégrales.

Descriptif :

Variables locales :

i, a_i, b_i

Préconditions :

$a_1 = a$

$a_i < b_i$, $\forall i \in \{1 \dots N\}$

$b_N = b$

Conditions :

$$S = \sum_{i=1}^{N} \int_{a_i}^{b_i} f(x)\,dx$$

Schéma de résolution :

[Méthodes d'intégrations par morceaux]

1.3) Intégration de Gauss–Legendre

Données : f, b, a , N

Résultats : S

Commentaire :

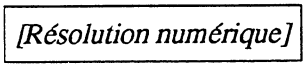
Si f est un polynôme de degré inférieur à 2N, alors cette méthode donne théoriquement une valeur exacte. Dans la pratique, il se peut qu'il y a des erreurs d'arrondis infimes.

Descriptif :

Préconditions :

$N \in 2\mathbb{N}^*$

Schéma de résolution :



2) Méthodes d'intégrations par morceaux

Langage = *Algorithmique*

Strate père : *[Intégration par morceaux]*

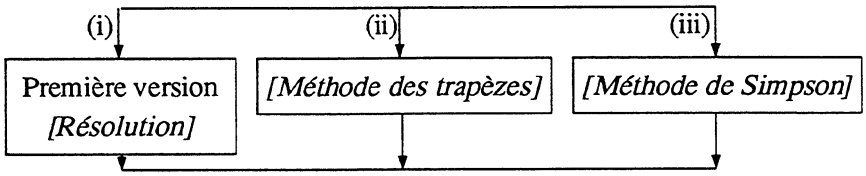
Données : *f*, a, b, N

Résultats : S

Commentaire :

L'utilisateur écrit sa méthode en deux temps : la première en faisant une décomposition de tous les sous–problèmes, le choix (i), puis une deuxième méthode en tenant compte de toutes les simplifications possibles en recomposant les sous–problèmes entre eux, le choix (ii).

Schéma de résolution :



2.1) Résolution

Strate père : *[Méthodes d'intégrations par morceaux]*

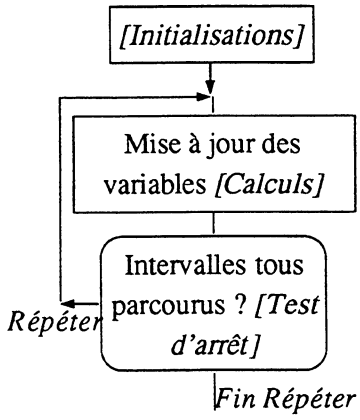
Données : *f*, a, b, N

Résultats : S

Commentaire :

Algorithme général. Les différentes variantes, donneront lieu à la méthode des trapèzes, de Simpson, et ainsi qu'une méthode générale d'intégration numérique.

Schéma de résolution :



2.2) Initialisations

Strate père : *[Résolution]*

Résultats : S, i

Commentaire :

Cette sous–strate est indépendante des données du problème.

Descriptif :

Préconditions :

S :: Réel

i :: Entier naturel

Conditions :

$S \leftarrow 0.0$

$i \leftarrow 0$

2.3) Calculs

Strate père : *[Résolution]*

Données : S, i, Aire

Résultats : S, i

Commentaire :

Nous calculons de nouvelles valeurs pour les variables S et *i*. Pour cela, à partir de leurs anciennes valeurs, nous les mettons à jour par effet de bord, ce qui explique pourquoi nous les retrouvons aussi bien dans les données que dans les résultats.

Le schéma propose un choix pour les intervalles et dans le calcul des aires. Pour les intervalles, seul celui des intervalles équidistants est précisé. Par contre, le choix dans le calcul des aires permet d'obtenir deux méthodes : celle des trapèzes, et celle de Simpson.

Remarque: ce schéma possède à la fois des renvois à des sous–strates, et des instructions algorithmiques.

Descriptif :

Variables locales :

a_i, b_i

Schéma de résolution :

choix (i) :
| *[Intervalles équidistants]*
choix (ii) :
| Intervalles non équidistants (comment les construire ?)

choix (i) :
| *[Aire d'un trapèze]*
choix (ii) :
| *[Aire selon Simpson]*
choix (iii) :
| *[Aire selon une interpolation]*

$S \leftarrow S + Aire$

$i \leftarrow i + 1$

2.4) Test d'arrêt

Strate père : *[Résolution]*

Données : i, N

Résultats : B

Commentaire :

Les sous-strates servant à un test doivent retourner un et un seul résultat de type booléen, ici B.

Schéma de résolution :

$B \leftarrow i \geq N$

2.5) Intervalles équidistants

Strate père : *[Calculs]*

Données : a, b, N, i

Résultats : a, b

Schéma de résolution :

$$h \leftarrow \frac{b-a}{N}$$

$a[i] \leftarrow a + (i-1)\,h$

$b[i] \leftarrow a[i] + h$

2.6) Aire d'un trapèze

Strate père : *[Calculs]*

Données : f, a[i], b[i]

Résultats : Aire

Commentaire :

L'aire d'un trapèze est donné par : $\frac{f(a[i]) + f(b[i])}{2}(b[i] - a[i])$

Schéma de résolution :

$$Aire \leftarrow \frac{f(a[i]) + f(b[i])}{2}\left(b[i] - a[i]\right)$$

2.7) Aire selon Simpson

Strate père : *[Calculs]*

Données : f, a[i], b[i]

Résultats : Aire

Commentaire :

Sur chaque intervalle, nous effectuons le calcul suivant :

$$\frac{f(a[i]) + 4f\left(\frac{a[i] + b[i]}{2}\right) + f(b[i])}{6}$$

Arguments :

Intégration de l'interpolation des points (a[i],f(a[i])) , (c[i],f(ci[i]) et (b[i],f(b[i])), où c[i] désigne le milieu, dans notre cas, de a[i] et de b[i].

Schéma de résolution :

$$Aire \leftarrow \frac{f(a[i]) + 4f\left(\frac{a[i] + b[i]}{2}\right) + f(b[i])}{6}$$

2.8) Aire selon une interpolation

Strate père : *[Calculs]*

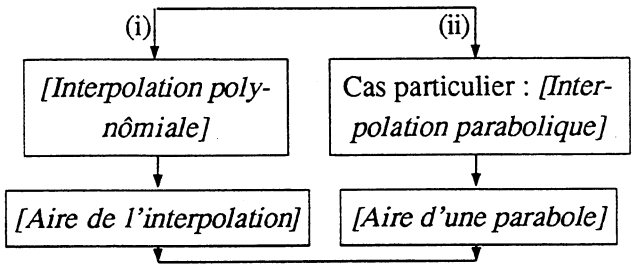
Données : f, a[i], b[i]

Résultats : Aire

Commentaire :

Soit une interpolation de la fonction f sur l'intervalle $[a_i, b_i]$

Schéma de résolution :



2.9) Interpolation polynômiale

Strate père : *[Aire selon une interpolation]*

Données : *f*, a[i], b[i]

Résultats : P

Paramètres : d

Commentaire :

Nous interpolons une fonction *f* sur l'intervalle $[a_i, b_i]$ à l'aide d'un polynôme de degré au plus *d*. Pour cela nous calculons *d* valeurs distinctes pour la fonction *f*.

Descriptif :

Variables locales :

k

Préconditions :

a[i] < b[i]

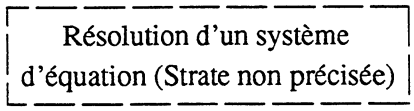
Conditions :

$P(c_k) = f(c_k)$, avec k compris entre 0 et d.

Postconditions :

$$P(x) = \sum_{i=0}^{d} x^i p_i$$

Schéma de résolution :



2.10) Aire de l'interpolation

Strate père : *[Aire selon une interpolation]*

Données : P, a[i], b[i]

Résultats : Aire

Commentaire :

Sachant intégrer un polynôme P, nous obtenons directement la valeur de l'intégrale de P sur le domaine [a[i] , b[i]].

Descriptif :

Préconditions :

$$P(x) = \sum_{i=0}^{d} x^i p_i$$

Conditions :

$$\text{Aire} = \sum_{j=0}^{d} p_j \int_{a[i]}^{b[i]} x^j = \sum_{j=0}^{d} p_j \frac{b[i]^{j+1} - a[i]^{j+1}}{j+1}$$

Schéma de résolution :

$Aire \leftarrow 0$

$j \leftarrow 0$

tant que j < d

faire

$j \leftarrow j + 1$

$Aire \leftarrow \text{Aire} + p[j] \frac{b[i]^j - a[i]^j}{j}$

2.11) Interpolation parabolique

Strate père : *[Aire selon une interpolation]*

Données : f, a_i, b_i

Résultats : P

Commentaire :

Par rapport à l'interpolation décrite en *(2.9)*, strate intitulée *[Interpolation polynômiale]*, le degré *d* est fixé et choisi à 2, et le point c[i] de [a[i] ,b[i]] est choisi comme étant le milieu. Le schéma de résolution est fourni après avoir réalisé des calculs intermédiaires sous Reduce, calculs explicités à la strate *(5.5) [Session Reduce pour l'interpolation parabolique]*.

Descriptif :

Variables locales :

p, q, r, h, c[i]

Préconditions :

$P(x) \leftarrow p x^2 + q x + r$

Conditions :

$h \leftarrow b[i] - a[i]$

$$c[i] \leftarrow \frac{a[i] + b[i]}{2}$$

$$\left\{ \begin{array}{l} P(ai) = f(ai) \\ P(ci) = f(ci) \\ P(bi) = f(bi) \end{array} \right\}$$

Schéma de résolution :

$$p \leftarrow \frac{2f(a[i]) + 2f(b[i]) - 4f(c[i])}{h^2}$$

$$q \leftarrow -2p + \frac{-3f(a[i]) - f(b[i]) + 4f(c[i])}{h}$$

$$r \leftarrow a[i]^2 p + a[i](-2p - q) + f(a[i])$$

2.12) Aire d'une parabole

Strate père : *[Interpolation polynômiale]*

Données : P, a[i], b[i]

Résultats : Aire

Commentaire :

Nous appliquons le calcul effectué en *(2.10)*, strate intitulée *[Aire de l'interpolation]* pour le cas d = 2.

Schéma de résolution :

$$Aire \leftarrow p\frac{b[i]^2 - a[i]^2}{2} + q\left(b[i] - a[i]\right)$$

3) Méthode des trapèzes

Langage = *Algorithmique*

Strate père : *[Résolution]*

Données : f, a[ ], b[ ], N

Résultats : S

Commentaire :

La première méthode des trapèzes effectue des calculs redondants d'un intervalle à un autre. En effet, sur deux intervalles consécutifs, nous aurons à calculer une somme de deux aires :

$$(b[i]-a[i])\frac{f(a[i])+f(b[i])}{2}+(c[i]-b[i])\frac{f(b[i])+f(c[i])}{2}$$

Cette somme se simplifie :

$$(b[i]-a[i])\frac{f(a[i])}{2}+(c[i]-a[i])f(b[i])+(c[i]-b[i])\frac{f(c[i])}{2}$$

En généralisant à N intervalles, nous effectuons deux fois moins d'évaluations de la fonction *f* que dans la version des trapèzes proposée en *(2.1)*, la strate *[Résolution]*.

Schéma de résolution :

[Résolution des trapèzes améliorée]

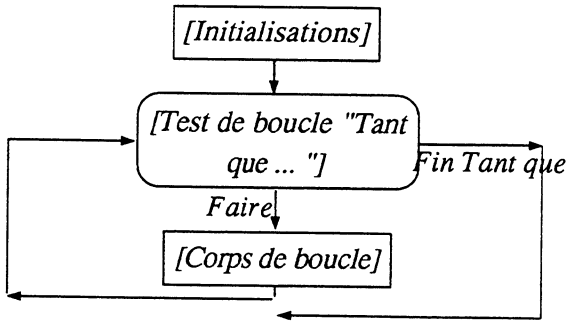
3.1) Résolution des trapèzes améliorée

Strate père : *[Méthode des trapèzes]*

Données : *f*, a, b, N

Résultats : S

Schéma de résolution :



3.2) Initialisations

Strate père : *[Résolution des trapèzes améliorée]*

Données : *f*, a, b, N

Résultats : S, i, h

Commentaire :

La mesure des intervalles équidistants est h. Nous initialisons S sur les bords de l'intervalle [a , b]. Nous pouvons vérifier que dans le cas extrême où N vaut 1, nous avons la formule de la moyenne.

Schéma de résolution :

$i \leftarrow 1$

$h \leftarrow \frac{b-a}{N}$

$S \leftarrow h \frac{f(a)+f(b)}{2}$

3.3) Test de boucle "Tant que ... "

Strate père : *[Résolution des trapèzes améliorée]*

Données : i, N

Résultats : B

Commentaire :

Ce n'est pas tout à fait le même test d'arrêt de boucle qu'en *(2.4)*, car nous avons une boucle "tant que ..." d'une part, et d'autre part nous avons N – 2 itérations du fait de cette méthode.

Schéma de résolution :

$B \leftarrow i < N-1$

3.4) Corps de boucle

Strate père : *[Résolution des trapèzes améliorée]*

Données : *f*, a, i, S, h

Résultats : i, S

Commentaire :

Le calcul itératif ne porte que sur les points intérieurs à l'intervalle [a , b].

Schéma de résolution :

$S \leftarrow S + 2h\,f(a + ih)$

$i \leftarrow i + 1$

4) Méthode de Simpson

Langage = *Algorithmique*

Données : f, a, b, N

Résultats : S

Commentaire :

La stratégie de résolution (fondée sur la même simplification) est calquée sur celle des trapèzes définie dans la strate . En effet :

$$S = \sum_{i=1}^{N-1} \left(h \frac{f(a\,i) + 4\,f(a\,i + h/2) + f(a\,i + h)}{6} \right)$$

La formule précédente se réécrit comme étant :

$$S = h\frac{f(a) + f(b)}{6} + h\frac{2}{3}\sum_{i=1}^{N-1} f(a\,i + h/2) + h\frac{2}{6}\sum_{i=2}^{N-1} f(a\,i)$$

Schéma de résolution :

[Résolution de Simpson]

4.1) Résolution de Simpson

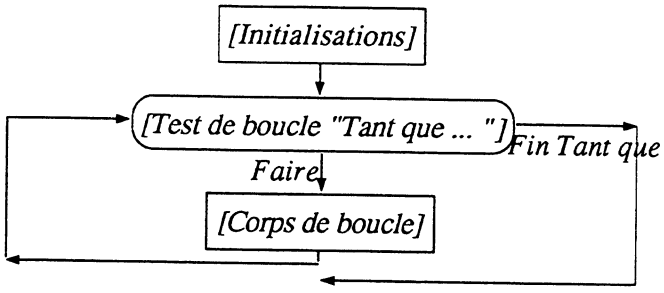
Données : f, a, b, N

Résultats : S

Commentaire :

Nous copions le schéma de résolution de la sous–strate en modifiant les références de sous–strates des éléments du schéma.

Schéma de résolution :



4.2) Initialisations

Strate père : *[Résolution des trapèzes améliorée]*

Données : f, a, b, N

Résultats : S, i, h

Commentaire :

C'est une copie de la strate *[Initialisations]* de *(3)*, seuls le calcul de S et l'initialisation de *i* diffèrent :

La mesure des intervalles équidistants est h.Nous initialisons S sur les bords de l'intervalle [a,b]. Nous pouvons vérifier que dans le cas extrême où N vaut 1, nous avons la formule de la moyenne.

Schéma de résolution :

$$i \leftarrow 2$$

$$h \leftarrow \frac{b-a}{N}$$

$$S \leftarrow h\frac{f(a)+f(b)}{2} + h\frac{2}{3} f\left(a+\frac{h}{2}\right)$$

4.3) Test de boucle "Tant que ... "

Strate père : *[Résolution des trapèzes améliorée]*

Données : i, N

Résultats : B

Commentaire :

Ce n'est pas tout à fait le même test d'arrêt de boucle qu'en *(2.4)*, car nous avons une boucle "tant que ..." d'une part, et d'autre part nous avons N – 2 itérations du fait de cette méthode.

Schéma de résolution :

$$B \leftarrow i<N-1$$

4.4) Corps de boucle

Strate père : *[Résolution des trapèzes améliorée]*

Données : *f*, a, i, S, h

Résultats : i, S

Commentaire :

C'est une copie de la strate *[Corps de boucle]* de *(3)* où le calcul de S est modifié

Schéma de résolution :

$$S \leftarrow S+ h\frac{2}{6} f(a+ih) + h\frac{2}{3} f\left(a+\frac{h}{2}\right)$$

$$i \leftarrow i+1$$

5) Résolution formelle

Langage = *Reduce*

Strate père : *[Strate initiale]*

Données : *f*, a, b

Résultats : I

Commentaire :

Cette strate est la traduction des données de la strate précédente. Le problème lui–même est traduit dans ce nouveau langage au sein de la strate *(5.1)*.

Equations de liens :

$F \leftarrow f$

$A \leftarrow a$

$B \leftarrow b$

$RES \rightarrow I$

Schéma de résolution :

[Intégration algébrique]

5.1) Intégration algébrique

Données : F, A, B

Résultats : RES

Commentaire :

Le système de calcul formel Reduce propose deux méthodes d'intégration formelle dont l'une, celle ayant recours au module ALGINT, de Davenport.

Descriptif :

Variables locales :

P

Conditions :

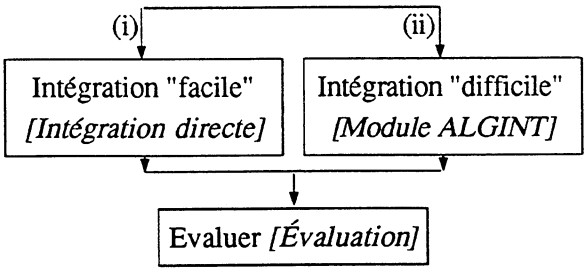
$P(x) \rightarrow \int F(x)\ dx$

$RES \leftarrow F(B) - F(A)$

Postconditions :

numberp RES

Schéma de résolution :



5.2) Intégration directe

Strate père : *[Intégration algébrique]*

Données : F

Résultats : P

Descriptif :

Conditions :

$P(x) \rightarrow \int F\, dx$

Schéma de résolution :

P := INT (F , X);

5.3) Module ALGINT

Strate père : *[Intégration algébrique]*

Données : F

Résultats : P

Descriptif :

Conditions :

$P(x) \rightarrow \int F\, dx$

Schéma de résolution :

% Activation du module ALGINT
on ALGINT;
P := INT(F , X);

5.4) Évaluation

Strate père : *[Intégration algébrique]*

Données : P, A, B

Résultats : I

Commentaire :

Où l'on se rend compte que l'on peut tester ici, si P contient oui ou non une intégrale indéfinie. Ceci permet d'éviter d'avoir à définir un test de présence d'une ou plusieurs intégrales dans l'expression algébrique fournie par Reduce. En cas d'échec, ces dernières pourraient faire l'objet d'une intégration numérique.

Descriptif :

Préconditions :

numberp A

numberp B

Conditions :

$I \leftarrow P(B) - P(A)$

Postconditions :

numberp I

5.5) Session Reduce pour l'interpolation parabolique

Strate père : *[Interpolation parabolique]*

Données : a, b, c

Résultats : p, q, r

Commentaire :

Ceci illustre une phase de calculs intermédiaire lors de l'écriture d'une strate, qui est ensuite escamotée pour laisser place à la strate précédente.

Equations de liens :

$a \leftarrow a[i]$

$b \leftarrow b[i]$

$c \leftarrow c[i]$

$\{ p , q , r \} \rightarrow P(x)$

Descriptif :

Variables locales :

P, h, fa, fb, fc

Schéma de résolution :

```
% Définition du polynôme P
for all x let P(x) = p * x**2 + q*x + r;

% Résolution d'un système linéaire
res := solve ( {P(a) = fa , P(b) = fb , P(c) = fc }, { p , q , r } );

% Modification des variables a, b, c pour simplifier le résultat "res"
b := a + h;
c := (a+b)/2;
res;
```

6) Résolution numérique

Langage = *C*

Strate père : *[Intégration de Gauss–Legendre]*

Données : *f*, a, b, N

Résultats : S

Schéma de résolution :

[Procédure de la bibliothèque scientifique ESSL]

6.1) Procédure de la bibliothèque scientifique ESSL

Données : *f*, a, b, N

Résultats : S

Commentaire :

Au moyen de l'interface avec la bibliothèque de calcul scientifique, ESSL, et d'un programme développé en C, les intégrales sont calculées par la méthode de Gauss–Laguerre.

Schéma de résolution :

$$S \leftarrow \int_a^b f(x)\,dx$$

Chapitre VI

Représentation logique et algébrique des strates

Suite au chapitre précédent où nous avons introduit les strates, nous étudions dans les sections qui suivent, des outils algébriques qui vont nous permettre de vérifier la cohérence des strates. A toute strate, nous associons une interprétation, définie à la section VI.1, qui est une représentation logique sur un corps fini de la strate, logique associée à un langage de description de strate. Ceci nous permet de vérifier la cohérence d'une strate ainsi que la cohérence des décompositions, ou des recompositions, des strates. L'interprétation des strates est à distinguer de la sémantique des strates, qui est le sens ou la représentation que nous attribuons à la formulation d'un problème mathématique ou de modélisation. Or, à toute formulation logique sur $\mathbf{Z}/p\mathbf{Z}$, nous avons vu que nous pouvions associer un polynôme sur $\mathbf{Z}/p\mathbf{Z}$, aussi pouvons–nous avoir une représentation algébrique des strates. Les questions de vérification de la cohérence d'une strate ou d'une décomposition en une autre strate, sont des questions qui sont formulées au niveau algébrique.

Aussi, nous nous intéressons dans un premier temps à l'aspect algébrique, avant d'aborder l'aspect logique des strates. Auparavant, nous définissons la résolution de polynômes en « x puissance d », puis ce qu'est un système polynômial bien formé à la section III.5. Ces définitions vont nous permettre de vérifier si une strate est cohérente en–elle même, c'est–à–dire si ses spécifications ne sont pas contradictoires ou inconsistantes par rapport à la logique associée au langage des spécifications.

Puis, aux sections VI.2 et VI.3, nous arbordons la vérification des décompositions d'une strate en une autre. Ceci revient à étudier au niveau algébrique un problème de déduction : quand avons–nous un polynôme qui peut–être déduit d'un système polynômial ? Une méthode, la méthode de Wu [45] permet d'y répondre dans certains cas. Nous

proposons d'autres approches, par des méthodes, plus coûteuses, mais qui permettent de répondre à ce problème lorsque le corps est fini.

Enfin, nous proposons une méthodologie qui permet de vérifier l'ensemble de toutes les strates ainsi que les décompositions entre elles sans avoir à considérer un système polynômial représentant toutes les strates et toutes les décompositions. Pour cela nous introduisons à la section VI.4 une propriété sur la vérification de la cohérence mais à un niveau local seulement. Les algorithmes qui suivent sont basés sur ce principe de vérification locale afin de vérifier l'ensemble d'une résolution, et ce en considérant seulement les systèmes polynômiaux des strates concernées, sans se préoccuper des autres. Ceci nous permet de traiter des systèmes polynômiaux de taille bien inférieure que l'ensemble de tous les systèmes.

Nous terminons ce chapitre en présentant quelques exemples de logiques que nous pouvons associer à des langages de programmation. Ce sont aussi bien des langages de programmation classiques que des langages temps réel comme Signal. A la section VI.5.2, nous abordons le problème de la dynamique des algorithmes représentés par un système algébrique statique. Pour le résoudre, nous proposons la notion de superviseur, ce qui nous permet de décrire des algorithmes séquentiels à partir d'un formalisme développé initialement pour des algorithmes temps réel synchronisés.

VI.1 Strates et interprétations

A toute strate, nous y associons une interprétation. Cette dernière permet de vérifier la cohérence d'une strate ainsi que la cohérence des décompositions ou des recompositions des strates. L'interprétation des strates est à distinguer de la sémantique des strates, qui est le sens ou la représentation que nous attribuons à la formulation d'un problème mathématique ou de modélisation. Nous rappelons dans les sections qui suivent une notion relative aux formules logiques du premier ordre qui est l'interprétation de formules logiques sur un domaine D de valeurs. Nous redonnons les définitions de VI.1.1 et VI.1.2, extraites de [20].

VI.1.1 Interprétation d'une formule logique (rappels)

Définition VI.1 :

Soient :

- une formule logique G bien formulée, définie en I.2.1,
- un choix d'un domaine D non vide,
- l'assignation à chaque constante de G d'un élément de D,
- les fonctions d'arité *n* présentes dans G sont des applications de D^n dans D,
- les prédicats d'arité *n* présentes dans G sont des applications de D^n dans { Vrai, Faux },

- les propositions de G prennent une valeur { Vrai, Faux}

Nous avons alors une interprétation I de G sur un domaine D.

Définition VI.2 :

Une interprétation I d'une formule logique G sur un domaine D est soit vraie, soit fausse, selon le choix des valeurs dans D pour les variables de G. Nous dirons que nous avons une valeur vraie ou fausse pour G selon I pour le choix des valeurs des variables.

VI.1.2 Validité et consistance (rappels)

Définition VI.3 :

Une interprétation d'une formule logique est valide si et seulement si pour toute interprétation sur un domaine D sa valeur est vraie, sinon elle est invalide. Nous employerons aussi le terme de tautologie au lieu de valide.

Définition VI.4 :

Une interprétation d'une formule logique est inconsistante si et seulement si pour toute interprétation sur un domaine D sa valeur est fausse, sinon, elle est inconsistante. Nous dirons ausi que c'est une contradiction au lieu d'inconsistance.

VI.1.3 Interprétation d'une strate

Définition VI.5 :

Nous dirons que l'interprétation associée à une strate est l'interprétation sur D de l'ensemble des formules logiques associées à l'ensemble des variables de la strate, pour un choix de valeurs pour les variables représentant les données et les paramètres de la strate.

Nous retrouvons les définitions précédentes portant sur la validité et sur la consistance, d'une interprétation d'une strate.

Définition VI.6 :

Une interprétation d'une strate est valide si et seulement si pour toute interprétation de la strate sur un domaine D sa valeur est vraie, sinon, elle est invalide.

Définition VI.7 :

Une interprétation d'une strate est inconsistante si et seulement si pour toute interprétation de la strate sur un domaine D sa valeur est fausse, sinon, elle est inconsistante.

VI.1.4 Cas des formules logiques sur $\mathbf{Z}/p\mathbf{Z}$

Pour les strates, nous leur associons un ensemble de formules logiques sur $\mathbf{Z}/p\mathbf{Z}$. Aussi, avons–nous la particularité d'avoir le domaine D inclus dans $\mathbf{Z}/p\mathbf{Z}$. Comme nous pouvons de plus associer à toute formule logique sur $\mathbf{Z}/p\mathbf{Z}$ une représentation algébrique, nous pouvons aisément vérifier si une formule logique sur $\mathbf{Z}/p\mathbf{Z}$ est valide ou non, consistante ou non sur un domaine D. Ceci nous permet ainsi de préciser quels sont les domaines D sur lesquels l'interprétation d'une formule ou d'un système de formules logiques représentant une strate est vérifié ou non.

VI.1.5 Strates et systèmes bien formés

Nous avons proposé auparavant une définition sur un polynôme ou un système polynômial bien formé, c'est–à–dire général en un choix de variables et selon une résolution donnée sur les variables, une résolution en « x puissance d ». Par rapport aux interprétations, cela signifie qu'il existe un domaine D_k pour chaque variable v_k sur lequel les formules logiques sont valides. Lorsque tous les domaines D_k sont égaux à un même domaine D, nous retrouvons les définitions précédentes des interprétations sur un domaine D. L'autre différence réside dans les résolutions en « x puissance d ». Dans ce cas nous ne sommes pas intéressés par la ou les valeurs possibles pour une variable x, mais pour la valeur de l'expression x^d. Ainsi, écrivons–nous que x^d appartient à D. Soit ici, $v_k^{d_k}$ appartient à D_k.

Nous avons vu que nous pouvions caractériser les systèmes bien formés par une condition portant seulement sur un choix de variables. Dans le cas des strates, nous choisissons ces variables comme étant les données et les paramètres. Nous justifions ce choix du fait que les résultats, ainsi que les variables locales, dépendent des données et des paramètres.
En revanche, le choix d'une résolution dépendra de la sémantique attribuée à chacune des variables. Ce peut–être le type d'une variable: booléen ou non, ou bien un état portant sur une variable: absent, présent, non mis à jour, ou sur le traitement des cas exceptionnels: nombre trop grand, nombre trop petit, etc. Mais en toute généralité nous verrons que la sémantique dépend fortement du langage dans lequel la variable évolue. Ce peut être aussi bien un langage de programmation classique ou non, comme les langages temps réel, qu'un langage de spécifications, ou de manière plus générale, un langage de très haut niveau comme un langage naturel. Nous étudions quelques langages à la section VI.5.

VI.2 Strates et décompositions

Nous avons vu qu'à toute strate nous associons un schéma de résolution, sauf lorsque nous sommes au niveau d'une strate qui est décrite dans un langage de programmation. Ce schéma, lorsqu'il ne comporte qu'un seul élément qui réfère à une autre strate, une sous–strate, représente la décomposition d'une strate en une autre strate. Nous voulons savoir si une telle décomposition est cohérente, cohérence dans le sens suivant : avons–nous

l'interprétation de la sous–strate contenue dans celle dela strate, en supposant que chaque strate est a fortiori cohérente. Cette question revient à étudier l'implication entre deux systèmes algèbriques qui représentent chacun une strate. Cette étude fait l'objet de cette section.

VI.2.1 Formulation du problème

Nous supposerons que nous avons choisi de façon correcte les paramètres $\{a_j\}_{j=1...q}$, et que le reste des variables $\{x_k\}_{k=1...m}$ est algébriquement dépendant de ces paramètres à travers les polynômes $\{P_i\}_{i=1...r}$ de $\mathbf{K}[x_1, ..., x_m, a_1, ..., a_q]$, $\mathbf{K}$ un corps. Soient donc :

- un système (S) polynômial $\{P_i\}_{i=1...r}$,
- un polynôme $Q(x_1, ..., x_m, a_1, ..., a_q)$.

Le problème est de décider si le polynôme Q peut se déduire du système (S) des polynômes $\{P_i\}_{i=1...r}$, ou s'ils sont équivalents.

Nous avons un problème similaire lorssque nous remplaçons le polynôme Q par un système polynômial (S'). Il suffit dans ce cas de déterminer le polynôme Q équivalent au système (S'), puis de considérer la formulation du problème précédent. Les méthodes que nous proposons en VI.2.2 ne perdent en rien de leur généralité.

VI.2.1.1 Cas des corps infinis

Pour les corps de caractéristique nulle, la méthode de Wu [45], [46], inspirée de l'algorithme de Ritt [40], permet de décider en un nombre fini de pas si un polynôme Q peut être déduit d'un système polynômial. Cependant, cette méthode est appliquée à des problèmes de géométrie – géométries métriques – où tout polynôme représente une propriété géométrique particulière, et l'ordre des polynômes et des variables est donné par l'ordre de la construction. Actuellement, les recherches de Chou [12], [13] et Gao [14] sur la méthode de Wu portent sur la démonstration de théorèmes de géométrie faisant inclure des relations d'ordre, et ce à l'aide de règles de réécriture ou des bases de Gröbner. D'après les auteurs, la méthode de Wu, est peu coûteuse en mémoire, efficace en temps de calculs, et permet de démontrer de nombreux théorèmes de géométrie, [12] et [14].

VI.2.1.2 Cas des corps finis

Nous aurions pu adapter la méthode de Wu au cas des corps finis $\mathbf{Z}/p\mathbf{Z}$, mais cela semble délicat pour les raisons suivantes :

- il faut donner un ordonnancement sur les hypothèses, les polynômes $\{P_i\}_{i=1...r}$,
- il faut choisir un ordre d'élimination sur les variables $\{x_k\}_{k=1...m}$.

Ces choix, dans le cas des corps infinis, sont guidés, par exemple, par l'ordre des constructions géométriques effectuées. Ainsi, dans l'exemple de la démonstration de la droite d'Euler pour un triangle (A,B,C), si nous changeons l'ordre des hypothèses, ou des variables, nous

constatons[1] que la méthode de Wu est mise en échec. De même, le « théorème de Von Aubler » concernant les carrés construits sur les côtés d'un quadrilatère convexe quelconque, n'est pas résolu par la méthode de Wu lorsque nous mélangeons l'ordre des hypothèses du système polynômial.

Une autre méthode basée sur l'algorithme d'élimination des variables décrit au chapitre précédent peut s'adapter à ce problème. En effet, dans un corps $\mathbf{Z}/p\mathbf{Z}$ nous pouvons déterminer les zéros du système polynômial, puis vérifier si ces zéros annulent bien le polynôme Q. Bien que coûteuse, cette méthode présente aussi l'avantage de vérifier en même temps si le système polynômial est bien formulé.

VI.2.2 Formulation des méthodes

VI.2.2.1 Première méthode

Une première méthode est de vérifier si le système (S) est général en $\{a_j\}_{j=1...q}$ pour $\left\{ a_j^{d_j} \right\}_{j=1...q}$ et pour $\left\{ x_k^{d_k} \right\}_{k=1...m}$, puis de rechercher toutes les racines du système et vérifier si elles annulent bien le polynôme Q.

Ce qui revient à déterminer la condition d'existence des racines du système (S) en fonction des paramètres $\{a_j\}_{j=1...q}$ et du choix d'une résolution en $\left\{ a_j^{d_j} \right\}_{j=1...q}$ et en $\left\{ x_k^{d_k} \right\}_{k=1...m}$, que nous noterons par $C_{(S)}(a_1, \ldots, a_q)=0$. Puis à rechercher les racines de (S) par une résolution en $\left\{ x_k^{d_k} \right\}_{k=1...m}$ afin de les substituer dans le polynôme $Q(x_1, \ldots, x_m, a_1, \ldots, a_q)$.

Nous avons vu au chapitre II que le coût est en $O(p^{m+q})$ pour déterminer les conditions d'existence sur les paramètres $\{a_j\}_{j=1...q}$ et pour déterminer les racines, avant de vérifier que $Q(x_1, \ldots, x_m, a_1, \ldots, a_q)$ s'annule.

VI.2.2.2 Deuxième méthode

Une autre méthode, serait de vérifier au préalable, avant de calculer les racines, si l'implication entre le système (S) et le polynôme Q est vérifiée ou non au niveau des conditions d'existence de racines pour le système (S) et le polynôme Q. En effet, nous avons l'implication suivante entre les conditions d'existence de solutions :

$$(\forall x, [(S) \Rightarrow Q(x,a)=0]) \Rightarrow [C_{(S)}(a)=0 \Rightarrow C_Q(a)=0] \qquad (1)$$

Le premier intérêt est le coût de cette méthode. En effet, il est beaucoup moins coûteux de déterminer la condition d'existence de racines, que de déterminer les racines elles-mêmes. Et

---

(1) Nous avons programmé sous Reduce la méthode de Wu

(2) Temps *cpu* obtenus sur un IBM RS/6000

le second intérêt de cette méthode est qu'elle permet de décider si l'implication est fausse en examinant l'implication au niveau de la condition d'existence de racines pour (S) et Q. En revanche, si cette dernière est vraie nous ne pourrons rien en déduire. Nous serions alors obligés d'examiner si l'ensemble des zéros de (S) est bien contenu dans celui de Q au moyen de la première méthode.

Comment pouvons–nous vérifier que $C_{(S)}(a)=0 \Rightarrow C_Q(a)=0$?

Les lemmes précédents des connecteurs logiques nous permettent d'écrire que pour tout paramètre $\{a_j\}_{j=1...q}$:

$$\left(1-C_{(S)}(a)^{p-1} \right)^k C_Q(a)^k = 0, \text{ avec } k \in \{1, \dots, p-1\} \qquad (2)$$

Nous choisirons $k=1$ pour simplifier les calculs. Le coût devient alors inférieur à la première méthode du simple fait que nous évitons le calcul des solutions $\{x_k\}_{k=1...m}$.

Remarque

Il se peut que nous obtenions de cette manière une condition pour que l'implication se réalise. Notons–la $C_{(S)\Rightarrow Q}(a)$:

$$C_{(S)\Rightarrow Q}(a) = \left(1-C_{(S)}(a)^{p-1} \right) C_Q(a) \qquad (3)$$

Si pour tout paramètre $\{a_j\}_{j=1...q}$, $C_{(S)\Rightarrow Q}(a)$ ne s'annule jamais, alors l'implication est toujours fausse. Mais, s'il existe un et un unique uplet de paramètres $\{a_j\}_{j=1...q}$ tels qu'il annule $C_{(S)\Rightarrow Q}(a)$ alors cela signifie que l'implication « (S) $\Rightarrow$ $Q(x,a)=0$ » n'est pas générale en les paramètres. En revanche, s'il existe plus d'un n–uplet de paramètres $\{a_j\}_{j=1...q}$ tels qu'il annulent $C_{(S)\Rightarrow Q}(a)$ alors cela signifie que l'implication « (S) $\Rightarrow$ $Q(x,a)=0$ » reste à vérifier, et que si elle est vraie, elle l'est sous la condition $C_{(S)\Rightarrow Q}(a)$. A ce moment –là nous la considérerons comme une contrainte de plus sur les paramètres, et nous l'ajouterons aux contraintes du système (S) et du polynôme Q.

VI.2.2.3 Troisième méthode

Nous avons vu au chapitre I que nous pouvions réécrire toute formule logique sous forme prénexe. De plus, cette réécriture associée à l'élimination de variables, nous permet de donner une condition nécessaire et suffisante pour notre problème énoncé en VI.2.1.2. Cependant, bien que nous obtenions une condition ne dépendant que des paramètres, nous montrons aussi que le calcul devient très coûteux.

Nous avons donc :

$\forall x, [\, P(x,a) = 0 \Rightarrow Q(x,a) = 0 \,]$

La double négation nous donne :

$\Leftrightarrow$ non ($\exists x$, non [$P(x,a) = 0 \Rightarrow Q(x,a) = 0$])

$\Leftrightarrow$ non ($\exists x$, [$P(x,a) = 0$ et $Q(x,a) \neq 0$])

Puis la réécriture à l'aide des lemmes sur les connecteurs logiques :

$$\Leftrightarrow \text{non} (\exists\, x, [P(x,a)^q + (1 - Q(x,a)^{p-1})^q = 0])$$

Soit encore :

$$\Leftrightarrow \text{non} (\exists\, x, [P(x,a)^q + 1 - Q(x,a)^{p-1} = 0])$$

Si nous notons $R(x,a)$ la fonction précédente, et $C_R(a)$ la condition d'existence de zéro associée à R, nous avons finalement :

$$\Leftrightarrow 1 - C_R(a)^{p-1} = 0$$

VI.3 Equivalence de strates

A la section précédente nous nous sommes intéressé à la cohérence d'une décomposition où l'interprétation de la sous–strate est contenue dans celle de la strate. Dans cette section, nous nous intéressons de savoir si l'interprétation de deux strates est équivalente.

Nous rappelons les remarques au sujet des compositions successives des lemmes sur les connecteurs logiques, en particulier celui du « et », vu en III.4.2. Car une méthode pour montrer si deux systèmes sont équivalents, est de déterminer les polynômes équivalents à chacun des systèmes respectifs, puis de les comparer terme à terme. Or, selon le choix du lemme « et » ce n'est pas toujours possible. Lorsque cela l'est, l'emploi d'un tel lemme, que nous nommons symétrique, est d'un coût disproportionné par rapport aux autres lemmes "et", à cause des calculs de polynômes intermédiaires denses.

Aussi, à la suite des méthodes vues en VI.2 nous en proposons une en VI.3.

VI.3.1 Le problème de l'équivalence

Suite aux remarques précédentes, et à la section sur le problème de l'implication entre deux systèmes polynômiaux, nous pouvons reprendre ces méthodes. Nous pouvons également réécrire la formulation logique en termes algébrique comme ci–dessous.

Nous avons donc :

$$\forall\, x, [P(x,a) = 0 \Leftrightarrow Q(x,a) = 0]$$

La double négation nous donne :

$$\Leftrightarrow \text{non} (\exists\, x, \text{non} [P(x,a) = 0 \Leftrightarrow Q(x,a) = 0])$$

Puis la réécriture à l'aide des lemmes sur les connecteurs logiques :

$$\Leftrightarrow \text{non} (\exists\, x, [P(x,a) = 0 \text{ et } Q(x,a) \neq 0] \text{ ou } [P(x,a) \neq 0 \text{ et } Q(x,a) = 0])$$

Soit encore :

$$\Leftrightarrow \text{non} (\exists\, x, [P(x,a)^q + (1 - Q(x,a)^{p-1})^q] [Q(x,a)^q + (1 - P(x,a)^{p-1})^q])$$

$$\Leftrightarrow \text{non}\,(\,\exists\, x,\,[\,(P(x,a)\,Q(x,a))^{q} + (1 - P(x,a)^{p-1})\,(1 - Q(x,a)^{p-1}) = 0\,]\,)$$

Si nous notons $R(x,a)$ la fonction précédente, et $C_R(a)$ la condition d'existence de zéro associée à R, nous avons finalement l'équation de contrainte suivante :

$$\Leftrightarrow 1 - C_R(a)^{p-1} = 0$$

VI.4 Strates et schémas de résolutions

Nous considérons maintenant l'ensemble de toutes les strates issues par des décompositions successives les unes des autres à partir d'une unique strate. Vérifier à partir d'un seul système polynômial représentant toutes ces strates, si cet ensemble est cohérent, est impraticable. En effet, les algorithmes d'éliminations dont nous disposons sont de complexité exponentielle. Aussi, leur application sur un système comportant plusieurs dizaines, voire centaines ou même bien plus, de variables, n'est actuellement pas praticable. En revanche, nous proposons une méthodologie qui nous permet de vérifier, grâce à la structure arborescente des strates, si l'ensemble des décompositions est cohérent. Et ce sans à avoir à considérer tous les systèmes polynômiaux des strates, mais seulement lorsqu'une nouvelle strate est définie, ou lorsqu'une strate est modifiée.

VI.4.1 Strates, schémas de résolution et systèmes polynômiaux

La propriété qui suit nous permet de justifier l'algorithme proposé en VI.4.

Propriété VI(4) :

Soient un système polynômial (S) sur $\mathbf{Z}/p\mathbf{Z}$ et r systèmes polynômiaux (S_k) sur $\mathbf{Z}/p\mathbf{Z}$ qui représentent une décomposition de (S).

Si le système (S) ainsi que chaque système (S_k) est cohérent, et si le système formé par l'ensemble des (S_k) est cohérent, et enfin, si le système (S) implique l'ensemble des (S_k), alors nous aurons une décomposition cohérente. Ceci se schématise comme suit en (4) :

$$\left\{\begin{array}{l} \quad (S) \\ (S_k),\, k \in \{1, \ldots, r\} \\ \left\{\begin{array}{l} \ldots \\ (S_k),\, k \in \{1, \ldots, r\} \\ \ldots \end{array}\right. \\ (S) \Rightarrow \left\{\begin{array}{l} \ldots \\ (S_k),\, k \in \{1, \ldots, r\} \\ \ldots \end{array}\right. \end{array}\right. \qquad (4)$$

La propriété précédente, permet de vérifier si la décomposition d'une strate qui a un schéma de résolution associé, c'est–à–dire qui se décompose en un agencement particulier de strates, est cohérente ou non. Pour cela, nous avons quatre niveaux de cohérence à vérifier afin que l'ensemble le soit.

VI.4.2 Algorithmes de vérifications

La propriété VI(4) nous donne une méthodologie qui nous permet en vérifiant de façon locale si la cohérence est vérifiée, de la vérifier au niveau global. Le premier algorithme permet de vérifier de manière statique si une résolution donnée de strates est cohérente. Tandis que le second algorithme, sous l'hypothèse que nous avons une résolution cohérente, n'effectue que les vérifications nécessaires lorsque nous ajoutons ou modifions une strate. Ce dernier, par opposition au premier algorithme, s'adapte bien pour une vérification dynamique des strates, lorsque la résolution est en cours de développement, ou lorsque nous modifions une strate.

VI.4.2.1 Algorithme de vérification générale

Suite à la propriété énoncée en VI.4.1, nous proposons l'algorithme suivant. Il se décompose en deux trois parties :

1. la vérification de tous les systèmes polynômiaux associés à chaque strate,
2. la vérification, pour tous les schémas de résolutions, du système polynômial formé par les systèmes de chaque élément du schéma qui se réfèrent à une strate,
3. la vérification des décompositions à partir d'une strate et de son schéma de résolution, ceci en vérifiant si le système polynômial déduit bien celui formé par le schéma de résolution de la strate.

Toutes les fois que nous faisons appel à un test de cohérence, nous faisons appel à une méthode algébrique de vérification de la cohérence vue à l'une des sections précédentes.

Algorithme de vérification d'une résolution de strates :

Entrées : Strates
Sorties : réponse

---

réponse ← vrai

Tant que (réponse = vrai) **et** (Strates non toutes parcourues) **faire**
Strate ← *PrendreUneStrateDans* : Strates
S ← *ExtraireSystèmeDe* : Strate
réponse ← réponse **et** (S cohérent ?)
fin Tant que

Tant que (réponse = vrai) **et** (Strates non toutes parcourues) **faire**
Strate ← *PrendreUneStrateDans* : Strates
SchémaRésolution ← *PrendreSchémaDans* : Strate
S' ←∅
Pour chaque Strate **de** SchémaRésolution **faire**
S' ← S' *union* : (*ExtraireSystèmeDe* : Strate)
fin Pour chaque
réponse ← réponse **et** (S' cohérent ?)
fin Tant que

Tant que (réponse = vrai) **et** (Strates non toutes parcourues) **faire**
Strate ← *PrendreUneStrateDans* : Strates
SchémaRésolution ← *PrendreSchémaDans* : Strate
S1 ← *ExtraireSystèmeDe* : Strate
S2 ←∅
Pour chaque Strate **de** SchémaRésolution **faire**
S2 ← S2 *union* : (*ExtraireSystèmeDe* : Strate)
fin Pour chaque
réponse ← réponse **et** ((S1⇒ S2) cohérent ?)
fin Tant que

Retourner réponse

VI.4.2.2 Algorithmes de vérification ponctuelle

Les algorithmes qui suivent effectuent des vérifications de manière ponctuelle lorsque nous :

1. définissons une nouvelle strate,
2. définissons un schéma de résolution,
3. et si le schéma de résolution est cohérent avec la strate qui le contient.

La variable Choix, est une liste de vérification(s) à réaliser selon l'énumération précédente.

Algorithme de vérification d'une strate :

Entrées : Strate,
Sorties : réponse

---

réponse ← vrai

Si (1 ∈ Choix) **alors**
S ← *ExtraireSystèmeDe* : Strate
réponse ← réponse **et** (S cohérent ?)
fin Si

Si (2 ∈ Choix) et (réponse = vrai) **alors**
SchémaRésolution ← *PrendreSchémaDans* : Strate
S' ←∅
Pour chaque Strate **de** SchémaRésolution **faire**
S' ← S' *union* : (*ExtraireSystèmeDe* : Strate)
fin Pour chaque
réponse ← réponse **et** (S' cohérent ?)
fin Si

Si (3 ∈ Choix) et (réponse = vrai) **faire**
SchémaRésolution ← *PrendreSchémaDans* : Strate
S1 ← *ExtraireSystèmeDe* : Strate
S2 ←∅
Pour chaque Strate **de** SchémaRésolution **faire**
S2 ← S2 *union* : (*ExtraireSystèmeDe* : Strate)
fin Pour chaque
réponse ← réponse **et** ((S1⇒ S2) cohérent ?)
fin Si

Retourner réponse

VI.4.3 Opérateurs de décomposition des schémas de résolutions

Aux constructions élémentaires qui décrivent un schéma de résolution des strates, nous leur associons des opérateurs de décompositions. Ces opérateurs dépendent de la représentation algébrique associée à un langage. Cependant, nous pouvons définir de façon générale comment sont formés les systèmes selon une décomposition.

Nous groupons les variables d'uns strate représentant les données et les paramètres en un ensemble D, les résultats en un ensemble R. Pour simplifier les notations, nous suffixons ces

variables par le nom de la strate, ainsi "D.S1" désigne les données et les paramètres de la strate S1.

Considérons la relation binaire "$\rightarrow$" qui définit un ensemble d'équations polynômiales à partir de deux ensembles de variables. Le premier désigne les données et/ou les paramètres, et le second les résultats. De plus, nous indicerons cette relation par le nom de la strate. Supposons que nous ayons une strate père, notée Sp, que nous décomposons à partir d'une des constructions de schéma de résolution. Cette strate admet comme représentation algébrique: $E.Sp \rightarrow_{Sp} R.Sp$

a) **Décomposition séquentielle**

De nouvelles variables, les variables locales, peuvent apparaître entre la séquence des strates S1, S2. Cette décomposition va nous construire le système suivant et les hypothèses ensemblistes sur les variables suivantes:

$$\begin{cases} E.S1 \rightarrow_{S1} R.S1 \\ E.S2 \rightarrow_{S2} R.S2 \end{cases} \text{, avec } \begin{cases} R.Sp \subseteq (R.S1 \cup R.S2) \\ E.S1 \subseteq E.Sp \\ E.S2 \subseteq (E.Sp \cup R.S1) \end{cases}$$

b) **Décomposition parallèle**

C'est une décomposition qui va séparer en deux groupes distincts les variables résultats de la strate père.

$$\begin{cases} E.S1 \rightarrow_{S1} R.S1 \\ E.S2 \rightarrow_{S2} R.S2 \end{cases} \text{, avec } \begin{cases} R.Sp \subseteq (R.S1 \cup R.S2) \\ R.S1 \cap R.S2 = \varnothing \\ E.S1 \subseteq E.Sp \\ E.S2 \subseteq E.Sp \end{cases}$$

c) **Décomposition choix**

Cette décomposition nous donne un choix de résolution. Aussi, aurons-nous l'un ou l'autre système associé :

$$\begin{cases} E.S1 \rightarrow_{S1} R.S1 \end{cases} \textbf{ ou } \begin{cases} E.S2 \rightarrow_{S2} R.S1 \end{cases}$$

avec les hypothèses sur les ensembles de variables des strates S1 et S2 :

$$\begin{cases} R.Sp \subseteq R.S1 \\ E.S1 \subseteq E.Sp \end{cases} \textbf{ et } \begin{cases} R.Sp \subseteq R.S2 \\ E.S2 \subseteq E.Sp \end{cases}$$

d) **Décomposition test**

Nous avons une strate, St, qui réalise un test, elle ne retourne qu'une seule valeur de type booléen, que nous notons T. Les sytèmes des strates S1 et

S2 tiennent compte de T dans leurs équations, aussi nous le signalons par les produits par T, et par (non T)

$$\begin{cases} E.St \rightarrow_{St} \{ T \} \\ (T) (E.S1 \rightarrow_{S1} R.S1) \\ (\text{non } T) (E.S2 \rightarrow_{S2} R.S2) \end{cases} \text{, avec } \begin{cases} R.Sp \subseteq R.S1 \\ R.Sp \subseteq R.S2 \\ E.S1 \subseteq E.Sp \\ E.S2 \subseteq E.Sp \end{cases}$$

e) **Décomposition boucle**

Nous avons une strate, St, qui réalise un test d'arrêt, elle ne retourne qu'une seule valeur de type booléen, que nous notons T. Le sytème de la strate S1 tient compte de T dans ses équations, aussi nous le signalons par un produit par T.

$$\begin{cases} E.St \rightarrow_{St} \{ T \} \\ (T) (E.S1 \rightarrow_{S1} R.S1) \end{cases} \text{, avec } \begin{cases} R.Sp \subseteq R.S1 \\ E.S1 \subseteq E.Sp \end{cases}$$

VI.5 Logique associées à quelques langages

Pour un langage donné, nous construisons une logique à partir des différents états possibles, en nombre fini, que peut prendre comme type de valeur une variable donnée. D'autre part, lorsque nous pouvons effectuer un classement, une taxinomie, des états qui entraînent d'autres états – sans avoir de circuits – nous verrons plus loin que nous pouvons réduire par un artifice judicieux le nombre des états. Ceci nous permet de choisir un corps fini $\mathbf{Z}/p\mathbf{Z}$ de caractéristique inférieur au nombre total des états..

VI.5.1 Description des états de variables d'un langage

Soit **L** un langage auquel nous pouvons associer différents états aux variables de ce langage. Ces états peuvent admettre des enchaînements finis de la forme :

Si l'état d'une variable est A alors c'est aussi un état B.

Et nous supposerons que n'avons pas de circuits dans ces séquences d'états.

Puis, nous pouvons classer les différents états, soit par une technique sur les graphes comme celle donnée par le diagramme de Hasse, soit en regroupant les états selon leur type. Il reste à définir les applications permettant de passer d'une classes d'états vers une autre classe. De façon arbitraire nous nous sommes limités aux applications puissances, car elles présentent une facilité d'interprétation des polynômes ou des résultats obtenus.

VI.5.1.1 Logique d'un langage synchrone, Signal

Le langage Signal est un langage temps réel, synchrone, qui permet de spécifier des circuits intégrés [26], mais aussi pour la vérification de spécifications pour des systèmes temps réels [26]. Ce langage associe à tout instruction une équation algébrique, ce qui lui permet de vérifier la cohérence des programmes Signal.

Ainsi, à toute variable Signal est associé l'un des deux états suivants :

a) 0 si le signal est indéfini,

b) 1 si le signal est défini.

Pour les variables Signal booléennes, il y a 3 états qui sont :

a) 0 si le signal booléen est indéfini,

b) 1 si le signal booléen est vrai,

c) -1 [(3)] si le signal booléen est faux.

Or, un signal booléen vrai ou faux est aussi un signal défini. Cette remarque permet de réduire le nombre des états dans le système Signal à 3 états seulement, au lieu de 4 états nécessaires (ou 5 états si l'on distingue les états indéfinis des signaux booléens ou non). Nous avons donc la taxinomie suivante des états :

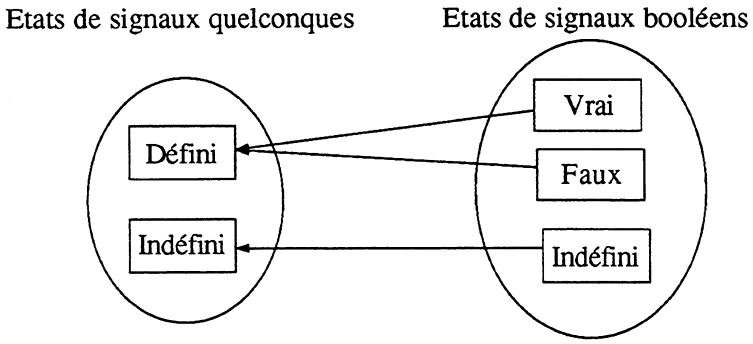


Fig. 6.1 : Taxinomie des états du langage Signal

Dans un corps **Z**/3**Z**, l'application qui aux états vrai, faux, respectivement 1, -1, associe l'état défini, 1, est donnée par x^2. Nous pouvons ainsi distinguer les variables d'une catégorie d'une autre. Cet artifice permet de réduire le nombre d'états, et d'éviter d'avoir à étudier des systèmes polynômiaux dans **Z**/5**Z**.

Ces états sont indépendants du temps t, ceci permet d'éviter des contradictions en raisonnant sur le temps. Aux différentes instructions Signal, traduites en français, nous pouvons leur associer une formule logique ou une expression algébrique sur **Z**/3**Z**. Nous redonnons les instructions du langage Signal et les polynômes associés. Les rapports sur Signal de Le Guernic *et al* [25], [26], et la thèse de Le Borgne [24], justifient ces choix de polynômes. Nous précisons seulement les actions sur les états de ces instructions.

---

(3) Aussi noté 2 au lieu de -1, car $2 \equiv -1$ [3]

Pour les instructions sur les variables booléennes nous avons :

- $x \leftarrow$ **non** a $\quad x \equiv -a\,[3]$
- $x \leftarrow$ a **ou** b $\quad x \equiv a\,b\,(1 - a\,b - a - b)\,[3]$
- $x \leftarrow$ a **et** b $\quad x \equiv a\,b\,(a\,b - 1 - a - b)\,[3]$
- $x \leftarrow a$ **retardé** $\quad \begin{cases} x \equiv a^2\,\xi\,[3] \\ \xi' \equiv a + (1 - a^2)\,\xi \end{cases}$

  L'état de la variable x prend celui de a mais avec un décalage d'une unité de temps. L'introduction de ξ et ξ' permet de mémoriser l'ancien état a.

Les instructions sur les variables non booléennes sont :

- $a_{n+1} \leftarrow f(a_1, \ldots, a_n) \quad a_{n+1}^2 \equiv a_1^2 \equiv \ldots \equiv a_n^2\,[3]$

  Si toutes les variables a_i sont définies alors a_{n+1} l'est aussi. Le calcul de f sera possible, ainsi que l'affectation.
- $a \leftarrow b$ **retardé** $\quad a^2 \equiv b^2$

  L'état de la variable x prend celui de a mais avec un décalage d'une unité de temps..
- $a \leftarrow b$ **défaut** c $\quad a^2 \equiv b^2 + c^2 - b^2\,c^2\,[3]$

  Si b est indéfini, alors a prend par défaut l'état de la variable c, sinon celui de b.
- $a \leftarrow b$ **quand** c $\quad a^2 \equiv b^2(-c - c^2)\,[3]$

  Lorsque c est vrai et que b est défini, alors la variable a est définie, sinon a reste indéfini.

Ainsi à toute procédure ou programme Signal ayant des variables en entrées/sorties, nous pouvons lui associer un système d'équations polynômiales sur **Z**/3**Z**. Nous pouvons alors vérifier si l'interprétation du code est valide, consistante, ou générale en les entrées selon une résolution en x ou en "x puissance 2" selon que x est booléen ou non.

VI.5.1.2 Logique des langages de programmation classique

Soit un langage programmation classique de type Fortran, Pascal, C, Ada. Nous associons à toute variable l'un des 4 états suivants :

a) 0, si la variable est booléenne et a pour valeur Faux,

b) 1, si la variable est booléenne et a pour valeur Vrai,

c) 2, si le calcul ou l'accès à la valeur de la variable conduit à une erreur,

d) 3, si le calcul ou l'accès à la valeur de la variable conduit à une boucle infinie.

Ici aucun de ces états n'entraîne un autre. Un premier jeu d'instructions élémentaires est le suivant :

- la négation, Non (x),
- le connecteur logique "et", Et (x, y),

- le connecteur logique "ou", Ou (x, y),
- le branchement conditionnel "si ... alors ... sinon ...", Si (x, y, z).

Les tables de vérités de ces opérateurs logiques qui forment une logique 4–valuée, sont les suivantes:

Etat x	0	1	2	3
Non (x)	1	0	2	3

Fig. 6.2 : Une négation 4–valuée

Et (x, y)		Etats x			
		0	1	2	3
Etats y	0	0	0	0	0
	1	0	1	2	3
	2	0	2	2	2
	3	0	3	2	3

Fig. 6.3 : Un connecteur logique "et" 4–valué

Ou (x, y)		Etats x			
		0	1	2	3
Etats y	0	0	1	2	3
	1	1	1	1	1
	2	2	1	2	2
	3	3	1	2	3

Fig. 6.4 : Un connecteur logique "ou" 4–valué

Nous définissons le connecteur séquentiel "si ... alors ... sinon ..." de la manière suivante:

$$\text{Si}(x, y, z) = \begin{cases} y, \text{ si } x = 1 \\ z, \text{ si } x = 0 \\ x, \text{ sinon} \end{cases}$$

Fig. 6.5 : Un connecteur "si" 4–valué

Nous pouvons associer à chacune de ces expressions logiques des polynômes de **Z/5Z** les caractérisant. Nous employons pour cela les polynômes d'interpolation de Lagrange dans les corps finis définis au chapitre I. Cependant, nous n'avons pas défini de cinquième état, aussi aurons–nous non unicité dans le choix de nos interpolations de Lagrange. Par exemple, la négation admet cinq interpolations possibles qui sont:

a) $4\left(x^4 + 3x^3 + 2x + 4\right)$

b) $3\left(x^4 + x^3 + 3x^2 + 3x + 2\right)$

c) $2\left(x^4 + 2x^3 + 4x^2 + 3\right)$

d) $x^4 + 2x^2 + x + 1$

e) $x^3 + x^2 + 2x + 1$

Le choix d'un de ces polynômes associés à la négation est arbitraire. Par souci de réduction des opérations d'évaluation d'un polynôme en une valeur de x, pour des sommes et produits de polynômes, nous pouvons prendre un critère sur le nombre de monômes, puis si cela ne suffit pas, prendre un critère sur le degré total. Les critères précédents nous amènent à retenir pour la négation 4–valuée, le dernier des polynômes précédents, soit:

$$\text{Non}(x) = x^3 + x^2 + 2x + 1 \qquad (5)$$

Pour les autres opérateurs, les critères de choix, nombre total de termes puis degré total, nous donnent:

- $\text{Et}(x, y) = xy\left(4x^3y^3 + x^2y^2 + 2xy + 2x + 2y\right)$
- $\text{Ou}(x, y) = x^4y^4 + 4x^3y^2 + 4x^2y^3 + 2x^2y + 2xy^2 + xy + x + y$
- $\text{Si}(x, y, z) = 4\,(\,x^3y^4z^3 + 3x^3y^4z^2 + 3x^3y^4z + x^3y^3z^4 + x^3y^3z^3 + 2x^3y^3z^2$
  $+ 4x^3y^3 + 3x^3y^2z^4 + 2x^3y^2z^3 + 3x^3y^2z^2 + 2x^3y^2z + 2x^3y^2 + 3x^3yz^4$
  $+ 2x^3yz^2 + x^3yz + 2x^3y + 4x^3z^3 + 2x^3z^2 + 2x^3z + 3x^2y^4z^4 + 3x^2y^4z^3$
  $+ x^2y^4z^2 + 2x^2y^4 + 4x^2y^3z^4 + 4x^2y^3z^3 + 3x^2y^3z^2 + x^2y^3 + 4x^2y^2z^4$
  $+ 4x^2y^2z^3 + 3x^2y^2z^2 + x^2y^2 + 3x^2yz^4 + 3x^2yz^3 + x^2yz^2 + 2x^2y + 2x^2z^4$

$$+2x^2z^3+4x^2z^2+3x^2+2xy^4z^4+2xy^4z^3+4xy^4z^2+3xy^4+2xy^3z^4$$
$$+4xy^3z^3+xy^3z+3xy^3+4xy^2z^4+xy^2z^2+3xy^2z+xy^2+xyz^3+3xyz^2$$
$$+3xyz+3xz^4+3xz^3+xz^2+2x+4y^4z^3+2y^4z^2+2y^4z+2y^3z^3+y^3z^2$$
$$+y^3z+2y^2z^3+y^2z^2+y^2z+4yz^3+2yz^2+2yz+z^3+3z^2+3z\ )$$

Le polynôme correspondant à l'instruction "si" comporte 73 monômes, et a été déterminé à l'aide des critères précédents en employant seulement 3 degrés de libertés correspondants aux situations suivantes: Si $(0, y, 4) = u_1$, Si $(1, 4, z) = u_2$, Si $(4, y, z) = u_3$. Dans [11], Chazarain *et al.* ont sans doute effectué une minimisation basée sur les critères précédents en employant bien plus de degrés de libertés, au moins 35, et proposent le polynôme suivant :

$$\text{Si}\,(x, y, z) = 3x^3y+4x^3z+2x^3+x^2z+3xy+4xz+3x+z \qquad (6)$$

Nous pouvons remarquer que les tautologies ou les contradictions de la logique classsique, 2–valuée, ne s'appliquent plus dans ce cadre. En effet :

- $\text{Ou}(\text{Non}(x), x) = 4\left(x^3+4x^2+4\right) \neq 1$
- $\text{Et}(\text{Non}(x), x) = 2x\left(x^3+x+3\right) \neq 0$

Cependant, nous aurions pu définir le connecteur "ou" à partir de l'expression suivante Non (Et (Non x, Non y)), mais nous obtiendrions à ce moment–là un polynôme comportant 14 termes au lieu des 8 du polynôme précédent associé au "ou".

VI.5.2 Dynamique pris en compte par une représentation statique

Pour les langages séquentiels, au contraire de Signal par exemple, un inconvénient majeur des représentations logiques, ou algébriques, est qu'elles ne prennent pas en compte la notion de d'ordre dans une séquence d'instruction. Afin, de pouvoir tenir compte de la dynamique d'un algorithme séquentiel, nous proposons en VI.5.2.1 un artifice qui permet de la représenter par un système statique.

VI.5.2.1 La notion de superviseur

Nous considérons pour cela le langage synchrone Signal vu en VI.5.1.1, et nous voulons représenter une affectation au sens des langages de programmation classiques. C'est–à–dire, à un instant t_0 donné, cette affectation a eu lieu, et depuis nous avons en mémoire une valeur pour la variable y. Et, pour préciser que cette opération a eu lieu, pour tous les instants, quel que soit l'état de x, nous ne faisons plus l'affectation. Nous avons besoin de mémoriser si cette instruction a été réalisée ou non, grâce à un superviseur associé à cette instruction.

Nous notons la composition séquentielle par $Y \leftarrow f(X)$, et nous la définissons comme suit en termes de signaux :

A l'instant t, le signal Y est défini lorsque la première valeur du signal X est arrivée à un instant antérieur ou égal à t. Ceci nous donne une équation Signal de filtrage :

$$Y := X \textbf{ quand } \xi$$

où ξ est un signal booléen qui filtre les signaux X.

Les équations s'écrivent en fonction du temps t comme suit :

$$\text{Pour tout instant } t, \xi_t = \begin{cases} \text{est indéfini , si } \xi_{t-1} \text{ et } x_t \text{ sont indéfinis} \\ \text{est vrai, si } \xi_{t-1} \text{ est indéfini et } x_t \text{ est défini,} \\ \text{est faux, si } \xi_{t-1} \text{ est défini} \end{cases} \quad (7)$$

Par convention, nous initialisons ξ selon la présence ou l'absence de x avec ξ indéfini à l'instant 0.

En termes d'équations sur le corps $\mathbf{Z}/3\mathbf{Z}$, le système s'écrit :

$$\text{Pour tout instant } t, \xi_t = \begin{cases} -1 \text{ , si } \xi_{t-1}^2 \equiv 1 \ [3] \\ \text{sinon} \begin{cases} 1 \text{ , si } x_t^2 \equiv 1 \\ 0, \text{ sinon} \end{cases} \end{cases} \quad (8)$$

Nous obtenons une relation dynamique, qui est :

$$\text{Pour tout instant } t, \xi_t \equiv -\xi_{t-1}^2 + x_t^2\left(1 - \xi_{t-1}^2\right) [3] \quad (9)$$

Cette équation, peut s'écrire indépendemment du temps, comme étant :

$$\xi \equiv -\xi'^2 + x^2\left(1 - \xi'^2\right) [3] \quad (10)$$

Cette condition de filtrage des signaux X nous permet de définir les équations associées à la composition séquentielle. Nous avons donc :

$$Y \leftarrow f(X) \Leftrightarrow \begin{cases} y^2 \equiv x^2 (-\xi - \xi^2) \ [3] \\ \xi \equiv -\xi'^2 + x^2\left(1 - \xi'^2\right) [3] \end{cases} \quad (11)$$

Conclusion

Durant ce travail de thèse, nous avons développé deux logiciels importants qui permettent d'une part d'éditer des formules mathématiques en lien avec des systèmes de calculs présenté en annexe B, et d'autre part un éditeur des strates présenté au chapitre V et en annexe C. Ces éditeurs ont été développé à l'aide des langages de Grif et peuvent s'insérer dans tout document structuré de Grif, comme dans la présente thèse. Ces réalisations nous ont permis de mettre au point la structure des strates développée initialement sous forme de prototype dans le système orienté–objet Smalltalk, mais également de montrer l'apport d'un tel environnement pour le calcul scientifique et la modélisation.

D'autre part plusieurs problèmes restent en suspens qui sont les suivants :

- le développement d'un éditeur de formules mathématiques qui soit moins contraignant que celui que nous proposons, GramMath, mais dont l'analyse sémantique des formules ne soit pas sujette à des ambiguïtés d'interprétations. D'autres problèmes comme celui de la définition d'une grammaire mathématique suffisamment riche, de la césure, du choix de la présentation pour les dérivées par exemple, du calcul par manipulation directe de formules, etc, peuvent être abordés ;
- l'étude des logiques multi–valuées associée aux langages qu'ils soient de haut niveau comme les langage naturel et mathématiques, ou qu'ils soient de nature informatique comme les langages de programmation, les langages temps réel, etc ;
- la représentation des polynômes sur les corps finis n'est pas toujours mise sous forme canonique, une voie de recherche sur d'autres représentations est proposée en ce sens par Le Borgne [24] sur des représentations ensemblistes ou au moyen de graphes de Bryant. Des algorithmes restent à développer sur ce thème ;
- l'interprétation des équations de contraintes obtenues après élimination, nous conduit souvent à une étude au cas par cas afin de vérifier si ces l'interprétation de ces équations est bien celle que nous attendons. Aussi un problème est d'analyser ces équation de contraintes ;
- l'implémentation du calcul sur les corps finis en lien avec les strates,
- et enfin, l'implémentation de algorithmes de projection des strates conduisent à des problèmes de simplification des schémas de résolutions.

Annexes

Annexe A

Programmes Reduce

Nous avons développé les programmes Reduce correspondant aux algorithmes d'éliminations du chapitre III, et à la méthode de Dixon–Biard généralisée du chapitre IV. Nous donnons dans cette annexe A les descriptifs des procédures employées.

A.1 Calcul formel sur $\mathbf{Z}/p\mathbf{Z}$

Nous avons plusieurs manières d'implémenter le calcul sur $\mathbf{Z}/p\mathbf{Z}$ sous un système de calcul formel comme Reduce. Nous proposons deux procédés qui utilisent les commandes Reduce, le premier est standard, le second nous permet en revanche de tenir compte des hypothèses lors des calculs d'inversion. Dans la pratique, nous avons toujours choisi celui décrit en A.1.1 pour définir les calculs sur $\mathbf{Z}/p\mathbf{Z}$ du fait de sa simplicité, bien que cela ne puisse pas tenir compte des inversions.

A.1.1 L'approche standard

Afin de définir le calcul sur le corps fini $\mathbf{Z}/p\mathbf{Z}$, nous avons à coder le petit théorème de Fermat, et les congruences modulo p pour les scalaires.

```
% Fixer un nombre premier p
p := 3;

% Petit théorème de Fermat :
for all x let x**p = x;
% Ou bien pour les versions 3.5 et plus de Reduce :
let ~x**p => x;
```

```
% Congruences modulo p :
setmod p;
% Activation du calcul modulaire :
on modular;
```

L'inconvénient majeur de ces règles porte sur le calcul des inverses. En effet, lorsque nous avons un calcul d'inverse, nous pourrions avoir une règle de réécriture de la forme :

```
for all x let 1/x = x**(p-2);
```

Cependant le système de calcul formel Reduce[1] ne nous permet pas de réécrire un quotient comme $\frac{1}{x}$. Aussi, proposons-nous à la section suivante un autre procédé pour le calcul sur $\mathbf{Z}/p\mathbf{Z}$.

A.1.2 Une autre approche

Une tentative pour remédier au problème du calculs des inverses est de définir des opérateurs correspondant au calcul sur $\mathbf{Z}/p\mathbf{Z}$. Cependant ces opérateurs nécessitent une mémoire assez importante car nous conservons les hypothèses des calculs d'inverses. Les opérateurs sont les suivants :

a)	a **mod** p	applique le calcul de a sur $\mathbf{Z}/p\mathbf{Z}$ comme à la section précédente;
b)	b**inv** p	calcule l'inverse de b sur $\mathbf{Z}/p\mathbf{Z}$, et mémorise le fait que b doit être non nul;
c)	**hyp** (c , p)	applique l'ensemble des hypothèses établies lors des calculs d'inverses précédents sur $\mathbf{Z}/p\mathbf{Z}$ sur l'expression c.

Les règles de calculs de ces opérateurs formels apparaissent dans la liste REGLESmodp ci-dessous, et utilisent deux procédures : "memorise" et "hypothese".

La première procédure détermine un inverse modulo p, et mémorise dans la liste globale *!hyp!* l'expression avant inversion. La seconde procédure, "hypothese", détermine à partir des expressions de *!hyp!* un ensemble de règles de simplifications basées sur le petit théorème de Fermat :

$$x \neq 0\ [p] \Leftrightarrow x^{p-1} \equiv 1\ [p]$$

Cependant, nous pouvons avoir des situations qui ne permettent pas de simplifier une expression, car le système de calcul formel Reduce ne sait pas toujours factoriser sur les corps finis.

```
% Opérations dans Z/pZ
% en mémorisant les hypothèses de calcul d'inverses dans !hyp!
```

(1) La version 3.5

```
!!hyp!!:={};

infix mod, inv;
operator hyp;

procedure memorise(obj, p);
begin
if not {obj, p} member !!hyp!!
    then !!hyp!! := {obj, p} . !!hyp!!;
return mod(obj^(p-2), p);
end;

procedure hypothese(x,p);
begin scalar listeRegles;
listeRegles:= for i:=1:length(!!hyp!!) join
     if part(!!hyp!!, i, 2) eq p
     then  { << mod( part(!!hyp!!, i, 1)^(p-1) , p )>> => 1 }
     else  {};
let listeRegles;
x:=mod(x, p);
clearrules listeRegles;
return x;
end;

REGLESmodp := {
    % Inverse dans Z/pZ
    inv(~x*~y, ~p) => if numberp p and primep p
                          then  inv(x,  p)  *inv(y,  p),
    inv(~x, ~p) => if numberp p and primep p
                          then   memorise(x,   p),

    % Application des hypothèses x non nul
    hyp(~x, ~p) => <<hypothese(x, p)>>,

    % Petit théorème de Fermat
    mod(~x^~n, ~p) => if numberp p and primep p and numberp n
        then x^<<if remainder(n, p-1)=0 then p-1 else remainder(n, p-1)>>,

    % Linéarité
    mod( - ~a, ~p) => - mod(a, p),
    mod(~a + ~b, ~p) => mod(a, p) + mod(b, p),
```

```
mod(~a * ~b, ~p) => mod(a, p) * mod(b, p),
mod(~a / ~b, ~p) => mod(a * inv(b, p), p),

% Calcul modulo p
mod(~x, ~p) => remainder(x, p)
};

let REGLESmodp;
```

A.2 Élimination sur le corps fini Z/3Z

Pour l'élimination sur un corps $\mathbf{Z}/p\mathbf{Z}$, p supérieur à 3, nous pouvons seulement déterminer l'équation de contrainte associée. Dans les descriptifs suivants, les exemples que nous donnons sont tous réalisés en ayant au préalable défini des règles de calcul modulo 3.

A.2.1 Opérateur infixes : "yEst" et "Present"

Cet opérateur nous permet de tester la présence ou non d'une variable dans un polynôme. Il est différent de l'opérateur "member" qui teste la présence d'un élément dans une liste car il est basé sur une comparaison d'objets identiques, alors que l'opérateur "yEst" est basé sur les congruences modulo p. Par exemple, en ayant activé le calcul sur $\mathbf{Z}/3\mathbf{Z}$, nous aurons : 2 member {3, 4, 5}, qui nous retourne une valeur faux, tandis que : 2 yEst {3, 4, 5}, nous retourne une valeur vraie car 2 est ici congrue à 5 modulo 3.

Syntaxe : a yEst L

yEst (a, L)

Entrées : un élément a, une liste L

Sortie : une valeur booléenne

Traitement effectué : Similaire à l'opérateur Reduce member sauf que les comparaisons sont effectuées modulo p.

Exemples

```
> if (2 member {3, 4, 5}) then Vrai else Faux;
FAUX
> if (2 yEst {3, 4, 5}) then Vrai else Faux;
VRAI
```

Le second opérateur porte sur la présence de variables dans un polynôme sans faire appel aux opérateurs Reduce sur la dépendance. En effet, il se peut qu'il y ait des dépendances non définies, c'est une situation que nous voulons éviter.

Syntaxe : x Present P

Present (x, P)

Entrées : un nom de variable x, un polynôme P

Sortie : une valeur booléenne

Traitement effectué : Retourne une valeur vraie si le polynôme P contient dans son expression la variable x.

Exemple

```
>if x present (x-y)*(x+y2) then Vrai else Faux;
VRAI
> if (x present x^3-x) then Vrai else Faux;
FAUX
```

A.2.2 Procédures sur les équations : "Scal", "idNul", "extrait"

Les procédures qui suivent nous permettent de faire quelques opérations élémentaires sur les équations :

1. **procédure Scal :**

   Syntaxe : Scal (a = b);
   Scal (equa);

   Entrée : une équation

   Sortie : un scalaire

   Traitement effectué : réalise la soustraction des deux membres de l'équation, a – b.

   Exemple

   ```
   > Scal ( x^2 + x = x + y );
   ```
   $X^2 - Y$

2. **procédure idNul :**

   Syntaxe : idNul (a = b);
   idNul (equa);

   Entrée : une équation

   Sortie : un booléen

   Traitement effectué : teste si l'équation est identiquement nulle

   Exemples

   ```
   > if idNul ( x = x ) then Vrai else Faux;
   FAUX
   > if idNul ( 0 = 0 ) then Vrai else Faux;
   VRAI
   ```

3. **procédure extrait :**

Entrées : un polynôme, une liste de variables

Sortie : une autre liste de variables

Traitement effectué : parmi la liste des variables fournie en entrée, cette procédure retourne celles qui sont présentes dans le polynôme donné en entrée.

Exemple

```
> Extrait ( x*(y-z)^2 = x*w, {u,v,w,x});
{ W , X }
```

A.2.3 Procédure "Reevalue"

Entrées : un polynôme, une variable, un code, et une solution

Sortie : un polynôme

Traitement effectué : substitue la variable par une solution dans un polynôme. La substitution diffère selon que nous ayons une résolution en x, codé par 0, ou en x^2, codé par 1.

Exemples

```
> Reevalue ( x**2 + x - y, x, 0, y);
```

Y^2

```
> Reevalue ( x**2 + x - y, x, 1, y);
RAC ( Y )
```

A.2.4 Procédures "GenTyp", "GenEqu"

Les procédures qui suivent nous permettent de générer soir le type de résolution, soit une liste de variables pour chacune des équations d'une liste.

1. **Procédure "GenTyp"**

Entrée : une liste de monômes comportant une seule variable chacun, et tous ces monômes sont indépendants 2 à 2.

Sortie : une liste composée de deux listes : une liste de nom de variables et une liste d'entiers

Traitement effectué : Selon la puissance, détermine un code correspondant au type de résolution associé à une variable.

Exemple

```
> GenTyp({a,b**2,c**2});
{ {C, B, A}, {1, 1, 0 } }
```

2. **Procédure "GenEqu"**

Entrées : une liste d'équations, une liste de variables

Sortie : une liste de paires : le premier élément est une équation et le second élément une liste de variables

Traitement effectué : Pour chaque équation, nous appliquons la procédure "extrait" qui nous retourne une liste de variables prises dans la liste fournie en entrée et présentes dans l'équation.

Exemple

```
> GenEqu ( { x^2 = y^2, z^2 = x^2*(-h-h^2)}, {x, y, z, h} );
```

$\{ \{ X^2 = Y^2, \{ X, Y \} \},$

$\{ Z^2 = -H^2*X^2 - H*X^2, \{X, Z, H\} \} \}$

A.2.5 Procédure "resoudTout"

Entrées : un polynôme, un nom de variable, un terme

Sortie : une liste de deux éléments : liste des coefficients, liste des monômes multipliés par le terme

Traitement effectué : Similaire à celui effectué par la procédure COEFF de Reduce, sauf que nous supprimons les éléments nuls de la liste fournie par COEFF, et nous associons pour les éléments non nuls , les monômes de la forme : variable * terme

Exemple

```
> mycoeff (a*x^10+b*x+c, x, 2);
{ {A, B, C} , {2*X, 2*X, 2} }
```

A.2.6 Algorithme principal : procédure "resoud"

La procédure "résoud" est en quelque sorte l'algorithme autour duquel les résolutions sont basées pour l'éliminations de variables d'un système polynômial sur $\mathbf{Z}/p\mathbf{Z}$.

Entrées : une liste de paires dont le premier élément est une équation, et le second une liste de variables contenues dans l'équation, une expression algébrique, une liste de variables, une liste d'entiers compris entre 0 et 1

Sortie : une liste de trois éléments : deux listes d'équations, et l'expression algébrique

Traitement effectué : Cette procédure réalise l'élimination des variables données en entrée du système d'équation contenu dans la liste des paires. Pour chacune des variables à éliminer nous avons un choix de résolution proposé par la liste des entiers compris entre 0 et 1. Et, lors de chaque élimination, la solution obtenue pour telle ou telle variable est substituée dans l'expression algébrique fournie en entrée.

Les éléments de la liste obtenus en sortie correspondent respectivement à la liste des équations de contraintes, à la liste des solutions obtenues pour la liste des variables éliminées, et enfin à l'expression algébrique destinée à être réévaluée.

Exemples

```
> Resoud ({{x^2=y^2*(-h-h^2), {x, y}}, {x^2=y^2, {x, y}}} , x + Rac(y) , {x, y},
{0, 1});
{ {} ,
{ Y^2 = 2*H^2*PHI#_9^2 + 2*H*PHI#_9^2,
  X = 2*SIGNE_(!8_NUMERO_)*H^2*PHI#_9^2 +
    2*SIGNE_(!8_NUMERO)*H*PHI#_9^2 } ,
2*SIGNE_(!8_NUMERO_)*PHI#_9^2*H*(H+1) +
    RAC(2*H^2*PHI#_9^2 + 2*H*PHI#_9^2)
}
```

A.2.7 Procédures "resoudre" et "resoudTout"

Suite à la procédure "resoud", ces procédures nous servent d'interface pour les entrées. En effet, il nous suffit de donner pour les entrées un système d'équations et une liste de variables pour lesquelles nous désirons réaliser l'élimination. La différence entre ces deux procédures est que la seconde, "resoudTout", tente de résoudre les équations de contraintes obtenues afin de simplifier l'interprétation des résultats.

- **procédure "resoudre"**

  Entrées : une liste d'équations, une liste de variables

  Sortie : une liste de trois éléments : les équations de contraintes, les solutions et une liste vide par défaut.

Traitement effectué : Nous employons la procédure "resoud" pour éliminer les variables du systèmes polynômial donné en entrée.

Exemple

> Resoudre({x^2=y^2*(-h-h^2), x^2=y^2}, {x, y^2});

{ {} ,

{ $X = 2*H^2*PHI\#\_11 + 2*H*PHI\#\_11$,

$Y^2 = 2*H^2*PHI\#\_11^2 + 2*H*PHI\#\_11^2$ },

{ } }

- **procédure "resoudTout"**

Entrées : une liste d'équations, deux liste de variables

Sortie : une liste de trois éléments : les équations de contraintes, les solutions correspondant à la première liste de variables, E, puis celle correspondant à la seconde liste de variables, S.

Traitement effectué : Nous réalisons de même une élimination des variables S du système initial, et nous résolvons en plus les équations de contraintes en éliminant les variables E.

Exemple

> ResoudTout({x^2=y^2*(-h-h^2), x^2=y^2}, {h^2},{x^2});

{ { $2*Y^2 = 0$ },

{ $H^2 = 2*Y^2*PHI\#\_14^2 + 2*Y^2 + PHI\#\_14^2$ },

{ $X^2 = 2*RAC(2*Y^2*PHI\#\_14^2 + 2*Y^2 + PHI\#\_14^2) + Y^2$ } }

A.2.8 Procédures "solution" et "contrainte"

Lorsque nous voulons déterminer les solutions des variables éliminées, nous disposons de 2 procédures qui fournissent la solution sous une forme générale, et une équation de contrainte associée.

- **Procédure "solution"**

Entrées : une équation, une variable, et un entier

Sortie : une expression algébrique

Traitement effectué : Selon l'entier fourni, cette procédure détermine une solution pour une variable

selon une équation. Si l'entier vaut 1, nous effectuons une résolution en x^2, sinon une résolution en x.

Exemples

> Solution (a*x^2=–b*x–c, x, 1);

$2*PHI\#\_7^2 *(A^2*B^2*C^2 + 2*A^2*B^2 + 2*A^2*C^2 + A + 2*B^2*C^2 + B^2 + C^2 + 2) - A*C$

> Solution (a*x^2=–b*x–c, x, 0);

$SIGNE(!9\_NUMERO)*RAC(- A*C + B^2)*A + 2*PHI\#\_8 *(A^2*B^2*C^2 + 2*A^2*B^2 + 2*A^2*C^2 + A + 2*B^2*C^2 + B^2 + C^2 + 2) + B*(A^2*C + A - C)$

- **Procédure "contrainte"**

Entrées : une équation, une variable, et un entier

Sortie : une expression algébrique

Traitement effectué : Selon l'entier fourni, cette procédure détermine la contrainte sur l'existence d'une solution. Si l'entier vaut 1, nous donnons l'équation de contrainte associée à une résolution en x^2, sinon celle associée à une résolution en x.

Exemple

> Contrainte (a*x^2=–b*x–c, x, 1

$-(A^2*B^2 - A*B^2*C - A*C - C^2)$

> Contrainte (a*x^2=–b*x–c, x, 0);

$C*(A^2 + 2*A*C - B^2 + 1)$

A.2.9 Opérateurs algébriques : "signe" et "rac"

Ces opérateurs nous permettent de décrire les solutions obtenues sous une forme la plus générale possible avec des règles de calculs spécifiques. Ainsi avons–nous l'opérateur "signe" qui correspond au ± indexé :

Syntaxe : signe(A)

Entrée : une expression A

Sortie : une autre expression

Traitement effectué : Toutes les fois q'un opérateur signe est appliqué, nous y associons un numéro afin de les distinguer les uns des autres.

Exemples

> signe(A);
SIGNE_(!7_NUMERO_)*A
> signe(B) + signe(−B);
SIGNE_(!8_NUMERO_)*B − SIGNE_(!9_NUMERO_)*B

Dans le corps **Z/3Z**, la racine suit les règles de calcul suivantes :

1. si $x \equiv x^2$ [3] alors $\sqrt{\ } x \equiv x$ [3], et
2. $\sqrt{\ x^2} \equiv x$ [3]

Syntaxe : rac (x)

Entrée : une expression algébrique

Sortie : une autre expression algébrique

Traitement effectué : Calcule la racine modulo 3.

Exemples

> Rac (h + h^2);
RAC (H^2 + H)
> Rac (− h − h^2);
− H*(H + 1)

A.2.10 Programme Reduce

```
% Resolution d'un systeme d'equations de Z/3Z

infix yEst;
symbolic operator yEst;
symbolic procedure yEst(elt,liste);
if liste={} then nil
else (elt=(car liste)) or yEst(elt,cdr liste);

infix present;
symbolic operator present;
symbolic procedure present(var,poly);
(algebraic prsnt(var,poly)) neq 0;

procedure prsnt(var,poly);
begin
coeff(poly,var);
return lowpow!*^2+hipow!*^2;
```

```
end;

procedure scal(equ);
(lhs(equ)-rhs(equ));

symbolic operator idNul;
symbolic procedure idNul(equ);
(lhs(equ)=0) and (rhs(equ)=0);

procedure extrait(equ,listeVar);
for each obj in listeVar join
if obj present scal(equ) then {obj} else {};

procedure reevalue(obj,var,typ,sol);
begin
if typ=1 then
     begin
     let var^2=sol;
     let var=RAC(sol);
     obj:=obj;
     clear var,var^2;
     end
else begin
     let var=sol;
     obj:=obj;
     clear var;
     end;
return obj;
end;

procedure genTyp(listeVar);
begin scalar va,typ;
va:=typ:={};
for each v in listeVar do
  if v^2 = v
     then begin va:=part(v,1).va; typ:=(1).typ; end
     else begin va:=v.va; typ:=(0).typ; end;
return {va,typ};
end;

procedure genEqu(equa,var);
```

```
for each elt in equa collect
{elt,extrait(elt,var)};

procedure resoudTout(equa,entrees,sorties);
begin scalar sol,vt, tout;
sol:=resoudre(equa,sorties);
vt:=genTyp(entrees);
tout:=resoud(genEqu(first(sol),first(vt)),second(sol),first(vt),second(vt));
off modular; on factor;
tout:=<<tout>>;
off factor; on modular;
return tout;
end;

procedure resoudre(equa,var);
begin scalar vt,tout;
vt:=genTyp(var);
tout:=resoud(genEqu(equa,first(vt)),{},first(vt),second(vt));
off modular; on factor;
tout:=<<tout>>;
off factor; on modular;
return tout;
end;


procedure resoud(equa,eqfix,var,typ);
begin scalar Xi,Ti,eqXi,solXi,contXi,newEq,keepEq;
keepEq:={};
while var neq {} do
    begin
    Xi:=first(var);  Ti:=first(typ);
    typ:=rest(typ);   var:=rest(var);
    eqXi:=0=0;
    newEq:={};
    for each elt in equa do
        if Xi yEst <<second(elt)>>
        then eqXi:=<<scal(eqXi)^2+scal(first(elt))^2>>=0
        else newEq:=elt.newEq;
    if not idNul(eqXi)
    then begin
        % Resolution en Xi
```

```
            contXi:=<<Contrainte(eqXi,Xi,Ti)>>=0;
            if not idNul(contXi) then
                  newEq:={contXi,extrait(contXi,var)}.newEq;
            %
            solXi:=Solution(eqXi,Xi,Ti);
            newEq:=reevalue(newEq,Xi,Ti,solXi);
            keepEq:=reevalue(keepEq,Xi,Ti,solXi);
            eqFix:=reevalue(eqFix,Xi,Ti,solXi);
            if Ti=1 then keepEq:=(Xi^2=solXi).keepEq
            else keepEq:=(Xi=solXi).keepEq;
            end; % for...
      equa:=newEq;
      end; % while...
return {for each elt in equa collect first(elt), keepEq, eqFix};
end;

operator signe, signe_;
factor signe_;
for all x let signe(x) = signe_(mkid(mygensym(!!),!_numero!_))*x;
for all x let signe_(x)^2 = 1;


operator rac;
for all x such that x=x^2 let rac(x)=x;
for all x let rac(x)^2=x;


procedure solution(equ,var,typ);
begin scalar a,b,c,r,v,cff,delta;
cff:=append(coeff(scal(equ),var),{0,0});
a:=third(cff);
b:=second(cff);
c:=first(cff);
v:=mygensym(phi!#!_);
factor v;
if typ=1
     then r:=v^2*(1-a^2)*(1-b^2)*(1-c^2) - a*c
     else r:=<<delta:=<<signe( rac(b^2-a*c))>>;
v*(1-a^2)*(1-b^2)*(1-c^2) - b*c*(1-a^2) + a*(b+delta)>>;
return r;
end;
```

```
procedure contrainte(equ,var,typ);
begin scalar a,b,c,r,cff;
cff:=append(coeff(scal(equ),var),{0,0});
a:=third(cff);
b:=second(cff);
c:=first(cff);
if typ=1
     then r:=<<a*b^2*(c-a)+a*c+c^2>>
     else r:=<<c*((a+c)^2-b^2)>>;
return r;
end;

procedure mygensym(var);
begin integer no;
off modular;
no:=mkid(var,symbolic digitpart gensym());
on modular;
return no;
end;

% Pris dans scope.red des sources Reduce
symbolic procedure digitpart(name);
% Effet: Digitpart of Name returned, i.e. 55 of aa55bbb

write "Activation du calcul sur Z/3Z";
for all x let x^3=x;
setmod 3;
on modular;
```

A.3 Résultant de Dixon–Biard généralisé

Par rapport à méthode décrite au chapitre <SystZpZ.chap.2i>, nous ne déterminons pas toute la matrice C. En effet, l'algorithme tel que nous l'avons implémenté sous Reduce, extrait les coefficients à partir d'un polynôme, aussi les lignes ou colonnes identiquement nulles n'apparaissent pas. Le programme principal se trouve en A.3.7, et le programme Reduce complet en A.3.8.

A.3.1 Procédure "myCoeff"

Entrées : un polynôme, un nom de variable, un terme

Sortie : une liste de deux éléments : liste des coefficients, liste des monômes multipliés par le terme

Traitement effectué : Similaire à celui effectué par la procédure COEFF de Reduce, sauf que nous supprimons les éléments nuls de la liste fournie par COEFF, et nous associons pour les éléments non nuls , les monômes de la forme : variable * terme

Exemple

```
> mycoeff (a*x^10+b*x+c, x, 2);
```

{ {A, B, C} , {2*X^{10}, 2*X, 2} }

A.3.2 Procédure "galCoeff"

Entrées : un polynôme, une liste de variables

Sortie : une liste de deux éléments : liste de coefficients, liste de monômes.

Traitement effectué : Généralise myCoeff sur une liste de variables : extraction de tous les monômes et coefficients possibles employant au moins une variable de la liste des variables donnée en entrée.

Exemple

```
> galCoeff ( (x+1)^2 * (y–z^2), {x, y, z} );
```

{ {–1, 1, –2, 2, –1, 1}, {X^2 * Z^2, X^2 * Y, X * Z^2, X*Y, Z^2, Y } }

A.3.3 Procédure "ajoutListe"

Entrées : un élément, une liste

Sortie : une liste

Traitement effectué : Ajoute l'élément à liste que s'il n'appartient pas à cette liste. Il est placé en premier.

Exemples

```
> ajoutListe( z, {a, b, c});
{Z, A, B, C}
> ajoutListe( b , {a, b, c});
{A, B, C}
```

A.3.4 Procédure "PPCMdeListes"

Entrées : une liste de listes

Sortie : une liste

Traitement effectué : Calcule la plus petite liste commune à toutes les listes fournies en entrée.

Exemples

```
> PPCMdeListes ( { {a, b, c}, {b, c, d}, {c, d, e} } );
{E, D, A, B, C}
> PPCMdeListes ( {a, b}, {x, y}, {} } );
{Y, X, A, B}
```

A.3.5 Procédure "supCoeff"

Entrées : un polynôme, une liste de variables, une autre liste de variables

Sorties : une liste contenant le vecteur vg, une matrice matT, et un vecteur vd. Ces variables vg, matT, vd sont globales.

Traitement effectué : Réécrit le polynôme sous forme d'un produit vg*matT*vd où le vecteur vd (resp. vg) contient toutes les variables de la première (resp. seconde) liste présentes dans le polynôme.

Exemple

```
> supCoeff( (x+u)^2*(y+v)^2, {u}, {v});
```

$$\left\{ \left[1, U, U^2 \right], \begin{bmatrix} X^2 & 2*X^2*Y & X^2*Y^2 \\ 2*X & 4*X*Y & 2*X*Y^2 \\ 1 & 2*Y & Y^2 \end{bmatrix}, \begin{bmatrix} V^2 \\ V \\ 1 \end{bmatrix} \right\}$$

A.3.6 Procédure "detDCalc"

Entrées : une liste de polynômes, une liste de variables, une autre liste de variables de même longueur que la précédente et différente.

Sortie : un polynôme

Traitement effectué : Le polynôme obtenu en sortie est le déterminant simplifié de la matrice M de Dixon–Biard généralisée. Nous faisons correspondre ici la nième variable V de la première liste, à la nième variable U de la deuxième liste. En général, nous avons comme nom de variable pour U : V!#

Exemples

```
> detDCalc ( {x-t^2, y-t}, {t}, {t!#} );
 - T*Y + T*T# + X - Y*T#
> detDCalc ({x^2+y^2-R^2, x^2+y^2+z^2 - R^2, z - u}, {R, u}, {R!#, u!#} );
 - Z * (R + R#)
```

A.3.7 Procédure "DixonBiard"

Entrées : une liste de N+1 polynômes, une liste N de variables

Sorties : une liste contenant vg, matT, vd

Traitement effectué : Elimination des N variables parmi les N+1 polynômes du système afin d'obtenir une équation implicite du système.

Exemples

- Exemple de la demi-droite paramétrée par t qui admet pour équation implicite $x=y$

  ```
  > DixonBiard ( { x-t^2, y-t}, {t} );
  - X + Y
  ```

- Condition nécessaire d'existence d'une intersection d'un cercle et d'une droite :

  ```
  > DixonBiard ({x^2+y^2-R^2, y-a*x-b}, {x, y});
  B - A * R
  ```

- Condition nécessaire d'existence d'une intersection d'une hyperbole et d'un cercle :

  ```
  > DixonBiard ( {x*y - c, x^2+y^2-R^2}, {x, y});
  C
  ```

- Exemple de recherche de l'intersection d'un cercle de rayon R centré à l'orgine et appartenant au plan (O, x, y), d'une sphère centrée à l'origine de rayon U, et du plan x+y+z=0 :

  ```
  > DixonBiard ( {x^2+y^2-R^2, x^2+y^2+z^2 - U^2, x+y+z }, {R, U} );
  X**4 + 4*X**3*Y + 4*X**3*Z + 6*X**2*Y**2 + 12*X**2*Y*Z +
  6*X**2*Z**2 + 4*X*Y**3 + 12*X*Y**2*Z + 12*X*Y*Z**2 +
  4*X*Z**3 + Y**4 + 4*Y**3*Z + 6*Y**2*Z**2 + 4*Y*Z**3 + Z**4
  ```

  Un simple contrôle de la matrice T en faisant `matT` permet de voir que l'équation obtenue précédemment est en fait :

  $(x+y+z)^4=0$

- Exemple d'élimination de six paramètres u1, u2, u3, u4, u5, u6 :
  > DixonBiard ({x1–u1*u2+u3+1, x2–u2*u3+u4+1, x3–u3*u4+u5+1, x4–u4*u5+u6+1, x5–u5*u6+u1+1, x6–u6*u1+u2+1 } , {u1, u2, u3, u4, u5, u6});
  X2*X4**2*X5*X6 + X2*X4**2*X5 + X2*X4**2*X6 + X2*X4**2 + 2*X2*X4*X5*X6 + 2*X2*X4*X5 + 2*X2*X4*X6 + 2*X2*X4 + X2*X5*X6 + X2*X5 + X2*X6 + X2 + X4**2*X5*X6 + X4**2*X5 + X4**2*X6 + X4**2 + 2*X4*X5*X6 + 2*X4*X5 + 2*X4*X6 + 2*X4 + X5*X6 + X5 + X6 + 1

A.3.8 Programme Reduce

Les % commentent une ligne, et les points d'exclamations précèdent un caractère spécial.

```
% Méthode de Dixon–Biard généralisée

if not numberp !%in!% then
      begin
      !%in!%:=0;
      in "Bareiss";
      end;

procedure myCoeff(poly, var, prod);
comment
Procedure similaire a la commande Reduce coeff(poly,var) sauf que nous
supprimons les coefficient nuls et nous leur associons le monôme var^i multiplié
par un terme constant "prod".
    Sortie: une liste de deux elements:
            1) la liste des coefficients
            2) la liste des monômes fois prod;
begin
scalar i, cff, elt, cffct, monom;
cffct:={};
monom:={};
cff:=coeff(poly, var);
for i:=lowpow!* : hipow!* do
     if (elt:=part(cff, i+1)) neq 0
     then  begin
           cffct:=elt.cffct;
           monom:=(var^i*prod).monom
           end;
return {cffct, monom};
```

```
end;

procedure galCoeff(poly, varListe);
comment
Généralise myCoeff sur une liste de variables.
Extraction de tous les monômes et coefficients possibles employant au moins
une variable de varListe;
begin
scalar res, var, cff, cffct, monom, rescff, resmono;
res:=myCoeff(poly, first(varListe),1);
varListe:=rest(varListe);
while varListe neq {} do
      begin
      var:=first(varListe);
      varListe:=rest(varListe);
      cffct:=first(res);
      monom:=second(res);
      rescff:=resmono:={};
      for i:=1 : length(cffct) do
            begin
           cff:=myCoeff( part(cffct,i), var, part(monom,i));
            rescff:=append(first(cff),rescff);
           resmono:=append(second(cff),resmono);
            end;
     res:={rescff, resmono};
      end;
return res;
end;

procedure ajoutListe(elt, liste);
begin
scalar et;
et:=0;
for each obj in liste do
     if obj=elt then et:=1;
if et=0 then liste:=elt.liste;
return liste;
end;

procedure PPCMdeListes(listeDeListe);
comment
```

```
Calcul de la plus petite liste commune à toutes
les listes contenue dans listeDeListe;
begin
scalar ppcm, elt, obj;
ppcm:=first(listeDeListe);
for each elt in rest(listeDeListe) do
      for each obj in elt do
           ppcm:=ajoutListe(obj,ppcm);
return ppcm;
end;


procedure supCoeff(poly, var1Liste, var2Liste);
comment
Réécriture du polynome poly sous forme
d'un produit vecteur1–matriceT–vecteur2
Le vecteur1 (resp. vecteur2) contient tous les monômes
formés des variables de var1Liste (resp. var2Liste)
et la matriceT contient l'ensemble des coefficients;
begin
scalar i, j, ligne, monom, coeffct,
        gcff, vect2, temp, vect1, coeff1;
%
%       Calcul de vecteur 1 et 2
%       matT est sous forme condensée dans temp
%
vect1:=galCoeff(poly,var1Liste);
coeff1:=first(vect1);
vect2:=temp:={};
for i:=1 : length(coeff1) do
      begin
      gcff:=galCoeff(part(coeff1,i),var2Liste);
      temp:=  gcff.temp;
      vect2:=second(gcff).vect2;
      end;
vect2:=PPCMdeListes(vect2);
%
%       Expansion de temp en la matrice (globale) matT
%
matrix matT(length(temp),inutile:=length(vect2));
i:=0;
```

```
for each ligne in temp do
      begin
      i:=i+1;
      coeffct:=first(ligne);
      monom:=second(ligne);
      while monom neq {} do
            begin
            j:=1;
            terme:=first(monom);
           while (terme neq part(vect2,j)) and (j<inutile)
                 do   j:=j+1;
           if terme neq part(vect2,j)
                 then rederr "supCoeff: le test inutile a servi !";
           matT(i,j):=  first(coeffct);
            coeffct:=rest(coeffct);
            monom:=rest(monom);
             end;
      end;
%
% Mise sous format matrice des vecteurs gauche et droite
%
vect1:=reverse(second(vect1));
matrix vg(1,length(vect1)), vd(length(vect2),1);
for j:=1 : length(vect1) do
vg(1,j):=part(vect1,j);
for i:=1 : length(vect2) do
     vd(i,1):=part(vect2,i);
return {vg, matT, vd};
end;


procedure detDCalc(fList,vList,vConjList);
comment
Construit la matrice initiale sur laquelle
nous effectuons le premier calcul de déterminant;
begin
scalar n,i,j;
n:=length(fList);
%
% Construction de la matrice de Dixon
%
```

```
matrix M(n,n);
for i:=1:n do m(i,1):=part(fList,i);
for j:=2:n do for i:=1:n do
                              M(i,j):=sub(part(vList,j-1)=part(vConjList,j-1), M(i,j-1));
%
% Simplification du déterminant par (x-x#)
%
return det(M)/(for i:=1:<<n-1>> product part(vList,i)-part(vConjList,i));
end;

Procedure Dixon(polyListe,varListe);
begin
scalar varConjListe;
%
% Variables conjuguées
%
VarConjListe:=for each var in varListe collect mkid(var,!#);
%
% Calcul du déterminant de la matrice de Dixon
%
Poly:=detDCalc(polyListe,varListe,varConjListe);
return supCoeff(poly,varListe,varConjListe);
end;
```

A.4 Algorithme de Bareiss modifié

Nous nous sommes inspirés de l'algorithme de Bareiss modifié décrit dans Biard [7]. Le seul cas particulier pour lequel cette méthode peut ne pas marcher concerne les matrices diagonales dont le produit des éléments diagonaux non nuls est congru à zéro modulo p.

A.4.1 Procédure "Bareiss"

Entrée : une matrice globale, matM, pas forcément carrée.

Sortie : une liste de deux éléments : le déterminant de la mineure de matM, et la liste des positions (i , j) des pivots employés. De plus, la matrice matM est triangularisée.

Traitement effectué : Application de la méthode de Gauss sur la matrice matM, mais à chaque itération les sous-matrices qui restent à diagonaliser sont divisées par l'avant-dernier pivot. Ceci nous permet

d'extraire une mineure de matM et de connaître son déterminant grâce au dernier pivot non nul employé.

Exemple

```
> matM := MAT( (0, 1, 1, 1), (2, 4, 8, 16) , (3, 9, 27, 81) );
```

$$\text{MATM} := \begin{bmatrix} 0 & 1 & 1 & 1 \\ 2 & 4 & 8 & 16 \\ 3 & 9 & 27 & 81 \end{bmatrix}$$

```
> Bareiss;
{12,{1,2,3}}
> matM;
```

$$\text{MATM} := \begin{bmatrix} 2 & 4 & 8 & 16 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 12 & 54 \end{bmatrix}$$

A.4.2 Programme Reduce

```
% Recherche d'une sous-matrice extraite inversible
% de taille egale au rang d'une matrice donnee (une mineure)

let Bareiss = Bareiss();
procedure Bareiss();
begin
scalar m, n, i, j, pivots, diagonale, k, l, p0, p1, p2, temp;

n:=first(length(matM));
m:=second(length(matM));
i:=j:=1;
% Initialisation du diviseur de Bareiss
p1:=1;
pivots:={}; diagonale:={};
while i<=n and j<=m do
    begin
    k:=i;

    % Recherche d'un pivot non nul
    while k<n and matM(k,j)=0
        do  k:=k+1;
```

```
        if matM(k,j)=0 then j:=j+1% Changement de colonne
        else begin
            pivots:=j . pivots;
            % Permutation de deux lignes
            if k>i
            then for l:=j:m do
                  begin
                  temp:=matM(i,l);
                  matM(i,l):=matM(k,l);
                  matM(k,l):=temp;
                  end;
            % Recherche la présence de pivots non nuls
             temp:=0;
            for l:=k+1 : m do
                if matM(i,l) neq 0 then temp:=1;
            % Application du pivot de Gauss avec division
            if temp=0 then diagonale:= j . diagonale
            else  begin
                % Pour avoir un Gauss habituel, faire p0:=1au lieu de p0:=1/p1;
                 p0:=1/p1;
                p1:=matM(i,j);% Elimination de Gauss avec division
                for  k:=i+1:n  do
                    if  (p2:=matM(k,j))  neq  0
                    then  for  l:=j:m  do
                            begin
                        on modular; % Activation du calcul sur Z/pZ
                        matM(k,l):=( p1*matM(k,l) – p2*matM(i,l) )* p0;
                        off modular; % Desactivation du calcul sur Z/pZ
                        end; % for ... if ...
                 end; % if temp=0 ... else ...
             i:=i+1;
             j:=j+1;
            end; % else
        end; % while

% Multiplication des éléments diagonaux
p1 := p1 * (for each diag in diagonale product diag);
return p1.{reverse pivots};
end;
```

Annexe A

Programmes Reduce

Annexe B

Un éditeur mathématique Grif orienté par la sémantique

Cette annexe décrit un éditeur de formules mathématiques de Grif, qui permet à un utilisateur de rédiger un document tout en éditant des formules mathématiques destinées à être évaluées, manipulées, calculées par un système de calcul symbolique ou numérique. Les résultats de ces calculs s'insèrant à nouveau dans le document. La construction de ces formules mathématiques permet d'en extraire directement leur sémantique, ce qui évite le problème difficile de la génération de la sémantique d'une formule à partir de sa syntaxe. Cette nouvelle version complète celle décrite dans [21].

B.1 Introduction

L'objectif de cette interface de calcul scientifique est de regrouper au sein d'un même environnement l'édition et le calcul de formules mathématiques. L'utilisateur édite sous un traitement de texte un document comportant des formules. Ces dernières peuvent être transmises et évaluées par un système de calcul formel ou numérique, au moyen d'un langage de programmation ou à l'aide d'une bibliothèque scientifique. En retour, le ou les résultats fournis s'insèrent dans le document en cours d'édition.

Les interfaces réalisées à ce jour sont :

- interface Grif / Reduce décrite dans [21],
- interface Grif / ESSL–C.

D'une part, ceci permet à l'utilisateur d'employer un langage naturel – le langage mathématique – comme interface homme–machine pour les calculs tout en rédigeant son document scientifique. D'autre part, ceci lui permet d'effectuer des calculs sans utiliser directement un système ou un langage de calcul.

Ce type d'interface présente plusieurs avantages tels que :

- la restitution d'un cadre de travail habituel pour l'écriture de formules mathématiques,
- la décharge de l'utilisateur de toute utilisation directe d'un système ou langage de calcul. Ce qui lui évite un mode d'utilisation technique d'un système ou langage de calcul, d'où une utilisation rapide pratiquement sans apprentissage et comportant aussi moins d'erreurs de manipulation,
- la récupération sous l'éditeur des résultats des calculs, ce qui évite d'avoir à les réécrire.

Tous ces avantages permettent une rédaction rapide d'un article scientifique qui fait intervenir des calculs mathématiques, symboliques ou numériques.

Pour l'environnement d'édition, nous employons l'éditeur de documents structurés Grif, [34] et [36]. Il présente l'avantage de pouvoir définir des éditeurs spécialisés, des traducteurs vers d'autres formalismes comme TEX, LaTEX, et qui propose plus récemment des outils pour construire de nouvelles applications traitant les documents structurés sous Unix et X–Windows.

Les systèmes de calculs employés sont actuellement :

- Reduce (version 3.4.1 ou 3.5), un système de calcul formel, et
- ESSL (version 4), une bibliothèque de calcul numérique.

Nous pouvons réaliser des calculs par d'autres systèmes formels, numériques, ou par des bibliothèques ou langages différents. Notre choix s'est porté sur Reduce du fait de sa disponibilité, mais nous aurions pu prendre un autre système de calcul formel comme Maple, Mathematica, ... etc.

B.1.1 Architecture de l'interface

Il existe deux catégories d'utilisateurs de cette interface : d'une part l'utilisateur final qui crée et manipule des formules mathématiques en s'appuyant sur l'éditeur Grif de documents structurés et sur les systèmes de calculs, d'autre part, l'administrateur de l'interface de calcul qui l'adapte à de nouveaux besoins ou à d'autres langages.

La figure suivante résume l'architecture de l'interface :

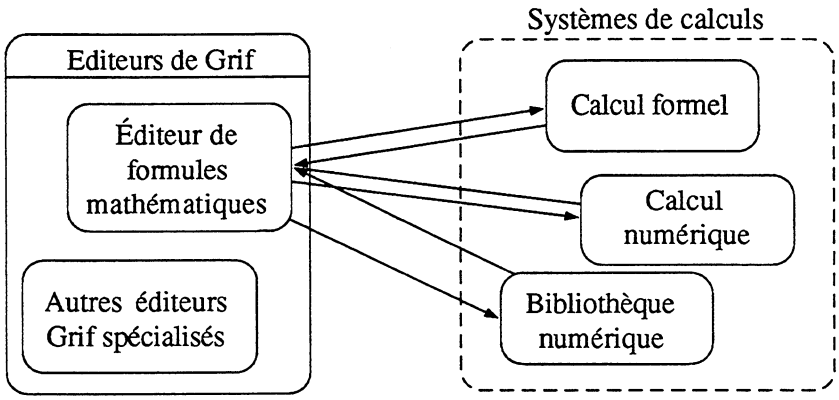


Fig. 2.1 : Architecture de l'interface de calcul scientifique

L'utilisateur final n'a accès qu'à l'éditeur de formules, et lorsqu'il désire réaliser un calcul, il peut suivre les processus de traductions et de calculs en cours. Ceci lui permet d'avoir une trace et éventuellement un diagnostic d'erreur en cas d'échec lors d'une traduction ou d'un calcul. En revanche, l'administrateur du système peut accéder aux systèmes de calculs pour définir de nouvelles fonctionnalités de plusieurs manières : ce peut être soit la définition d'une nouvelle fonction ou algorithme de calcul au sein d'un système de calcul existant, soit l'ajout d'un nouveau système ou langage de calcul.

B.1.2 Comparaison des éditeurs Grif : Maths et GramMath

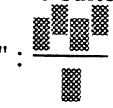
Les éditions de formules mathématiques à l'aide des éditeurs Grif de type Math et GramMath présentent une grande différence : bien que le logiciel Grif ne propose que des éditeurs syntaxiques, l'éditeur Math propose des constructions mathématiques d'ordre syntaxique, tandis que GramMath propose quant à lui, des constructions d'ordre sémantique. En effet, sous l'hypothèse d'avoir écrit une formule mathématique correcte d'un point de vue syntaxique, la représentation interne d'une formule de type Math ne permet pas toujours d'en extraire son contenu sémantique contrairement à une formule de type GramMath. L'une est orientée par la syntaxe tandis que l'autre est plus orientée par la sémantique des constructions mathématiques.

B.1.2.1 Exemple d'édition Math

Nous notons entre guillemets les constructions proposées par les différents éditeurs de formules.

Par exemple, l'écriture de la dérivée partielle en x et y de : $|y|+yx^3$, nécessite les constructions Math suivantes :

1. une "Fraction", qui crée deux boîtes, le "Numérateur" et le "Dénominateur", comme suit : ▒ ▒

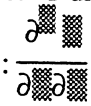
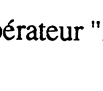
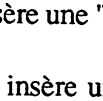
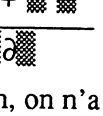
2. au "Numérateur", on insère une suite de deux éléments de type "Simple_Texte" suivis de l'option "Exposant" : 

L'utilisateur n'a plus qu'à choisir les polices employant le symbole mathématique «∂», et un style italique pour les variables. Ces constructions fournissent finalement la représentation graphique suivante :

$$\frac{\partial^2 |y| + yx^3}{\partial x \partial y}$$

B.1.2.2 Exemple d'édition GramMath

L'écriture de cette même dérivée partielle sous l'éditeur GramMath demandera les constructions suivantes :

1. une "Derivees", qui crée deux boîtes, en haut une "fonction", et en bas un "objet_derivee_part", comme suit : 
2. l'insertion des options "ordre", et d'une deuxième variable de dérivation "objet_derivee_part", ce qui donne : 
3. dans la "fonction", insertion de l'opérateur "Addition" : 
4. pour le premier "objet_add", on insère une "ValeurAbsolue" : 
5. et pour le second "objet_add", on insère une "Multiplication" dont le deuxième terme est une "Puissance" : 

Pour chacun des éléments de la construction, on n'a plus qu'à choisir si l'on y place une "Constante" ou une "Variable". Les premières restent en style normal, tandis que les suivantes sont en italiques. Au bout du compte, nous obtenons la même formule mathématique qu'en d'un point de vue représentation graphique :

$$\frac{\partial^2 |y| + y x^3}{\partial x \partial y}$$

B.1.2.3 Comparaison des représentations internes

Sur les exemples précédents, nous avons vu que pour une même formule mathématique, on emploie des constructions différentes. Afin de pouvoir réécrire une formule vers d'autres formalismes, nous devons, dans la plupart des cas, connaître son contenu sémantique. Mais extraire le contenu sémantique d'une formule de type Math est un problème très difficile,

comme le montre Bouhier dans son étude réalisée en Prolog [8]. En effet, pour certaines formules mathématiques, plusieurs interprétations sont possibles. Une manière de contourner cette difficulté est d'établir un dialogue avec l'utilisateur pour guider, préciser le contexte, voire se substituer à l'analyseur sémantique. Une autre manière d'éviter le problème de la génération de la sémantique, est d'écrire directement les formules mathématiques dans un langage qui précise leur sémantique, ici GramMath. Les formules GramMath permettent ainsi des réécritures directes vers d'autres langages.

Tout document Grif est représenté de façon interne par un arbre abstrait. Ainsi, l'arbre de la formule Math de B.1.2.1 comporte des informations de nature syntaxique :

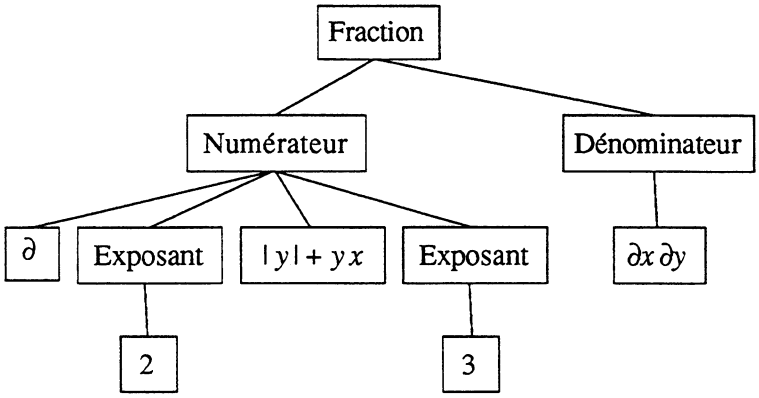


Fig. 2.2 : Structure interne d'une formule Math

Par contre, l'arbre de la formule GramMath de B.1.2.2 comporte des informations de nature sémantique :

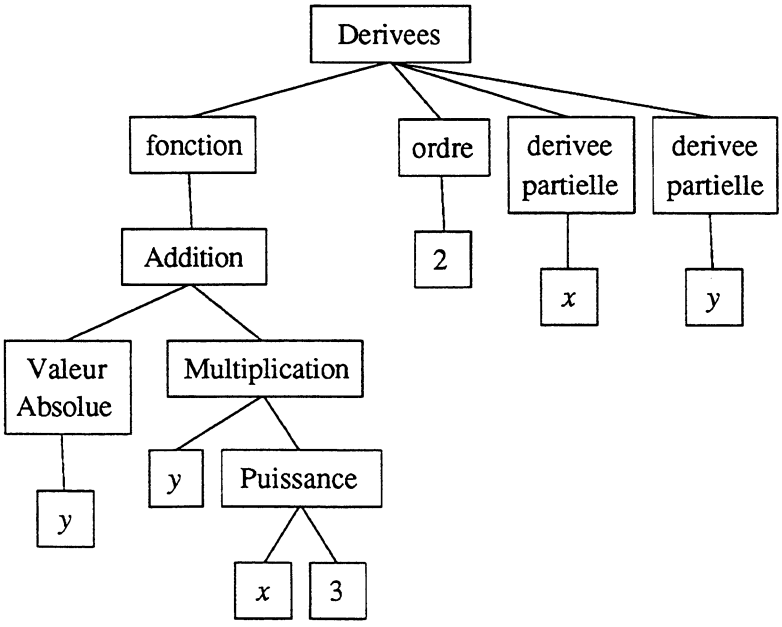


Fig. 2.3 : Structure interne d'une formule GramMath

B.2 Edition de formules mathématiques

Les formules mathématiques que l'on désire évaluer doivent être éditées sous l'éditeur GramMath de Grif. Cet éditeur comporte une grande différence avec l'éditeur existant Math de Grif, car il propose des constructions sémantiques au lieu de constructions syntaxiques. Ceci permet d'avoir directement une interprétation sémantique de toute formule, et de la traduire vers d'autres formalismes. Cependant, l'édition d'une formule GramMath devient moins naturelle puisque l'écriture n'est plus linéaire de la gauche vers la droite. Elle nécessite d'avoir une vision globale des formules avant de préciser leur contenu. La sous-section suivante aborde et précise ces différences d'éditions.

B.2.1 Edition GramMath

Nous avons déjà eu un aperçu d'édition à la section précédente. Ici, nous présentons les possibilités offertes par l'éditeur GramMath avec quelques exemples, puis nous nous intéressons à l'édition proprement dite.

B.2.1.1 Règles d'éditions de GramMath

Chacune de ces règles crée une ou plusieurs boîtes vides. Ces boîtes sont de deux types : "Texte" et "Expression". Lorsqu'on a une boîte de type "Expression", on peut à nouveau insérer une nouvelle règle de construction. Au plus haut niveau, les règles d'éditions sont la définition d'objets de base de type "Expression", hormis les règles "Arithmetique" et "Operateurs" qui renvoient vers un autre choix de règles.

Règles *Expression*

1.	**Constante**	crée une boîte texte : ▒
2.	**Variable**	crée du texte en italique : ▒
3.	**Fonction**	crée une fonction : ▒(▒), avec la possibilité d'insérer plusieurs arguments à la fonction : ▒(▒,▒,▒)
4.	**Equation**	crée l'équation ▒ = ▒
5.	**Vecteur_ou_Matrice**	crée une matrice de dimension 1×1 :(▒), puis nous pouvons insérer des éléments en colonne (▒ / ▒), ou rajouter une colonne :(▒ ▒)
6.	**Liste**	crée une liste : {▒}, avec la possibilité d'insérer plusieurs éléments : {▒,▒,▒}
7.	**Liste_Verticale**	crée une liste verticale : {▒}, avec la possibilité d'insérer plusieurs éléments : {▒ / ▒}
8.	**Arithmetique**	fait appel au règles arithmétiques

9. **Operateurs** — fait appel à des opérateurs mathématiques
10. **Affectation** — crée une affectation : ▒ ← ▒
11. **Regle** — crée une définition de fonction, ou une réécriture d'expressions mathématiques : ▒ → ▒

Règles *Arithmétiques*

Les opérations arithmétiques sont :

1. **Addition** — crée une addition de deux termes : ▒ + ▒, nous pouvons rajouter de nouveaux termes additifs : ▒ + ▒ + ▒ + ▒ + ▒
2. **Signe_moins** — crée un signe moins : −▒
3. **Soustraction** — crée une soustraction : ▒ − ▒
4. **Multiplication** — crée une multiplication de deux termes : ▒ ▒, on peut rajouter de nouveaux termes multiplicatifs : ▒ ▒ ▒ ▒
5. **Fraction** — crée une fraction : $\frac{▒}{▒}$
6. **Racine** — crée une racine : $\sqrt{▒}$
7. **ValeurAbsolue** — crée une valeur absolue : $|▒|$
8. **Transposee** — crée une transposée : $▒^t$

Règles *opérateurs*

Les opérateurs mathématiques définis sont :

1. **Evaluation** — crée une évaluation, par exemple $[\ f(x)]_{x=0}$. Afin d'éviter toute confusion possible avec la règle précédente Equation, la substitution $x=0$ est notée $x \leftarrow 0$. Nous avons donc : $\left[▒ \right]_{▒ \leftarrow ▒}$. Nous pouvons définir d'autres substitutions pour l'évaluation : $\left[▒ \right]_{▒ \leftarrow ▒}$ $▒ \leftarrow ▒$
2. **Primitive** — crée une primitive : $\int ▒\ d▒$
3. **Integrale** — crée une intégrale avec bornes : $\int_{▒}^{▒} ▒\ d▒$
4. **Sommation** — crée l'opérateur de sommation. Pour préciser que les systèmes de calculs effectuent une affectation sur la variable de sommation,

l'égalité est remplacée par un symbole d'affectation : $\sum_{\blacksquare \leftarrow \blacksquare}^{\blacksquare} \blacksquare$

5. **Produit** crée un produit. Pour la même raison que la sommation, nous notons l'égalité par un symbole d'affectation : $\prod_{\blacksquare \leftarrow \blacksquare}^{\blacksquare} \blacksquare$

6. **Derivees** crée une ou plusieurs dérivées partielles : $\frac{\partial^{\blacksquare}}{\partial \blacksquare}$, nous pouvons insérer l'ordre total, ainsi que de nouvelles variables de dérivation avec un ordre partiel : $\frac{\partial^{\blacksquare} \blacksquare}{\partial \blacksquare \partial \blacksquare^{\blacksquare} \partial \blacksquare^{\blacksquare}}$

B.2.1.2 Modifier une formule GramMath

On peut modifier une formule en changeant le contenu des boîtes "Texte", mais aussi changer leur structure. Pour une formule GramMath qui a tous ses éléments de type "Expression" définis, les modifications possibles sont :

1. enlever une structure existante pour y placer une autre, ou
2. insérer une structure entre deux structures existantes.

La première modification peut se réaliser au moyen d'un couper, d'un détruire de l'expression comme nous avons vu à la fin de la section précédente.

En revanche, si nous voulons insérer une nouvelle structure entre deux autres, nous avons à distinguer deux cas :

1. une insertion dans une liste, ce qui est immédiat, et
2. une insertion « verticale », moins facile que nous illustrons par l'exemple ci–dessous.

Soit le polynôme suivant :

$$x^7 - 2x^5 - 2x^4 - 2x^3 + x^2 + x + 1$$

où l'on désire parenthéser les termes négatifs, puis les monômes de degré inférieur ou égal à 2 afin d'avoir :

$$x^7 + \left(-2x^5 - 2x^4 - 2x^3\right) + \left(x^2 + x + 1\right)$$

Si l'on étudie l'arbre de construction du polynôme à modifier :

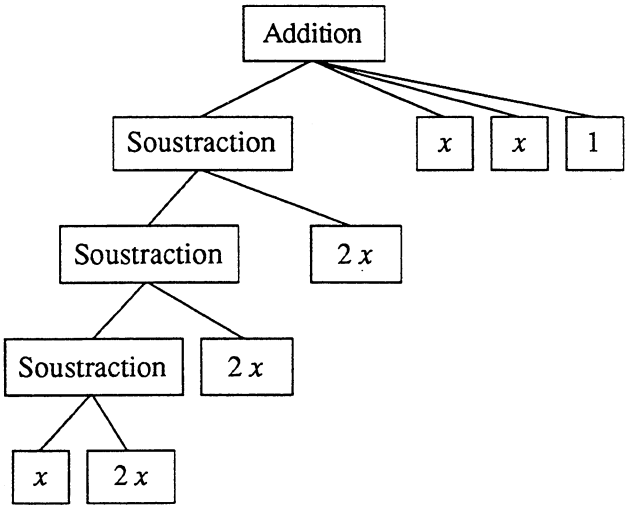


Fig. 2.4 : Arbre représentant $x^7 - 2x^5 - 2x^4 - 2x^3 + x^2 + x + 1$

Nous aurons à insérer de nouveaux nœuds, nœuds ayant un cadre ombré, de manière à obtenir parmi plusieurs arbres possibles, un tel arbre :

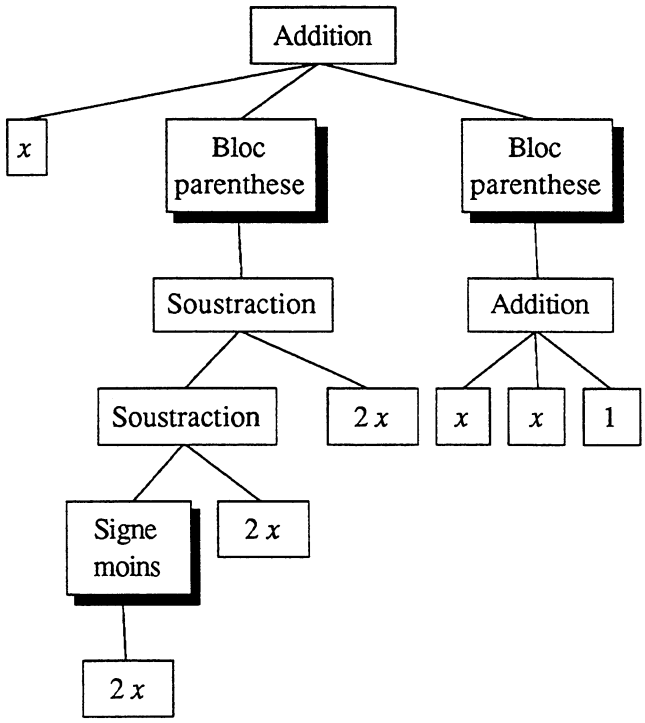


Fig. 2.5 : Arbre représentant $x^7 + \left(-2x^5 - 2x^4 - 2x^3\right) + \left(x^2 + x + 1\right)$

Une des manières de procéder est de suivre les étapes suivantes :

- sélectionner le nœud Soustraction contenant toutes les autres Soustractions,
- le couper,

- insérer un terme de l'addition avant, de type BlocParenthèse, n'y mettre aucune construction,
- coller à l'intérieur le nœud Soustraction.

Nous aurons alors la formule :

$$\left(x^7 - 2x^5 - 2x^4 - 2x^3\right) + x^2 + x + 1$$

Reste à séparer le monôme de tête du bloc de parenthèse. Il suffit de :

- sélectionner ce monôme, ici ce sera une Puissance,
- le couper,
- puis sélectionner le bloc de parenthèse au niveau "objet_add" afin de coller le nouveau terme additif.

$$x^7 + \left(-2x^5 - 2x^4 - 2x^3\right) + x^2 + x + 1$$

Nous avons terminé avec le premier parenthésage. Il est à noter que nous avons laissé en blanc le terme gauche de la soustraction. Ici, nous pouvons remplacer cette soustraction incomplète par la construction "Signe_moins". A la section B.2.1.5 nous verrons que certaines constructions peuvent rester incomplètes.

Finalement, le dernier parenthésage est plus simple. En effet, on doit :

- sélectionner le premier terme additif jusqu'au dernier,
- le couper,
- insérer un nouveau terme pour l'addition de type BlocParenthèse, ne rien mettre comme construction,
- coller à l'intérieur la sélection.

$$x^7 + \left(-2x^5 - 2x^4 - 2x^3\right) + \left(x^2 + x + 1\right)$$

Nous avons finalement les parenthésages voulus. Lorsque les modifications sont trop importantes, il est préférable de construire la structure dans laquelle viendront s'insérer des éléments par le copier/coller. Pour illustrer notre propos, supposons que l'on veuille factoriser les termes ayant le facteur $-2\,x$. Nous construisons une addition ayant pour premier terme une soustraction :

$$\square - \square + \square$$

Nous copions/collons le monôme de tête dans la première boîte, puis les monômes restants qui ne contiennent pas le facteur $-2\,x$ dans la dernière boîte. Nous avons :

$$x^7 - \square + \left(x^2 + x + 1\right)$$

Dans la boîte restante, nous y définissons une multiplication de trois termes : une constante, une puissance, et un bloc parenthèse :

$$x^7 - 2x^3(\square) + \left(x^2 + x + 1\right)$$

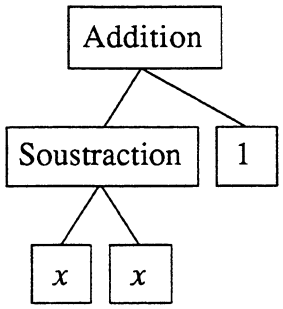
Il ne reste plus qu'à copier/coller les monômes correspondants dans la parenthèse afin d'obtenir :

$$x^7 - 2x^3\left(x^2+x+1\right) + \left(x^2+x+1\right)$$

B.2.1.3 Manipuler les soustractions

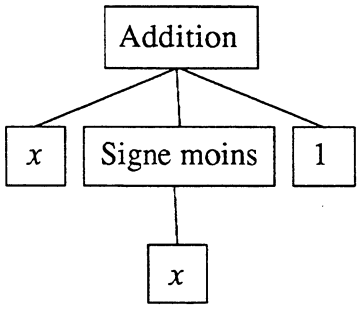
L'opérateur de soustraction peut poser quelques problèmes lorsqu'elles appararaissent plus d'une fois sur une même ligne. Une première solution, pour éviter tout risque d'erreur, est de parenthéser systématiquement le terme droit de la soustraction. Nous proposons un outil de parenthésage automatique en B.2.1.4. Mais ce genre d'écriture peut s'avérer peu pratique lorsqu'on doit manipuler plusieurs soustractions et additions imbriquées. Une autre solution, est de combiner l'addition et le signe moins. Ceci présente un avantage d'un point de vue de manipulation des différents termes, mais d'un point de vue présentation graphique nous aurons les symboles +– qui se suivent. Une fois les manipulations réalisées, les expressions ainsi éditées peuvent être réécrites avec l'opérateur soustraction pour avoir une présentation graphique correcte. La commande Unix qui permet de telles réécritures se nomme *soustraction*.

Par exemple, si l'on veut écrire le polynôme : $x^2 - x + 1$, la première façon est de construire une addition, puis pour le terme de gauche insérer une soustraction; la deuxième méthode est de construire une addition de trois termes, avec l'insertion du signe poins pour le terme du milieu. Les arbres suivants montrent ces deux constructions :



Arbre représentant la formule :

$$x^2 - x + 1$$



Arbre représentant la formule :

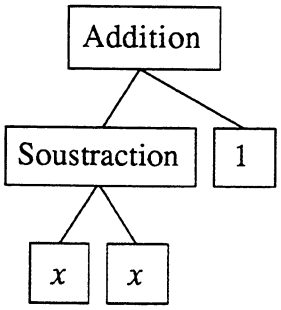
$$x^2 + -x + 1$$

Ainsi, si l'on doit pour une raison ou une autre, réécrire ce polynôme selon les puissances croissantes, nous pourrons difficilement manipuler l'arbre comportant la soustraction (voir la section B.2.1.2). En revanche, les termes de l'addition peuvent facilement être permutés au moyen de copier(s) et coller(s).

B.2.1.4 Parenthésage automatique

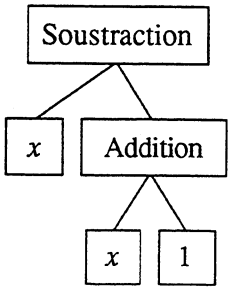
Il se peut que nous éditions une formule mathématique qui admette une représentation interne différente de sa représentation graphique. Ainsi, lorsque nous écrivons : $x^2 - x + 1$, nous interprètons cette expression comme étant : $\left(x^2 - x\right) + 1$, mais sa représentation interne peut être différente. En effet, nous pouvons construire cette formule de deux manières selon que nous construisons l'addition ou la soustraction en premier.

Les arbres suivants montrent ces deux constructions possibles :



Arbre représentant la formule :

$$\left(x^2 - x\right) + 1$$



Arbre représentant la formule :

$$x^2 - (x + 1)$$

La première construction a une représentation interne qui correspond à l'interprétation de l'utilisateur, tandis que la seconde donne une interprétation différente de celle de l'utilisateur.

Un moyen de contrôler si l'aspect interne de la formule est en accord avec notre interprétation visuelle est de réaliser un parenthésage des constructeurs, ce qui permet de faire apparaître la représentation interne des formules. Nous proposons un outil qui insère automatiquement les parenthèses lorsque cela est nécessaire. Ce parenthésage automatique est réalisé au moyen de la commande Unix *parenthese*.

Soit par exemple le polynôme suivant construit par l'utilisateur :

$$P(x, y, z) = -a\,x + b + y + z - c + -x$$

La commande *parenthese* fournit le résultat suivant qui précise la représentation interne de la formule :

$$P(x, y, z) = -a\left(x + \left(b + y\right) + z\right) - \left(c + \left(-x\right)\right)$$

L'exemple qui suit montre quelques cas de « sur-parenthésages » dus à la commande *parenthese*, la formule suivante :

$$1 + \frac{2 - \sqrt{3 + 4}}{5 - 6 - 7}$$

donne :

$$1 + \frac{\left(2 - \sqrt{\left(3 + 4\right)}\right)}{\left(\left(5 - 6\right) - 7\right)}$$

B.2.1.5 Quelques raccourcis

Nous pouvons éviter dans certains cas l'emploi des règles de constructions de formules GramMath. Par exemple, si une formule ne fait intervenir que des additions, soustractions, multiplications et parenthèses, nous pourrons écrire un objet de type "Variable" qui contiendra en fait le texte suivant :

$$P(x, y, z) = -a(x + b + y + z) - c + x$$

Cependant, si nous voulons qu'une telle expression soit traduite vers un autre formalisme, nous devrons éventuellement vérifier si les traductions obtenues sont correctes.

Un autre raccourci est d'amputer une construction d'un de ses éléments sans la remplacer par une structure adéquate. Ainsi avons–nous, à la section B.2.1.2, supprimé le terme gauche d'une soustraction sans la remplacer par un signe moins.

Les constructions qui peuvent rester incomplètes sont :

- les soustractions, qui peuvent avoir soit le terme gauche, soit le terme droit, absent;
- les matrices, qui peuvent comporter des colonnes de tailles inégales, les éléments manquant, c'est–à–dire ceux du bas de la colonne, sont mis à zéro;
- les dérivées, qui peuvent ne pas comporter d'ordre total.

B.2.2 Exemples d'éditions GramMath

Nous donnons ci–dessous quelques exemples de formules éditées dans le langage GramMath de Grif.

B.2.2.1 Définitions de fonctions

La construction "Règle" permet de manière générale de définir une règle de calcul ou de réécriture. Nous pouvons en particulier définir une fonction. Les arguments de la fonction qui sont de type "Variable" permettent de définir la fonction quelque soient ces variables. Cependant, les arguments qui sont de type "Constante" définissent la fonction pour une constante donnée. Ce qui permet de distinguer une constante d'une variable dans une formule, est que les premières sont en style Roman, et les suivantes en italiques.

Par exemple, soit la fonction f suivante :

$$f(x, \mathrm{a}) \rightarrow \sqrt{\frac{x^2 - \mathrm{a}^2}{x^4 + 1}}$$

Ici, la variable sera "x" tandis que "a" est une constante. Ainsi, nous n'avons pas défini la fonction $f(x, 0)$.

Voici d'autres exemples de définition de fonctions :

1. $\mathrm{C}(n, p) \rightarrow \prod_{j \leftarrow 1}^{p} \frac{n+1-j}{j}$
2. $\Gamma(z) \rightarrow \int_0^{\infty} t^{z-1} \mathrm{e}^{-t} \mathrm{d}t$

B.2.2.2 Equations

Une équation exprime simplement une égalité entre deux formules, à ne pas confondre avec une affectation ou une définition de fonction. Ainsi, la formule de Wallis pour le calcul de $\pi \div 2$ s'écrit :

$$\frac{\pi}{2} = \prod_{n \leftarrow 1}^{\infty} \frac{1}{1 - \frac{1}{4n^2}}$$

B.2.2.3 Affectations

Cette construction fait partie du langage algorithmique, cependant les langages Reduce et C autorisent l'emploi de l'affectation au sein d'une expression algébrique, arithmétique ou logique, aussi l'avons–nous incluse.

Nous avons par exemple l'affectation :

$$z \leftarrow \ln\left(x^2 + y^2\right),$$

mais aussi des affectations multiples comme :

$$x \leftarrow y \leftarrow \frac{\sqrt{2}}{2}$$

Pour accélérer les calculs, pour avoir des résultats intermédiaires, nous pouvons définir des variables Y, Y' et Y'' à l'aide d'affectations dans une expression comme suit :

$$-\left(1+x^2\right)\left[Y \leftarrow \frac{\sin(x)}{1-x^2}\right] - 4x\left[Y' \leftarrow \frac{\partial Y}{\partial x}\right] + \left(1-x^2\right)\left[Y'' \leftarrow \frac{\partial^2 Y}{\partial x^2}\right]$$

B.2.2.4 Listes

La construction liste permet de définir des ensembles ordonnés d'objets. Nous proposons deux types de listes, l'une horizontale, l'autre verticale, qui ont une présentation différente. Voici un exemple de liste horizontale :

$$\{2, 3, 5, 7, 11, 13, 17, 19\}$$

et de liste verticale :

$$\left\{ \begin{array}{l} x^2 + y^2 = z^2 \\ xy + z^3 = 1 \\ x^3 - y^3 = z^2 \end{array} \right\}$$

Nous pouvons aussi employer les deux types de constructions pour des listes imbriquées :

$$\left\{ \left\{ \begin{array}{l} x = 0 \\ y = 1 \end{array} \right\}, \left\{ \begin{array}{l} x = \mathrm{i} \\ y = -1 \end{array} \right\}, \left\{ \begin{array}{l} x = -1 \\ y = 1 \end{array} \right\} \right\}$$

B.2.2.5 Calcul intégral

Nous disposons de deux règles : "Primitive" et "Integrale". Voici quelques exemples de primitives :

1. $\int x\sqrt{1+x^2}\,dx$
2. $\int \sqrt{1+x^2}\,dx$
3. $\int e^{x^2}\,dx$
4. $\int \frac{\log(x)}{x^2}\,dx$
5. $\int x\,e^{x}\log(x)\,dx$
6. $\int e^{\frac{1+\log\left(z^3+1\right)}{2}}\,dz$
7. $\int\int e^{x^2+y^2}\,dx\,dy$

De la même manière, nous avons les intégrales :

1. $\int_0^{10} x\sqrt{1+x^2}\,dx$
2. $\int_0^{2\pi}\int_0^{\sqrt{x^2+y^2}} e^{\left(\rho\,e^{i\theta}\right)^2}\,d\rho\,d\theta$

B.2.2.6 Equations différentielles

En utilisant la règle "Equation" puis "Derivees", nous écrirons ces équations différentielles:

1. $\left(1-x^2\right)\frac{\partial^2 y}{\partial x^2}-4x\frac{\partial y}{\partial x}-\left(1+x^2\right)y=0$
2. $\left(1-x^2\right)\frac{\partial y}{\partial x}=1+xy$
3. $x\frac{\partial y}{\partial x}=x^n+n\,y$

4. $\left(\frac{\partial z}{\partial x}\right)^2 + \left(\frac{\partial z}{\partial y}\right)^2 = 4\,e^{-z}$

Sans exprimer d'égalité, nous écrivons :

1. $\frac{\partial \ln\left(\sqrt{\frac{1+x}{1-x}}\right)}{\partial x}$
2. $\frac{\partial\left(\frac{1+\sin(x)}{1+\cos(x)}\right)}{\partial x}$
3. $\frac{\partial \sin(|x|)}{\partial x}$
4. $\frac{\partial^n\left(\left(x^2-1\right)\frac{\partial y(x)}{\partial x} + x\,y(x) + 1\right)}{\partial x^n}$

B.2.2.7 Evaluations

L'évaluation permet d'évaluer une fonction en attribuant des valeurs ou des expressions aux variables de la fonction. Ainsi : $\left[z^2 + z + 1\right]_{z \leftarrow x^2}$

Nous pouvons par cette construction vérifier si les solutions suivantes vérifient bien les équations différentielles précédentes :

1. $\left[\left(1-x^2\right)\frac{\partial^2 y}{\partial x^2} - 4x\frac{\partial y}{\partial x} - \left(1+x^2\right)y = 0\right]_{y \leftarrow \frac{\sin(x)}{1-x^2}}$
2. $\left[\left(1-x^2\right)\frac{\partial y}{\partial x} = 1 + xy\right]_{y \leftarrow \frac{\arcsin(x)}{\sqrt{1-x^2}}}$
3. $\left[x\frac{\partial y}{\partial x} = x^n + n\,y\right]_{y \leftarrow x^n \mathrm{Ln}(x)}$
4. $\left[\left(\frac{\partial z}{\partial x}\right)^2 + \left(\frac{\partial z}{\partial y}\right)^2 = 4\,e^{-z}\right]_{z \leftarrow \ln\left(x^2 + y^2\right)}$

B.2.2.8 Calcul matriciel

Les vecteurs ou les matrices sont construits à partir de la même règle "Vecteur_ou_Matrice". La seule opération mathématique spécifique aux matrices que nous avons retenue est la construction "Transposee". Mais, nous pouvons faire des produits vecteur–vecteur, vecteur–matrice, matrice–matrice, des calculs d'inverses, de puissance, ... etc.

$$\begin{pmatrix} 1 & 4 & 7 \\ -4 & 1 & -2 \\ 0 & 2 & 1 \end{pmatrix} \overset{t}{\begin{pmatrix} x \\ y \\ z \end{pmatrix}} = 0$$

Nous pouvons calculer un déterminant, par exemple :

$$M \leftarrow \begin{pmatrix} 2a & b & c & d \\ b & 2a^p & b^p & c^p \\ c & b^p & 2a^{p^2} & b^{p^2} \\ d & c^p & b^{p^2} & 2a^{p^3} \end{pmatrix}$$

$$\det(M)$$

L'exemple qui suit est relatif aux matrices de spin de Pauli qui interviennent dans la mécanique quantique, ou au groupe de Lie SU(2). Nous définissons une fonction j, qui, selon la constante fournie, retourne telle ou telle matrice :

$$\left\{ \begin{array}{l} j(x) \rightarrow \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ j(y) \rightarrow \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \\ j(z) \rightarrow \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \end{array} \right\}$$

B.2.2.9 Sommes et produits finis

Les règles "Sommation" et "Produit" permettent de définir des sommes et produits finis. Soit la formule de Li Shanlan où $l \geq k \geq 0$ et l, k, j, x sont des entiers naturels :

$$\sum_{j \leftarrow 0}^{k} C(k,j)\ C(l,j)\ C(x+k+l-j, k+l) = C(x+k, k)\ C(x+l, l)$$

Et pour le produit, nous avons par exemple la factorisation d'un polynôme P :

$$P(x) = x \prod_{k \leftarrow 1}^{n} x^2 + k^2$$

B.2.2.10 Définitions de règles de calculs

Nous pouvons employer la construction "Règle" qui permet non seulement de définir une fonction, mais permet aussi de définir des règles de calculs. Par exemple, nous pouvons en employant la construction "Liste_Verticale" définir des règles de calcul trigonométrique :

$$Trigo \leftarrow \left\{ \begin{array}{l} \cos(x)^2 \rightarrow \dfrac{1+\cos(2x)}{2} \\ \sin(x)^2 \rightarrow \dfrac{1-\cos(2x)}{2} \\ \cos(x)\ \cos(y) \rightarrow \dfrac{\cos(x+y)+\cos(x-y)}{2} \\ \cos(x)\ \sin(y) \rightarrow \dfrac{\sin(x+y)-\sin(x-y)}{2} \\ \sin(x)\ \sin(y) \rightarrow \dfrac{\cos(x-y)-\cos(x+y)}{2} \end{array} \right\}$$

B.3 Evaluation de formules GramMath

Nous illustrons les possibilités offertes par la commande grifred de l'interface Grif/Reduce en présentant des extraits de *Algebraic Computing with Reduce* [29].

B.3.1 Quelques intégrales

Voici un document Grif contenant une liste de primitives à calculer. Une fois ce document enregistré sous le format pivot de Grif nous pouvons traiter ce document par la commande Unix *grifred*.

$$\left\{ \begin{array}{l} \displaystyle\int \cos(x)^3\,dx \\ \displaystyle\int x^2 e^x\,dx \\ \displaystyle\int \frac{7}{(3-x)(1+2x)}\,dx \\ \displaystyle\int \frac{1}{(x-1)(2x+1)}\,dx \end{array} \right\}$$

Le résultat des intégrations nous donne le document Grif suivant :

$$\left\{ \begin{array}{l} \dfrac{\sin(x)\left(-\sin(x)^2+3\right)}{3} \\ e^x\left(x^2-2x+2\right) \\ \ln(2x+1)-\ln(x-3) \\ \dfrac{-\ln(2x+1)+\ln(x-1)}{3} \end{array} \right\}$$

B.3.2 Intégration

Soient les deux intégrales suivantes enregistrées dans le fichier Grif AlgInt :

$$\left\{ \begin{array}{l} Facile \leftarrow \int x\sqrt{1+x^2}\,dx \\ Difficile \leftarrow \int \sqrt{1+x^2}\,dx \end{array} \right\}$$

La commande Unix, grifred AlgInt, crée un nouveau fichier Grif, AlgInt.RG, qui contient les résultats suivants :

$$\left\{ \frac{\sqrt{x^2+1}\left(x^2+1\right)}{3}, \frac{\sqrt{x^2+1}\,x+\displaystyle\int \frac{\sqrt{x^2+1}}{x^2+1}\,dx}{2} \right\}$$

L'ordre est respecté, le premier élément de la liste correspond à la variable *Facile*, et le second à *Difficile*.

Cependant, Reduce n'a pas su intégrer jusqu'au bout *Difficile*. Nous essayons alors d'activer le switch algint de Reduce en faisant :

```
grifred AlgInt -mode algint
```

L'exécution fournira le même fichier AlgInt.RG bien que grifred signale une erreur de Reduce. La lecture du message d'erreur de Reduce nous apprend que algint n'est plus un switch pour les versions 3.4 de Reduce :

```
***** ALGINT not defined as switch
```

Le manuel de Reduce 3.4.1 signale que le module ALGINT ne se charge plus automatiquement. Ce qui explique pourquoi le switch algint n'est plus connu par défaut. Afin de remédier à ceci, nous rajoutons à la fin du document, une commande Reduce chargeant le module ALGINT.

Pour insérer une commande Reduce sous Grif, il suffit de l'écrire entre les super–parenthèses Reduce << et >>. Nous l'écrivons en majuscules de manière à ce que Reduce puisse l'interpréter correctement :

$$\left\{ \begin{array}{l} Facile \leftarrow \int x\sqrt{1+x^2}\,dx \\ Difficile \leftarrow \int \sqrt{1+x^2}\,dx \\ \text{<<LOAD\_PACKAGE ALGINT>>} \end{array} \right\}$$

Une fois cette mise à jour sauvegardée sous Grif, nous refaisons la commande Unix, grifred AlgInt. L'exécution fournit alors le résultat suivant :

$$\left\{ \frac{\sqrt{x^2+1}\left(x^2+1\right)}{3}, \frac{2\sqrt{x^2+1}\,x - \ln\left(\sqrt{x^2+1}-x\right) + \ln\left(\sqrt{x^2+1}+x\right)}{4}, ALGINT() \right\}$$

Bien que l'instruction Reduce chargeant le module ALGINT soit à la fin, nous avons l'intégrale *Difficile* qui a été résolue. Ceci s'explique par le fonctionnement de l'interface Grif/Reduce. Les expressions algébriques sont évaluées une première fois lors de la lecture du fichier Reduce AlgInt.red créé par grifred. L'intégrale *Difficile* n'est pas résolue puisque le mode ALGINT est chargé après son calcul. Cependant, lorsque nous analysons les expressions, un second passage sur les résultats obtenus est effectué. A ce moment–là Reduce en profite pour terminer, grâce au module ALGINT gardé en mémoire, ses calculs sur l'intégrale *Difficile*.

Afin d'éviter à l'utilisateur une telle technicité, nous proposons d'activer l'option –V de la commande grifred pour accéder aux facilités et enrichissements de Reduce. Nous reprenons alors la toute première version du document AlgInt :

```
grifred –v AlgInt –mode algint
```

Par cette option, nous laissons les affectations telles quelles et le module ALGINT est chargé par défaut. Nous accédons ainsi au module ALGINT sans qu'il n'y ait d'erreur. De façon plus aisée, nous obtenons le document AlgInt.RG suivant :

$$\left\{ \begin{array}{l} Facile \leftarrow \dfrac{\sqrt{x^2+1}\left(x^2+1\right)}{3} \\ Difficile \leftarrow \dfrac{2\sqrt{x^2+1}\,x - \ln\left(\sqrt{x^2+1}-x\right) + \ln\left(\sqrt{x^2+1}+x\right)}{4} \end{array} \right\}$$

B.3.3 Intégration avec bornes

Soit l'intégrale suivante à déterminer :

$$\int_0^1 \left(x^2\right)\left(x+1\right) e^{x^2}\,dx$$

La commande *grifred* appliquée sur cette dernière nous fournit le résultat suivant :

$$\frac{-\left[\int e^{x^2}\,dx\right]_{x\leftrightarrow 1}+\left[\int e^{x^2}\,dx\right]_{x\leftrightarrow 0}+e+1}{2}$$

Or, nous aurions préféré que grifred nous retourne une intégrale non résolue mais avec les bornes. Pour cela, il suffit de rajouter l'option –V à la commande grifred et de reprendre le document initial. Nous obtenons alors le résultat suivant :

$$\frac{-\int_0^1 e^{x^2}\,dx+e+1}{2}$$

Nous voulons maintenant intégrer :

$$\int_0^1 \sqrt{4-x^2}\,dx$$

La commande grifred sans options nous retourne le résultat suivant :

$$\frac{\sqrt{3}-4\left[\int \frac{\sqrt{-x^2+4}}{x^2-4}\,dx\right]_{x\leftrightarrow 1}+4\left[\int \frac{\sqrt{-x^2+4}}{x^2-4}\,dx\right]_{x\leftrightarrow 0}}{2}$$

Mais en voulant procéder comme sur le premier exemple, la commande grifred –V avec le mode algint activé, donne une erreur irrémédiable de Reduce :

***** Zero divisor

En effet, si nous cherchons la primitive précédente avec la commande grifred –V et avec le mode algint, nous obtenons :

$$\frac{\sqrt{-x^2+4}\,x+2\,\mathrm{Atan}\left(\frac{\sqrt{-x^2+4}\left(-x^2+2\right)}{x\left(x^2-4\right)}\right)}{2}$$

Cette fonction n'est pas définie en zéro, à moins de faire un calcul de limites.

B.3.4 Dérivation

Soient les trois dérivées du premier ordre enregistrées dans le document Grif, Chap1.1 :

$$\left\{ \begin{array}{l} \dfrac{\partial \ln\left(\sqrt{\dfrac{1+x}{1-x}}\right)}{\partial x} \\ \dfrac{\partial \dfrac{1+\sin(x)}{1+\cos(x)}}{\partial x} \\ \dfrac{\partial \sin(|x|)}{\partial x} \end{array} \right\}$$

La commande grifred Chap1.1 fournit le document Grif, Chap1.1.RG, suivant :

$$\left\{ \frac{-1}{x^2-1}, \frac{\sin(x)^2+\sin(x)+\cos(x)^2+\cos(x)}{\cos(x)^2+2\cos(x)+1}, \frac{|x|\cos(|x|)}{x} \right\}$$

Nous remarquons que Reduce considère les dérivées des valeurs absolues comme le rapport de $|x|$ sur x. En revanche, la fonction mathématique *ln* a été correctement traitée par Reduce qui ne connaît que la fonction *log*. En fait, c'est la table lexicale qui fait la correspondance entre ces deux fonctions.

Nous pouvons introduire des règles de calcul trigonométriques qui simplifient quelques unes des expressions :

$$\left\{ \begin{array}{l} (\cos(x))^2 \rightarrow \dfrac{1+\cos(2x)}{2} \\ (\sin(x))^2 \rightarrow \dfrac{1-\cos(2x)}{2} \\ \cos(x)\cos(y) \rightarrow \dfrac{\cos(x+y)+\cos(x-y)}{2} \\ \cos(x)\sin(y) \rightarrow \dfrac{\sin(x+y)-\sin(x-y)}{2} \\ \sin(x)\sin(y) \rightarrow \dfrac{\cos(x-y)-\cos(x+y)}{2} \end{array} \right\}$$

Les variables x et y apparaîssant dans le membre gauche des règles, doivent être construites à l'aide de la règle "Variable" de GramMath, sinon les mécanismes de de réécritures de Reduce ne s'appliqueront que pour les constantes "x" et "y".

Une fois ces règles sauvegardées dans le répertoire des définitions Reduce (donné par la variable Unix $REDLOC) sous le nom Trigo, nous faisons :

```
regle Trigo
```

Ceci a pour effet de générer un fichier texte Trigo qui nous servira de définition pour des calculs Reduce ultérieurs. Si nous reprenons l'exemple précédent en précisant avec l'option –def le fichier de définition contenant les règles trigonométriques :

```
grifred Chap1.1 –def Trigo
```

Ou bien sur le fichier résultat :

```
grifred Chap1.1.RG –def Trigo
```

Nous obtenons le résultat suivant dans le document Grif, Chap1.1.RG pour la première commande, ou Chap1.1.RG.RG pour la seconde commande :

$$\left\{ \frac{-1}{x^2-1}, \frac{2\left(\cos(x)+\sin(x)+1\right)}{\cos(2x)+4\cos(x)+3}, \frac{|x|\cos(|x|)}{x} \right\}$$

B.3.5 Equations différentielles

Nous pouvons écrire des équations différentielles, et vérifier si les solutions que nous proposons sont correctes ou non. Nous proposons ci–dessous une démarche qui montre comment nous pouvons interagir avec l'interface Grif/Reduce. De plus, nous montrons un cas très particulier d'erreur Reduce lorsqu'on évalue une équation.

Soit une équation différentielle, et une solution z fonction de x, y. Nous définissons z par une affectation, puis nous posons l'équation :

$$\left\{ \begin{array}{l} z \leftarrow \log\left(x^2+y^2\right) \\ \left(\frac{\partial z}{\partial x}\right)^2 + \left(\frac{\partial z}{\partial y}\right)^2 = 4\,e^{-z} \end{array} \right\}$$

La commande grifred appliquée sur ce document fournit le résultat suivant :

$$\left\{ \log\left(x^2+y^2\right), \frac{\partial \log\left(x^2+y^2\right)^2}{\partial x} + \frac{\partial \log\left(x^2+y^2\right)^2}{\partial y} = \frac{4}{x^2+y^2} \right\}$$

Le membre gauche de l'équation n'a pas été calculé car Reduce n'évalue pas les membres gauches des équations. Une autre façon de procéder est de demander explicitement une évaluation de l'équation en une ou plusieurs variables comme suit :

$$\left[\left(\frac{\partial z}{\partial x}\right)^2 + \left(\frac{\partial z}{\partial y}\right)^2 = 4\,e^{-z} \right]_{z \leftrightarrow \log\left(x^2+y^2\right)}$$

Cependant, nous obtenons le résultat suivant :

$$0 = \frac{4}{x^2+y^2}$$

Ce n'est pas le résultat attendu. Reduce a considéré la variable z comme étant une variable sans dépendance, en particulier indépendante de x et de y. Pour préciser qu'une variable dépend d'une autre, il faut écrire :

$$\left\{ \left[\left(\frac{\partial z}{\partial x}\right)^2 + \left(\frac{\partial z}{\partial y}\right)^2 = 4\,e^{-z} \right]_{z \leftrightarrow \log\left(x^2+\,y^2\right)} \right\} \quad \text{<<DEPEND } z, x, y \text{>>}$$

Mais la commande grifred appliquée sur le document précédent conduit à l' échec suivant:

***** SUB({ .Z=.L.O.G(.X**2 +.Y**2)}, DF(.Z,.X)**2 +DF(.Z,.Y)**2 =4/.E**.Z) invalid as scalar

Pour éviter cette erreur Reduce, il suffit de ne pas écrire d'équations dans les évaluations. Nous pouvons écrire une soustraction au lieu d'une équation :

$$\left\{ \left[\left(\frac{\partial z}{\partial x}\right)^2 + \left(\frac{\partial z}{\partial y}\right)^2 - 4\,e^{-z} \right]_{z \leftrightarrow \log\left(x^2+\,y^2\right)} \right\} \quad \text{<<DEPEND } z, x, y \text{>>}$$

Et la commande grifred nous retourne cette fois–ci le bon résultat :

$$\{\blacksquare, 0\}$$

Le premier élément correspond à l'instruction DEPEND qui ne retourne rien, et le second correspond au résultat de la soustraction.

B.3.6 Dérivées n–ièmes

L'option –V de la commande grifred permet entre autres d'accéder aux dérivées n–ièmes des fonctions en appliquant le théorème de Leibniz et quelques dérivées n–ièmes usuelles. Par exemple, nous voulons calculer la dérivée n–ième de :

$$\frac{\partial^n\left(\left(x^2-1\right)\frac{\partial y(x)}{\partial x} + x\,y(x) + 1\right)}{\partial x^n}$$

La commande grifred –V donne le résultat suivant :

$$\frac{\partial^{n-1} y(x)}{\partial x^{n-1}}\,n^2 + \frac{\partial^{n+1} y(x)}{\partial x^{n+1}}\,x^2 - \frac{\partial^{n+1} y(x)}{\partial x^{n+1}} + 2\frac{\partial^n y(x)}{\partial x^n}\,x\,n + \frac{\partial^n y(x)}{\partial x^n}\,x$$

La factorisation de l'expression précédente peut se réaliser soit avec l'expression initiale, soit avec l'expression résultat. En général, il vaut mieux pour des questions de temps d'exécutions dus à Reduce, choisir de travailler sur des résultats, surtout lorsque nous

voulons changer la présentation. Ici la commande grifred –v fichier –mode factor nous permet d'obtenir :

$$(x+1)\,(x-1)\,\frac{\partial^{n+1}y(x)}{\partial x^{n+1}}+(2n+1)\,\frac{\partial^{n}y(x)}{\partial x^{n}}\,x+\frac{\partial^{n-1}y(x)}{\partial x^{n-1}}\,n^{2}$$

B.3.7 Les tables lexicales

Nous illustrons dans cette sous–section comment obtenir des documents par grifred sous forme de questions/réponses, mais aussi comment fonctionnent les tables lexicales et les appels de fonctions ou commandes Reduce.

La table lexicale de l'utilisateur comporte deux tables: celle des constantes et celle des fonctions. Pour les constantes le format est le suivant: c'est l'affectation de la variable LEXVARPERSO d'une liste d'équations dont le terme gauche est une expression constante de Reduce, et le terme droit une expression constante de Grif. Par exemple, nous définissons les correspondances suivantes :

$$LEXVARPERSO \leftarrow \left\{ \begin{array}{l} \frac{1+\sqrt{5}}{2} = \tau \\ x = X \\ Y = y \end{array} \right\}$$

La première équation signifie que toute expression représentant le nombre d'or sera remplacé par la lettre grecque τ. Les équations suivantes signifient que tous les x minuscules seront remplacés par un X majuscule, et tous les Y majuscules par des y minuscules. Il ne peut y avoir deux fois la même variable apparaissant dans deux équations différentes. Cela conduirait à une erreur Reduce.

Une fois ce document Grif sauvegardé sous un nom, par exemple LexTau, la commande Unix lexique crée un fichier Reduce qui nous servira de définition.

```
lexique LexTau
```

Par exemple, soit le fichier suivant sur lequel nous voulons appliquer notre table lexicale:

$$\left\{ \begin{array}{l} \frac{1+\sqrt{5}}{2} = \frac{1+\sqrt{5}}{2} \\ \tau^2 + -\tau + -1 = \tau{**}2 - \tau - 1 \\ x + y + X^2 + Y^2 = x + y + X^2 + Y^2 \end{array} \right\}$$

Nous jouons sur le fait que les équations, sous Reduce, n'ont que le terme droit qui est évalué. Ainsi, cela nous permettra d'avoir un document résultat qui comporte les questions/réponses pour chacune des équations. Enregistré sous le nom GoldenRatio, la commande grifred qui emploie la table lexicale précédente sera :

```
grifred GoldenRatio –def LexTau
```

Nous obtenons le fichier GoldenRatio.RG suivant :

$$\left\{ \begin{array}{l} \dfrac{\sqrt{5}+1}{2} = \tau \\ \tau^2 - \tau - 1 = 0 \\ X^2 + Y^2 + x + y = X^2 + X + y^2 + y \end{array} \right\}$$

Les tables lexicales des fonctions consistent à initialiser LEXFCTPERSO d'une paire de liste de règles. La première liste est celle qui associe les fonctions de Reduce vers Grif, et la seconde liste les fonctions de Grif vers Reduce. Par exemple, si nous voulons ni avoir à écrire SOLVE pour Reduce, ni avoir de ARBINT, ARBREAL et de ARBCOMPLEX provenant de Reduce, nous écrivons :

$$\textit{LEXFCTPERSO} \leftarrow \left\{ \begin{array}{l} \left\{ \begin{array}{l} \text{ARBINT}(n) \rightarrow \text{IN}(n) \\ \text{ARBREAL}(x) \rightarrow \text{IR}(x) \\ \text{ARBCOMPLEX}(z) \rightarrow \iota(z) \end{array} \right\} \\ \{\text{Resoudre}(eq, var) \rightarrow << \text{SOLVE } (eq, var) >>\} \end{array} \right\}$$

Nous appliquons sur ce document Grif, comme pour la table lexicale des variables, la commande lexique. Si nous avons mis entre super-parenthèses Reduce << et >> la commande SOLVE, c'est pour forcer Reduce à réécrire Resoudre en SOLVE puis à évaluer SOLVE. Sinon une erreur irrémédiable risque d'apparaître. D'autre part, par erreur nous avions donné pour ARBINT une association avec IN. Or IN est une commande Reduce, ce qui entraîne une erreur de Reduce. Dans ce cas, une astuce est d'utiliser l'alphabet grec, sinon il suffit de choisir un autre nom.

Par exemple, soient les cinq documents suivants :

1. $$\left\{ \begin{array}{l} \text{Resoudre}(x^2 = 1, x) \\ \text{Resoudre}(x^7 + -x^6 + x^2 = 1, x) \\ \text{Resoudre}(x^2 = 2x + -1, x) \end{array} \right\}$$
2. $$\text{Resoudre}(\ln(\sin(x+\pi))^3 = 8, x)$$
3. $$\text{Resoudre}(a \ln(\sin(x+3))^5 - b, \sin(x+3))$$
4. $$\left\{ \begin{array}{l} \text{Resoudre}\left(\left\{ \begin{array}{l} a x + y = 3 \\ y = -2 \end{array} \right\}, \{x, y\} \right) \\ \text{Resoudre}\left(\left\{ \begin{array}{l} x + 3y = 7 \\ y - x = 1 \end{array} \right\}, \{x, y\} \right) \end{array} \right\}$$
5. $$\text{Resoudre}(\{x+y = 0, 2*x+2*y = 0\}, \{x, y\})$$

Après avoir fait: grifred fichiers... -def LexiqueFct -mode multiplicities solvesingular, options qui nous permettent de donner les racines multiples ou bien de résoudre des systèmes admettant une infinité de solutions, nous obtenons :

1. $$\left\{ \begin{array}{l} \{x = -1, x = 1\} \\ \{x = 1, x = \text{ROOTOF}(!x\_^6 + !x\_ + 1, !x\_)\} \\ \{x = 1, x = 1\} \end{array} \right\}$$

2. $$\left\{ \begin{array}{l} x = 2\,\text{IN}(2)\,\pi + \text{Asin}\left(\dfrac{1}{e^{\sqrt{3}\,i}}\right) - \pi \\ x = 2\,\text{IN}(2)\,\pi - \text{Asin}\left(\dfrac{1}{e^{\sqrt{3}\,i}}\right) \\ x = 2\,\text{IN}(1)\,\pi + \text{Asin}\left(\dfrac{e^{\sqrt{3}\,i}}{e}\right) - \pi \\ x = 2\,\text{IN}(1)\,\pi - \text{Asin}\left(\dfrac{e^{\sqrt{3}\,i}}{e}\right) \\ x = 2\,\text{IN}(3)\,\pi + \text{Asin}(e^2) - \pi \\ x = 2\,\text{IN}(3)\,\pi - \text{Asin}(e^2) \end{array} \right\}$$

3. $$\{\ln(\sin(x+3))^5 a - b = 0\}$$

4. $$\left\{ \left\{ \left\{ x = \frac{5}{a}, y = -2 \right\} \right\}, \{\{x = 1, y = 2\}\} \right\}$$

5. $$\{\{x = -\iota(1), y = \iota(1)\}\}$$

B.3.8 Les sommes finies et indéfinies

Les sommes et produits finis comme :

$$\left\{ \begin{array}{l} \displaystyle\sum_{k \leftarrow 1}^{10} \frac{1}{k^2} \\ \displaystyle\sum_{i \leftarrow 1}^{3} \prod_{j \leftarrow 0}^{i} \left(x^i - j\right) \end{array} \right\}$$

donnent par la commande grifred sans aucune option :

$$\left\{ \frac{1968329}{1270080}, x\left(x^{11} - 6x^8 + 12x^5 - 3x^3 - 6x^2 + 3x - 1\right) \right\}$$

Si l'on souhaite calculer les sommes indéfinies suivantes :

$$\left\{ \begin{array}{l} \displaystyle\sum_{n\leftarrow 1}^{N} \frac{\sin\left(n^2 x\right)}{n^2} \\ \displaystyle\frac{\partial\left(\sum_{n\leftarrow 1}^{N} \frac{\sin\left(n^2 x\right)}{n^2}\right)}{\partial x} \end{array} \right\}$$

où N et n sont deux variables différentes dans notre interface, la commande grifred sans options retourne le message d'erreur suivant :

***** N – 1 invalid in FOR statement

En effet, Reduce essaye toujours d'exécuter une boucle FOR, et ici il ne peut pas itérer un nombre indéfini de fois. D'où l'erreur. Pour remédier à ce problème d'abstraction, nous avons défini un ensemble de règles pour des sommes indéfinies, mais aussi dans une moindre mesure pour des produits indéfinis. Ces règles sont accessibles en choisissant l'option –V. Ainsi, l'exemple précédent nous fournira le résultat suivant :

$$\left\{ \sum_{n\leftarrow 1}^{N} \frac{\sin\left(n^2 x\right)}{n^2}, \sum_{n\leftarrow 1}^{N} \cos\left(n^2 x\right) \right\}$$

Le premier élément de la liste reste inchangé, et le second a été dérivé correctement.

L'exemple suivant montre une sommation indéfinie que Reduce sait résoudre :

$$\left\{ \begin{array}{l} \displaystyle\sum_{j\leftarrow \mathrm{N}}^{2\,\mathrm{N}} (\mathrm{j}-1)^7 \\ \displaystyle\sum_{k\leftarrow 1}^{\mathrm{n}} \mathrm{k}^{12} \end{array} \right\}$$

En choisissant la commande grifred –V, nous obtenons :

$$\left\{ \begin{array}{l} \displaystyle\frac{255\,N^8 - 500\,N^7 + 238\,N^6 + 168\,N^5 - 315\,N^4 + 280\,N^3 - 166\,N^2 + 56\,N - 8}{8} \\ \displaystyle\frac{n\left(210\,n^{12} + 1365\,n^{11} + 2730\,n^{10} - 5005\,n^8 + 8580\,n^6 - 9009\,n^4 + 4550\,n^2 - 691\right)}{2730} \end{array} \right\}$$

Ce genre de calcul sur des sommes indéfinies est hautement récursif. Aussi avons–nous défini des règles de calcul qui retiennent au fur et à mesure les sommes déjà calculées, et ce indépendamment des bornes et du nom de variable choisi. Ceci permet un gain de temps non négligeable, mais peut–être plus gourmand en mémoire.

Si l'on essaye l'option de factorisation, –mode factor, sur la liste des sommes indéfinies, nous n'aurions pas pu obtenir le résultat escompté. Cependant, nous pouvons le faire sur les résultats.

$$\left\{ \begin{array}{l} \dfrac{\left(85\,N^6 - 195\,N^5 + 201\,N^4 - 141\,N^3 + 76\,N^2 - 26\,N + 4\right)\,\left(3N-2\right)\,\left(N+1\right)}{8} \\ \textit{... Expression trop longue ...} \end{array} \right\}$$

De même, l'option de propagation du diviseur dans le numérateur, mode –div, appliqué sur les résultats précédents donne :

$$\left\{ \begin{array}{l} \dfrac{255}{8}N^8 - \dfrac{125}{2}N^7 + \dfrac{119}{4}N^6 + 21\,N^5 - \dfrac{315}{8}N^4 + 35\,N^3 - \dfrac{83}{4}N^2 + 7N - 1 \\ n\left(\dfrac{1}{13}n^{12} + \dfrac{1}{2}n^{11} + n^{10} - \dfrac{11}{6}n^8 + \dfrac{22}{7}n^6 - \dfrac{33}{10}n^4 + \dfrac{5}{3}n^2 - \dfrac{691}{2730}\right) \end{array} \right\}$$

Supposons que nous veuillons manipuler l'expression comme les précédentes afin de grouper certains monômes entre eux. Ces manipulations sous l'éditeur Grif sont assez délicates à cause des soustractions imbriquées les unes dans les autres. Pour de tels exemples de manipulations, voir le manuel sur l'édition GramMath . Nous pouvons cependant choisir une autre présentation en demandant à la commande grifred de ne pas insérer l'opérateur soustraction. Il suffit pour cela de donner l'option –s à la commande grifred, et nous conservons l'option de propagation du diviseur dans les numérateurs :

grifred –s fichier –mode div

Nous obtenons des sommes de monômes qui, s'il comportent un coefficient négatif, ont un signe moins. La présentation est un peu inhabituelle, mais elle permet de comprendre la structure du document :

$$\left\{ \begin{array}{l} \dfrac{255}{8}N^8 + -\dfrac{125}{2}N^7 + \dfrac{119}{4}N^6 + 21\,N^5 + -\dfrac{315}{8}N^4 + 35\,N^3 + -\dfrac{83}{4}N^2 + 7\,N + -1 \\ n\left(\dfrac{1}{13}n^{12} + \dfrac{1}{2}n^{11} + n^{10} + -\dfrac{11}{6}n^8 + \dfrac{22}{7}n^6 + -\dfrac{33}{10}n^4 + \dfrac{5}{3}n^2 + -\dfrac{691}{2730}\right) \end{array} \right\}$$

Nous pouvons faire de même pour le parenthésage avec l'option –p, ou les deux à la fois avec l'option –ps. Cependant ces deux dernières options présentent un intérêt moindre.

Annexe B

Un éditeur mathématique Grif orienté par la sémantique

Annexe C

Activités de modélisation au moyen des strates

A travers un problème de modélisation inspiré d'un cas réel en industrie, ici un scanner de forme, nous illustrons les strates et les algorithmes présentés aux chapitres V et VI.

Nous rappelons que les numérotations apparaissant en italique et entre parenthèses, ainsi que les caractères en oblique entre crochets, sont des renvois à d'autres strates. Sous le logiciel Grif, ce sont des liens hypertextes.

Par langage de programmation, nous désignons un langage informatique classique appartenant à la famille des langages de type C, Fortran, et Pascal.

C.1 Premier jet

Nous modélisons un problème en restant dans un premier temps général. Ceci explique le type de langage employé qui est d'abord naturel, puis mathématique, mais tout en restant imprécis quant à la manière de déterminer les résultats de notre problème.

1) Problème de modélisation

Langage = *Naturel*

Données : ρ_i, z_i, i=1..512

Résultats : $\bar{\rho}_i$, $\bar{z}_i$, i=1..512

Commentaire :

Étant donné un contour connu sous la forme de coordonnées cylindriques bruitées à angles "équidistants", le problème est de définir la forme dans le plan, ce qui revient à déterminer les coordonnées polaires du contour dans le plan qui l'approche au mieux.

Mais nous devons aussi calculer la distance du plan à un point du contour afin de pouvoir avoir une idée sur l'erreur commise.

Arguments :

Décomposition séquentielle du problème.

Descriptif :

Variables locales :

P

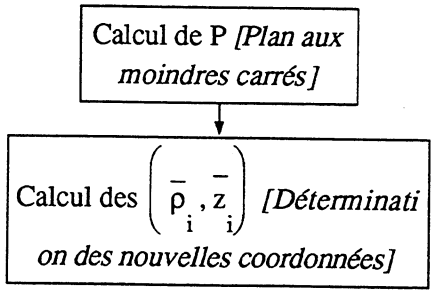
Conditions :

P = plan aux moindres carrés de $\left\{\rho_i, z_i\right\}_{i = 1 .. 512}$

$\left(\frac{2\pi i}{512}, \bar{\rho}_i\right)$ = coordonnées polaires de $\left(\rho_i, z_i\right)$ dans le plan P.

$\bar{z}_i$ = distance de $\left(\rho_i, z_i\right)$ au plan P.

Schéma de résolution :



2) Plan aux moindres carrés

Langage = *Naturel – Mathématiques*

Strate père : *[Problème de modélisation]*

Données : ρ_i, z_i, i=1..512

Résultats : ρ_i, z_i, i=1..512, P

Commentaire :

On doit préciser la représentation mathématique du plan P, et pour cela représenter les points en coordonnées cartésiennes.

Arguments :

Modélisation.

Descriptif :

Variables locales :

x_i, y_i, i=1..512, α, β, γ

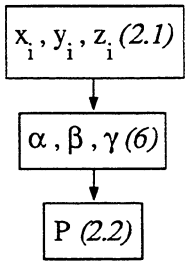
Conditions :

P = plan aux moindres carrés de $\{\rho_i, z_i\}_{i=1..512}$

$P \Leftrightarrow (\alpha, \beta, \gamma), z = \alpha x + \beta y + \gamma,$

$$\underset{(\alpha,\beta,\gamma)}{\text{Min}} \sum_i [z_i - (\alpha x_i + \beta y_i + \gamma)]^2$$

Schéma de résolution :



2.1) Passage en coordonnées cartésiennes

Strate père : *[Plan aux moindres carrés]*

Données : ρ_i, z_i, i

Résultats : x_i, y_i, i

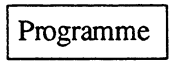
Descriptif :

Conditions :

$$x_i = \rho_i \cos\left(\frac{2\pi i}{512}\right)$$

$$y_i = \rho_i \sin\left(\frac{2\pi i}{512}\right)$$

Schéma de résolution :



2.2) Traduction retour

Strate père : *[Plan aux moindres carrés]*

Données : α, β, γ

Résultats : P

Commentaire :

Nous faisons une représentation mathématique du plan P.

Descriptif :

Conditions :

$P \Leftrightarrow (\alpha, \beta, \gamma), z = \alpha x + \beta y + \gamma,$

3) Détermination des nouvelles coordonnées

Langage = *Naturel – Mathématiques*

Strate père : *[Problème de modélisation]*

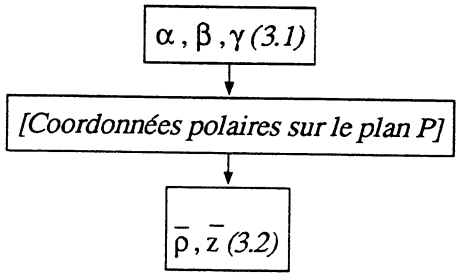
Données : ρ_i, z_i, P

Résultats : $\bar{\rho}_i$, $\bar{z}_i$

Commentaire :

Nous devons ici préciser comment nous calculons les nouvelles coordonnées dans le plan des moindres carrés P.

Schéma de résolution :



3.1) Traduction aller

Strate père : *[Détermination des nouvelles coordonnées]*

Données : P

Résultats : α, β, γ

Commentaire :

Nous faisons l'inverse de la strate *[Traduction retour]*

Descriptif :

Conditions :

$(\alpha, \beta, \gamma), z = \alpha x + \beta y + \gamma \quad \Leftrightarrow P$

3.2) Ajustement des nouvelles coordonnées

Strate père : *[Détermination des nouvelles coordonnées]*

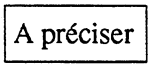
Données : $\alpha, \beta, \gamma, \rho_i, z_i, \alpha, \beta, \gamma$

Résultats : $\bar{\rho}_i$, $\bar{z}_i$

Commentaire :

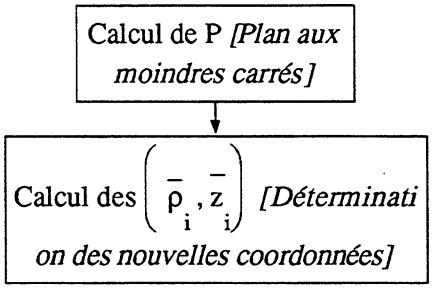
A partir des coordonnées projetées sur le plan aux moindres carrés, nous devons obtenir des coordonnées polaires d'angle équidistants, et déterminer la nouvelle distance du contour au plan.

Schéma de résolution :

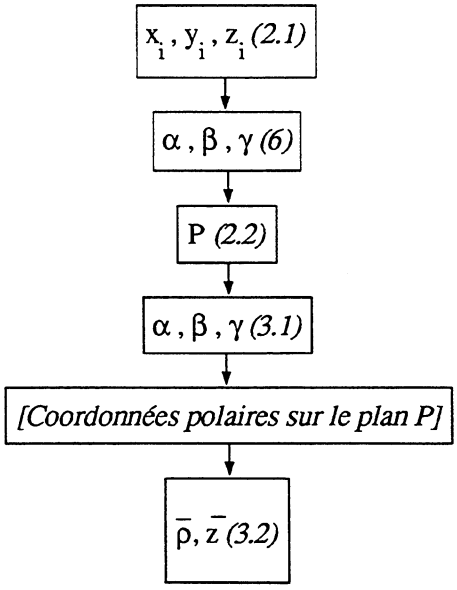


C.2 Une première projection

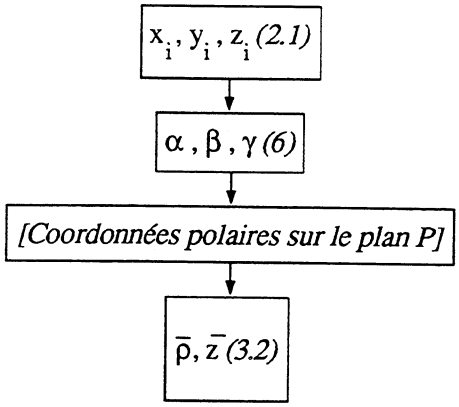
Les strates *[Problème de modélisation]*, *[Plan aux moindres carrés]* et *[Détermination des nouvelles coordonnées]* étant définies, nous pouvons appliquer l'algorithme de projection des strates. L'algorithme substitue successivement les schémas de résolution au schéma initial :



Ce qui nous donne :



L'utilisateur peut se rendre compte *a fortiori* que les traductions sur la représentation du plan P entre le langage naturel et le langage mathématique, permettent de simplifier le schéma de résolution :



Ce schéma de résolution peut ainsi remplacer celui de la strate initiale.

C.3 Raffinement des strates

Dans un second temps, nous allons préciser les calculs et les méthodes de résolutions pour chacune des strates précédentes. La strate qui suit reprend la projection précédente, et est décrite en langage mathématique. Les autres caractéristiques sont simplement une copie de la strate initiale *[Problème de modélisation]* mises à jour en remplaçant P par le triplet (α,β,γ)

4) Projection des strates de C.1

Langage = *Mathématique*

Strate père : *[Problème de modélisation]*

Données : ρ_i, z_i, i=1..512

Résultats : $\bar{\rho}_i$, $\bar{z}_i$, i=1..512

Commentaire :

Étant donné un contour connu sous la forme de coordonnées cylindriques bruitées à angles "équidistants", le problème est de définir la forme dans le plan, ce qui revient à déterminer les coordonnées polaires du contour dans le plan qui l'approche au mieux. Mais nous devons aussi calculer la distance du plan à un point du contour afin de pouvoir avoir une idée sur l'erreur commise.

Arguments :

Décomposition séquentielle du problème.

Descriptif :

Variables locales :

α, β, γ , P

Conditions :

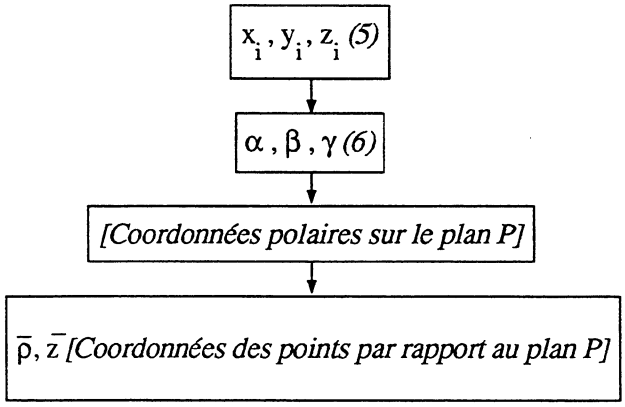
P = (α, β, γ)

P = plan aux moindres carrés de $\left\{\rho_i, z_i\right\}_{i = 1 .. 512}$

$\left(\frac{2\pi i}{512}, \bar{\rho}_i\right)$ = coordonnées polaires de $\left(\rho_i, z_i\right)$ dans le plan P.

$\bar{z}_i$ = distance de $\left(\rho_i, z_i\right)$ au plan P.

Schéma de résolution :



5) Polaires – Cartésiennes

Langage = *Programmation*

Strate père : *[Projection des strates de]*

Données : ρ_i, z_i, i

Résultats : x_i, y_i, i

Descriptif :

Variables locales :

PI

Conditions :

PI ← 3.1415

Pour i **allant de** 1 **à** 512 **faire**

X[i] ← RHO[i] * cos(2.0 * PI * i / 512.0)

Y[i] ← RHO[i] * sin(2.0 * PI * i / 512.0)

6) Calcul du plan aux moindres carrés

Langage = *Mathématiques*

Strate père : *[Plan aux moindres carrés]*

Données : x_i, y_i, z_i

Résultats : α, β, γ

Commentaire :

Le système suivant est obtenu en utilisant la théorie des moindres carrés.

Descriptif :

Variables locales :

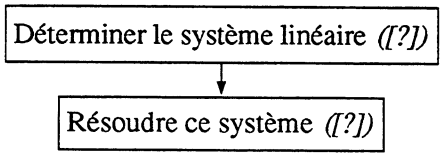
A, b

Conditions :

$$A = \begin{pmatrix} \sum_i x_i^2 & \sum_i x_i y_i & \sum_i x_i \\ \sum_i x_i y_i & \sum_i y_i^2 & \sum_i y_i \\ \sum_i x_i & \sum_i y_i & 512 \end{pmatrix}, \quad b = \begin{pmatrix} \sum_i x_i z_i \\ \sum_i y_i z_i \\ \sum_i z_i \end{pmatrix}$$

$$A \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = b$$

Schéma de résolution :



7) Coordonnées polaires sur le plan P

Langage = *Mathématiques*

Strate père : *[Détermination des nouvelles coordonnées]*

Données : $x_i, y_i, z_i, \alpha, \beta, \gamma$

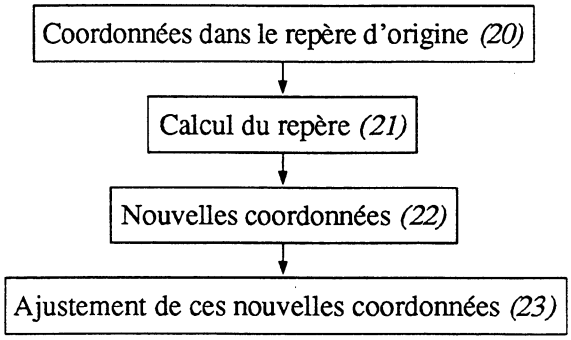
Résultats : $\bar{x}_i, \bar{y}_i, \bar{z}_i$

Descriptif :

Variables locales :

$\tilde{x}_i, \tilde{y}_i, \tilde{z}_i$

Schéma de résolution :



8) Coordonnées des points par rapport au plan P

Langage = *Mathématiques*

Données : $\bar{x}_i, \bar{y}_i, \bar{z}_i$

Résultats : $\bar{\rho}_i, \bar{z}_i$

Descriptif :

Conditions :

$$\bar{\rho}_i = \sqrt{\bar{x}_i^2 + \bar{y}_i^2}$$

Schéma de résolution :

Programme

C.4 La résolution de la strate "Coordonnées polaires sur P"

20) Résolution des 512 systèmes 4×4

Langage = *Algorithmique*

Strate père : *[Coordonnées polaires sur le plan P]*

Données : $x_i, y_i, z_i, \alpha, \beta, \gamma$

Résultats : $\tilde{x}_i, \tilde{y}_i, \tilde{z}_i$

Commentaire :

Nous projetons sur le plan aux moindres carrés. L'inconnue δ_i est la distance du point à sa projection.

Descriptif :

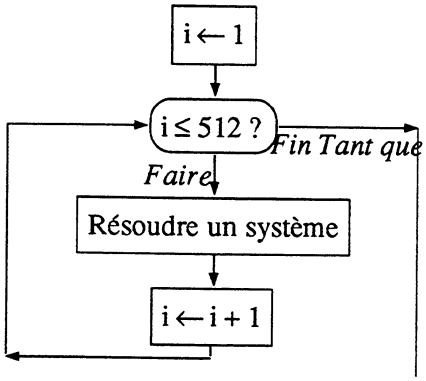
Variables locales :

δ_i

Conditions :

$$\begin{cases} (\tilde{x}_i - x_i,\ \tilde{y}_i - y_i, \tilde{z}_i - z_i) = \delta_i(\alpha, \beta, -1) \\ \tilde{z}_i = \alpha \tilde{x}_i + \beta \tilde{y}_i + \gamma \end{cases}$$

Schéma de résolution :



21) Repère dans le plan P

Langage = *Mathématiques*

Strate père : *[Coordonnées polaires sur le plan P]*

Données : $\tilde{x}_i, \tilde{y}_i, \tilde{z}_i$

Résultats : $\vec{i}, \vec{j}, \vec{k}$

Commentaire :

L'origine du repère initial doit être également projetée sur le plan aux moindres carrés afin de pouvoir déterminer le nouveau repère. Soit nous considérons ce calcul imprévu dans cette strate, soit nous propageons à l'ensemble des strates le fait que les indices i varient entre 0 et 512, au lieu de 1 et 512, pour inclure ce nouveau point. Nous retenons ici la première solution.

Descriptif :

Variables locales :

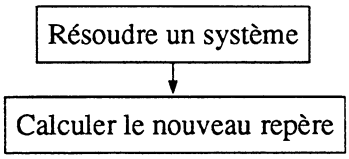
$\tilde{x}_0, \tilde{y}_0, \tilde{z}_0$

Conditions :

$$\begin{cases} (\tilde{x}_0 - x_0,\ \tilde{y}_0 - y_0, \tilde{z}_0 - z_0) = \delta_0(\alpha, \beta, -1) \\ \tilde{z}_0 = \alpha \tilde{x}_0 + \beta \tilde{y}_0 + \gamma \end{cases}$$

$$\begin{cases} \bar{i} = (\tilde{x}_1 - \tilde{x}_0, \tilde{y}_1 - \tilde{y}_0, \tilde{z}_1 - \tilde{z}_0) \\ \bar{j} = \bar{k} \wedge \bar{i} \\ \bar{k} = (\alpha, \beta, -1) \end{cases}$$

Schéma de résolution :



22) Nouvelles coordonnées

Langage = *Mathématiques*

Strate père : *[Coordonnées des points par rapport au plan P]*

Données : $\tilde{x}_0, \tilde{y}_0, \tilde{z}_0, \tilde{x}_i, \tilde{y}_i, \tilde{z}_i$

Résultats : $\hat{x}_i, \hat{y}_i, \hat{z}_i$

Descriptif :

Variables locales :

V

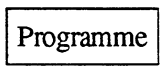
Préconditions :

$V = (\tilde{x}_i - \tilde{x}_0, \tilde{y}_i - \tilde{y}_0, \tilde{z}_i - \tilde{z}_0)$

Conditions :

$$\begin{cases} \hat{x}_i = < V , \bar{i} > \\ \hat{y}_i = < V , \bar{j} > \\ \hat{z}_i = < V , \bar{k} > \end{cases}$$

Schéma de résolution :



23) Ajustement de ces nouvelles coordonnées

Langage = *Mathématiques*

Strate père : *[Coordonnées polaires sur le plan P]*

Données : $\hat{x}_i, \hat{y}_i, \hat{z}_i$

Résultats : $\bar{x}_i, \bar{y}_i, \bar{z}_i$

Commentaire :

Nous devons déterminer de nouveaux points du plan aux moindres carrés de telle sorte que leurs coordonnées polaires soient à angles équidistants. L'idée de la méthode algorithmique est, pour un angle θ donné, de rechercher l'intersection entre la droite D passant par l'origine d'angle θ et la droite D' joignant les deux points les plus proches de part et d'autre de la droite D. Cette intersection nous donnera les nouvelles coordonnées recherchées.

Schéma de résolution :

Programme *[Ajustement]*

C.5 Ajustement des coordonnées

30) Ajustement

Langage = *Algorithmique*

Strate père : *[Ajustement de ces nouvelles coordonnées]*

Données : $\hat{x}_i, \hat{y}_i, \hat{z}_i$

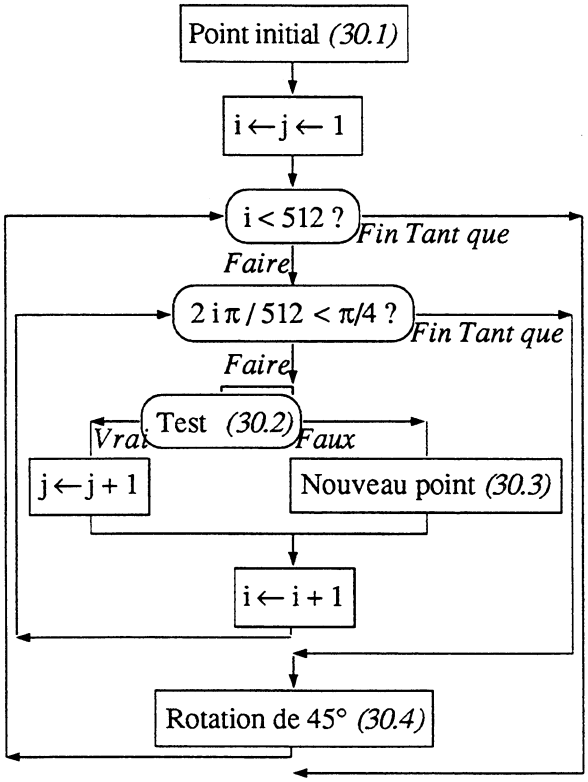
Résultats : $\bar{x}_i, \bar{y}_i, \bar{z}_i$

Descriptif :

Variables locales :

j

Schéma de résolution :



30.1) Point initial

Strate père : *[Ajustement]*

Données : $\hat{x}_1, \hat{y}_1, \hat{z}_1$

Résultats : $\bar{x}_1, \bar{y}_1, \bar{z}_1$

Commentaire :

Ce premier point a été choisi auparavant, *[Repère dans le plan P]*, pour le repère.

Descriptif :

Conditions :

$\bar{x}[1] \leftarrow \hat{x}[1]$

$\bar{y}[1] \leftarrow \hat{y}[1]$

$\bar{z}[1] \leftarrow \hat{z}[1]$

30.2) Test

Strate père : *[Ajustement]*

Données : $\hat{x}_j, \hat{y}_j, \hat{z}_j, i$

Résultats : bool

Commentaire :

Recherche du point le plus proche de la droite passant par l'origine d'angle (2πi/512) au moyen de la formule des tangentes. Cette strate retourne une valeur booléenne.

Descriptif :

Conditions :

$$\text{bool} \leftarrow \frac{\hat{y}_i}{\hat{x}_i} < \text{tg}\left(\frac{2\pi i}{512}\right)$$

30.3) Intersection

Strate père : *[Ajustement]*

Données : $\hat{x}_i, \hat{y}_i$, i, j

Résultats : $\bar{x}_i, \bar{y}_i$

Commentaire :

Le nouveau est calculé en prenant l'intersection de la droite passant par l'origine d'angle (2πi/512) et de la droite $[\hat{x}_{i-1}\ \hat{y}_{i-1}, \hat{x}_i\ \hat{y}_i]$.

Descriptif :

Variables locales :

a, b, p

Préconditions :

$$p = \text{tg}\frac{2\pi j}{512}$$

$$\begin{cases} \hat{y}_i = a\hat{x}_i + b \\ \hat{y}_{i-1} = a\hat{x}_{i-1} + b \end{cases}$$

Conditions :

$$\begin{cases} \bar{x}_j = \dfrac{b}{p-a} \\ \bar{y}_j = \dfrac{bp}{p-a} \end{cases}$$

Schéma de résolution :

Programme

30.4) Rotation de 45°

Strate père : *[Ajustement]*

Données : $\hat{x}_i, \hat{y}_i, \hat{z}_i$

Résultats : $\hat{x}_i, \hat{y}_i, \hat{z}_i$

Descriptif :

Variables locales :

u, v

Conditions :

$$\begin{cases} u \leftarrow (\hat{x}_i - \hat{y}_i)\dfrac{\sqrt{2}}{2} \\ v \leftarrow (\hat{x}_i + \hat{y}_i)\dfrac{\sqrt{2}}{2} \end{cases}$$

$$\begin{cases} \hat{x}_i \leftarrow u \\ \hat{y}_i \leftarrow v \end{cases}$$

Schéma de résolution :

Programme

C.6 Développement du programme

Nous présentons ci–dessous le code associé à la résolution précédente. Nous employons pour cela une représentation sous forme d'organigramme qui représente toutes les étapes algorithmiques.

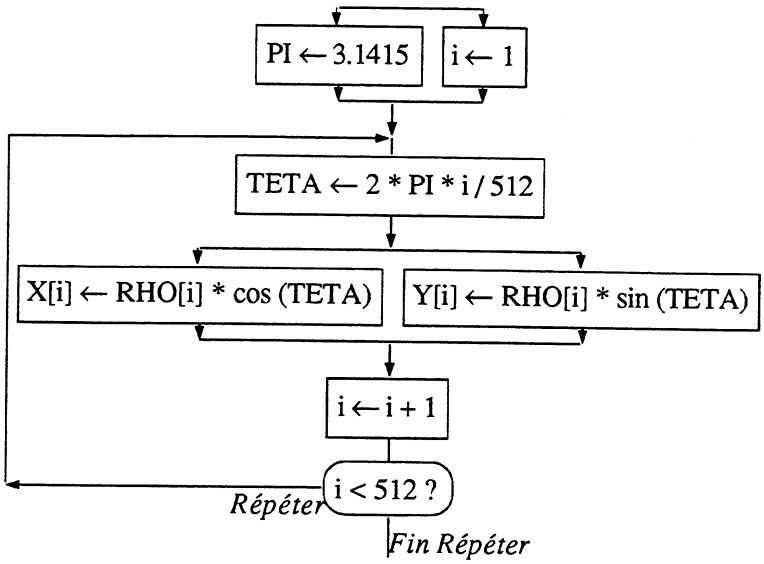
66) Polaires – Cartésiennes

Langage = *Pascal*

Données : RHO[ ]

Résultats : X[ ], Y[ ]

Schéma de résolution :



67) Plan aux moindres carrés

Langage = *Pascal*

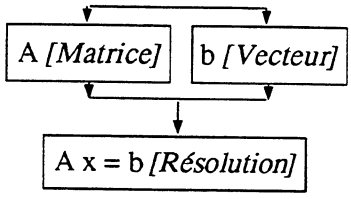
Données : X[ ], Y[ ], Z[ ]

Résultats : α, β, γ

Commentaire :

Nous pouvons déterminer en parallèle la matrice A et le vecteur b.

Schéma de résolution :

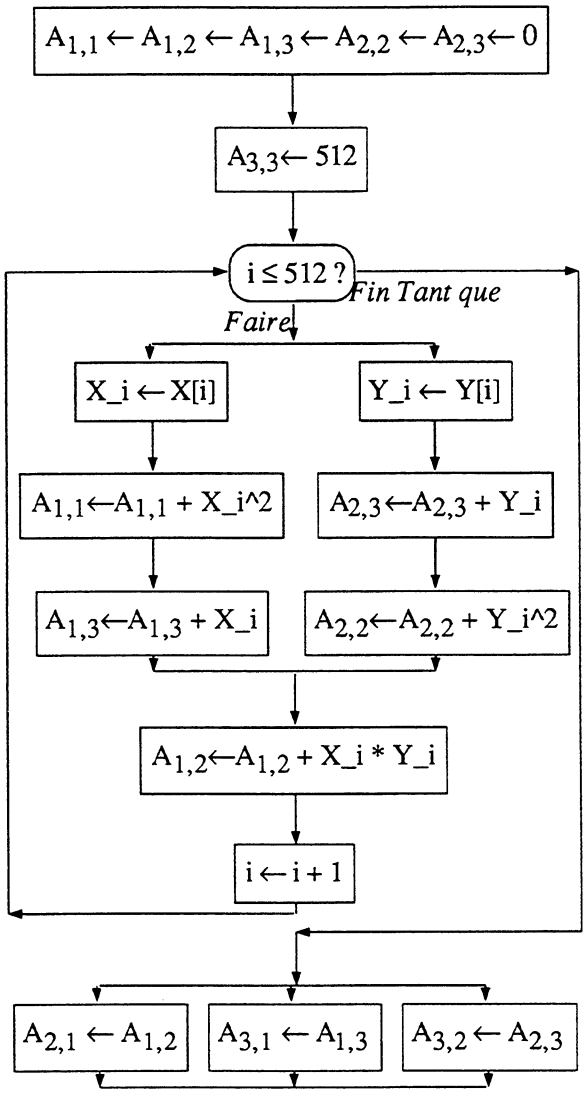


67.1) Matrice

Données : X[ ], Y[ ]

Résultats : A

Schéma de résolution :

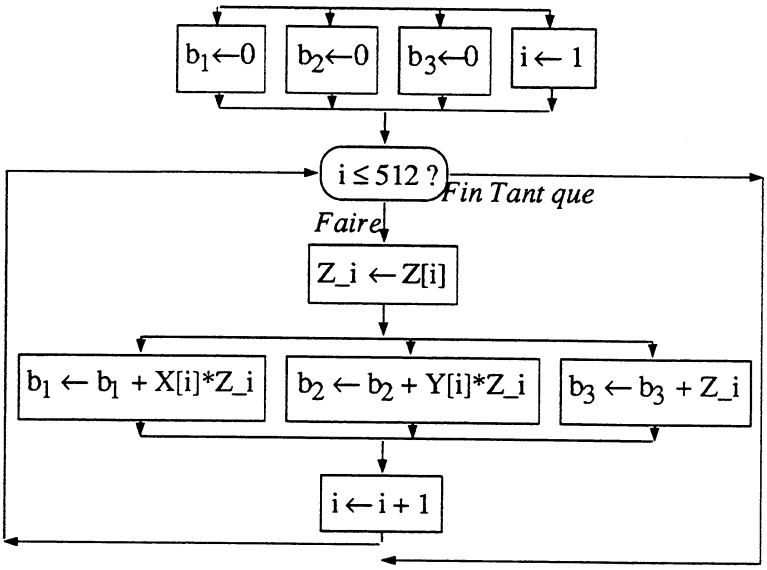


67.2) Vecteur

Données : X[ ], Y[ ]. Z[ ]

Résultats : b

Schéma de résolution :



67.3) Résolution

Données : A, b

Résultats : α, β, γ

Equations de liens :

$N \leftarrow 3$

$M \leftarrow$ A

$v \leftarrow$ b

w[1] $\rightarrow \alpha$

w[2] $\rightarrow \beta$

w[3] $\rightarrow \gamma$

Schéma de résolution :

Résolution par la méthode de Gauss *(10)*

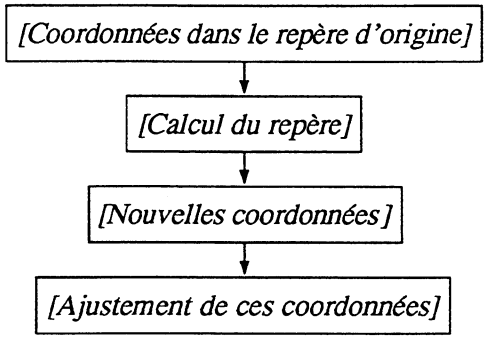
68) Coordonnées polaires sur P

Langage = *Pascal*

Données : α, β, γ, X[ ], Y[ ], Z[ ]

Résultats : X_chapeau[ ], Y_chapeau[ ]

Schéma de résolution :



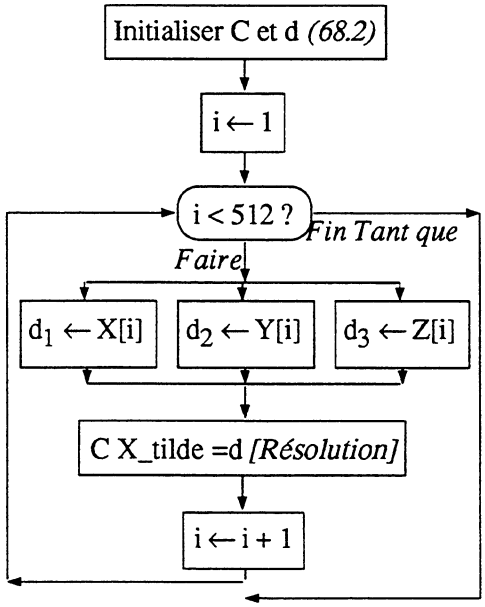
68.1) Coordonnées dans le repère d'origine

Strate père : *[Coordonnées polaires sur P]*

Données : α, β, γ, X[ ], Y[ ], Z[ ]

Résultats : X_tilde[ ], Y_tilde[ ], Z_tilde[ ]

Schéma de résolution :



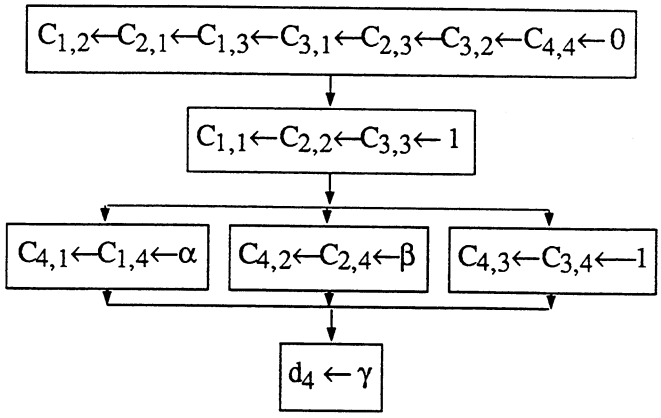
68.2) Initialisations

Strate père : *(68.1)[Coordonnées dans le repère d'origine]*

Données : α, β, γ

Résultats : C, d

Schéma de résolution :



68.3) Résolution

Strate père : *(68.1)[Coordonnées dans le repère d'origine]*

Données : C, d

Résultats : X_tilde[ ], Y_tilde[ ], Z_tilde[ ], Delta[ ]

Equations de liens :

$N \leftarrow 4$

$M \leftarrow C$

$v \leftarrow d$

w[1] → X_tilde[i]

w[2] → Y_tilde[i]

w[3] → Z_tilde[i]

w[4] → Delta[i]

Schéma de résolution :

Résolution par la méthode de Gauss *(10)*

68.4) Calcul du repère

Strate père : *[Coordonnées polaires sur P]*

Données : X[ ], Y[ ], Z[ ]

Résultats : i_barre, j_barre, k_barre

Schéma de résolution :

[Résoudre un système]

[Calculer le nouveau repère]

68.5) Résoudre un système

Strate père : *[Calcul du repère]*

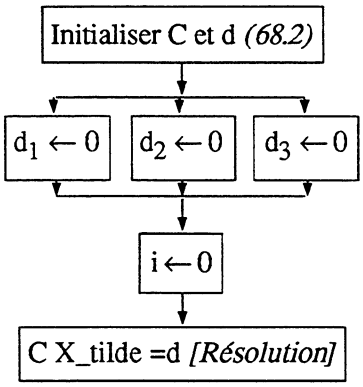
Données : X[ ], Y[ ], α, β, γ

Résultats : X_tilde[ ], Y_tilde[ ]

Commentaire :

Nous réutilisons les résolutions des 512 systèmes pour calculer la projection de l'origine sur le plan au moindres carrés.

Schéma de résolution :



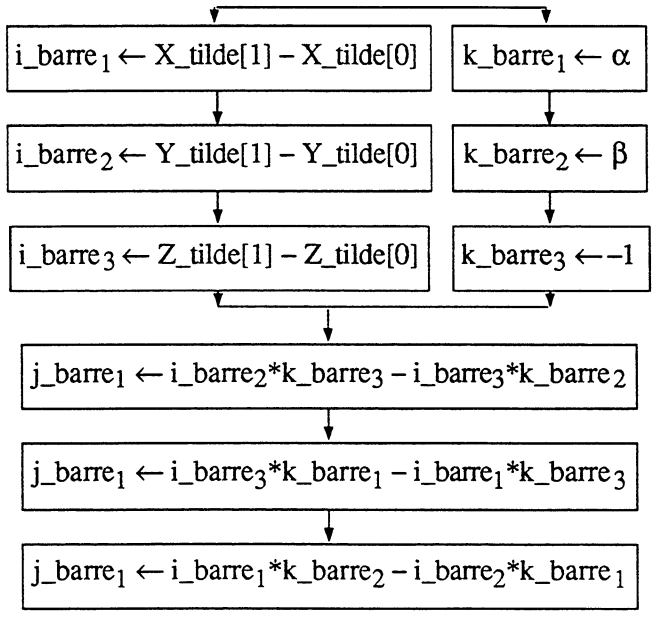
68.6) Calculer le nouveau repère

Strate père : *[Calcul du repère]*

Données : X_tilde[ ], Y_tilde[ ], Z_tilde[ ], α, β

Résultats : i_barre, j_barre, k_barre

Schéma de résolution :



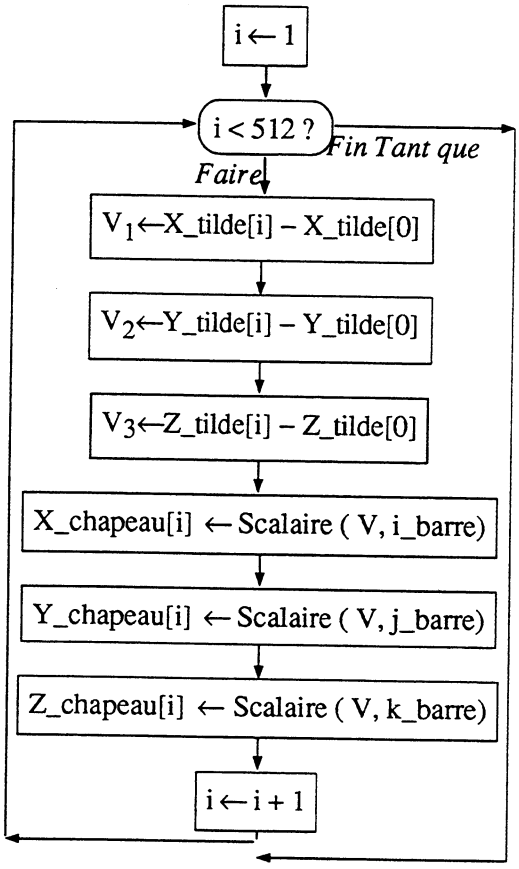
68.7) Nouvelles coordonnées

Strate père : *[Coordonnées polaires sur P]*

Données : X_tilde, Y_tilde, Z_tilde, i_barre, j_barre, k_barre

Résultats : X_chapeau, Y_chapeau, Z_chapeau

Schéma de résolution :



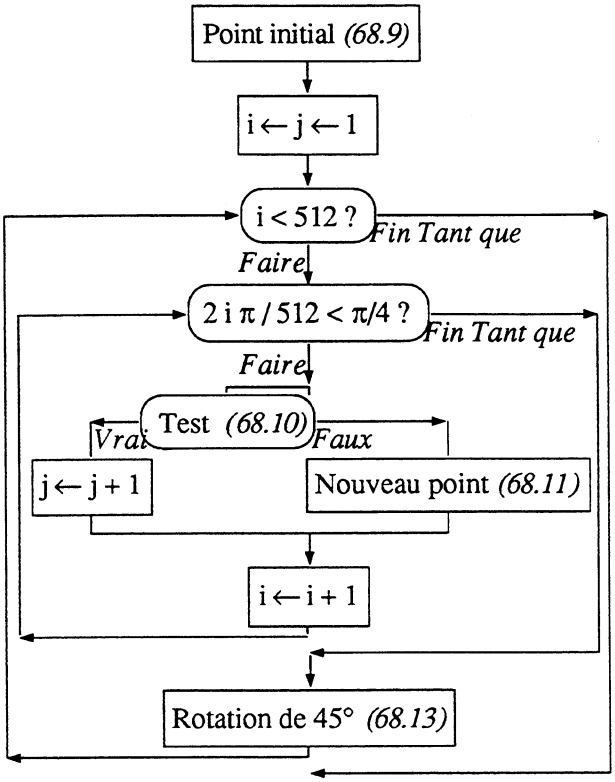
68.8) Ajustement de ces coordonnées

Strate père : *[Coordonnées polaires sur P]*

Données : X_chapeau[ ], Y_chapeau[ ]

Résultats : X_barre[ ], Y_barre[ ]

Schéma de résolution :



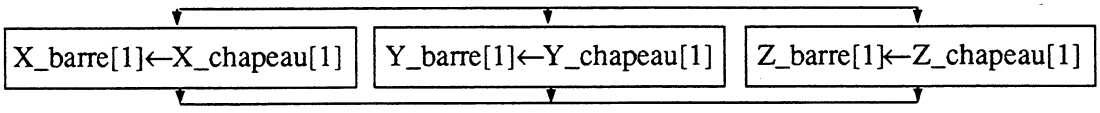
68.9) Point initial

Strate père : *[Ajustement de ces coordonnées]*

Données : X_chapeau[ ], Y_chapeau[ ], Z_chapeau[ ]

Résultats : X_barre, Y_barre, Z_barre

Schéma de résolution :



68.10) Test de boucle

Strate père : *[Ajustement de ces coordonnées]*

Données : X_chapeau[ ], Y_chapeau[ ], i

Résultats : bool

Schéma de résolution :

bool ← (Y_chapeau[i] / X_chapeau[i]) < tg (2*PI* i / 512)

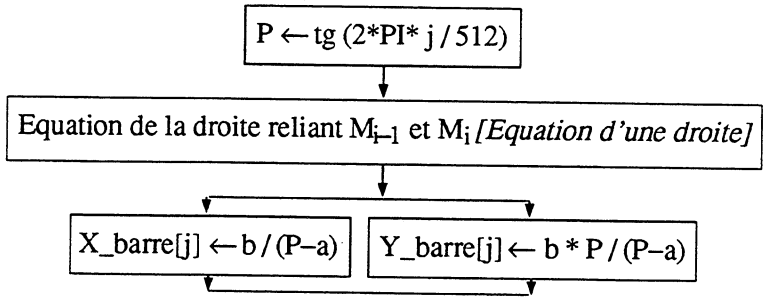
68.11) Intersection

Strate père : *[Ajustement de ces coordonnées]*

Données : i, j, X_chapeau[ ], Y_chapeau[ ]

Résultats : X_barre[ ], Y_barre[ ]

Schéma de résolution :



68.12) Equation d'une droite

Strate père : *[Intersection]*

Données : X_chapeau[ ], Y_chapeau[ ]

Résultats : a, b

Equations de liens :

N ← 2

M[1,1] ← X_chapeau[i]

M[1,2] ← 1

M[2,1] ← X_chapeau[i–1]

M[2,2] ← 1

v[1] ← Y_chapeau[i]

v[2] ← Y_chapeau[i–1]

w[1] → a

w[2] → b

Schéma de résolution :

Résolution par la méthode de Gauss *(10)*

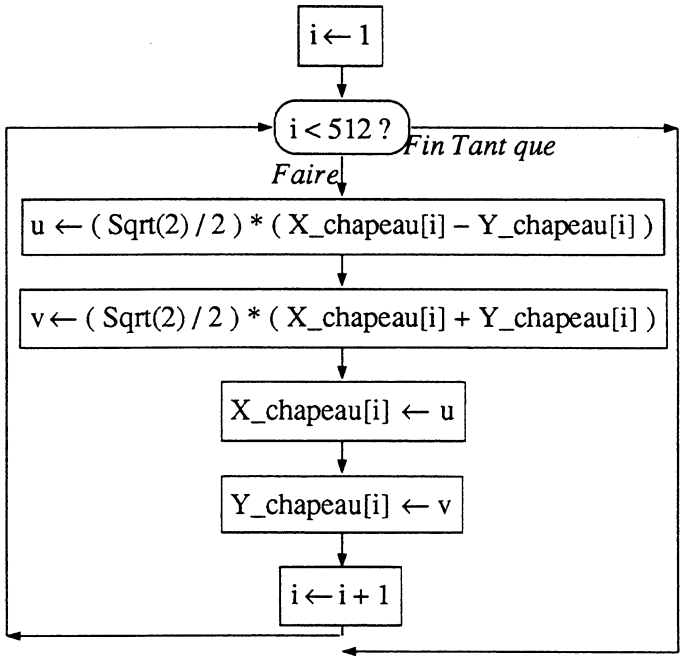
68.13) Rotation de 45°

Strate père : *[Ajustement de ces coordonnées]*

Données : X_chapeau[ ], Y_chapeau[ ]

Résultats : X_chapeau[ ], Y_chapeau[ ]

Schéma de résolution :



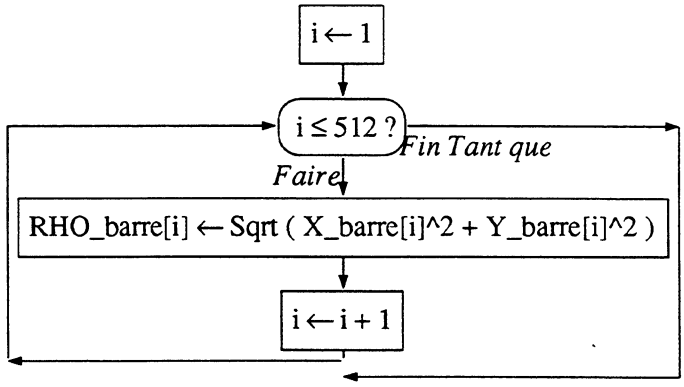
69) Coordonnées des points par rapport au plan P

Langage = *Pascal*

Données : X_barre[ ], Y_barre[ ]

Résultats : RHO_barre[ ]

Schéma de résolution :



C.7 Interprétation Signal de ces strates

Nous donnons ci–dessous les interprétations en termes du langage Signal pour les strates. Nous montrons sur ces exemples comment nous appliquons les algorithmes d'élimination de variables et comment nous pouvons interpréter les équations de contraintes obtenues. Tous

ces calculs ont été réalisés au moyen de l'interface Grif–Reduce par l'entremise de l'éditeur GramMath présenté dans l'annexe B.

C.7.1 Strate (1)

1) Problème de modélisation

Langage = *Naturel*

Données : ρ_i, z_i, i=1..512

Résultats : $\bar{\rho}_i$, $\bar{z}_i$, i=1..512

Descriptif :

Variables locales :

P

Conditions :

P = plan aux moindres carrés de $\{\rho_i, z_i\}_{i=1..512}$

$\left(\frac{2\pi i}{512}, \bar{\rho}_i\right)$ = coordonnées polaires de (ρ_i, z_i) dans le plan P.

$\bar{z}_i$ = distance de (ρ_i, z_i) au plan P.

Nous avons dans la description de cette strate trois calculs différents qui reviennent à déterminer le plan P en fonction des coordonnées cylindriques, et des nouvelles coordonnées cylindriques, $\rho\_$ et $z\_$, respectivement en fonction des anciennes coordonnées cylindriques pour l'une, ρ et z, et pour l'autre en fonction de ρ, z et de P. Ceci nous donne le système suivant :

$$\left\{ \begin{array}{l} P^2 = \rho^2 z^2 \\ \rho\_^2 = \rho^2 z^2 \\ z\_^2 = \rho^2 z^2 P^2 \end{array} \right\}$$

Comme nous voulons savoir si ce système détermine bien les résultats, $\rho\_$ et $z\_$, en fonction des données, ρ et z, et ce indépendemment de la variable locale P, nous faisons une élimination de cette variable P. Nous obtenons ainsi :

$$\left\{ \begin{array}{l} \rho\_^2 = \rho^2 z^2 \\ z\_^2 = \rho^2 z^2 \end{array} \right\}$$

Ce qui donne bien le résultat attendu.

C.7.2 Strate (2)

2) Plan aux moindres carrés

Langage = *Naturel – Mathématiques*

Strate père : *[Problème de modélisation]*

Données : ρ_i, z_i, i=1..512

Résultats : ρ_i, z_i, i=1..512, P

Descriptif :

Variables locales :

x_i, y_i, i=1..512, α, β, γ

Conditions :

P = plan aux moindres carrés de $\left\{\rho_i, z_i\right\}_{i=1..512}$

$$P \Leftrightarrow (\alpha, \beta, \gamma),\ z = \alpha x + \beta y + \gamma,$$

$$\underset{(\alpha,\beta,\gamma)}{\text{Min}} \sum_i [\, z_i - (\alpha x_i + \beta y_i + \gamma)\,]^2$$

Maintenant les données sont { ρ, z } et le résultat est P. Aussi devons–nous éliminer les variables locales présentes dans ce système, soit x, y, α, β, γ.

$$\left\{ \begin{array}{l} P^2 = \rho^2 z^2 \\ P^2 = \alpha^2 \beta^2 \gamma^2 \\ \alpha^2 = x^2 y^2 z^2 \\ \beta^2 = x^2 y^2 z^2 \\ \gamma^2 = x^2 y^2 z^2 \end{array} \right\}$$

Nous obtenons les équations de contraintes suivantes.

$$\left\{ \begin{array}{l} P^2 = \rho^2 z^2 \\ P^2\left(z^2 - 1\right) = 0 \end{array} \right\}$$

Cependant pour les interpréter, nous sommes obligés, ici, de faire une substitution de P de la première équation dans la seconde. Ceci nous donne :

$$\left\{ \begin{array}{l} P^2 = \rho^2 z^2 \\ 0 = 0 \end{array} \right\}$$

L'interprétation nous renseigne que la variable P dépend des variables ρ et z, ce qui correspond bien à la description de cette strate.

C.7.3 Strate (2.1)

2.1) Passage en coordonnées cartésiennes

Strate père : *[Plan aux moindres carrés]*

Données : ρ_i, z_i, i

Résultats : x_i, y_i, i

Descriptif :

Conditions :

$$x_i = \rho_i \cos\left(\frac{2\pi i}{512}\right)$$

$$y_i = \rho_i \sin\left(\frac{2\pi i}{512}\right)$$

Cette fois–ci nous avons le système :

$$\left\{ \begin{array}{l} x^2 = \rho^2 \\ y^2 = \rho^2 \end{array} \right\}$$

Ce qui nécessite aucune élimination car les résultats x et y sont bien fonction des données ρ.

C.7.4 Strate (2.2)

2.1) Traduction retour

Strate père : *[Plan aux moindres carrés]*

Données : α, β, γ

Résultats : P

Descriptif :

Conditions :

$P \Leftrightarrow (\alpha, \beta, \gamma), z = \alpha x + \beta y + \gamma,$

Nous avons une seule condition qui a pour interprétation Signal l'équation suivante :

$$\left\{ P^2 = \alpha^2 \beta^2 \gamma^2 x^2 y^2 z^2 \right\}$$

L'élimination des variables locales nous donne l'équation de contrainte suivante :

$$\left\{ P^2 \left(\alpha^2 \beta^2 \gamma^2 - 1 \right) = 0 \right\}$$

Nous ne pouvons pas interpréter directement cette équation. Seule l'étude cas par cas est envisageable. Supposons que P est indéfini, soit P=0, nous aurons 0=0 ce qui est vrai. Supposons que P est défini, soit P^2=1, alors nous aurons : $\alpha^2 \beta^2 \gamma^2 = 1$, ce qui signifie que les variables α, β, γ sont définies. Cette étude de cas ne nous a pas révélé de contradictions, aussi l'équation de contrainte est–elle cohérente par sa signification.

C.7.5 Strate (3)

3) Détermination des nouvelles coordonnées

Langage = *Naturel – Mathématiques*

Strate père : *[Problème de modélisation]*

Données : ρ_i, z_i, P

Résultats : $\bar{\rho}_i$, $\bar{z}_i$

Comme nous n'avons pas de descriptif au sein de cette strate, nous ne pouvons pas faire d'interprétation.

C.7.6 Strate (3.1)

3.1) Traduction aller

Strate père : *[Détermination des nouvelles coordonnées]*

Données : P

Résultats : α, β, γ

Descriptif :

Conditions :

$(\alpha, \beta, \gamma), z = \alpha x + \beta y + \gamma \Leftrightarrow P$

Nous retrouvons le système vu en C.7.4 :

$$\left\{P^2 = \alpha^2 \beta^2 \gamma^2 x^2 y^2 z^2\right\}$$

Et l'étude sur l'équation de contrainte nous conduit à la même conclusion qu'en C.7.4.

$$\left\{P^2\left(\alpha^2 \beta^2 \gamma^2 - 1\right) = 0\right\}$$

C.7.7 Strate (3.2)

3.1) Ajustement des nouvelles coordonnées

Strate père : *[Détermination des nouvelles coordonnées]*

Données : $\alpha, \beta, \gamma, \rho_i, z_i, \alpha, \beta, \gamma$

Résultats : $\bar{\rho}_i$, $\bar{z}_i$

Comme nous n'avons pas de descriptif au sein de cette strate, nous ne pouvons pas faire d'interprétation.

C.7.8 Strate (4)

4) Projection des strates de C.1

Langage = *Mathématique*

Strate père : *[Problème de modélisation]*

Données : ρ_i, z_i, i=1..512

Résultats : $\bar{\rho}_i$, $\bar{z}_i$, i=1..512

Descriptif :

Variables locales :

α, β, γ, P

Conditions :

$P = (\alpha, \beta, \gamma)$

P = plan aux moindres carrés de $\left\{\rho_i, z_i\right\}_{i = 1 .. 512}$

$\left(\frac{2\pi i}{512}, \bar{\rho}_i\right)$ = coordonnées polaires de $\left(\rho_i, z_i\right)$ dans le plan P.

$\bar{z}_i$ = distance de $\left(\rho_i, z_i\right)$ au plan P.

Les données sont { ρ, z } et les résultats { p_, z_ }. Aussi devons–nous éliminer les variables locales présentes dans ce système, soit P, α, β, γ.

$$\left\{ \begin{array}{l} P^2 = \alpha^2 \beta^2 \gamma^2 \\ P^2 = \rho^2 z^2 \\ \rho\_^2 = \rho^2 z^2 \\ z\_^2 = \rho^2 z^2 P^2 \end{array} \right\}$$

Nous obtenons des contraintes qui correspondent à ce que nous attendions.

$$\left\{ \begin{array}{l} \rho\_^2 = \rho^2 z^2 \\ z\_^2 = \rho^2 z^2 \end{array} \right\}$$

C.7.9 Strate (5)

5) Polaires – Cartésiennes

Langage = *Programmation*

Strate père : *[Projection des strates de]*

Données : ρ_i, z_i, i

Résultats : x_i, y_i, i

Descriptif :

Variables locales :

PI

Conditions :

PI ← 3.1415

Pour i **allant de** 1 **à** 512 **faire**

X[i] ← RHO[i] * cos(2.0 * PI * i / 512.0)

Y[i] ← RHO[i] * sin(2.0 * PI * i / 512.0)

Nous définissons une variable locale, Pi, et nous calculons les résultats x et y. Les équations sont :

$$\left\{ \begin{array}{l} Pi^2 = 1 \\ x^2 = \rho^2 Pi^2 \\ y^2 = \rho^2 Pi^2 \end{array} \right\}$$

Et nous avons une équation de contrainte après élimination qui est difficilement interprétable :

$$\left\{ \rho^2 x^2 y^2 + \rho^2 x^2 + \rho^2 y^2 - \rho^2 + x^2 y^2 + x^2 - y^2 = 0 \right\}$$

Une élimination de la variable ρ nous renseigne que les résultats x et y sont définis ou non, simultanément car :

$$\left\{ x^2 = y^2 \right\}$$

Et une substitution dans l'équation de contrainte, nous donne finalement :

$$\left\{ \rho^2 = x^2 y^2 \right\}$$

C.7.10 Strate (6)

6) Calcul du plan aux moindres carrés

Langage = *Mathématiques*

Strate père : *[Plan aux moindres carrés]*

Données : x_i , y_i , z_i

Résultats : α , β , γ

Descriptif :

Variables locales :

A, b

Conditions :

$$A = \begin{pmatrix} \sum_i x_i^2 & \sum_i x_i y_i & \sum_i x_i \\ \sum_i x_i y_i & \sum_i y_i^2 & \sum_i y_i \\ \sum_i x_i & \sum_i y_i & 512 \end{pmatrix}, \quad b = \begin{pmatrix} \sum_i x_i z_i \\ \sum_i y_i z_i \\ \sum_i z_i \end{pmatrix}$$

$$A \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = b$$

Le calcul du plan en fonction des variables locales A et b et des données x, y, et z nous donne l'interprétation Signal suivante :

$$\left\{ \begin{array}{l} A^2 = x^2 y^2 \\ b^2 = x^2 y^2 z^2 \\ \alpha^2 = A^2 b^2 \\ \beta^2 = A^2 b^2 \\ \gamma^2 = A^2 b^2 \end{array} \right\}$$

Soit, après élimination de A et b :

$$\{ \alpha^2 \beta^2 \gamma^2 = x^2 y^2 z^2 \}$$

Cette équation de contrainte répond au fait que les résultats α, β, γ dépendent des variables x, y, z. Si, nous étudions les solutions fournies par notre algorithme sur les solutiopns A et b, nous avons :

$$\left\{ \begin{array}{l} A^2 = -\alpha^2 \beta^2 \gamma^2 x^2 y^2 - \alpha^2 \beta^2 \gamma^2 - x^2 y^2 z^2 + x^2 y^2 \\ b^2 = -\alpha^2 \beta^2 \gamma^2 x^2 y^2 z^2 + \alpha^2 \beta^2 \gamma^2 + x^2 y^2 z^2 \end{array} \right\}$$

Soit encore par simplification en tenant compte de l'équation de contrainte :

$$\left\{ \begin{array}{l} A^2 = x^2 y^2 \\ b^2 = x^2 y^2 z^2 \end{array} \right\}$$

Ce dernier système signifie que les variables locales A et b ne dépendent bien que des données x, y et z.

C.7.11 Strate (7)

7) Coordonnées polaires sur le plan P

Langage = *Mathématiques*

Strate père : *[Détermination des nouvelles coordonnées]*

Données : $x_i, y_i, z_i, \alpha, \beta, \gamma$

Résultats : $\bar{x}_i, \bar{y}_i, \bar{z}_i$

Descriptif :

Variables locales :

$\tilde{x}_i, \tilde{y}_i, \tilde{z}_i$

Comme nous n'avons pas de descriptif au sein de cette strate, nous ne pouvons pas faire d'interprétation.

C.7.12 Strate (8)

8) Coordonnées des points par rapport au plan P

Langage = *Mathématiques*

Données : $\bar{x}_i, \bar{y}_i, \bar{z}_i$

Résultats : $\bar{\rho}_i, \bar{z}_i$

Descriptif :

Conditions :

$$\bar{\rho}_i = \sqrt{\bar{x}_i^{-2} + \bar{y}_i^{-2}}$$

Le système correspondant à cette strate est :

$$\{\rho\_^2 = x\_^2 \; y\_^2\}$$

Il met en relief le fait que la variables z_ est inutilisée.

C.7.13 Strate (20)

20) Résolution des 512 systèmes 4×4

Langage = *Algorithmique*

Strate père : *[Coordonnées polaires sur le plan P]*

Données : $x_i, y_i, z_i, \alpha, \beta, \gamma$

Résultats : $\tilde{x}_i, \tilde{y}_i, \tilde{z}_i$

Descriptif :

Variables locales :

δ_i

Conditions :

$$\begin{cases} (\tilde{x}_i - x_i,\ \tilde{y}_i - y_i,\ \tilde{z}_i - z_i) = \delta_i(\alpha, \beta, -1) \\ \tilde{z}_i = \alpha \tilde{x}_i + \beta \tilde{y}_i + \gamma \end{cases}$$

Nous avons ici une double affectation ou un effet de bord. Le système :

$$\left\{ \begin{array}{l} \tilde{x}^2 \tilde{y}^2 \tilde{z}^2 x^2 y^2 z^2 = \delta^2 \alpha^2 \beta^2 \\ \tilde{z}^2 = \alpha^2 \beta^2 \gamma^2 \tilde{x}^2 \tilde{y}^2 \end{array} \right\}$$

nous donne ces équations de contraintes

$$\left\{ \begin{array}{l} \tilde{z}^2 = \alpha^2 \beta^2 \gamma^2 \tilde{x}^2 \tilde{y}^2 \\ \tilde{x}^2 \tilde{y}^2 \tilde{z}^2 x^2 y^2 z^2 \left(\alpha^2 \beta^2 - 1\right) = 0 \end{array} \right\}$$

Une substitution de la première équation dans la seconde nous donne :

$$\left\{ \begin{array}{l} \tilde{z}^2 \tilde{x}^2 \tilde{y}^2 = \alpha^2 \beta^2 \gamma^2 \\ 0 = 0 \end{array} \right\}$$

C'est–à–dire une équation toujours vérifiée et une autre qui précise quelles sont les variables calculées en fonction des autres.

C.8 Algorithme de Gauss

10) Résolution de système par la méthode de Gauss

Langage = *Programmation*

Données : M, v, N

Résultats : w

Commentaire :

Ceci est une strate qui nous sert de modèle pour des résolution particulières de systèmes linéaires. Nous supposons que la matrice M est inversible.

Descriptif :

Préconditions :

M :: Matrice carrée d'ordre N

v :: Vecteur de taille N

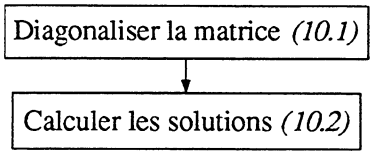
Conditions :

M w = v

Postconditions :

w :: Vecteur de taille N

Schéma de résolution :



10.1) Trianguler le système

Strate père : *[Résolution de système par la méthode de Gauss]*

Données : M, v, N

Résultats : M, v

Commentaire :

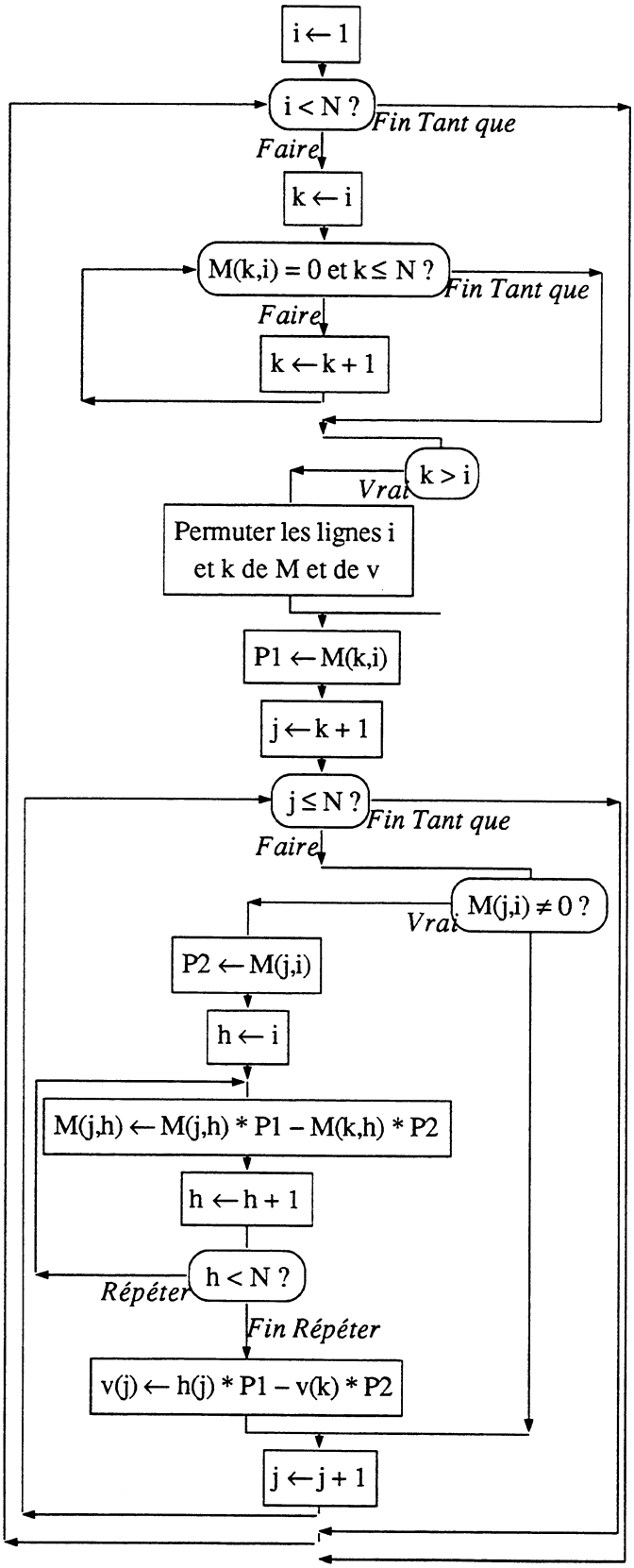
Un schéma de résolution graphique trop détaillé peut être difficilement lisible par rapport à un schéma de résolution textuel.

Descriptif :

Variables locales :

i, j, k, h, P1, P2

Schéma de résolution :



10.2) Calcul d'une solution d'un système linéaire

Strate père : *[Résolution de système par la méthode de Gauss]*

Données : M, v, N

Résultats : w

Commentaire :

Ici nous présentons un schéma de résolution sous forme textuelle.

Descriptif :

Variables locales :

i, j, S

Préconditions :

M :: Matrice carrée triangulaire supérieure

Schéma de résolution :

```
i ← N
répéter
    | S ← 0
    | j ← i
    | tant que j < N
    | faire
    |     | j ← j + 1
    |     | S ← S + w(j) * M(i,j)
    | w(i) ← S / M(i,i)
    | i ← i - 1
jusqu'à i < 1
```

Références bibliographiques

[1] S.S. Abhyankar, "Historical ramblings in algebraic geometry and related algebra", *American Mathematical monthly*, (83), pp. 409–448, 1976.

[2] "Algorithms in Real Algebra Geometry", *J. Symbolic Computing*, 5 (1 & 2), pp. 1–267, février/avril 1988.

[3] J.A. Alonso, E. Briales et A. Riscos, "Preuves Automatiques dans le Calcul Propositionnel et des Logiques Trivalentes", *Proc. Congress on Computationnal Geometry and Topology and Computation*, Sevilla, 1987.

[4] D.S. Arnon, R. Beach, K. McIsaac et C. Waldspurger, *Camino Real : An Interactive Mathematical Notebook*, (RR), Xerox Corporation, Palo Alto, 1988.

[5] D.S Arnon, "Geometric Reasoning with Logic and Algebra", *Geometric Reasoning*, édité par D. Kapur et J. Mundy, pp. 37–60, MIT Press, 1989.

[6] D. Arnon, "On mechanical quantifier elimination for elementary algebra and geometry : solution for a non trivial problem", *Proc. EUROCAL 85, vol. 2*, pp. 270–271, LNCS 204, Springer–Verlag, 1985.

[7] L. Biard, *Méthode algorithmique d'implicitisation et d'inversion, Application au lancer de rayons*, Thèse, Université Joseph–Fourier, 1990.

[8] M. Bouhier, *Génération de la sémantique d'une formule mathématique*, (RR 783–M), TIM3–Imag, 1988.

[9] H.–J. Bürckert, "A Resolution Principe for a Logic with Restricted Quantifiers", *Lecture Notes in Artificial Intelligence 568*, 1991.

[10] M. Chavret, P. Chenin, P.–O. Garreau et R. Rivaux, *Création d'un outil en Smalltalk–80 pour l'automatisation et la simplification du système d'équations d'une carrière*, (Rapport de maîtrise), LMC–Imag, juin 1991.

[11] J. Chazarain, A. Riscos, J.A. Alonso et E. Briales, "Multi–valued Logic and Gröbner Bases with Applications to Modal Logic", *J. Symbolic Computation*, 11, pp. 181–194, 1991.

[12] S.–C. Chou, "A Method for the Mecanical Derivation of Formulas in Elementary Geometry", *Journal of Automated Reasoning*, 2(4), pp. 291–299, 1986.

[13] S.–C. Chou, *Mecanical geometry theorem proving*, Reidel (éd.), 1988.

[14] S.–C. Chou et X. S. Gao, "Proving Geometry Statements of Constructive type", édité par Kapur (éd.), CADE–11, LNAI 607, juin 1992.

[15] P. J. Cohen, "Decision Procedures for Real and p–Adic Fields", *Communications*

on Pure and Applied Mathematics, 32, pp. 131–151, 1969.

[16] M. Coste, ''Géométrie semi–algébrique effective'', *Calcul formel et outils algébriques pour la modélisation géométrique, sur le thème : « Outils mathématiques et informatiques des modèles géométriques »*, CNRS, Paris, 9–10 mars 1988.

[17] J.H. Davenport, Y. Siret, et E. Tournier, *Calcul formel, Systèmes et algorithmes de manipulations algèbriques*, Masson, coll. ERI, 1987.

[18] J. Davenport, ''A "piano movers" problem'', *SIGSAM Bulletin*, 20 (1 & 2), pp. 15–17, 1986.

[19] J.–P. Desclés et R. Fraïssé, *Quelques réflexions sur les rapports entre linguistique et mathématiques*, in Penser les mathématiques, appendice 1, partie I : Des mathématiques au langage, Points Sciences, 1982.

[20] H.Farreny et M. Ghalla, *Eléments d'intelligence artificielle*, TNT, Série IA Hermès, 1987.

[21] P.–O. Garreau, *Manuel utilisateur de l'interface Grif/Reduce*, (RT 82), LMC–Imag, Grenoble, mai 1992.

[22] S. Lang, *Algebra*, Addison Wesley, 1965.

[23] H. Laurent, *L'élimination*, n° 7 Scientia, Gauthiers–Villars, 1900.

[24] M. Le Borgne, *Systèmes dynamiques sur les corps finis*, Thèse, Université de Rennes I, septembre 1993.

[25] P. Le Guernic et A. Benveniste, *Real–Time, Synchronous, Data–Flow Programming: the Language SIGNAL and its Mathematical Semantics*, (298), IRISA/INRIA, Rennes, juin 1986.

[26] P. Le Guernic et T. Gautier, *Data–Flow to Von Neumann: The SIGNAL Approach*, (1229), INRIA, Rocquencourt, mai 1990.

[27] F.S. Macaulay, ''On some formula in elimination'', *Proceedings of London Mathematical society*, pp. 3–27, mai 1902.

[28] F.S. Macaulay, ''Note on the resultant of a number of polynomials of the same degree'', *Proceedings of London Mathematical society*, pp. 14–21, juin 1921.

[29] M. MacCallum et F. Wright, *Algebraix Computing with Reduce*, vol. 1, Lecture Notes from the First Brazilian School on Computer Algebra, Clarendon Press . Oxford, 1991.

[30] D. Manocha, ''MultiPolynomial Resultant Algorithms'', *J. Symbolic Computation*, 15, pp. 99–122, 1993.

[31] J. Marchand, ''Elimination des quantificateurs et décomposition algébrique'',

Calcul formel et outils algébriques pour la modélisation géométrique, sur le thème : « Outils mathématiques et informatiques des modèles géométriques », CNRS, Paris, 9–10 mars 1988.

[32] H. Minkowski, *La géométrie des nombres*, 1896.

[33] H. Pollard, *The Theory of Algebraic Numbers*, Mathematical Association of America, 1950.

[34] V. Quint, *Les langages de Grif*, Grif S.A., Bull–Imag, juillet 1993.

[35] V. Quint, I. Vatton, J. André et H. Richy, ''Grif et l'édition de documents structurés : nouveaux développements'', *Cahiers GUTenberg*, (9), pp. 49–65, juillet 1991.

[36] V. Quint, *La représentation pivot de Grif*, version 2.0, Bull–Imag, décembre 1989.

[37] V. Quint, I. Vatton et H. Bedor, ''Grif, an Interactive Environment for TEX'', *TEX for Scientific Documentation*, édité par J. Désarménien, pp. 145–158, Springer–Verlag, 1986.

[38] J. Rengar, ''On the Computational Complexity and Geometry of the First–Order Theory of Reals – Part III: Quantifier Elimination'', *J. Symbolic Computing*, _(_), pp. 255–352, 1993.

[39] P. Ribenboim, *L'arithmétique des corps*, Hermann, Paris, 1972.

[40] R. F. Ritt, *Differential Algebra*, AMS Colloquium Publications, New–York, 1950.

[41] I. Stewart et D. Tall, *Algebraic Number Theory*, Chapman and Hall, 1987.

[42] A. Tarski, *A decision method for elementary algebra and geometry*, 2nd. ed. Univ. Calif. Press, Berkely, 1951.

[43] G. Viry, ''Factorisation of Multivariate Polynomails with Coefficients in F_p'', *J. Symbolic Computing*, (15), pp. 371–391, 1993.

[44] S. Wolfram, *Mathematica – A System for Doing Mathematics by Computer*, Addison–Wesley, 1988.

[45] W.–T. Wu, ''Basic Principles of Mecanical Theorem Proving in Elementary Geometries'', *Journal of Automated Reasoning*, 2(4), pp. 221–252, 1986.

[46] W.–T. Wu, *Mecanical geometry theorem proving*, Springer–Verlag, 1994.

Annexe C

Activités de modélisation au moyen des strates