THÈSE de DOCTORAT de L'UNIVERSITÉ PARIS 6

spécialité :

Informatique

présentée par :

Marc MORENO MAZA

Sujet de la Thèse :

Calculs de Pgcd au-dessus des Tours d'Extensions Simples et Résolution des Systèmes d'Équations Algébriques

Soutenue le vendredi 13 juin 1997 devant le jury composé de :

M. Claude BERGE, président

Mme Dominique DUVAL, rapporteur

Mme Marie-Françoise ROY, rapporteur

M. James DAVENPORT, rapporteur

Mme Michèle SORIA, examinateur

M Daniel LAZARD, directeur de thèse

THÈSE de DOCTORAT de L'UNIVERSITÉ PARIS 6

spécialité :

Informatique

présentée par :

Marc MORENO MAZA

Sujet de la Thèse :

Calculs de Pgcd au-dessus des Tours d'Extensions Simples et Résolution des Systèmes d'Équations Algébriques

Soutenue le vendredi 13 juin 1997 devant le jury composé de :

M. Claude BERGE, président

Mme Dominique DUVAL, rapporteur

Mme Marie-Françoise ROY, rapporteur

M. James DAVENPORT, rapporteur

Mme Michèle SORIA, examinateur

M Daniel LAZARD, directeur de thèse

A mes parents,
A mon frère,
A Fanou.

Remerciements

Ce travail est avant toute chose la réunion d'un ensemble de circonstances favorables à sa réalisation. J'ai tout d'abord eu la chance de travailler avec Daniel Lazard, mon directeur de thèse. Il m'a offert la possibilité de prolonger ses idées qui, sur le sujet de la résolution des systèmes d'équations algébriques, me semblent être parmi les plus élégantes et les plus performantes. Avec son équipe, dont chaque membre possède, de remarquables qualités tant humaines que scientifiques, il m'a donné un environnement de travail exceptionnel.

Certaines parties de cette thèse sont des travaux communs avec d'autres membres de l'équipe de Daniel Lazard. La troisième section du second chapitre et l'intégralité du quatrième chapitre constituent un travail commun avec Renaud Rioboo. C'est de plus sur ce travail que s'appuie le cinquième chapitre. Les quatrième et sixième sections du troisième chapitre, ainsi que le sixième chapitre constituent un travail commun avec Philippe Aubry. Ce travail a permis un rapprochement et des avancées, tant théoriques qu'expériementales, sur différents travaux ([Wu87, Laz91a, Kal91, Wan93a]). Jean-Charles Faugère en nous assistant grâce à son logiciel GB ([Fau94]) nous a permis d'effectuer des calculs qui n'auraient pas été possibles sans. En particulier, ce logiciel nous a permis de valider nos propres programmes.

C'est un grand honneur pour moi, Mesdames et Monsieurs les membres du jury, que vous ayez accepter cette fonction.

Je remercie également les personnes qui ont eu la patience de relire tout ou partie de ce document. En particulier les membres de mon jury, Philippe Aubry, Renaud Rioboo et Fanou, ma compagne. Par leurs suggestions, ils ont grandement contribué à la mise en place de ce document.

Je remercie également les différentes équipes enseignantes des universités Paris 6 et Paris 7 dont j'ai eu la chance de faire partie. Je remercie en particulier l'équipe de MIAS11 qui m'a permis d'adapter mon emploi du temps dans les phases de travail intensif sur ce document

Je dédie également ce travail à mes parents. Pendant mes longues années d'étude, leur affection et leur soutien ne m'ont jamais manqué. Ma chance ne s'arrêté pas là. J'ai aussi des amis formidables qui, avec mes parents et mon directeur de thèse, m'ont aidé à surmonter les difficultés que j'ai rencontré. Vous m'avez pardonné mes humeurs (enfin, je le crois ...) et été une merveilleuse équipe de supporters. Je ne voudrais oublier personne. Alors, si vous me le permettez, je me contenterai de pousser avec vous, notre cri de guerre ([Aug93, Fau94, Reb97]).

¡ Cucarachas !

¡ A por ellas !

¡ Uno, dos, très !

Introduction

Pour résoudre un système de deux équations algébriques à une inconnue, on sait que l'on peut calculer le pgcd des polynômes définissant les équations. Les racines communes de nos polynômes sont alors les racines de leur pgcd. Lorsque nos polynômes ont deux inconnues, ce procédé échoue. Ainsi, les polynômes x^2+y^2 et x^2-y^2 n'ont pas de facteur commun, leur pgcd est 1 et pourtant $x=0$ et $y=0$ est une solution. Pour résoudre de tels systèmes, les méthodes les plus plus étudiées, surtout depuis le milieu des années 60 ([Buc65]), généralisent celles de l'algèbre linéaire en calculant des systèmes particuliers, appelés *bases de Gröbner*. Ces méthodes sont aussi les plus répandues dans les systèmes de calcul formel et l'on dispose aujourd'hui d'implantations très performantes ([Fau94]).

Lorsqu'une de ces méthodes est appliquée à un système d'équations algébriques, un certain nombre d'informations capitales peuvent être obtenues, comme le nombre de solutions du système. Cependant la lisibilité et l'utilisation de ces solutions, surtout lorsqu'elles sont en nombre infini, ne sont pas toujours plus aisées que dans le système initial. En particulier, une base de Gröbner peut avoir beaucoup plus d'équations que le système de départ.

À ce stade, nous devons préciser ce que nous souhaitons avoir comme description commode des solutions. Dans le cadre où nous nous plaçons, à savoir celui du calcul symbolique, la lisibilité des solutions d'un système est généralement aisée si celui-ci est *triangulaire*. On entend par système triangulaire, un système de polynômes, où, si un ordre a été fixé sur les variables, deux polynômes distincts n'ont pas la même plus grande variable. L'intérêt est bien sûr analogue à celui de la triangulation des systèmes d'équations linéaires. Le remplacement d'un système quelconque par un système triangulaire n'est pas toujours possible. Cependant, tout système peut être remplacé par une réunion finie de systèmes triangulaires. Cela signifie que toute solution du système de départ est solution de l'un des systèmes triangulaires le remplaçant, et inversement. Les méthodes de triangulation des systèmes sont apparues dans les années 30 ([Rit32]) et sont développées essentiellement depuis les années 80 ([Wu84a]).

Ces méthodes ont aussi leurs points faibles. Certaines remplacent par exemple un système sans solutions par un autre sans découvrir que ce remplacement est inutile. De plus, les programmes implantant ces méthodes ne produisent pas toujours de solutions en un temps raisonnable là où les méthodes fondées sur les bases de Gröbner y parviennent.

Dans [Laz92a], pour un système de départ avec un nombre fini de solutions, Daniel Lazard a proposé un algorithme, appelé *lexTriangular*, remplaçant une base de Gröbner de ce système par une réunion finie de systèmes triangulaires ayant de bonnes propriétés (comme avoir nécessairement des solutions). D'autre part dans [Laz91a], Daniel Lazard a proposé un nouvel algorithme remplaçant tout système par une réunion finie de systèmes triangulaires ayant de bonnes propriétés. D'un point de vue théorique, ce second algorithme est mieux adapté aux systèmes ayant un nombre infini de solutions que ne le sont les méthodes calculant des bases de Gröbner.

Dans un premier temps, Daniel Lazard m'a proposé de mettre en œuvre le premier de ces deux algorithmes dans le système de calcul formel AXIOM ([JS92]). Cependant, cet algorithme nécessitait des *calculs de pgcd* qui s'avéraient très rapidement impraticables. Renaud Rioboo et moi-même ([MR95]) avons alors mis au point un nouvel algorithme de calcul de pgcd permettant de rendre très performant l'algorithme *lexTriangular*. Des systèmes d'équations algébriques qui ne pouvaient jusqu'alors être résolus de façon automatisée ont pu l'être par nos programmes. Ce premier travail est rapporté dans le quatrième chapitre de cette thèse.

Dans un second temps, j'ai adapté la méthode de calcul de pgcd réalisé avec Renaud Rioboo à l'algorithme de Daniel Lazard exposé dans [Laz91a]. Ceci fait l'objet de notre cinquième chapitre. Nous verrons que la situation est relativement plus compliquée que pour le premier algorithme de pgcd, essentiellement à cause du fait que l'on souhaite traiter des systèmes ayant éventuellement un nombre infini de solutions.

La découverte de ces algorithmes performants de pgcd à engendrer un certain travail théorique. En effet, nous avons dit au début de cette introduction, que la notion habituelle de pgcd était inadaptée à la résolution des systèmes d'équations à plusieurs inconnues. Il a donc fallu préciser le cadre de nos calculs de pgcd. Cette partie théorique se trouve rassemblée dans les trois premiers chapitres de cette thèse.

On y rappelle tout d'abord les définitions et théorèmes sur lesquels reposent en général les méthodes de résolution symbolique des systèmes d'équations algébriques. Dans les second et troisième chapitres, on rappelle ou introduit les notions plus spécifiques aux méthodes développées dans cette thèse. On introduit en particulier la notion de *scindage d'un idéal* dans un anneau commutatif unitaire. Cette notion est bien adaptée à la présentation et à l'étude des méthodes de résolution qui remplace un système d'équations par une réunion de systèmes. Elle permet aussi de donner une présentation en termes d'idéaux de l'*évaluation dynamique* du système D5 de calcul avec les nombres algébriques ([DDD85]). Plus généralement, elle permet l'étude des algorithmes qui, travaillant dans un anneau $\mathbf{A}$ modulo un idéal I de cet anneau, cherchent à scinder leurs calculs en différents cas pour travailler en fait dans une structure-produit isomorphe à $\mathbf{A}/I$.

On présente ensuite la notion de pgcd mise en œuvre dans nos algorithmes. Son utilisation repose tout d'abord sur la remarque suivante. Si $\mathbf{A}$ est un anneau nœthérien réduit alors $\mathbf{fr}(\mathbf{A})$, l'anneau total des fractions de $\mathbf{A}$, est un produit de corps et, par conséquent, l'anneau des polynômes univariés à coefficients dans $\mathbf{fr}(\mathbf{A})$ est principal. Elle repose aussi sur une généralisation de la méthode des sous-résultants pour le calcul d'un pgcd de deux polynômes univariés à coefficients dans un anneau intègre ([MR95]).

On fait ensuite le lien entre les *bons* systèmes triangulaires (ceux qui ont des solutions) et cette vision *univariée* des polynômes à travers les notions duales de *tour d'extensions simples* et d'*ensemble régulier* que nous introduisons dans le troisième chapitre. L'étude des ensembles réguliers nous a conduit à établir avec Philippe Aubry et Daniel Lazard un théorème donnant quatre caractérisations équivalentes d'un ensemble régulier. L'une d'elles est d'être une *chaîne régulière*, notion introduite indépendamment par Yang Lu et Zhang Jingzhong ([YZ91]) d'une part et Michael Kalkbrener ([Kal91]) d'autre part. Une autre est d'être un ensemble caractéristique ([Rit32]) de son idéal saturé. Ce théorème est rapporté dans [ALM97]. Cet article reprend essentiellement notre troisième chapitre. Il contient également des résultats sur les ensembles caractéristiques et les bases de Gröbner que j'ai obtenus peu avant la soutenace de cette thèse et qui ne figurent dans ce document.

Nous étudions ensuite les zéros des systèmes triangulaires et rappelons différents résultats donnant une décomposition d'une variété affine comme réunion des zéros de systèmes triangulaires. Le cinquième chapitre présente du reste un algorithme permettant d'obtenir l'une de ces décompositions à l'aide d'ensembles triangulaires particuliers, les *ensembles de Lazard* ([Laz91a]). Il s'agit d'ensembles réguliers dont les tours d'extensions simples associées sont des produits de corps. D'où l'intervention de nos algorithmes de pgcd.

Le quatrième chapitre de cette thèse est en fait la version française de l'article [MR95] écrit avec Renaud Rioboo. C'est le plus ancien des textes de cette thèse. Il ne bénéficie donc pas autant des trois premiers chapitres et peut paraître assez technique. Cependant, la lecture de certains passages est essentielle pour aborder le cinquième chapitre. En particulier, la première section, qui a été réécrite, présente la problématique. De même les algorithmes des sections 2.3 et 2.4 doivent être parcourus. On les retrouvera au cinquième chapitre avec quelques *complications supplémentaires*. L'algorithme de pgcd de ce quatrième chapitre a permis une première implantation performante de l'algorithme *lexTriangular* et une résolution complètement automatisée du système-test `Cyclic-7` ([BF91]).

Le cinquième chapitre de cette thèse présente un algorithme de résolution de systèmes d'équations algébriques basé sur celui de Daniel Lazard ([Laz91a]). Cet algorithme possède des avantages que ne réunissent aucun des autres algorithmes comparables. Les systèmes triangulaires qu'il produit ont toujours des solutions. En outre, leur lisibilité et leur évaluation numérique sont commodes. De plus chaque décomposition produite par l'algorithme est sans redondances. Cela signifie qu'aucun des systèmes triangulaires n'a toutes ses solutions contenues parmi celles d'un autre système de la décomposition. Notre algorithme diffère de celui de Daniel Lazard en cela qu'il repose essentiellement sur la notion de pgcd étudié dans le second chapitre. Plus précisément, lorsque notre algorithme rencontre deux polynômes dont on cherche les racines communes (et que l'on peut interpréter comme deux polynômes unitaires en une même variable et à coefficients dans un produit de corps) il les remplace par leur pgcd. Si cet algorithme semble naturel, sa description précise est compliquée.

Cela tient pour partie à la nature du problème. Ainsi, les solutions d'un système d'équations à trois inconnues correspondent à une famille de régions de l'espace, parmi lesquelles il peut y avoir des points, des courbes, des surfaces. À chacun de ces types de région va correspondre un type de systèmes triangulaires où zéro, une ou deux indéterminées ne sont pas fixées. Par suite, au cours du déroulement de l'algorithme, une même indéterminée pourra être soit fixée, soit libre, soit variable d'un polynôme (donc en instance de devenir fixée ou libre).

La difficulté tient aussi de la méthode. Pour calculer nos pgcd, nous devons voir nos polynômes comme étant à coefficients dans un produit de corps, ce produit de corps comportant lui-même des quantités algébriques. En travaillant dans chacun des corps, on est amené à calculer des inverses de quantités algébriques. Ceci se fait au moyen d'une astuce bien connue fondée sur l'identité de Bezout, autrement dit sur des calculs de pgcd ! Bref, la méthode est récursive. En fait, elle comprend une quinzaine d'algorithmes qui s'appellent mutuellement, directement ou indirectement.

Les algorithmes et résultats de [Laz91a] étaient donnés sans démonstration. Nous prouvons ceux que nous utilisons ainsi que nos propres algorithmes et nous illustrons le tout à l'aide de nombreux exemples. Si l'ensemble de la méthode est compliqué, celle-ci a permis la résolution de problèmes inaccessibles par d'autres méthodes et a conduit à un brevet industriel ([FMR96]).

Nous avons déjà à dit qu'une décomposition calculée par cette méthode possédait de bonnes propriétés. On peut se demander si elle est pour autant compétitive (en terme de temps de calcul) vis à vis des autres méthodes calculant des décompositions en systèmes triangulaires. On peut aussi se demander si, en pratique, les autres méthodes ne donnent pas des décompositions souvent exploitables. Une comparaison expérimentale de quatre méthodes calculant des décompositions en systèmes triangulaires fait l'objet du sixième chapitre.

Les méthodes les plus connues (au moment où nous avons commencé cette thèse) sont dûes à Daniel Lazard ([Laz91a]), Michael Kalkbrener ([Kal91, Kal93]), Dongming Wang ([Wan93b]) et Wu Wen-Tsün ([Wu87]). Ce sont celles que nous avons retenues. De l'opinion même de ces auteurs, il semblait nécessaire de confronter ces méthodes, qui, lorsqu'elles ont été implantées, ne l'ont pas été dans des conditions comparables (logicielles, matérielles, humaines). Une terminologie et des concepts tentant d'unifier tous ces points de vue ont été proposés par Daniel Lazard ([Laz92b]). Nous les rappelons dans le troisième chapitre. A partir de là, nous avons implanté en AXIOM les outils communs à ces méthodes. Le but était de faire en sorte que les implantations de nos méthodes ne diffèrent effectivement que par leurs algorithmes, toutes les routines et structures de données devant être communes. De plus, les calculs effectués par chaque méthode peuvent être vérifiés à l'aide des techniques fondées sur les bases de Gröbner ([CLO91]) et du logiciel GB de J.-C. Faugère ([Fau94]). Ainsi, a-t-on pu mettre au point les implantations de ces différentes méthodes. Nous détaillons largement la façon dont ces méthodes ont été programmées. Celles de Michael Kalkbrener et Dongming Wang ont été implantées par Philippe Aubry.

Ces méthodes ont été testées sur de nombreux exemples, pour la plus part tirés du jeu d'essais du projet européen PoSSo ([Eur92]). Nous rapportons ici non seulement les temps de calcul, mais aussi les décompositions calculées par les quatre méthodes sur 25 exemples. Ceci explique la longueur de ce dernier chapitre, à la fin duquel vous trouverez quelques enseignements sur cette comparaison. Ce travail de comparaison est rapporté dans [AM97a, AM97b].

Table des matières

Chapitre I

Systèmes d'équations algébriques : notions classiques

Introduction

Le sujet principal de cette thèse est l'étude d'une famille d'algorithmes de résolution des systèmes d'équations algébriques fondés sur la notion de pgcd. Les trois premiers chapitres décrivent les outils nécessaires à la présentation de ces algorithmes. La première partie de ce premier chapitre passe en revue les notions classiques de *base de Gröbner*, de *résultant* et de *variété affine*. Le lecteur familier avec ces concepts peut passer directement à la seconde partie consacrée essentiellement à la *fermeture de Zariski* et au *saturé d'un idéal par un élément*.

Nous nous adresserons donc maintenant au lecteur non familier avec la résolution des systèmes d'équations algébriques. Nous commencerons par un exemple que nous traiterons volontairement de manière naïve, maladroite, et imprécise. Nous supposerons ne pas connaître autre chose de l'algèbre que ce que l'on a appris au lycée. Considérons le système d'équations :

$$\left\{\begin{array}{rcl} x^2+y+z & = & 1 \\ x+y^2+z & = & 1 \\ x+y+z^2 & = & 1 \end{array}\right.$$

Nous cherchons les triplets (x, y, z) satisfaisant simultanément les trois équations précédentes.

Une première idée est de procéder comme pour les systèmes d'équations linéaires. Autrement dit on va chercher à réécrire le système de façon à produire 2 équations à 2 inconnues, puis une seule équation à une seule inconnue. Et ceci en utilisant deux règles pour produire de nouvelles équations :

1) multiplication d'une équation par un nombre,

2) addition (ou soustraction) membre à membre de deux équations.

En effet, si le triplet (x, y, z) satisfait les équations $E(x, y, z) = 0$ et $F(x, y, z) = 0$ il satisfait aussi l'équation

$$\alpha E(x, y, z) + \beta F(x, y, z) = 0$$

o α et β sont des nombres (éventuellement nuls). Nous savons que la réciproque est généralement fausse. En particulier si α et β sont tous deux nuls !

Si l'on multiplie la seconde équation de notre système par x, en la retranchant ensuite à la première, on obtient :

$$(-y^2 - z + 1)x + y + z - 1 = 0 \tag{I.1}$$

Puis, en multipliant la seconde équation (du système de départ) par $(-y^2 - z + 1)$ et en la retranchant à l'équation précédente, il vient une première équation sans x :

$$y^4 + (2z - 2)y^2 + y + z^2 - z = 0 \tag{I.2}$$

D'autre part en soustrayant membre à membre les deux dernières équations du système de départ, on a aussi :

$$y^2 - y - z^2 + z = 0 \tag{I.3}$$

Nous cherchons maintenant à éliminer y dans le nouveau système :

$$\left\{ \begin{array}{rcl} y^4 + (2z-2)y^2 + y + z^2 - z & = & 0 \\ y^2 - y - z^2 + z & = & 0 \end{array} \right.$$

En multipliant la dernière équation par y^2 et en la retranchant à la première, on obtient :

$$y^3 + (z^2 + z - 2)y^2 + y + z^2 - z = 0 \tag{I.4}$$

En multipliant la dernière équation du nouveau système cette fois-ci par y et la retranchant à celle ci-dessus il vient :

$$(z^2 + z - 1)y^2 + (z^2 - z + 1)y + z^2 - z = 0 \tag{I.5}$$

En continuant de même, on trouve :

$$2z^2y + z^4 - z^2 = 0 \tag{I.6}$$

Puis :

$$(-z^6 - z^5 + 4z^4 - z^3 + z^2)y + 2z^4 - 2z^3 = 0 \tag{I.7}$$

Et enfin :

$$z^{10} + z^9 - 5z^8 + 7z^6 - 5z^5 + z^4 = 0 \tag{I.8}$$

On peut remarquer que $z = 0$ et $z = 1$ sont des solutions évidentes de (I.8) et cette équation se réécrit :

$$(z-1)^2z^4(z^4 + 3z^3 - 3z + 1) = 0$$

Arrivé à ce stade, nous sommes pratiquement bloqués, sauf à remarquer encore que

$$(z^4 + 3z^3 - 3z + 1) = (z^2 + z - 1)(z^2 + 2z - 1)$$

Contrairement à ce qui se passe avec les systèmes d'équations linéaires, la *remontée* est plus ardue et parfois vaine. En effet le cas $z^2 + z - 1 = 0$ qui correspond à

$$z = -\frac{1-\sqrt{5}}{2} \text{ ou } z = -\frac{1+\sqrt{5}}{2}$$

donnent un système en (x, y) sans solutions. Pour cela, en utilisant l'équation $z^2 + z - 1 = 0$ avec celles du système de départ, on trouve successivement les équations $x + y - z = 0$, $2y - z = 0$ et $z^2 + 6z - 4 = 0$ et par suite :

$$z = -\frac{1 - \sqrt{13}}{2} \text{ ou } z = -\frac{1 + \sqrt{13}}{2}$$

Notons que cette contradiction peut aussi être découverte sans résoudre les équations précédentes en z. Il suffit d'appliquer l'algorithme d'Euclide pour calculer le pgcd des polynômes z^2+z-1 et $z^2 + 6z - 4$, ce qui donne 1. Ainsi, ces deux polynômes ne peuvent avoir de racines communes.

Heureusement, il existe un outil mathématique permettant de ne pas tomber sur ce cas. Cet outil, appelé *résultant*, nous apprend que le système en (y, z) écrit plus haut admet des solutions si et seulement si z satisfait :

$$z^8 - 4z^6 + 4z^5 - z^4 = 0$$

soit encore :

$$(z - 1)^2 z^4 (z^2 + 2z - 1) = 0$$

On peut se demander si l'on aurait pu mener les calculs en utilisant nos deux règles de départ de sorte à :

1) remplacer à chaque étape, le système de départ par un système ayant les mêmes solutions,
2) tout en éliminant successivement les variables, sans introduire de cas superflus.

Ceci est effectivement possible grâce à un outil, les *bases de Gröbner*. Cependant, il faut alors faire une concession : plusieurs polynômes *ayant la même variable principale* pourront apparaître dans le système final. Cet outil appliqué au système de départ nous produit le système d'arrivée :

$$\left\{\begin{array}{rcl} x + y + z^2 - 1 & = & 0 \\ y^2 - y - z^2 + z & = & 0 \\ 2z^2 y + z^4 - z^2 & = & 0 \\ z^6 - 4z^4 + 4z^3 - z^2 & = & 0 \end{array}\right.$$

Nous pouvons maintenant nous poser la question suivante : tenant compte du fait que z doit satisfaire l'équation :

$$z^6 - 4z^4 + 4z^3 - z^2 = 0$$

pouvons-nous remplacer les deux polynômes en y (et z) par un seul qui serait une *sorte de pgcd* de ces deux polynômes. Ceci est effectivement possible et c'est le sujet de cette thèse (chapitre 2 et suivants) Mais il faut là aussi faire une concession : le système précédent sera éventuellement remplacé par plusieurs systèmes dont la réunion des solutions est l'ensemble des solutions du système de départ. Sur notre exemple, on trouvera les deux systèmes suivants avec l'une des méthodes que nous étudierons plus loin :

$$\left\{\begin{array}{rcl} z^2 & = & 0 \\ y^2 - y + z & = & 0 \\ x + y - 1 & = & 0 \end{array}\right. \quad \text{ou} \quad \left\{\begin{array}{rcl} z^4 - 4z^2 + 4z - 1 & = & 0 \\ 2y + z^2 - 1 & = & 0 \\ 2x + z^2 - 1 & = & 0 \end{array}\right.$$

Si l'on remarque d'une part que :

$$z^4 - 4z^2 + 4z - 1 = (z-1)^2(z^2+2z-1)$$

et d'autre part que :

$$z^2 = 0 \ \Leftrightarrow\ z = 0 \quad \text{et} \quad (z-1)^2 = 0 \ \Leftrightarrow\ z - 1 = 0$$

on obtient la *réécriture suivante* du système de départ :

$$\left\{\begin{array}{lcl} z & = & 1 \\ y & = & 0 \\ x & = & 0 \end{array}\right. \text{ ou } \left\{\begin{array}{lcl} z & = & 0 \\ y & = & 1 \\ x & = & 0 \end{array}\right. \text{ ou } \left\{\begin{array}{lcl} z & = & 0 \\ y & = & 0 \\ x & = & 1 \end{array}\right. \text{ ou } \left\{\begin{array}{rcl} z^2+2z-1 & = & 0 \\ y & = & z \\ x & = & z \end{array}\right.$$

Enfin, on peut remplacer le dernier système ci-dessus par ceux obtenus en y injectant :

$$z = -1 - \sqrt{2} \text{ ou } z = -1 + \sqrt{2}$$

Dans ce chapitre, ainsi que dans les deux suivants, nous allons donc décrire les outils mentionnés précédemment. D'autre part, notre but étant d'automatiser la résolution des systèmes d'équations algébriques par une certaine classe de méthodes, nous devrons préciser ce que nous entendons par résoudre. Ainsi, dans l'exemple précédent, nous aurions pu nous arrêter à différentes étapes ou prendre d'autres directions. Nous décrirons à partir du quatrième chapitre différents *algorithmes* de résolution. Ceci conduit à la question : qu'est-ce qu'un système résolu ? Une famille de tels systèmes, les *systèmes triangulaires* sera étudiée dans le troisième chapitre. Une question plus fondamentale encore est : qu'est-ce qu'un système d'équations algébriques ? Il existe au moins deux concepts formalisant cette notion : celui d'*idéal d'un anneau de polynômes* et celui de *variété affine*. Le fait de choisir l'un plutôt que l'autre dépend du sens que l'on donne au mot *résoudre*, ces deux concepts étant de toute façon intimement liés.

Nous supposerons dans la suite de cette thèse que le lecteur est familier avec les notions d'*anneau* (*intègre*, *réduit*, ...) d'*idéal* (*principal*, *radical*, *premier*, ...) et de corps. Il sera également fait référence aux notions d'*anneau nœthérien* et *décomposition primaire* sans pour autant qu'une parfaite connaissance de ces concepts soit nécessaire. Pour toutes ces notions élémentaires d'algèbre, le lecteur pourra se reporter (entre autres) aux ouvrages suivants ([CLO91, Laf74, Sei56, van91]), le premier d'entre eux étant à la fois le plus élémentaire, le plus attrayant pour des néophytes et le plus proche des objectifs de cette thèse.

Donnons maintenant le plan de ce chapitre. Dans la première partie, nous précisons le principe de résolution utilisée sur l'exemple précédent, à savoir *l'élimination des variables et la remontée des solutions*. Pour cela nous rappelons deux théorèmes fondamentaux fondés sur les notions de *variété affine*, de *base de Gröbner* et de *résultant*.

Le but de la seconde partie est de réviser les liens entre les idéaux d'anneaux de polynômes et les variétés affines. Pour cela, nous rappelons les différents énoncés du théorème dit *des zéros de Hilbert*. Puis nous passerons en revue les différentes opérations dont nous aurons besoin dans la suite sur les idéaux et les variétés affines, en particulier le *saturé d'un idéal par rapport à un élément* et la *fermeture de Zariski*. Ces deux notions (souvent inutilisées dans les travaux voisins de cette thèse [Wu87, Wan93b, Kal93]) joueront un rôle capital dans la suite de cet exposé.

À l'exception de la section sur le saturé d'un idéal qui s'appuie sur [van91], les notes rassemblées dans ce chapitre sont essentiellement tirées de [CLO91].

1 Élimination des variables, extension des solutions

Dans cette partie, on rappelle les définitions et premières propriétés de notions fondamentales intervenant dans la résolution des systèmes d'équations algébriques et la description de leurs solutions. Il s'agit des notions de variété affine, de base de Gröbner, et de résultant. Indiquons en quelques mots ce que ces notions formalisent. Soient $f_1, \ldots, f_s$ des polynômes à coefficients dans un corps $\mathbf{k}$ et soit $\mathbf{K}$ un corps contenant $\mathbf{k}$. La variété affine des polynômes $f_1, \ldots, f_r$ relative à $\mathbf{K}$ est l'ensemble des solutions dans $\mathbf{K}$ du système $f_1 = \cdots = f_s = 0$ (voir définition I.2)

Moyennant le choix d'un ordre $<$ *admissible* (définition I.3) sur les monômes, une base de Gröbner des polynômes $f_1, \ldots, f_s$ est un système de générateurs privilégié (avec des propriétés remarquables, théorème I.1) parmi tous les systèmes de générateurs engendrant le même idéal que les polynômes $f_1, \ldots, f_s$. Par suite, pour $\mathbf{K}$ algébriquement clos (par exemple $\mathbf{K}$ égal au corps des nombres complexes) une base de Gröbner des polynômes $f_1, \ldots, f_s$ fournit un système canonique parmi tous les systèmes ayant exactement les mêmes solutions dans $\mathbf{K}$ que $f_1 = \cdots = f_s = 0$.

Pour deux polynômes f_1, f_2 en deux variables x et y le résultant de ces polynômes par rapport y fournit une condition nécessaire et suffisante pour que f_1 et f_2 vus comme polynômes univariés en x aient un facteur commun non trivial (définition I.9 p. 25 et proposition I.4 p. 26). Cette notion s'étend au cas de plusieurs polynômes $f_1, \ldots, f_s$ (définition I.10 p. 27 et proposition I.6 p. 28).

On rappelle également deux théorèmes illustrant le principe commun à de nombreuses méthodes de résolution : éliminer des indéterminées, puis étendre les solutions obtenues pour les indéterminées restantes aux indéterminées éliminées. Le premier (théorème I.3 p. 24) exprime le fait que pour un certain ordre sur les monômes (*l'ordre lexicographique* définition I.1 p. 21) le calcul d'une base de Gröbner des polynômes $f_1, \ldots, f_s$ est un bon procédé pour éliminer les indéterminées. Le second, fondé sur la notion de résultant, (théorème I.4 p. 29) donne une condition suffisante de *remontée des solutions.*

1.1 Variétés affines

On précise ici des notations employées dans toute la suite, et on rappelle les propriétés des variétés affines vis-à-vis des opérations ensemblistes élémentaires (proposition I.1).

Notations I.1 Soit $\mathbf{k}$ un corps, n un entier positif et V un ensemble de n variables que l'on suppose ordonnées comme suit

$$x_1 < x_2 < \cdots < x_n$$

Soit i un entier naturel tel que $1 \leq i \leq n$. On pose $\mathbf{P}_i = \mathbf{k}[x_1, \ldots, x_i]$. Soit $\mathbf{K}$ une extension du corps $\mathbf{k}$. On note $\mathbf{A}^i(\mathbf{K})$ l'espace affine de dimension i défini par $\mathbf{K}^i$. Pour des polynômes $f_1, \ldots, f_r$ dans $\mathbf{P}_n$ on note $\langle f_1, \ldots, f_r \rangle_{\mathbf{P}_n}$ (ou simplement $\langle f_1, \ldots, f_r \rangle$ si aucune confusion n'est à craindre) l'idéal de $\mathbf{P}_n$ engendré par $f_1, \ldots, f_r$ et on désigne par $\mathbf{V}_{\mathbf{K}}(f_1, \ldots, f_r)$ (que l'on abrège en $\mathbf{V}(f_1, \ldots, f_r)$ si possible) l'ensemble des points de $\mathbf{A}^n(\mathbf{K})$ annulant simultanément les polynômes $f_1, \ldots, f_r$.

Remarque I.1 L'ensemble $\mathbf{V}(f_1, \ldots, f_r)$ est aussi l'ensemble des points de $\mathbf{A}^n(\mathbf{K})$ annulant simultanément les polynômes de $\langle f_1, \ldots, f_r \rangle$. Par suite, étant donné un idéal I de $\mathbf{P}_n$ et $\{f_1, \ldots, f_r\}$ un ensemble de générateurs de I, on peut poser

$$\mathbf{V}_{\mathbf{K}}(\mathrm{I}) = \mathbf{V}_{\mathbf{K}}(f_1, \ldots, f_r)$$

On omettra là encore l'indice $\mathbf{K}$ si l'on ne considère qu'une extension de $\mathbf{k}$.

Définition I.1 *Pour une partie V de $\mathbf{A}^n(\mathbf{K})$ on appelle* idéal sur $\mathbf{k}$ associé à V *l'idéal de $\mathbf{P}_n$ noté $\mathbf{Id}_{\mathbf{k}}(V)$ formé par les polynômes de $\mathbf{P}_n$ s'annulant en tout point de V.*

Notations I.2 Lorsque $\mathbf{k}$ sera supposé algébriquement clos, cela sera dit dans les énoncés des propositions et théorèmes et l'on identifiera $\mathbf{k}$ et $\mathbf{K}$. Donc, sauf mention explicite du contraire, on ne suppose pas $\mathbf{k}$ algébriquement clos et $\mathbf{K}$ est n'importe quel sur-corps de $\mathbf{k}$ ou $\mathbf{k}$ lui-même. Enfin, pour $n = 3$, on écrira souvent z, y et x pour x_1, x_2 et x_3 respectivement. On aura donc : $z < y < x$.

Définition I.2 *Une partie V de $\mathbf{A}^n(\mathbf{K})$ est dite* variété affine sur $\mathbf{k}$ *s'il existe un idéal I de $\mathbf{P}_n$ tel que $V = \mathbf{V}_{\mathbf{K}}(I)$. Tout point de V est alors appelé* zéro *(dans $\mathbf{K}$) de I.*

Proposition I.1 *Soient I, J deux idéaux de $\mathbf{P}_n$ et V, W deux variétés affines sur $\mathbf{k}$ de $\mathbf{A}^n(\mathbf{K})$. On a alors les propriétés suivantes :*

(i) $I \subseteq J \Rightarrow \mathbf{V}(J) \subseteq \mathbf{V}(I)$

(ii) $V \subseteq W \iff \mathbf{Id}_{\mathbf{k}}(W) \subseteq \mathbf{Id}_{\mathbf{k}}(V)$

(iii) $\mathbf{V}(\mathbf{Id}_{\mathbf{k}}(V)) = V$

(iv) $I \subseteq \mathbf{Id}_{\mathbf{k}}(\mathbf{V}(I))$

(v) $\mathbf{V}(I) \cap \mathbf{V}(J) = \mathbf{V}(I + J)$

(vi) $\mathbf{V}(I) \cup \mathbf{V}(J) = \mathbf{V}(I \cap J) = \mathbf{V}(I \,.\, J)$

Preuve. $\triangleright$ Soit $\zeta \in \mathbf{A}^n(\mathbf{K})$. Supposons que l'on ait : $I \subseteq J$. Si ζ appartient à $\mathbf{V}(J)$ il annule en particulier les éléments de I, donc appartient à $\mathbf{V}(I)$. D'où (i). Vérifions (iii). Par définition même de $\mathbf{Id}_{\mathbf{k}}(V)$ on a :

$$V \subseteq \mathbf{V}(\mathbf{Id}_{\mathbf{k}}(V))$$

Inversement, soient $f_1, \ldots, f_r$ des polynômes de $\mathbf{P}_n$ tel que $V = \mathbf{V}(f_1, \ldots, f_r)$. On a clairement $\langle f_1, \ldots, f_r \rangle \subseteq \mathbf{Id}_{\mathbf{k}}(V)$. Par conséquent, avec (i), il vient :

$$\mathbf{V}(\mathbf{Id}_{\mathbf{k}}(V)) \subseteq V$$

Ceci prouve (iii). Vérifions (ii). L'implication directe est claire. Sa réciproque résulte de (i) et (iii). Vérifions (iv). Soient $f_1, \ldots, f_r$ et f dans $\mathbf{P}_n$ tel que $I = \langle f_1, \ldots, f_r \rangle$ et $f \in I$. On a donc : $\mathbf{V}(f_1, \ldots, f_r) \subseteq \mathbf{V}(f)$ D'où : $f \in \mathbf{Id}_{\mathbf{k}}(\mathbf{V}(f_1, \ldots, f_r))$. Ceci entrane (iv). La proposition (v) est triviale. Vérifions (vi). Comme on a les inclusions

$$I \,.\, J \subseteq I \cap J \subseteq I \,,\, J$$

il vient avec (i)

$$\mathbf{V}(I) \cup \mathbf{V}(J) \subseteq \mathbf{V}(I \bigcap J) \subseteq \mathbf{V}(I \,.\, J)$$

Inversement, supposons que ζ ne soit zéro ni de I ni de J. Comme $\mathbf{P}_n$ est intègre, ζ n'annule pas l'un des éléments de $I \,.\, J$ et n'est pas zéro de cet idéal, d'où (vi). $\triangleleft$

Remarque I.2 Pour la propriété (iv) précédente, on peut avoir l'égalité, comme ne pas l'avoir. Considérons tout d'abord avec $n = 3$, l'idéal $I = \langle g_1, g_2 \rangle$, où $g_1 = x - z^3$ et $g_2 = y - z^2$. Posons alors $V = \mathbf{V}(I)$ et $J = \mathbf{Id}_{\mathbf{k}}(V)$. On vérifie sans peine que :

$$V = \{(t, t^2, t^3) \mid t \in \mathbf{K}\}$$

On sait que l'on a : $I \subseteq J$. Soit alors $f \in \mathbf{k}[z, y, x]$. On vérifie qu'il existe des polynômes h_1, h_2 et r tel que $f = h_1 g_1 + h_2 g_2 + r$ où $r \in \mathbf{k}[z]$. Supposons que $f \in J$. Alors r doit être identiquement nul et l'on en déduit $J = I$. Toujours avec $n = 3$, considérons l'idéal $I = \langle g_1, g_2, g_3 \rangle$, où $g_1 = x^2$, $g_2 = y^2$ et $g_3 = z^2$. On pose encore $V = \mathbf{V}(I)$ et $J = \mathbf{Id}_{\mathbf{k}}(V)$. On vérifie sans peine que :

$$V = \{(0, 0, 0)\} \text{ et } J = \langle x, y, z \rangle$$

Bref, si l'on peut reconstruire une variété affine à partir de son idéal associé, on ne peut reconstruire un idéal de $\mathbf{P}_n$ à partir de sa variété affine associée. Il y a en effet une *perte d'information* qu'exprime la propriété (iv). Le théorème des zéros fort, que nous aborderons plus loin, précisera cette dernière remarque.

1.2 Bases de Gröbner

On rappelle tout d'abord dans cette section la notion d'ordre admissible (définition I.3). Un ordre admissible étant fixé, on révise le concept de S-polynôme (définition I.4 p. 22), puis la réduction d'un polynôme par rapport à une famille de polynômes (définition I.5 et définition I.6 p. 22). On passe alors à la définition d'une base de Gröbner d'un idéal de $\mathbf{P}_n$ sous forme de trois caractérisations équivalentes (théorème I.1 p. 23). On termine par quelques propriétés d'usage courant : théorème I.2 p. 23 et remarque I.6 p. 24.

Les bases de Gröbner sont un outil important, mais auxiliaire, dans la suite de notre exposé. Cette section n'est pas par conséquent aussi développée que les autres sections de ce chapitre. Ainsi, les énoncés sont donnés sans leurs preuves et celles-ci nécessitent parfois des résultats intermédiaires non rappelés ici (voir [CLO91]). De plus, on n'aborde pas la question du calcul des bases de Gröbner. Nous utiliserons les bases de Gröbner essentiellement dans la deuxième partie de ce chapitre (afin d'affirmer que certains objets peuvent être calculés), dans le quatrième chapitre (comme entrée de l'algorithme lexTriangular [Laz92a]) et dans le sixième (comme outil d'analyse des algorithmes qui y sont étudiés).

Définition I.3 *Une relation d'ordre totale $<$ sur les monômes de $\mathbf{P}_n$ est dite* ordre admissible *si pour tous les monômes m, n, p de $\mathbf{P}_n$ on a :*

(i) $m < n \;\Rightarrow\; mp < np$

(ii) $1 < m$

Exemple I.1 Donnons 3 exemples d'ordres admissibles sur les monômes de $\mathbf{P}_n$ fréquemment utilisés. Identifions les monômes de $\mathbf{P}_n$ à des éléments de $\mathbb{N}^n$, où $\mathbb{N}$ désigne le demi-anneau des entiers naturels. Soient alors $a = (a_1, \ldots, a_n)$ et $b = (b_1, \ldots, b_n)$ deux monômes de $\mathbf{P}_n$. On appelle *ordre lexicographique* l'ordre noté $<_{lex}$ et défini par :

$$a <_{lex} b \iff (\exists i \in \{1, \ldots, n\}) \text{ t.q. } \begin{cases} a_i < b_i \\ a_j = b_j \quad (\forall j < i) \end{cases}$$

On appelle *ordre degré-lexicographique* l'ordre noté $<_{deglex}$ et défini par :

$$a <_{deglex} b \quad \Longleftrightarrow \quad \left\{ \begin{array}{l} |\,a\,| < |\,b\,| \text{ ou} \\ |\,a\,| = |\,b\,| \text{ et } a <_{lex} b \end{array} \right.$$

où $|\,a\,|$ dénote la somme des composantes de a.

On appelle *ordre degré-lexicographique-inverse* l'ordre noté $<_{degrevlex}$ et défini par :

$$a <_{degrevlex} b \quad \Longleftrightarrow \quad \left\{ \begin{array}{l} |\,a\,| < |\,b\,| \text{ ou} \\ |\,a\,| = |\,b\,| \text{ et } a >_{lex} b \end{array} \right.$$

Remarque I.3 tant donné un ordre admissible $<$ sur les monômes de $\mathbf{P}_n$, ces derniers s'identifiant à des *vecteurs* d'entiers naturels à n composantes, il n'existe pas de suites infinies strictement décroissantes pour $<$.

Notations I.3 Soit $<$ un ordre admissible sur les monômes de $\mathbf{P}_n$. Tout polynôme $p \in \mathbf{P}_n$ peut se mettre sous la forme unique

$$p = \mathsf{lt}(p) + \mathsf{reductum}(p) \quad \text{avec} \quad \mathsf{lt}(p) = \mathsf{lc}(p)\,\mathsf{lm}(p)$$

où $\mathsf{lm}(p)$ est le plus grand monôme de p pour $<$ et $\mathsf{lc}(p)$ son coefficient (dans $\mathbf{k}$).

Définition I.4 *Soit $<$ un ordre admissible sur les monômes de $\mathbf{P}_n$. Soit $p, q \in \mathbf{P}_n$. On appelle* S-polynôme de *p et q le polynôme noté $\mathsf{spol}(p,q)$ et défini par*

$$\mathsf{spol}(p,q) \;=\; \frac{m\ p}{\mathsf{lt}(p)} - \frac{m\ q}{\mathsf{lt}(q)}$$

où m est le ppcm des monômes $\mathsf{lm}(p)$ et $\mathsf{lm}(q)$.

Définition I.5 *Soit $<$ un ordre admissible sur les monômes de $\mathbf{P}_n$. Soit $p \in \mathbf{P}_n$ et m un monôme de $\mathbf{P}_n$. On dit que m est* réductible par *p si $\mathsf{lm}(p)$ divise m, ce que l'on note : $\mathsf{lm}(p) \mid m$. Dans le cas contraire, on dit que m est* réduit par rapport à *p.*

Soit $q \in \mathbf{P}_n$ un second polynôme. On dit que q est réductible en tête par *p si $\mathsf{lm}(p)$ divise $\mathsf{lm}(q)$. Dans le cas contraire, on dit que q est* réduit en tête par rapport à *p. On définit alors l'élément $\mathsf{headRem}(q,p) \in \mathbf{P}_n$ comme suit :*

$$\mathsf{headRem}(q,p) = \left\{ \begin{array}{ll} \mathsf{headRem}(\mathsf{lc}(q)\,\mathsf{spol}(\mathsf{lm}(q),p) + \mathsf{reductum}(q), p) & \text{si } \mathsf{lm}(p) \mid \mathsf{lm}(q) \\ q & \text{sinon} \end{array} \right.$$

On dit que q est réductible par *p si $\mathsf{lm}(p)$ divise l'un des monômes de q. Dans le cas contraire, on dit que q est* réduit par rapport à *p. On définit alors l'élément $\mathsf{rem}(q,p) \in \mathbf{P}_n$ comme suit :*

$$\mathsf{rem}(q,p) = \left\{ \begin{array}{ll} \mathsf{rem}(\mathsf{headRem}(q,p),p) & \text{si } \mathsf{lm}(p) \mid \mathsf{lm}(q) \\ \mathsf{lt}(q) + \mathsf{rem}(\mathsf{reductum}(q),p) & \text{sinon} \end{array} \right.$$

Remarque I.4 Avec les hypothèses et notations précédentes, remarquons tout d'abord que si q est *réductible en tête par* un polynôme p on a :

$$\mathsf{lm}(\mathsf{spol}(q,p)) \ < \ \mathsf{lm}(q)$$

Par suite, à l'aide de la remarque I.3, on en déduit que $\mathsf{headRem}(q,p)$ et $\mathsf{rem}(q,p)$ sont bien définis pour tout p et tout q de $\mathbf{P}_n$. De plus, si q est réductible par p on a :

$$\mathsf{lm}(\mathsf{rem}(q,p)) \ < \ \mathsf{lm}(q)$$

Notons également que les polynômes $\mathsf{spol}(q,p)$, $\mathsf{headRem}(q,p)$ et $\mathsf{rem}(q,p)$ appartiennent à l'idéal de $\mathbf{P}_n$ engendré par p et q.

Remarque I.5 Avec les hypothèses et notations précédentes, soit de plus $G = \{g_1, \dots, g_s\}$ un ensemble fini de polynômes de $\mathbf{P}_n$. On dit que q est *réductible par* G s'il est réductible par l'un des éléments de G. Dans ce cas, on pose $q_1 = q$, puis, tant qu'il existe $g \in G$ tel que q_i soit réductible par rapport à g, on choisit un tel polynôme g et l'on pose $q_{i+1} = \mathsf{rem}(q_i, g)$. De la remarque précédente, on déduit que la suite $(q_1, q_2, \dots)$ est finie. Cependant les polynômes $q_2, \dots$ dépendent du choix des polynômes g utilisés. Considérer l'exemple suivant pour l'ordre lexicographique avec $x_1 < x_2$:

$$q = x_1 x_2 \ \text{ et } \ G = \{x_2 - x_1, x_2 + x_1\}$$

Définition I.6 *Avec les hypothèses et notations de la définition I.5 on appelle* réduction de *q par rapport à G toute suite finie $(q_1, \dots, q_k)$ de polynômes de $\langle q \rangle + \langle G \rangle$ telle que*

(*i*) $q_1 = q$

(*ii*) $(\forall i < k)\ (\exists g \in G) \ \mid \ q_{i+1} = \mathsf{rem}(q_i, g)$

(*iii*) *q_i est réduit par rapport à G si et seulement si $k = i$*

Le polynôme q_k est alors dit forme réduite *de q par rapport à G.*

Théorème I.1 *Soit $<$ un ordre admissible sur les monômes de $\mathbf{P}_n$. Soit I un idéal de $\mathbf{P}_n$ et G une famille de générateurs de I. Alors les propositions suivantes sont équivalentes :*

(*i*) *pour tout polynôme $p \in \mathbf{P}_n$ il existe un polynôme $r \in \mathbf{P}_n$ telle que toute forme réduite de p par rapport à G soit égale à r*

(*ii*) *toute forme réduite d'un polynôme p de I par rapport à G est nulle*

(*iii*) *pour toute paire $\{p, q\}$ de polynômes de G, toute forme réduite du polynôme $\mathsf{spol}(p,q)$ par rapport à G est nulle.*

Lorsque ces assertions sont vraies la famille de polynôme G est dite base de Gröbner de l'idéal I pour l'ordre $<$. *Si de plus, tout polynôme g de G est réduit par rapport à $G \setminus \{g\}$, alors G est dite* base de Gröbner minimale de l'idéal I pour l'ordre $<$. *Enfin, Si G est une base de Gröbner minimale de l'idéal I pour tout polynôme $p \in \mathbf{P}_n$ on appelle* forme normale de p par rapport à *G le polynôme noté $\mathsf{nf}(p, G)$ égal à toute forme réduite de p par G.*

Théorème I.2 *Soit $<$ un ordre admissible sur les monômes de $\mathbf{P}_n$. Soit I un idéal de $\mathbf{P}_n$. Alors il existe une unique famille de générateurs $g_1, \dots, g_s$ de I, avec $\mathsf{lc}(g_i) = 1$ pour chaque i, qui soit une base de Gröbner minimale de l'idéal I pour l'ordre $<$. De plus, il existe un algorithme pour la calculer.*

Remarque I.6 Nous renvoyons entre autre à [Buc65, CLO91, Fau94] pour des algorithmes de calcul des bases de Gröbner. Notons cependant que l'on peut déduire de la propriété [(*iii*)] ci-dessus un tel algorithme. Voir en particulier [DST93] pour un exposé de cette remarque. Nous retiendrons surtout les 3 corollaires suivants du théorème I.1 dont nous reprenons les notations. On suppose que $G = \{g_1, \ldots, g_s\}$ est une base de Gröbner minimale de I.

(i) Pour tout polynôme p il existe des polynômes $q_1, \ldots, q_s$ de $\mathbf{P}_n$ tel que :

$$p \;=\; q_1 g_1 + \cdots + q_n g_n + \mathsf{nf}(p, G)$$

(ii) Par suite, le théorème I.1 fournit une condition nécessaire et suffisante pour qu'un polynôme $p \in \mathbf{P}_n$ appartienne à l'idéal I. Cette condition est :

$$\mathsf{nf}(p, G) \;=\; 0$$

(iii) Soit de plus $\mathbf{K}$ une clôture algébrique de $\mathbf{k}$. Alors on a :

$$\mathbf{V}_{\mathbf{K}}(I) \neq \emptyset \quad \Longleftrightarrow \quad G \cap \mathbf{k} = \emptyset$$

1.3 Un théorème d'élimination

Définition I.7 *Soit I un idéal de $\mathbf{P}_n$. Soit k un entier tel que : $0 \leq k \leq n$. On appelle* $k-$ième idéal d'élimination *de I l'idéal noté I_k défini par : $I_k = I \bigcap \mathbf{P}_{n-k}$. Pour $0 < p < n-k$, on notera que I_{k+p} est le $p-$ième idéal d'élimination de I_k.*

Définition I.8 *Soit k un entier tel que : $1 \leq k \leq n$. Un ordre admissible $<$ sur les monômes de $\mathbf{P}_n$ est dit* $k-$éliminant *(ou* de $k-$élimination*) si pour tout monôme m de $\mathbf{P}_n$ de degré non nul en l'une des variables $x_{n-k+1}, \ldots, x_n$ et tout monôme m' de $\mathbf{P}_{n-k}$ on a : $m' < m$*

Exemple I.2 L'ordre lexicographique induit par $x_1 < x_2 < \cdots < x_n$ est $k-$éliminant pour tout entier k compris entre 1 et n. Plus généralement, étant donné k avec $1 \leq k \leq n$, supposons $\mathbf{P}_{n-k}$ et $\mathbf{k}[x_{n-k+1}, \ldots, x_n]$ munis respectivement de deux ordres admissibles $<_-$ et $<_+$. Pour tout monôme $m = x_1^{e_1} \cdots x_{n-k}^{e_{n-k}} x_{n-k+1}^{e_{n-k+1}} \cdots x_n^{e_n}$, on pose $m_k^- = x_1^{e_1} \cdots x_{n-k}^{e_{n-k}}$ et $m_k^+ = x_{n-k+1}^{e_{n-k+1}} \cdots x_n^{e_n}$. On définit alors un ordre $k-$éliminant sur les monômes de $\mathbf{P}_n$, soit$<$, en posant :

$$m < m' \text{ ssi } \left\{ \begin{array}{lcll} m_k^+ & <_+ & {m'}_k^+ & \text{ou,} \\ m_k^+ & = & {m'}_k^+ & \text{et } \; m_k^- \; <_- \; {m'}_k^- \end{array} \right.$$

Théorème I.3 *Soit k un entier tel que : $1 \leq k \leq n$. Soit I un idéal de $\mathbf{P}_n$ et G une base de Gröbner de I pour un ordre $k-$éliminant. Posons $G_k = G \bigcap \mathbf{P}_{n-k}$. Alors G_k est une base de Gröbner de I_k.*

Preuve. $\triangleright$ Posons $G = \{g_1, \ldots, g_r\}$ et supposons que les g_i soient indéxés de sorte que $G_k = \{g_1, \ldots, g_s\}$. On a : $\langle g_1, \ldots, g_s \rangle \subseteq I_k$. Soit maintenant $f \in I_k$. Comme $f \in I$ et comme G est une base de I, il existe des polynômes $h_1, \ldots, h_r$ tel que : $f = h_1 g_1 + \cdots + h_r g_r$. Comme les polynômes $g_{s+1}, \ldots, g_r$ ont un degré non nul en l'une des variables $x_{n-k+1}, \ldots, x_n$ et comme G est une base de Gröbner pour un ordre $k-$éliminant, $f - h_1 g_1 - \cdots - h_s g_s$ est réduit par rapport à chacun des polynômes $g_{s+1}, \ldots, g_r$. Par suite $h_{s+1} = \ldots = h_r = 0$ et $f \in \langle g_1, \ldots, g_s \rangle$. Bref, $\langle g_1, \ldots, g_s \rangle = I_k$. Vérifions maintenant que G_k est bien une base de Gröbner. Soient g et g' dans G_k. Comme le $S-$polynôme $S(g, g')$ appartient à I_k et est réduit à 0 par G, il réduit à 0 par G_k, ce qui termine la preuve. $\triangleleft$

1.4 Résultant de deux polynômes

Dans cette section, nous rappelons la notion de résultant (définition I.9 p. 25). Cette notion s'introduit naturellement à l'aide de la proposition I.2. Pour deux polynômes univariés à coefficients dans un corps, cette proposition donne une condition nécessaire et suffisante pour qu'ils aient un facteur commun non trivial. La proposition I.3 p. 26 précise ce résultat en donnant des propriétés du résultant. Enfin, la remarque I.8 et la proposition I.4 p. 26 étendent les énoncés précédents au cas de deux polynômes multivariés.

Nous ne donnons ici que les propriétés élémentaires du résultant nécessaires à la suite de cet exposé. Voir [DST93, Val95] pour des présentations plus détaillées.

Proposition I.2 *Soient f et g dans $\mathbf{k}[x]$. Les polynômes f et g possèdent un facteur commun d non trivial (i.e. de degré strictement positif) si et seulement si le problème suivant d'inconnue (u, v) dans $\mathbf{k}[x] \times \mathbf{k}[x]$ admet des solutions :*

$$uf + vg = 0 \quad \text{avec} \quad (u, v) \neq (0, 0) \text{ et } \mathsf{deg}(u) < \mathsf{deg}(g) \text{ et } \mathsf{deg}(v) < \mathsf{deg}(f)$$

Preuve. $\triangleright$ Supposons tout d'abord qu'il existe $d \in \mathbf{k}[x]$ avec $\mathsf{deg}(d) > 0$ divisant f et g. Alors, il existe $f_1, g_1 \in \mathbf{k}[x]$ tel que $f = f_1 d$ et $g = g_1 d$. et $(g_1, -f_1)$ est une solution du problème précédent. Supposons maintenant que f et g ne possèdent pas de facteur commun non trivial. Alors, il existe $a, b \in \mathbf{k}[x]$ tel que $af + bg = 1$. Supposons que le problème précédent admette une solution (u, v). On peut supposer $u \neq 0$. Il vient :

$$u = (af + bg)u = afu + bgu = (bu - va)g$$

Comme $u \neq 0$, on en tire : $\mathsf{deg}(u) \geq \mathsf{deg}(g)$. Une contradiction. $\triangleleft$

Remarque I.7 Nous allons réécrire le problème précédent sous la forme d'un système d'équations linéaires. Pour cela nous posons :

$$f = a_0 x^l + \cdots + a_l \text{ avec } a_0 \neq 0 \quad \text{et} \quad g = b_0 x^m + \cdots + b_m \text{ avec } b_0 \neq 0$$

On cherche u et v sous la forme :

$$u = c_0 x^{m-1} + \cdots + c_{m-1} \quad \text{et} \quad v = d_0 x^{l-1} + \cdots + d_{l-1}$$

Résoudre ledit problème équivaut alors à trouver $c_0, \ldots, c_{m-1}, d_0, \ldots, d_{l-1} \in \mathbf{k}$ non tous nuls et satisfaisant le système d'équations linéaires homogène suivant :

$$\left\{ \begin{array}{lllllllll} a_0 c_0 & & & + & b_0 d_0 & & & = & 0 \\ a_1 c_0 & + & a_0 c_1 & + & b_1 d_0 & + & b_0 d_1 & = & 0 \\ & \ddots & & & & \ddots & & & \vdots \\ & & & a_l c_{m-l} & & + & b_m d_{l-1} & = & 0 \end{array} \right.$$

Définition I.9 *Avec les notations et hypothèses de la remarque précédente, on appelle* résultant *des polynômes f et g le déterminant de la matrice du système d'équations linéaires ci-dessus.*

Cette matrice est appelée matrice de Sylvester *de f et g. Elle est notée* $\mathsf{syl}(f,g,x)$ *et le résultant de f et g est noté* $\mathsf{res}(f,g,x)$. *On a donc :*

$$\mathsf{res}(f,g,x) = \begin{vmatrix} a_0 & 0 & \cdots & 0 & b_0 & 0 & \cdots & 0 \\ a_1 & a_0 & \ddots & \vdots & b_1 & b_0 & \ddots & \vdots \\ a_2 & a_1 & \ddots & 0 & b_2 & b_1 & \ddots & 0 \\ \vdots & & \ddots & a_0 & \vdots & & \ddots & b_0 \\ & \vdots & & a_1 & & \vdots & & b_1 \\ a_l & & & & b_m & & & \\ 0 & a_l & & \vdots & 0 & b_m & & \vdots \\ \vdots & \ddots & \ddots & & \vdots & \ddots & \ddots & \\ 0 & \cdots & 0 & a_l & 0 & \cdots & 0 & b_m \end{vmatrix}$$

Proposition I.3 *Avec les notations et hypothèses de la définition I.9, on a :*

(i) $\mathsf{res}(f,g,x) \in \mathsf{Z}[a_0,\ldots,a_l,b_0,\ldots,b_m]$

(ii) $\mathsf{res}(f,g,x) = 0 \iff (\exists d \in \mathbf{k}[x]) \mid \mathsf{deg}(d) > 0$ et $f,g \in \langle d \rangle$

(iii) $(\exists u,v \in \mathsf{Z}[a_0,\ldots,a_l,b_0,\ldots,b_m][x]) \mid uf + vg = \mathsf{res}(f,g,x)$

Preuve. $\triangleright$ Le seul point à vérifier est la dernière assertion. Celle-ci est immédiate si le résultant de f et g est nul. On peut donc supposer $\mathsf{res}(f,g,x) \neq 0$. Par suite, il existe $a,b \in \mathbf{k}[x]$ tel que $af + bg = 1$. De plus, on peut utiliser les mêmes notations pour les coefficients de a et b que celles utilisées pour ceux de u et v respectivement. En effet, on peut supposer $\mathsf{deg}(a) < \mathsf{deg}(g)$ et $\mathsf{deg}(b) < \mathsf{deg}(f)$. Ceci résulte par exemple de l'algorithme d'Euclide étendu. On est alors conduit à résoudre le même système d'équations linéaires que dans la remarque précédente, à une modification près : dans le second membre de la dernière équation, 0 est remplacé par 1. Ce système a donc une unique solution. Les formules de Cramer donnant les valeurs des inconnues $c_0,\ldots,c_{m-1}$,$d_0,\ldots,d_{l-1}$ montrent que chacun des coefficients des polynômes a et b s'écrit comme le quotient d'un élément de $\mathsf{Z}[a_0,\ldots,a_l,b_0,\ldots,b_m]$ sur le résultant $\mathsf{res}(f,g,x)$. D'où la formule annoncée. $\triangleleft$

Remarque I.8 Soit $\mathbf{A}$ est un anneau intègre, et $f,g \in \mathbf{A}[x]$. Le résultant de f et g peut être défini de la même manière que dans la définition précédente. De plus, en appliquant la proposition ci-dessus au corps $\mathbf{k}$ des fractions de $\mathbf{A}$, on obtient la proposition suivante. En particulier avec $\mathbf{A} = \mathbf{k}[x_1,\ldots,x_n]$ on peut définir $\mathsf{res}(f,g,x_i)$, pour $1 \leq i \leq n$, comme le résultant de f et g vus comme éléments de $\mathbf{k}[x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n][x_i]$.

Proposition I.4 *Soient f et g dans $\mathbf{P}_n = \mathbf{k}[x_1,\ldots,x_n]$ de degré strictement positif en x_n. Alors on a :*

(i) $\mathsf{res}(f,g,x_n) \in \langle f,g \rangle \cap \mathbf{k}[x_1,\ldots,x_{n-1}]$

(ii) $\mathsf{res}(f,g,x_n) = 0 \iff (\exists d \in \mathbf{P}_n)\ f,g \in \langle d \rangle$ et $\mathsf{deg}(d,x_n) > 0$

Preuve. ▷ En effet, d'après la remarque précédente, on peut appliquer la proposition I.3 en y remplaçant $\mathbf{k}$ par $\mathbf{k}[x_1, \ldots, x_{n-1}]$. Comme on a ici

$$\mathsf{Z}[a_0, \ldots, a_l, b_0, \ldots, b_m] \subseteq \mathbf{k}[x_1, \ldots, x_{n-1}]$$

et comme la dernière assertion de la proposition I.3 signifie que le résultant de f et g appartient à l'idéal que ces polynômes engendrent, les points (i) et (ii) sont immédiats. ◁

1.5 Un théorème d'extension

Dans le cas d'un idéal I de $\mathbf{P}_n$ engendré par deux polynômes, la proposition I.5 suivante donne une condition suffisante pour qu'un zéro (dans la clôture algébrique de $\mathbf{k}$) du premier idéal d'élimination de I *s'étende* en un zéro de I. Puis, la définition I.10 p. 27 donne une généralisation de la notion de résultant au cas de plusieurs polynômes $f_1, \ldots, f_s$. La proposition I.6 p. 28 étend la propriété (i) de la proposition I.4 p. 26 à ce résultant généralisé. Ceci permet de généraliser la proposition I.5 au cas d'un idéal I quelconque de $\mathbf{P}_n$ et fournit le théorème I.4 p. 29.

Proposition I.5 *On suppose que* $\mathbf{k}$ *est un corps algébriquement clos. Soient* f *et* g *dans* $\mathbf{P}_n$ *de degré strictement positif en* x_n. *Vus comme polynômes de* $\mathbf{k}[x_1, \ldots, x_{n-1}][x_n]$ *écrivons* f *et* g *comme suit :*

$$f = a_0 x_n^l + \cdots + a_l \text{ avec } a_0 \neq 0 \quad \text{ et } \quad g = b_0 x_n^m + \cdots + b_m \text{ avec } b_0 \neq 0$$

On pose $I_1 = \langle f, g\rangle \cap \mathbf{k}[x_1, \ldots, x_{n-1}]$. *Alors on a :*

$$(\forall (c_1, \ldots, c_{n-1}) \in \mathbf{V}_{\mathbf{k}}(I_1) \setminus \mathbf{V}_{\mathbf{k}}(a_0, b_0)) \quad (\exists c_n \in \mathbf{k}) \quad (c_1, \ldots, c_n) \in \mathbf{V}_{\mathbf{k}}(f, g)$$

Preuve. ▷ Soit donc $c = (c_1, \ldots, c_{n-1}) \in \mathbf{V}_{\mathbf{k}}(I_1) \setminus \mathbf{V}_{\mathbf{k}}(a_0, b_0)$ Nous allons vérifier que $f(c, x_n)$ et $g(c, x_n)$ ont une racine commune dans $\mathbf{k}$. Supposons tout d'abord que $a_0(c) \neq 0$ et $b_0(c) \neq 0$. Puisque $\mathsf{res}(f, g, x_n) \in I_1$ on a $\mathsf{res}(f, g, x_n)(c) = 0$. Or, puisque $a_0(c) \neq 0$ et $b_0(c) \neq 0$, on a : $\mathsf{res}(f(c, x_n), g(c, x_n), x_n) = \mathsf{res}(f, g, x_n)(c)$ (écrire les deux matrices de Sylvester pour s'en convaincre). Par suite, $f(c, x_n)$ et $g(c, x_n)$ ont un facteur commun non trivial dans $\mathbf{k}[x_n]$. Comme $\mathbf{k}$ est algébriquement clos, ils ont donc une racine commune dans $\mathbf{k}$. Supposons maintenant $a_0(c) \neq 0$ et $b_0(c) = 0$. Nous n'avons plus l'égalité précédente entre les deux résultants. Soit alors un entier m tel que $\mathsf{deg}(x_n^m f, x_n) > \mathsf{deg}(g, x_n)$. Ainsi le coefficient dominant de $h = x_n^m f + g$ est a_0. Nous sommes donc ramenés au cas traité précédemment, puisque nous avons $\langle f, g\rangle = \langle f, h\rangle$, ce qui termine la preuve. ◁

Définition I.10 *Soient* $f_1, \ldots, f_s$ *dans* $\mathbf{P}_n$ *et* $u_2, \ldots, u_s$ *des variables supplémentaires (telles que les variables* $x_1, \ldots, x_n, u_2, \ldots, u_s$ *soient algébriquement indépendantes sur* $\mathbf{k}$ *dans leur ensemble). On appelle* résultant en x_n de la suite $(f_1, \ldots, f_s)$ *le polynôme de l'anneau* $\mathbf{k}[x_1, \ldots, x_{n-1}, u_2, \ldots, u_s]$ *noté* $\mathsf{res}(f_1, f_2, \ldots, f_s; x_n)$ *et défini par*

$$\mathsf{res}(f_1, f_2, \ldots, f_s; x_n) = \mathsf{res}(f_1, u_2 f_2 + \cdots + u_s f_s, x_n)$$

Soit $(h_\alpha)_{\alpha \in \Delta}$ *avec* $\Delta \subset \mathbb{N}^{s-1}$ *la famille (finie) de polynômes de* $\mathbf{k}[x_1, \ldots, x_{n-1}]$ *telle que*

$$\mathsf{res}(f_1, \ldots, f_s; x_n) = \sum_{\alpha \in \Delta} h_\alpha \overline{u}^\alpha$$

où si $\alpha = (\alpha_2, \ldots, \alpha_s) \in \mathbb{N}^{s-1}$ *alors* $\overline{u}^\alpha$ *dénote le monôme* $u_2^{\alpha_2} \cdots u_s^{\alpha_s}$. *Les polynômes de la famille* $(h_\alpha)_{\alpha \in \Delta}$ *sont appelés ici* composantes *du résultant de la séquence* $(f_1, \ldots, f_s)$.

Exemple I.3 On considère les polynômes f_1, f_2, f_3 de $\mathbf{k}[z, y, x]$ définis par

$$\begin{aligned} f_1 &= x^2 + y + z - 1 \\ f_2 &= x + y^2 + z - 1 \\ f_3 &= x + y + z^2 - 1 \end{aligned}$$

Dans ce cas nous avons :

$$\begin{aligned} \mathsf{res}(f_1, f_2, f_3; x) = {} & (y^4 + 2y^2z - 2y^2 + z^2 + y - z)u_2^2 + \\ & 2(y^2z^2 + y^3 + z^3 - y^2 - z^2 + yz)u_2u_3 + \\ & (z^4 + 2yz^2 + y^2 - 2z^2 - y + z)u_3^2 \end{aligned}$$

Notons $h_{2,0}, h_{1,1}, h_{0,2}$ les composantes de $\mathsf{res}(f_1, f_2, f_3; x)$ avec les conventions de la définition précédente. Soit I_1 le premier idéal d'élimination de $I = \langle f_1, f_2, f_3 \rangle$. Une base de Gröbner de I pour l'ordre lexicographique induit par l'ordre sur les variables $z < y < x$ est $G = \{g_1, g_2, g_3, g_4\}$ où

$$\begin{aligned} g_1 &= x + y + z^2 - 1 \\ g_2 &= y^2 - y - z^2 + z \\ g_3 &= 2z^2y + z^4 - z^2 \\ g_4 &= z^6 - 4z^4 + 4z^3 - z^2 \end{aligned}$$

Une base de Gröbner de I_1 pour l'ordre lexicographique induit par $z < y$ est donc $G_1 = \{g_2, g_3, g_4\}$. On vérifie qu'une base de Gröbner de $\{h_{2,0}, h_{1,1}, h_{0,2}\}$ pour le même ordre que G est $H = \{\tilde{h}_1, \tilde{h}_2, \tilde{h}_3\}$ où

$$\begin{aligned} \tilde{h}_1 &= 2g_1 + (z - 2)g_3 \\ \tilde{h}_2 &= 2g_2 - (z - 2)g_3 \\ \tilde{h}_3 &= z(z - 1)g_3 \end{aligned}$$

On remarque donc que l'idéal engendré par $\{h_{2,0}, h_{1,1}, h_{0,2}\}$ est contenu dans I_1 sans lui être égal à cause du polynôme $\tilde{h}_3$. De même, le second idéal I_2 d'élimination de I est engendré par le seul polynôme $g_4 = (z - 1)^2\ z^2\ (z^2 + 2\ z - 1)$ Or, on a :

$$\begin{aligned} \mathsf{res}(h_{2,0}, h_{1,1}, h_{0,2}; y) = {} & (z - 1)^3\ z^5\ (z^2 + 2\ z - 1) \\ & (u_3^2\ z^3 + (-2\ u_3^2 - u_2\ u_3)\ z + 4\ u_3^2 + 4\ u_2\ u_3 + u_2^2) \\ & (u_3^2\ z^3 + u_3^2\ z^2 + (-u_3^2 + u_2\ u_3)\ z - u_3^2 + u_2\ u_3 + u_2^2) \end{aligned}$$

Ceci montre que chaque composante de $\mathsf{res}(h_{2,0}, h_{1,1}, h_{0,2}; y)$ est divisible par g_4 sans lui être égale (ou associée). Plus généralement, on a le résultat suivant :

Proposition I.6 *Avec les notations et hypothèses de la définition I.10, les composantes du résultant* $\mathsf{res}(f_1, f_2, \ldots, f_s; x_n)$ *sont dans le premier idéal d'élimination de* $\langle f_1, \ldots, f_s \rangle$.

Preuve. $\triangleright$ D'après la proposition I.3, il existe A et B dans $\mathbf{k}[x_1, \ldots, x_n, u_2, \ldots, u_s]$ tel que :

$$Af_1 + B(u_2f_2 + \cdots + u_sf_s) = \mathsf{res}(f_1, \ldots, f_s; x_n)$$

crivons A et B comme polynômes de $\mathbf{k}[x_1, \ldots, x_n][u_2, \ldots, u_s]$

$$A = \sum_\alpha A_\alpha \overline{u}^\alpha \text{ et } B = \sum_\alpha B_\alpha \overline{u}^\alpha$$

où les polynômes A_α et B_α sont donc dans $\mathbf{k}[x_1, \ldots, x_n]$. On note e_i l'élément de $\mathbb{N}^{s-1}$ dont toutes les composantes sont nulles, sauf la i-ième qui vaut 1. Ainsi :

$$u_2 f_2 + \cdots + u_s f_s = \sum_{i \geq 2} \overline{u}^{e_i} f_i$$

On vérifie alors que l'on a :

$$\sum_\alpha h_\alpha \overline{u}^\alpha \;=\; \sum_\alpha \left(A_\alpha f_1 + \sum_{\substack{\beta + e_i = \alpha \\ i \geq 2}} B_\beta f_i \right) \overline{u}^\alpha$$

Ceci montre que chaque h_α appartient bien à $\langle f_1, \ldots, f_s \rangle$. $\triangleleft$

Théorème I.4 *Soit* $\mathbf{k}$ *un corps algébriquement clos. Soient* $f_1, \ldots, f_s$ *des polynômes de l'anneau* $\mathbf{k}[x_1, \ldots, x_n]$ *et* I *l'idéal qu'ils y engendrent. Pour* $1 \leq i \leq s$, *soit* $g_i(x_1, \ldots, x_{n-1})$ *le coefficient dominant de* f_i *vu comme polynôme de* $\mathbf{k}[x_1, \ldots, x_{n-1}][x_n]$. *Soit* I_1 *le premier idéal d'élimination de* I, *i.e.* $I_1 = I \cap \mathbf{k}[x_1, \ldots, x_{n-1}]$. *Alors on a :*

$$(\forall (c_1, \ldots, c_{n-1}) \in \mathbf{V}_\mathbf{k}(I_1) \setminus \mathbf{V}_\mathbf{k}(g_1, \ldots, g_s)) \quad (\exists c_n \in \mathbf{k}) \quad (c_1, \ldots, c_n) \in \mathbf{V}_\mathbf{k}(I)$$

Preuve. $\triangleright$ Notons que le cas $s = 2$ (et par suite le cas $s = 1$) est couvert par la proposition I.5. On suppose donc $s \geq 3$. Soit $c = (c_1, \ldots, c_{n-1}) \in \mathbf{V}_\mathbf{k}(I_1) \setminus \mathbf{V}_\mathbf{k}(g_1, \ldots, g_s)$. Quitte à renuméroter les f_i, on a : $g_1(c) \neq 0$. Notons $h(x_1, \ldots, x_{n-1}, u_2, \ldots, u_s)$ le résultant $\mathsf{res}(f_1, \ldots, f_s; x_n)$. D'après la proposition I.6, on a : $h_\alpha(c) = 0$ pour tout α. Par suite, $h(c, u_2, \ldots, u_s) = 0$.

Supposons tout d'abord que $\mathsf{deg}(f_2, x_n) > \mathsf{deg}(f_i, x_n)$ pour tout $i \geq 3$ et que $g_2(c) \neq 0$. Le polynôme $p = u_2 f_2 + \cdots + u_s f_s$ étant vu dans $\mathbf{k}[x_1, \ldots, x_{n-1}, u_2, \ldots, u_s][x_n]$ a pour coefficient $u_2 g_2$. Par conséquent l'évaluation au point $(c, u_2, \ldots, u_s)$ de h n'est autre que le résultant en x_n des polynômes $f_1(c, x_n)$ et $u_2 f_2(c, x_n) + \cdots + u_s f_s(c, x_n)$ de $\mathbf{k}[u_2, \ldots, u_s, x_n]$. Ainsi, ce dernier résultant est nul. D'après la proposition I.4 nous en déduisons qu'ils possèdent un facteur commun d dans $\mathbf{k}[u_2, \ldots, u_s, x_n]$ de degré positif en x_n. Puisque d divise $f_1(c, x_n)$, d appartient en fait à $\mathbf{k}[x_n]$. Il doit donc diviser chacun des polynômes $f_2(c, x_n), \ldots, f_s(c, x_n)$. Puisque $\mathbf{k}$ est algébriquement clos, il existe alors $c_n \in \mathbf{k}$ tel que $f_i(c, c_n) = 0$ pour tout i.

Supposons maintenant que l'on puisse avoir $\mathsf{deg}(f_2, x_n) \leq \mathsf{deg}(f_i, x_n)$ pour un certain $i \geq 3$ ou $g_2(c) = 0$. On se ramène alors au cas précédent par une astuce analogue à celle utilisée à la fin de la preuve de la proposition I.5. On choisit $m \in \mathbb{N}$ suffisamment grand pour que le polynôme $f_2 + x_n^m f_1$ ait g_1 pour coefficient dominant (en tant qu'élément de $\mathbf{k}[x_1, \ldots, x_{n-1}][x_n]$) et que son degré en x_n soit strictement plus grand que celui de chacun des polynômes $f_3, \ldots, f_s$. Enfin, il ne reste plus qu'à remarquer que l'idéal I est inchangé si l'on y rajoute le polynôme $f_2 + x_n^m f_1$ et que l'on a donc $\mathbf{V}_\mathbf{k}(I) = \mathbf{V}_\mathbf{k}(I + \langle f_2 + x_n^m f_1 \rangle)$. $\triangleleft$

2 Opérations sur les variétés et les idéaux

Soit I un idéal de $\mathbf{P}_n$ et f un polynôme de $\mathbf{P}_n$. On suppose que $\mathbf{K}$ est la clôture algébrique de $\mathbf{k}$ (notation I.1 p. 19). Le théorème des zéros de Hilbert (théorème I.5 p. 31) fournit une condition nécessaire et suffisante portant sur f et I pour que f s'annule en tout point de $\mathbf{V}_{\mathbf{K}}(I)$. Ce théorème fondamental se trouve dans la majorité des ouvrages traitant des anneaux de polynômes. Cependant, dans la première section de cette seconde partie, nous en reproduisons la preuve constructive telle qu'elle est donnée dans [CLO91], avec ses lemmes et corollaires, car elle éclaire différentes questions qui nous intéressent ici. Tout d'abord elle permet de rappeler comment, à l'aide des bases de Gröbner, il est possible de décider algorithmiquement de la vacuité de $\mathbf{V}_{\mathbf{K}}(I)$ (remarque I.6 p. 24), puis de l'appartenance de notre polynôme f au radical $\sqrt{I}$ de l'idéal I (théorème I.6 p. 32) et de l'inclusion éventuelle d'une variété affine dans une autre (remarque I.9 p. 33). Nous rappellerons ensuite comment l'intersection (proposition I.12 p. 34) et le quotient (définition I.12 p. 38, proposition I.19, remarque I.15) de deux idéaux peuvent être calculés à l'aide des bases de Gröbner.

Nous aurons également besoin de décrire la différence ensembliste $V_1 \setminus V_2$ de deux variétés affines V_1 et V_2. Or, cette différence n'est pas en général une variété affine (remarque I.13). À l'aide des notions de *fermeture de Zariski* (définition I.11 p. 35) et de *saturé d'un idéal par un élément* (définition I.13 p. 40) nous caractériserons la plus petite variété affine contenant $V_1 \setminus V_2$ (théorème I.12 p. 44). Enfin, une dernière application du théorème des zéros de Hilbert (et de la fermeture de Zariski) fera l'objet du *théorème de projection* (théorème I.7 p. 37).

2.1 Le théorème des zéros de Hilbert

Soit I un idéal de $\mathbf{P}_n$ et f un polynôme de $\mathbf{P}_n$. La proposition I.7 suivante exprime le fait que si I n'a pas de zéro dans la clôture algébrique de $\mathbf{k}$, alors I ne peut être que l'idéal-unité. Cette première formulation du théorème des zéros, appelé souvent *Théorème des zéros faible*, combinée avec le théorème I.2 p. 23, fournit immédiatement la preuve du point (*iii*) de la remarque I.6 p. 24. Le théorème des zéros (théorème I.5 p. 31) et son corollaire (proposition I.9) exprime le fait que f s'annule en tout zéro (dans la clôture algébrique de $\mathbf{k}$) de I si et seulement si f appartient au radical au I. Par conséquent, si $\mathbf{k}$ est algébriquement clos, les idéaux radicaux de $\mathbf{P}_n$ et les variétés affines de $\mathbf{A}^n(\mathbf{k})$ se correspondent en bijection. De plus, en utilisant les techniques de la preuve du théorème I.5, que $\mathbf{k}$ soit algébriquement clos ou pas, on obtient également un procédé algorithmique pour décider si f appartient au radical de I (théorème I.6 et remarque I.9 p. 32). On termine cette section en caractérisant le radical de I lorsque I est engendré par un seul polynôme et l'on rappelle la définition de la partie sans carrés d'un polynôme de $\mathbf{P}_n$ (proposition I.11 et remarque I.10 p. 33).

Proposition I.7 *Si* $\mathbf{k}$ *est algébriquement clos et si* I *est un idéal de* $\mathbf{P}_n$*, alors on a :*

$$\mathbf{V}_{\mathbf{k}}(I) = \emptyset \implies I = \mathbf{P}_n$$

Cet énoncé est souvent appelé Théorème des zéros faible.

Preuve. $\triangleright$ Par induction sur n. Soit I un idéal de $\mathbf{P}_n$ tel que $\mathbf{V}_{\mathbf{k}}(I) = \emptyset$. Si $n = 1$ alors $\mathbf{P}_n$ est principal. Il existe donc $f \in \mathbf{P}_n$ tel que $f \neq 0$ et $I = \langle f \rangle$. Par suite $\mathbf{V}_{\mathbf{k}}(I)$ est l'ensemble des zéros de f. Puisque $\mathbf{k}$ est algébriquement clos $\mathbf{V}_{\mathbf{k}}(I)$ ne peut être vide que si f est de degré zéro, c'est à dire si f appartient à $\mathbf{k}$, ou bien encore si f engndre $\mathbf{P}_n$ tout entier. On suppose maintenant que la proposition est vraie pour $n-1$ variables. Soient $f_1, \ldots, f_m$ des polynômes de $\mathbf{P}_n$ tels que $I = \langle f_1, \ldots, f_m \rangle$. Si $f_1, \ldots, f_m \in \mathbf{k}[x_1, \ldots, x_{n-1}]$, alors la propriété est démontrée. On peut donc supposer que f_1 est de degré non nul en x_n. Soient $h_0, \ldots, h_r$ des polynômes homogènes tels que h_i est de degré (total) i et $f_1 = h_r + \cdots + h_0$ où $h_r \neq 0$. Nous allons vérifier que l'on peut supposer qu'il existe $(a_1, \ldots, a_n) \in \mathbf{k}^n$ tel que $h_r(a_1, \ldots, a_n) \neq 0$. Ceci résultera du lemme suivant :

Proposition I.8 *Soit* $\mathbf{L}$ *un corps infini et* $p \in \mathbf{L}[x_1, \ldots, x_n]$. *Alors on a :*

$$p \neq 0 \quad \Longrightarrow \quad (\exists (a_1, \ldots, a_n) \in \mathbf{L}^n) \;\; p(a_1, \ldots, a_n) \neq 0$$

Preuve. $\triangleright$ Par récurrence sur $n \geq 1$. Supposons $n = 1$. Soit d le degré de p. Si $d = 0$ ou $d = 1$, la propriété est triviale. Pour $d \geq 2$, $\mathbf{L}[x_1]$ étant factoriel, il existe des polynômes $f_1, \ldots, f_r$ irréductibles sur $\mathbf{L}[x_1]$ dont le produit est p. Les polynômes $f_1, \ldots, f_r$ de degré au moins 2 ne s'annulent pas sur $\mathbf{L}$ (sinon, ils ne seraient pas irréductibles, penser à la division euclidienne). On conclut alors sans peine puisque $\mathbf{L}$ est un corps infini. Supposons maintenant $n > 1$. Si p est de degré zéro en x_n, alors on conclut par récurrence. Sinon, il existe des polynômes $q_0, \ldots, q_r \in \mathbf{L}[x_1, \ldots, x_{n-1}]$ tels que

$$p = q_r(x_1, \ldots, x_{n-1}) x_n^r + \cdots + q_0(x_1, \ldots, x_{n-1}) \;\text{ et }\; q_r(x_1, \ldots, x_{n-1}) \neq 0$$

Par hypothèse, il existe $(a_1, \ldots, a_{n-1}) \in \mathbf{L}^{n-1}$ tel que $q_r(a_1, \ldots, a_{n-1}) \neq 0$. Par suite, le polynôme

$$\tilde{p} = q_r(a_1, \ldots, a_{n-1}) x_n^r + \cdots + q_0(a_1, \ldots, a_{n-1})$$

de $\mathbf{L}[x_n]$ n'est pas nul. Il existe donc $a_n \in \mathbf{L}$ tel que $\tilde{p}(a_n) \neq 0$, ce qui termine la preuve de ce lemme. $\triangleleft$

Terminons maintenant celle de la proposition I.7. Comme $\mathbf{k}$ est algébriquement clos, il est aussi infini (penser aux racines de l'unité). Il existe donc $(a_1, \ldots, a_n) \in \mathbf{k}^n$ tel que $h_r(a_1, \ldots, a_n) \neq 0$. Puisque h_r est homogène de degré positif, quitte à renuméroter les variables, on peut supposer $a_n \neq 0$. De plus, quitte à remplacer x_n par $\frac{x_n}{a_n}$, on peut supposer $a_n = 1$. Pour $1 \leq i \leq n-1$, on pose alors $y_i = x_i - a_i x_n$. On vérifie qu'il existe $g \in \mathbf{P}_n$ tel que : $\mathsf{deg}(g, x_n) < \mathsf{deg}(f, x_n)$ et :

$$f(x_1, \ldots, x_n) = h_r(a_1, \ldots, a_{n-1}, 1) x_n^r + g(y_1, \ldots, y_{n-1}, x_n)$$

Soit alors I_1 le premier idéal d'élimination de I, i.e. $I_1 = I \cap \mathbf{k}[x_1, \ldots, x_{n-1}]$. D'après le théorème d'extension, s'il existe $(\zeta_1, \ldots, \zeta_{n-1}) \in \mathbf{V}_{\mathbf{k}}(I_1)$ alors il existe $\zeta_n \in \mathbf{k}$ tel que $(\zeta_1, \ldots, \zeta_n) \in \mathbf{V}_{\mathbf{k}}(I)$. Or $\mathbf{V}_{\mathbf{k}}(I)$ est vide. Par suite, $\mathbf{V}_{\mathbf{k}}(I_1)$ est vide et, par hypothèse de récurrence il vient : $I_1 = \mathbf{k}[x_1, \ldots, x_{n-1}]$. D'où la conclusion. $\triangleleft$

Théorème I.5 *Si* $\mathbf{k}$ *est algébriquement clos, pour des polynômes* $f, f_1, \ldots, f_s$ *de* $\mathbf{P}_n$ *on a :*

$$f \in \mathbf{Id}_{\mathbf{k}}(\mathbf{V}_{\mathbf{k}}(f_1, \ldots, f_s)) \quad \Longleftrightarrow \quad (\exists m \in \mathbb{N}^*) \;\; f^m \in \langle f_1, \ldots, f_s \rangle$$

Cet énoncé est souvent appelé Théorème des zéros de Hilbert.

Preuve. $\triangleright$ Il suffit de vérifier l'implication directe. Soit y une variable *supplémentaire*. On considère l'idéal de $\mathbf{k}[x_1, \ldots, x_n, y]$ défini par $\tilde{I} = \langle f_1, \ldots, f_n, 1 - yf \rangle$. Soit $(a_1, \ldots, a_{n+1}) \in \mathbf{k}^{n+1}$. Si $(a_1, \ldots, a_n) \in \mathbf{V_k}(f_1, \ldots, f_s)$, alors f s'annule aussi en ce point et $(1 - yf)$ vaut 1 en $(a_1, \ldots, a_{n+1})$. Par suite, $(a_1, \ldots, a_{n+1}) \notin \mathbf{V_k}(\tilde{I})$. Si $(a_1, \ldots, a_n) \notin \mathbf{V_k}(f_1, \ldots, f_s)$, on a clairement aussi $(a_1, \ldots, a_{n+1}) \notin \mathbf{V_k}(\tilde{I})$. Bref, $\mathbf{V_k}(\tilde{I})$ est vide. D'après le théorème des zéros faible, il existe alors $p_1, \ldots, p_s, q \in \mathbf{k}[x_1, \ldots, x_n, y]$ tel que

$$1 = \sum_{i=1}^{s} p_i(x_1, \ldots, x_n, y) f_i + q(x_1, \ldots, x_n, y)(1 - yf)$$

En remplaçant y par $1/f$, il vient :

$$1 = \sum_{i=1}^{s} p_i(x_1, \ldots, x_n, 1/f) f_i$$

En multipliant l'équation précédente par f^N, où N est le degré maximum en x_{n+1} des polynômes p_i, on obtient le résultat annoncé. $\triangleleft$

Proposition I.9 *Si V est une variété affine sur $\mathbf{k}$ de $\mathbf{A}^n(\mathbf{K})$ alors on a :*

$$\sqrt{\mathbf{Id_k}(V)} \;=\; \mathbf{Id_k}(V)$$

De plus si $\mathbf{k}$ est algébriquement clos et si I est un idéal de $\mathbf{P}_n$, alors on a :

$$\mathbf{Id_k}(\mathbf{V_k}(I)) \;=\; \sqrt{I}$$

Cet énoncé est souvent appelé Théorème des zéros fort.

Preuve. $\triangleright$ La première assertion est triviale ($\mathbf{P}_n$ est intègre). On suppose maintenant que $\mathbf{k}$ est algébriquement clos. Soit I est un idéal de $\mathbf{P}_n$ et $f \in \mathbf{P}_n$. Supposons que $f \in \sqrt{I}$. Alors il existe $n \in \mathbb{N}$ tel que $f^n \in I$. Donc f^n, et par suite f, s'annulent sur $\mathbf{V_k}(I)$. Bref, $f \in \mathbf{Id_k}(\mathbf{V_k}(I))$. Réciproquement, supposons que $f \in \mathbf{Id_k}(\mathbf{V_k}(I))$. Par définition, f s'annule sur $\mathbf{V_k}(I)$. D'après le théorème des zéros de Hilbert, il existe un entier $m \geq 1$ tel que $f^m \in I$ ce qui signifie $f \in \sqrt{I}$. $\triangleleft$

Théorème I.6 *Si $\mathbf{k}$ est un corps quelconque, $I = \langle f_1, \ldots, f_r \rangle$ un idéal de $\mathbf{P}_n$ et f un polynôme de $\mathbf{P}_n$, alors on a :*

$$f \in \sqrt{I} \iff \langle f_1, \ldots, f_n, 1 - yf \rangle = \mathbf{k}[x_1, \ldots, x_n, y]$$

Preuve. $\triangleright$ Dans la preuve du théorème des zéros de Hilbert, sans utiliser le fait que $\mathbf{k}$ était algébriquement clos, on a établi l'implication : $1 \in \tilde{I} \implies f \in \sqrt{I}$. Réciproquement, s'il existe un entier $m \geq 1$ tel que $f^m \in I$, il vient :

$$1 = y^m f^m + (1 - y^m f^m) = y^m f^m + (1 - yf)(1 + yf + \cdots + y^{m-1} f^{m-1}) \in \tilde{I}$$

puisque $\tilde{I} = I + \langle 1 - yf \rangle$. $\triangleleft$

Remarque I.9 Ce dernier théorème fournit un processus algorithmique pour décider si f appartient à $\sqrt{I}$ (pour $\mathbf{k}$ quelconque) ou, pour décider si f s'annule en tous les points de $\mathbf{V}_{\mathbf{k}}(I)$ (pour $\mathbf{k}$ algébriquement clos). Il suffit de calculer une base de Gröbner minimale de $\tilde{I} = I + \langle 1 - yf \rangle$. La réponse est oui si et seulement si cette base est $\{1\}$ (théorème I.1, remarque I.6 p. 23). On en déduit alors un processus algorithmique pour décider si une variété affine sur la clôture algébrique de $\mathbf{k}$ est contenue dans une autre. Ceci résulte immédiatement de la proposition suivante.

N.B. On rappelle que dans toute cette thèse les variétés affines sont par défaut relatives au corps $\mathbf{K}$, extension quelconque du corps $\mathbf{k}$ des coefficients des polynômes de $\mathbf{P}_n$ (notations I.1 et I.2 p. 19).

Proposition I.10 *Soient I et J deux idéaux de $\mathbf{P}_n$ et G un système de générateurs de J. On suppose que $\mathbf{K}$ est la clôture algébrique de $\mathbf{k}$. Alors on a :*

$$\mathbf{V}(I) \subseteq \mathbf{V}(J) \quad \Longleftrightarrow \quad (\forall g \in G)\ g \in \sqrt{I}$$

Preuve. $\triangleright$ Les propositions I.1 et I.9 fournissent l'équivalence : $\mathbf{V}(I) \subseteq \mathbf{V}(J) \iff \sqrt{J} \subseteq \sqrt{I}$. L'application *passage au radical* $I \longmapsto \sqrt{I}$ étant une fermeture (remarque I.12 p. 35), on en déduit l'équivalence : $\sqrt{J} \subseteq \sqrt{I} \iff (\forall g \in G)\ g \in \sqrt{I}$. $\triangleleft$

Proposition I.11 *Soit $f \in \mathbf{P}_n$ et I l'idéal principal de $\mathbf{P}_n$ engendré par f. Soit $f = f_1^{e_1} f_2^{e_2} \cdots f_r^{e_r}$ une factorisation de f où $f_1, \ldots, f_r$ sont irréductibles dans $\mathbf{P}_n$. Alors on a :*

$$\sqrt{I} \;=\; \langle f_1 \cdots f_r \rangle$$

Preuve. $\triangleright$ Il est clair que $\langle f_1 \cdots f_r \rangle \subseteq \sqrt{I}$. Réciproquement, soit $g \in \sqrt{I}$. Il existe donc $m \in \mathbb{N}$ et $h \in \mathbf{P}_n$ tel que $g^m = hf$. Soit alors $g = g_1^{d_1} g_2^{d_2} \cdots g_s^{d_s}$ une factorisation de g où $g_1, \ldots, g_s$ sont irréductibles dans $\mathbf{P}_n$. Par suite, chaque f_i doit être égal à une constante multiplicative près, à l'un des g_j. Bref, le produit $f_1 f_2 \cdots f_r$ divise g et $g \in \langle f_1 \cdots f_r \rangle$. $\triangleleft$

Remarque I.10 Notons $\mathsf{squareFreePart}(f)$ le polynôme $f_1 f_2 \cdots f_r$. Puisque $\mathbf{P}_n$ est factoriel, le pgcd d'une famille (finie) de polynômes de $\mathbf{P}_n$ peut se définir (et se calculer, sans que cela soit la méthode la plus efficace) à un facteur inversible près en comparant les décompositions en produit de facteurs irréductibles des polynômes de cette famille. On vérifie que si $\mathbf{k}$ est un corps de caractéristique zéro, alors le polynôme $\mathsf{squareFreePart}(f)$ est donné par ([CLO91]) :

$$\mathsf{squareFreePart}(f) = \frac{f}{\mathsf{pgcd}(f, \frac{\partial f}{\partial x_i}, \ldots, \frac{\partial f}{\partial x_i})}$$

Nous renvoyons à [Yun76] pour des algorithmes efficaces de partie sans carrés d'un polynôme.

2.2 Fermeture de Zariski

Soient I et J des idéaux de $\mathbf{P}_n$. Nous avons déjà vu (proposition I.1 p. 20) les formules :

$$\mathbf{V}(I) \cap \mathbf{V}(J) \;=\; \mathbf{V}(I + J) \quad \text{et} \quad \mathbf{V}(I) \cup \mathbf{V}(J) \;=\; \mathbf{V}(I \bigcap J) \;=\; \mathbf{V}(I \,.\, J)$$

Un ordre admissible sur les monômes étant fixé, on peut supposer que I et J sont respectivement décrits par des bases de Gröbner G_I et G_J. De la première formule, on déduit alors un procédé algorithmique pour décider si deux variétés affines (sur la clôture algébrique $\mathbf{K}$ de $\mathbf{k}$) ont des points communs : il suffit de calculer une base de Gröbner de $\mathbf{Id}_{\mathbf{K}}(G_I \cup G_J)$. Plus généralement, si l'on se contente de décrire une variété affine V de $\mathbf{A}^n(\mathbf{K})$ par un idéal H de $\mathbf{P}_n$, radical ou non, tel que $\mathbf{V}_{\mathbf{K}}(H) = V$, cette première formule fournit un procédé algorithmique pour décrire l'intersection de deux variétés affines. Avec cette même réserve, la seconde fournit un procédé algorithmique pour décrire leur réunion : à partir de G_I et G_J, il suffit soit de calculer une base de Gröbner du produit des idéaux I et J, soit de calculer une base de Gröbner de l'intersection de ces deux idéaux, en utilisant la proposition I.12

Cette proposition a une autre conséquence que nous utiliserons dans les chapitres suivants. Supposons que l'on souhaite décrire le radical d'un idéal H et que l'on ait à notre disposition un algorithme produisant à partir de H des idéaux radicaux $S_1, \ldots, S_t$ tel que :

$$\mathbf{V}(H) \;=\; \mathbf{V}(S_1) \cup \cdots \cup \mathbf{V}(S_t)$$

Utilisant le fait que le radical de l'intersection de deux idéaux d'un anneau commutatif est égal à l'intersection de leurs radicaux, il suffira d'appliquer la proposition I.12 pour calculer une base de Gröbner de $\sqrt{H}$ à partir des bases de Gröbner des idéaux $S_1, \ldots, S_t$. Nous reviendrons sur ce type de problème tout au long des chapitres 4 à 6. Nous renvoyons aussi à [GTZ88] pour des algorithmes de décomposition primaire minimale d'un idéal, les composantes primaires étant décrites par des bases de Gröbner.

Nous aurons également besoin de décrire la différence ensembliste des variétés affines $\mathbf{V}(I)$ et $\mathbf{V}(J)$ c'est-à-dire $\mathbf{V}(I) \setminus \mathbf{V}(J)$. Or, il ne s'agit pas nécessairement d'une variété affine (remarque I.13). Ceci conduit tout d'abord à utiliser *la fermeture de Zariski* (définition I.11 p. 35) qu'introduiront les propositions I.13 et I.14. On approche alors toute partie S de $\mathbf{A}^n(\mathbf{K})$, qui n'est pas une variété affine, par l'intersection, notée $\overline{S}$, des variétés affines qui contiennent S. Il nous restera ensuite à construire un idéal L de $\mathbf{P}_n$ tel que : $\overline{S} = \mathbf{V}(L)$. Cet objectif sera atteint dans les deux prochaines sections (théorème I.8 p. 38 et théorème I.12 p. 44). La topologie de Zariski peut également être introduite du point de vue des idéaux de $\mathbf{P}_n$ plutôt qu'à l'aide des variétés affines. Nous rappellerons aussi cette présentation (notation I.4 p. 36 et proposition I.15) qui sera utile par la suite.

Nous terminerons la présente section par un *théorème de projection* (théorème I.7 p. 37) permettant de *mesurer* à l'aide de la fermeture de Zariski les solutions qui ne peuvent être *remontées* dans un processus d'*élimination-extension* (voir la première section).

Proposition I.12 *Pour des idéaux I, J de $\mathbf{P}_n$ et une variable supplémentaire t, on a :*

$$I \cap J \;=\; (tI + (1-t)J) \cap \mathbf{P}_n$$

Preuve. ▷ Rappelons que tI est l'idéal de $\mathbf{k}[x_1, \ldots, x_n, t]$ engendré par $\{tp_1, \ldots, tp_s\}$ si $\{p_1, \ldots, p_s\}$ est un système de générateurs de I. On vérifie que cette construction de tI est indépendante du choix du système de générateurs de I et que pour tout $g(x_1, \ldots, x_n, t) \in tI$ et tout $a \in \mathbf{k}$ on a : $g(x_1, \ldots, x_n, a) \in I$. Vérifions maintenant l'égalité annoncée dans la proposition. Soit $f \in \mathbf{P}_n$. Supposons tout d'abord que : $f \in I \cap J$. Alors tf et $(1-t)f$ appartiennent respectivement à tI et $(1-t)J$. Donc $f = tf + (1-t)f$ appartient à la somme de ces idéaux. Comme $f \in \mathbf{P}_n$, on a aussi $f \in (tI + (1-t)J) \cap \mathbf{P}_n$. Inversement, si f appartient à ce dernier idéal, il existe $g(x_1, \ldots, x_n, t) \in tI$ et $h(x_1, \ldots, x_n, t) \in (1-t)J$, tel que

$$f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n, t) + h(x_1, \ldots, x_n, t)$$

Comme tout $p \in tI$ est multiple de t, en spécialisant t à 0, il vient $g(x_1, \ldots, x_n, 0) = 0$, donc $f(x_1, \ldots, x_n) = h(x_1, \ldots, x_n, 0)$, et par suite $f(x_1, \ldots, x_n) \in J$. De même, en spécialisant t à 1, on obtient $f(x_1, \ldots, x_n) \in I$, d'où la conclusion. $\lhd$

Proposition I.13 *Soit S une partie de $\mathbf{A}^n(\mathbf{K})$. Alors, $\mathbf{V_K}(\mathbf{Id_k}(S))$ est l'intersection de toutes les variétés affines sur $\mathbf{k}$ de $\mathbf{A}^n(\mathbf{K})$ contenant S.*

Preuve. $\rhd$ Soit donc S une partie de $\mathbf{A}^n(\mathbf{K})$. Puisque $\mathbf{V_K}(\mathbf{Id_k}(S))$ est une variété affine sur $\mathbf{k}$ contenant S, il suffit de vérifier que toute autre variété affine sur $\mathbf{k}$ contenant S contient $\mathbf{V_K}(\mathbf{Id_k}(S))$. Soit donc W une variété affine sur $\mathbf{k}$ telle que : $S \subseteq W$. Alors, à l'aide de la proposition I.1, il vient, $\mathbf{Id_k}(S) \supseteq \mathbf{Id_k}(W)$, d'où, $\mathbf{V_K}(\mathbf{Id_k}(S)) \subseteq \mathbf{V_K}(\mathbf{Id_k}(W)) = W$ $\lhd$.

Proposition I.14 *Les variétés affines sur $\mathbf{k}$ de $\mathbf{A}^n(\mathbf{K})$ forment les fermés d'une topologie dont la fermeture associée est l'application $S \longmapsto \overline{S}$ où $\overline{S}$ dénote $\mathbf{V_K}(\mathbf{Id_k}(S))$ pour une partie quelconque S de $\mathbf{A}^n(\mathbf{K})$.*

Preuve. $\rhd$ En effet, $\mathbf{A}^n(\mathbf{K}) = \mathbf{V_K}(\langle 0 \rangle)$ est une variété affine sur $\mathbf{k}$. D'autre part, d'après la proposition I.1, toute réunion finie de variétés affines sur $\mathbf{k}$ est encore une variété affine sur $\mathbf{k}$. Enfin, si S est une intersection quelconque de variétés affines sur $\mathbf{k}$, alors $\overline{S}$ est contenue dans chacune de ces variétés, donc dans leur intersection, et en fait $\overline{S} = S$. $\lhd$

Définition I.11 *Pour une extension $\mathbf{K}$ quelconque du corps $\mathbf{k}$, on appelle* topologie de Zariski relative à $\mathbf{k}$*, que l'on notera ici $\mathcal{F}_{\mathbf{k}}(\mathbf{A}^n(\mathbf{K}))$, la topologie de $\mathbf{A}^n(\mathbf{K})$ dont les fermés sont les variétés affines sur $\mathbf{k}$ de $\mathbf{A}^n(\mathbf{K})$.*

Remarque I.11 Rappelons quelques identités usuelles de topologie. Soit $(E, \mathcal{T})$ un espace topologique de fermeture $X \longmapsto \overline{X}$. Pour X, $Y \subseteq E$, on a :

(i) $X \subseteq \overline{Y} \iff \overline{X} \subseteq \overline{Y}$

(ii) $\overline{X \cup Y} = \overline{X} \cup \overline{Y}$

(iii) $X \subseteq Y \implies \overline{Y} = \overline{X} \cup \overline{Y \setminus X}$

Comme $\overline{X}$ est l'intersection des fermés contenant X, on a (i). Vérifions (ii). D'une part $\overline{X} \cup \overline{Y}$ est un fermé qui contient $X \cup Y$, donc qui contient $\overline{X \cup Y}$. D'autre part, $\overline{X}$ et $\overline{Y}$ sont contenus dans $\overline{X \cup Y}$, donc leur réunion aussi. Enfin, la propriété (iii) résulte de (ii).

Remarque I.12 Rappelons que si E est un ensemble et $X \longmapsto \overline{X}$ est une application de 2^E, l'ensemble des parties de E, dans 2^E, cette application est une *fermeture* dans E si et seulement si on a :

(a) $(\forall X \subseteq E)$ $X \subseteq \overline{X}$

(b) $(\forall X \subseteq E)$ $\overline{\overline{X}} = \overline{X}$

(c) $(\forall X, Y \subseteq E)$ $X \subseteq Y \implies \overline{X} \subseteq \overline{Y}$

La fermeture associée à un espace topologique $(E, \mathcal{T})$ est évidemment une fermeture au sens général ci-dessus. Cependant, toute fermeture dans E, satisfaisant les axiomes (a), (b) et

(c) ne définit pas nécessairement une topologie. Voici un contre-exemple. Soit $\mathbf{A}$ un anneau commutatif unitaire. L'application $X \longmapsto \sqrt{\mathbf{Id}_{\mathbf{A}}(X)}$ de l'ensemble des parties de $\mathbf{A}$ dans lui-même (qui à une partie de $\mathbf{A}$ fait correspondre l'idéal radical qu'elle engendre) est une fermeture au sens des axiomes (a), (b) et (c). De plus, pour des idéaux I et J de $\mathbf{A}$ on a :

$$\sqrt{I \cap J} \ = \ \sqrt{I} \cap \sqrt{J}$$

mais on a seulement :

$$\sqrt{I \cup J} \ = \sqrt{\sqrt{I} \cup \sqrt{J}}$$

Considérer l'anneau $\mathbf{A} = \mathbf{P}_n$, avec $n \geq 3$, et les idéaux $I = \langle (x_3 - x_1)(x_3 - x_2) \rangle$ et $J = \langle x_1, x_2 \rangle$. Ces deux idéaux sont radicaux. Cependant leur somme n'est pas un idéal radical puisque l'on a : $I + J = \langle x_1, x_2, x_3^2 \rangle$.

Par conséquent, les idéaux radicaux ne forment pas les fermés d'une topologie sur $\mathbf{A}$. Cependant, on peut munir l'ensemble des idéaux premiers de l'anneau commutatif unitaire $\mathbf{A}$ d'une topologie. Ceci fait l'objet de la proposition suivante.

Notations I.4 Soit $\mathbf{A}$ un anneau commutatif unitaire. On désigne par $\mathsf{spec}(\mathbf{A})$ l'ensemble des idéaux premiers de $\mathbf{A}$ distincts de $\mathbf{A}$. Pour un idéal I de $\mathbf{A}$, on désigne par $\mathcal{V}(I)$ l'ensemble des idéaux premiers contenant I. Ainsi :

$$\mathcal{V}(I) \quad = \quad \{\mathcal{P} \in \mathsf{spec}(\mathbf{A}) \ \mid \ I \subseteq \mathcal{P}\}$$

De plus, pour un élément a de $\mathbf{A}$ on écrira $\mathcal{V}_a$ au lieu de $\mathcal{V}(\langle a \rangle)$ et l'on dénotera par $\mathcal{D}_a$ le complémentaire de $\mathcal{V}_a$ dans $\mathsf{spec}(\mathbf{A})$. De sorte que :

$$\mathcal{D}_a \quad = \quad \{\mathcal{P} \in \mathsf{spec}(\mathbf{A}) \ \mid \ a \notin \mathcal{P}\}$$

Rappelons (voir [van91]) qu'avec ces notations on a :

$$\sqrt{I} \quad = \quad \cap_{\mathcal{P} \in \mathcal{V}(I)} \mathcal{P}$$

Proposition I.15 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire. Les sous-ensembles* $\mathcal{V}(I)$ *de* $\mathsf{spec}(\mathbf{A})$*, pour* I *décrivant l'ensemble des idéaux de* $\mathbf{A}$*, forment les fermés d'une topologie sur* $\mathsf{spec}(\mathbf{A})$*.*

Preuve. $\triangleright$ Notons tout d'abord que $\mathcal{V}(\mathbf{A}) = \emptyset$ et $\mathcal{V}(\langle 0 \rangle) = \mathsf{spec}(\mathbf{A})$. Soient maintenant I et J deux idéaux de $\mathbf{A}$. On a :

$$\mathcal{V}(I) \cup \mathcal{V}(J) \ = \ \{\mathcal{P} \in \mathsf{spec}(\mathbf{A}) \ \mid \ (I \subseteq \mathcal{P}) \text{ ou } (J \subseteq \mathcal{P})\} = \mathcal{V}(I \cap J)$$

Par suite les $\mathcal{V}(I)$ sont stables pour l'union finie. Soit maintenant $(I_\delta)_{\delta \in \Delta}$ une famille quelconque d'idéaux de $\mathbf{A}$. On a :

$$\cap_{\delta \in \Delta} \mathcal{V}(I_\delta) \ = \ \mathcal{V}(\Sigma_{\delta \in \Delta} I_\delta)$$

Par suite les $\mathcal{V}(I)$ sont stables pour l'intersection quelconque. $\triangleleft$

Proposition I.16 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire. On a les propriétés suivantes pour tout* a *et tout* b *de* $\mathbf{A}$ *:*

(i) $\{\mathcal{D}_a \mid a \in \mathbf{A}\}$ *est une base d'ouverts de* $\mathsf{spec}(\mathbf{A})$.

(ii) $\mathcal{D}_a \cap \mathcal{D}_b \; = \; \mathcal{D}_{ab}$

(iii) $\mathcal{D}_a = \emptyset \quad \Longleftrightarrow \quad a$ nilpotent

(iv) $\mathcal{D}_a = \mathsf{spec}(\mathbf{A}) \quad \Longleftrightarrow \quad a$ inversible

(v) $\mathcal{D}_a = \mathcal{D}_b \quad \Longleftrightarrow \quad \sqrt{\langle a \rangle} = \sqrt{\langle b \rangle}$

Preuve. $\triangleright$ Vérifions (i). Il s'agit de vérifier que pour chaque idéal I de $\mathbf{A}$ il existe une partie $X \subseteq \mathbf{A}$ tel que :

$$\mathsf{spec}(\mathbf{A}) \setminus \mathcal{V}(I) \quad = \quad \cup_{x \in X} \mathcal{D}_x$$

Il suffit de prendre pour X un système de générateurs de I. En effet, on a alors :

$$\begin{aligned} \cup_{x\in X} \mathcal{D}_x &= \cup_{x \in X} (\mathsf{spec}(\mathbf{A}) \setminus \mathcal{V}_x) \\ &= \mathsf{spec}(\mathbf{A}) \setminus (\cap_{x \in X} \mathcal{V}_x) \\ &= \mathsf{spec}(\mathbf{A}) \setminus \mathcal{V}(\Sigma_{x \in X} \langle x \rangle) \end{aligned}$$

La propriété (ii) résulte immédiatement de la définition d'un idéal premier. La propriété (iii) résulte du fait que :

$$\sqrt{\langle 0 \rangle} \quad = \quad \cap_{\mathcal{P} \,\in\, \mathsf{spec}(\mathbf{A})} \mathcal{P}$$

La propriété (iv) est trivial. La propriété (v) résulte de la définition-même de $\mathcal{V}(I)$ (voir notation I.4). $\triangleleft$

Remarque I.13 La différence ensembliste de deux variétés affines n'est pas en général une variété affine. En effet, si c'était le cas, tout ouvert de $\mathcal{F}_{\mathbf{k}}(\mathbf{A}^n(\mathbf{K}))$ serait une variété affine et inversement. Supposons alors que $\mathbf{k}$ et $\mathbf{K}$ soient égaux au corps $\mathbb{R}$ des nombres réels. Tout fermé (resp. ouvert) de $\mathcal{F}_{\mathbf{k}}(\mathbf{A}^n(\mathbf{K}))$ est alors également un fermé (resp. ouvert) pour la topologie *usuelle* de la distance euclidienne. En effet, les fonctions polynômiales de $\mathbb{R}^n$ dans $\mathbb{R}$ sont continues. Par suite, toute variété affine sur $\mathbb{R}$ serait un ouvert de la topologie *usuelle.* Finalement, tous les singletons de $\mathbb{R}^n$ seraient des ouverts pour cette topologie. Absurde!

Théorème I.7 *Soit* $\mathbf{k}$ *un corps algébriquement clos,* I *un idéal de* $\mathbf{P}_n$*, et* V *la variété affine de* $\mathbf{A}^n(\mathbf{k})$ *définie par* I*. Soit* $\pi_k : \mathbf{k}^n \longmapsto \mathbf{k}^{n-k}$ *la projection sur les* $n-k$ *premières composantes et* $I_k = I \cap \mathbf{k}[x_1, \ldots, x_{n-k}]$ *le* k*-ième idéal d'élimination de* I*. Alors on a :*

$$\mathbf{V}(I_k) \; = \; \overline{\pi_k(V)}$$

Preuve. $\triangleright$ On doit donc établir : $\mathbf{V}(I_k) \; = \; \mathbf{V}(\mathbf{Id}_{\mathbf{k}}(\pi_k(V)))$. Remarquons que l'on a : $\pi_k(V) \;\subseteq\; \mathbf{V}(I_k)$. En effet, les projetés sur les $n-k$ premières composantes des zéros de I sont des zéros des éléments de I qui ne dépendent que des $n-k$ premières variables. Comme $\mathbf{V}(\mathbf{Id}_{\mathbf{k}}(\pi_k(V)))$ est la plus petite variété affine sur $\mathbf{k}$ contenant $\pi_k(V)$, on en déduit l'inclusion : $\mathbf{V}(\mathbf{Id}_{\mathbf{k}}(\pi_k(V))) \;\subseteq\; \mathbf{V}(I_k)$.

Vérifions l'inclusion inverse. Soit $f \in \mathbf{Id}_{\mathbf{k}}(\pi_k(V))$ et $(a_1, \ldots, a_{n-k}) \in \pi_k(V)$. On a donc $f(a_1, \ldots, a_{n-k}) \; = \; 0$. Le polynôme f étant vu comme élément de $\mathbf{k}[x_1, \ldots, x_n]$ pour tout

$(a_1, \ldots, a_n) \in V$, on a : $f(a_1, \ldots, a_n) = 0$. Donc, d'après le théorème des zéros, il existe $m \in \mathbb{N}$ tel que : $f^m \in I$. Par suite $f^m \in I_k$ et donc $f \in \sqrt{I_k}$. Ainsi nous avons l'inclusion : $\mathbf{Id}_{\mathbf{k}}(\pi_k(V)) \subseteq \sqrt{I_k}$. D'où l'on tire $\mathbf{V}(\sqrt{I_k}) = \mathbf{V}(I_k) \subseteq \mathbf{V}(\mathbf{Id}_{\mathbf{k}}(\pi_k(V)))$. $\lhd$

2.3 Quotient d'idéaux

On rappelle dans cette section la notion de quotient de deux idéaux (définition I.12 et proposition I.17). Puis, pour deux idéaux I et J de $\mathbf{P}_n$, avec I radical, on donne une première réponse au problème de la description de la différence $\mathbf{V}(I) \setminus \mathbf{V}(J)$ (théorème I.8 et proposition I.18).

Définition I.12 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire,* I *un idéal de* $\mathbf{A}$ *et* P *une partie de* $\mathbf{A}$*. On appelle* conducteur *de* P *dans* I *la partie de* $\mathbf{A}$ *notée* $I : P$ *et définie par :*

$$I : P = \{a \in \mathbf{A} \ \mid \ (\forall p \in P) \ a\, p \in I\}$$

Lorsque P *est aussi un idéal de* $\mathbf{A}$*, on dit aussi que* $I : P$ *est le* quotient *de* I *par* P*.*

Remarque I.14 Soit $\mathbf{A}$ un anneau commutatif unitaire et I, J deux idéaux $\mathbf{A}$. On vérifie que $I : J$ est un idéal de $\mathbf{A}$. De plus, on a :

$$(I : J).J \ \subseteq \ I \ \subseteq \ I : J$$

Lorsque $I = I : J$ on dit que J est *relativement premier à* I. Ceci se produit en particulier si I est premier et si $J \not\subseteq I$. Donnons quelques exemples de quotients avec $\mathbf{A} = \mathbb{Z}$:

$$\begin{array}{lll} <12>:<0>=<1> & <12>:<1>=<12> & <12>:<2>=<6> \\ <12>:<3>=<4> & <12>:<5>=<12> & <12>:<8>=<3> \end{array}$$

Proposition I.17 *Pour des idéaux* $I, I_1, \ldots, I_r, J, J_1, \ldots, J_s, K$ *d'un anneau commutatif unitaire* $\mathbf{A}$*, on a les formules suivantes :*

$$\begin{array}{rlrl} (i) & I : \mathbf{A} = I & (v) & (\cap_{i=1}^r I_i) : J = \cap_{i=1}^r (I_i : J) \\ (ii) & IJ \subseteq K \iff I \subseteq K : J & (vi) & I : (\sum_{i=1}^s J_i) = \cap_{i=1}^r (I : J_i) \\ (iii) & J \subseteq I \iff I : J = \mathbf{A} & (vii) & I = \sqrt{I} \implies \sqrt{I : J} = I : J \\ (iv) & (I : J) : K = I : (JK) & & \end{array}$$

Preuve. $\rhd$ La vérification de chaque formule est facile. Vérifions (vi). Soit $x \in \mathbf{A}$. On a les équivalences suivantes : $x \in I : (\sum_i J_i) \iff (\forall y \in \sum_i J_i) xy \in I \iff (\forall i)(\forall y_i \in J_i) x y_i \in I \iff (\forall i) x \in I : J_i \iff x \in \cap_i I : J_i$. Vérifions (vii). On suppose donc que I est radical. On a bien sr $I : J \subseteq \sqrt{I : J}$. Inversement, soit $x \in \sqrt{I : J}$. Il existe donc $m \in \mathbb{N}$ tel que : $x^m \in I : J$. Donc pour tout $y \in J$ on a : $x^m y \in I$ et donc : $x^m y^m \in I$ Donc pour tout $y \in J$ on a : $xy \in \sqrt{I}$. Or, I est radical. On en tire : $x \in I : J$. $\lhd$

Théorème I.8 *Pour deux idéaux* I *et* J *de* $\mathbf{P}_n$*, on a :*

$$\mathbf{V}(I : J) \ \supseteq \ \overline{\mathbf{V}(I) \setminus \mathbf{V}(J)}$$

De plus, si $\mathbf{k}$ *est algébriquement clos et si* I *est radical, l'inclusion précédente est une égalité.*

Preuve. ▷ Grâce à la proposition I.1 p. 20, l'inclusion annoncée résultera de :

$$I : J \subseteq \mathbf{Id_k}(\mathbf{V}(I) \setminus \mathbf{V}(J))$$

Pour cela, soit $f \in I : J$ et $x \in \mathbf{V}(I) \setminus \mathbf{V}(J)$. Pour tout $g \in J$ on a alors $fg \in I$, donc $f(x)g(x) = 0$, et par suite $f(x) = 0$. Bref $f \in \mathbf{Id_k}(\mathbf{V}(I) \setminus \mathbf{V}(J))$, et l'on a bien l'inclusion désirée. Supposons $\mathbf{k}$ algébriquement clos et I radical. Soit $x \in \mathbf{V}(I : J)$ et $h \in \mathbf{Id_k}(\mathbf{V}(I) \setminus \mathbf{V}(J))$. Pour tout $g \in J$ le produit hg s'annule sur $\mathbf{V}(I)$. Comme $\mathbf{k}$ est algébriquement clos, d'après le théorème des zéros fort (proposition I.9 p. 32), le produit hg est dans $\sqrt{I}$. Or, I est radical. Ainsi, pour tout $g \in J$ on a : $hg \in I$. Donc $h \in I : J$ et $h(x)$ est nul. Bref, x appartient à $\mathbf{V}(\mathbf{Id_k}(\mathbf{V}(I) \setminus \mathbf{V}(J)))$ c'est à dire à $\overline{\mathbf{V}(I) \setminus \mathbf{V}(J)}$. ◁

Proposition I.18 *Pour deux variétés V et W de $\mathbf{A}^n(\mathbf{K})$ affines sur $\mathbf{k}$, on a :*

$$\mathbf{Id_k}(V) : \mathbf{Id_k}(W) = \mathbf{Id_k}(V \setminus W)$$

Preuve. ▷ Soient donc V et W deux variétés de $\mathbf{A}^n(\mathbf{K})$ affines sur $\mathbf{k}$. En appliquant l'inclusion du théorème I.8 aux idéaux de $\mathbf{P}_n$ engendrés par ces variétés, il vient :

$$\mathbf{Id_k}(V) : \mathbf{Id_k}(W) \subseteq \mathbf{Id_k}(\mathbf{V}(\mathbf{Id_k}(V)) \setminus \mathbf{V}(\mathbf{Id_k}(W)))$$

En utilisant le point (iii) de la proposition I.1, on en tire

$$\mathbf{Id_k}(V) : \mathbf{Id_k}(W) \subseteq \mathbf{Id_k}(V \setminus W)$$

Inversement, soit $f \in \mathbf{Id_k}(V \setminus W)$. Pour tout $g \in \mathbf{Id_k}(W)$, on a $fg \in \mathbf{Id_k}(V)$. D'où l'inclusion inverse. ◁

Proposition I.19 *Soient $h_1, \ldots, h_s, g$ des polynômes dans $\mathbf{P}_n$ et I un idéal de cet anneau. Alors, on a :*

$$I \cap \langle g \rangle = \langle h_1, \ldots, h_s \rangle \implies I : \langle g \rangle = \langle h_1/g, \ldots, h_s/g \rangle$$

Preuve. ▷ Soient donc $h_1, \ldots, h_s, g \in \mathbf{P}_n$ et I un idéal de $\mathbf{P}_n$ tel que : $I \cap \langle g \rangle = \langle h_1, \ldots, h_s \rangle$. Notons que chaque h_i est bien multiple de g puisqu'élément de $\langle g \rangle$. Soit $f \in \langle h_1/g, \ldots, h_s/g \rangle$. Pour tout $p \in \langle g \rangle$ on a : $fp \in \langle h_1, \ldots, h_s \rangle$, et donc $fp \in I$. D'où : $f \in I : \langle g \rangle$. Inversement, si $f \in I : \langle g \rangle$, alors $fg \in I \cap \langle g \rangle$. Il existe donc $p_1, \ldots, p_s \in \mathbf{P}_n$ tel que : $fg = \sum_i p_i h_i$. D'où : $f = \sum_i p_i \, h_i/g$. ◁

Remarque I.15 Soient I et J des idéaux de $\mathbf{P}_n$. La proposition I.19, combinée avec la propriété (vi) de la proposition I.17 p. 38, conduit à un algorithme pour calculer une base de Gröbner de $I : J$. Par suite, à l'aide de la proposition I.18 p. 39 on en déduit une base de Gröbner de l'idéal associé à la différence de deux variétés affines.

2.4 Saturé d'un idéal

Dans cette section, nous rappelons la notion de saturé d'un idéal I par un élément f, et même plus généralement, par une partie multiplicativement stable S (définition I.13).

Nous nous plaçons pour cela dans le contexte général d'un anneau commutatif unitaire $\mathbf{A}$. Lorsque $\mathbf{A}$ est nœthérien, nous exprimons une décomposition primaire du saturé de I par S à partir d'une décomposition primaire de I (proposition I.9) et les liens entre les composants isolés d'un idéal (notation I.5) et ses saturés (théorème I.10 et théorème I.11). Revenant à notre anneau de polynômes $\mathbf{P}_n$, nous rappelons une formule (proposition I.27) fournissant un algorithme pour calculer une base de Gröbner du saturé de I par un élément f (voir aussi [GTZ88] pour cette question). Nous terminons ce chapitre par une réponse générale au problème de la description de la différence de deux variétés affines évoquée au début de cette partie (théorème I.12). Ce résultat, probablement connu sous une autre forme, généralise une proposition de [Laz91a]. Des exemples détaillés p. 44 et suivantes illustrent les notions abordées.

Définition I.13 *Soit $\mathbf{A}$ un anneau commutatif unitaire, I un idéal de $\mathbf{A}$ et S une partie multiplicativement stable de $\mathbf{A}$. On appelle* saturé de I par S *la partie de $\mathbf{A}$ dénotée ici par $S^{-1}I$ et définie par :*

$$S^{-1}I = \{a \in \mathbf{A} \ \mid \ (\exists s \in S) \ \mid \ sa \in I\}$$

Soit $h \in \mathbf{A}$. On appelle saturé de I par h *la partie de $\mathbf{A}$ dénotée ici par $I : h^\infty$ et définie par :*

$$I : h^\infty \ = \ \{a \in \mathbf{A} \mid (\exists m \in \mathbb{N}) \ h^m a \in I\}$$

Remarque I.16 Il serait plus correcte de dénoter par $S^{-1}I \cap \mathbf{A}$ le saturé de l'idéal I par la partie multiplicativement stable S, comme le font certains auteurs. D'autant plus que nous utiliserons plus loin des modules de fractions (notation III.7 p. 91). Cependant, la notation $S^{-1}I \cap \mathbf{A}$ peut devenir relativement encombrante dans certaines formules et rendre obscures ces dernières. Comme nous n'utiliserons pas les saturés d idéaux et les modules de fractions dans les mêmes contextes, aucune confusion n'est à craindre. Avec les notations précédentes, la partie $S^{-1}I$ de $\mathbf{A}$ est donc un idéal de cet anneau. De plus, pour $h \in \mathbf{A}$ on a : $S^{-1}I = I : h^\infty$ où S est la partie multiplicativement stable de $\mathbf{A}$ engendré par h.

Proposition I.20 *Soit $\mathbf{A}$ un anneau commutatif unitaire nœthérien, I un idéal de $\mathbf{A}$ et h un élément de $\mathbf{A}$. Alors on a :*

$$(\exists N \in \mathbb{N}) \ I : h^N \ = \ I : h^\infty$$

De plus :

$$I = \sqrt{I} \quad \Longrightarrow \quad I : h \ = \ I : h^\infty$$

Preuve. $\triangleright$ Pour tout $n \in \mathbb{N}$ on a $I : h^n \subseteq I : h^{n+1}$. Comme $\mathbf{A}$ est nœthérien, la suite des idéaux $(I : h^n)_{n\in\mathbb{N}}$ est stationnaire à partir d'un rang N. Comme $I : h^\infty$ est la réunion des termes de cette suite, il vient $I : h^N \ = \ I : h^\infty$. Supposons maintenant que I soit radical. On a bien sur $I : h \ \subseteq I : h^\infty$. Inversement, soit $f \in I : h^\infty$. Alors, il existe $m \in \mathbb{N}$ tel que $fh^m \in I$. Par suite, $(fh)^m \in I$. Comme I est radical, on a $fh \in I$ c'est-à-dire $f \in I : h$. $\triangleleft$

Proposition I.21 *Soit $\mathbf{A}$ un anneau commutatif et unitaire. Soit $\mathcal{Q}$ un idéal primaire de $\mathbf{A}$ et $\mathcal{P}$ son radical. Soit h un élément de $\mathbf{A}$. Alors on a :*

(i) $h \in \mathcal{P} \ \Rightarrow \ \mathcal{Q} : h^\infty = \mathbf{A}$

(ii) $h \notin \mathcal{P} \ \Rightarrow \ \mathcal{Q} : h^\infty = \mathcal{Q}$

Preuve. ▷ Supposons tout d'abord que $h \in \mathcal{P}$. Il existe alors $m \in \mathbb{N}$ tel que $h^m \in \mathcal{Q}$. Par conséquent pour tout $a \in \mathbf{A}$, on a $h^m a \in \mathcal{Q}$, c'est-à-dire $a \in \mathcal{Q} : h^\infty$, ce qui montre (i).

Considérons maintenant un élément h de $\mathbf{A}$ tel que $h \notin \mathcal{P}$. On a clairement : $\mathcal{Q} \subseteq \mathcal{Q} : h^\infty$. Pour montrer l'inclusion réciproque on raisonne par l'absurde. On suppose donc qu'il existe un élément $a \in \mathcal{Q} : h^\infty$ tel que $a \notin \mathcal{Q}$. Il existe alors $m \in \mathbb{N}$ tel que $h^m a \in \mathcal{Q}$. Puisque a n'appartient pas à $\mathcal{Q}$ qui est primaire, il existe une puissance de h^m qui appartient à $\mathcal{Q}$. On a alors $h \in \mathcal{P}$. Une contradiction. Ceci prouve (ii). ◁

Proposition I.22 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire,* S *une partie multiplicativement stable de* $\mathbf{A}$ *et* I*,* J *deux idéaux de* $\mathbf{A}$*. Alors on a :*

$$S^{-1}(I \cap J) \quad = \quad (S^{-1}I) \cap (S^{-1}J)$$

Preuve. ▷ Soit $a \in \mathbf{A}$. Si $a \in S^{-1}(I \cap J)$ alors il existe $s \in S$ tel que $as \in I \cap J$ donc tel que $as \in I$ et $as \in J$. Par suite $a \in (S^{-1}I) \cap (S^{-1}J)$. Inversement, si $a \in (S^{-1}I) \cap (S^{-1}J)$ alors il existe $s_I, s_J \in S$ tel que $as_I \in I$ et $as_J \in J$. Par suite $as_I s_J \in J \cap J$. La partie S étant multiplicativement stable, on a bien $a \in S^{-1}(I \cap J)$. ◁

Théorème I.9 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire nœthérien et* I *un idéal de* $\mathbf{A}$*. Soit* $\Pi = (\mathcal{Q}_1, \ldots, \mathcal{Q}_l)$ *une décomposition primaire de* I*. Soit* h *un élément de* $\mathbf{A}$*. Alors on a :*

$$I : h^\infty \quad = \bigcap_{\{\mathcal{Q} \in \Pi \,|\, h \,\notin \sqrt{\mathcal{Q}}\}} \mathcal{Q}$$

Plus généralement, si S *est une partie multiplicativement stable, une décomposition primaire de* $S^{-1}I$ *s'obtient en ne conservant que les* $\mathcal{Q} \in \Pi$ *dont le radical ne rencontre pas* S*. On a donc :*

$$S^{-1}I \quad = \bigcap_{\{\mathcal{Q} \in \Pi \,|\, S \cap \sqrt{\mathcal{Q}} = \emptyset\}} \mathcal{Q}$$

Preuve. ▷ Le cas de $I : h^\infty$ résulte des propositions I.21 et I.22. Traitons maintenant le cas de $S^{-1}I$. Supposons que $\mathcal{Q}_1, \ldots, \mathcal{Q}_l$ soient numérotées de telle sorte qu'il existe un entier $m \in \{0, \ldots, l\}$ tel que pour tout entier $i \in \{1, \ldots, m\}$ on ait $\sqrt{\mathcal{Q}_i} \cap S = \emptyset \iff i \leq m$. Si $m = 0$, autrement dit si chaque $\sqrt{\mathcal{Q}_i}$ rencontre S, alors pour chaque $i \in \{1, \ldots, l\}$ il existe $s_i \in S$ tel que $s_i \in \sqrt{\mathcal{Q}_i}$. Il existe donc une puissance ${s_i}^{e_i}$ de s_i appartenant à $\mathcal{Q}_i$. Le produit s des ${s_i}^{e_i}$ appartient à la fois à S, puisque S est multiplicativement stable, et à I, puisque I est l'intersection des $\mathcal{Q}_i$. Le produit s *conduit* donc tout élément $a \in \mathbf{A}$ dans I. Par suite $S^{-1}I = \mathbf{A}$ et la formule annoncée est correcte pour $m = 0$. Supposons maintenant $m > 0$. En raisonnant comme précédemment, on a :

$$S^{-1}(\cap_{m+1 \leq i \leq l}\, \mathcal{Q}_i) \quad = \quad \mathbf{A}$$

Par suite, à l'aide de la proposition I.22 il vient :

$$S^{-1}I \quad = \quad \cap_{1 \leq i \leq m}\, (S^{-1}\mathcal{Q}_i)$$

Soit $i \in \{1, \ldots, m\}$. Il suffit de vérifier que l'on a $S^{-1}\mathcal{Q}_i = \mathcal{Q}_i$. On raisonne comme dans la preuve de la proposition I.21. On remarque tout d'abord que l'on a clairement $\mathcal{Q}_i \subseteq S^{-1}\mathcal{Q}_i$.

Inversement, soit $a \in S^{-1}\mathcal{Q}_i$ et $s \in S$ tel que $as \in \mathcal{Q}_i$. Supposons que a n'appartienne pas à $\mathcal{Q}_i$. Comme $\mathcal{Q}_i$ est une primaire, une puissance de s appartient alors à $\mathcal{Q}_i$. Autrement dit s appartient à $\sqrt{\mathcal{Q}_i}$, Or, par hypothèse, S ne rencontre pas $\sqrt{\mathcal{Q}_i}$. $\lhd$

Proposition I.23 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire nœthérien,* I *un idéal de* $\mathbf{A}$ *et* S *une partie multiplicativement stable de* $\mathbf{A}$*. Alors on a :*

$$S \cap \sqrt{I} \neq \emptyset \implies S^{-1}I = \mathbf{A}$$

Preuve. $\rhd$ Résulte immédiatement du théorème I.9. $\lhd$

Théorème I.10 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire nœthérien et* I *un idéal de* $\mathbf{A}$*. Alors, les assertions suivantes sont équivalentes :*

(*i*) I *est primaire*

(*ii*) $(\forall h \in \mathbf{A})\ \ (I : h^\infty = I)$ ou $(I : h^\infty = \mathbf{A})$

Preuve. $\rhd$ Nous avons déjà vu l'implication $(i) \implies (ii)$ avec la proposition I.21. Réciproquement, supposons que pour tout élément h de $\mathbf{A}$ le saturé $I : h^\infty$ de I par h soit I ou $\mathbf{A}$. Vérifions que I est primaire. Soit a et b deux éléments de $\mathbf{A}$ dont le produit est dans I. D'après le point (ii) de la proposition I.17 p. 38, il vient $a \in I : b$. Par suite $a \in I : b^\infty$. Si $I : b^\infty = I$, une puissance de a (en fait a lui-même) est dans I. Supposons maintenant $I : b^\infty = \mathbf{A}$. Puisque $\mathbf{A}$ est nœthérien, d'après la proposition I.20, il existe $N \in \mathbb{N}$ tel que $I : b^N = \mathbf{A}$. Avec le point (iii) de la proposition I.17, il vient alors $b^N \in I$. $\lhd$

Proposition I.24 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire nœthérien,* I *un idéal de* $\mathbf{A}$ *et* S *une partie multiplicativement stable de* $\mathbf{A}$*. Alors on a :*

$$\sqrt{S^{-1}I} = S^{-1}\sqrt{I}$$

Preuve. $\rhd$ D'après le théorème I.9 on a :

$$\sqrt{S^{-1}I} = \bigcap_{\{\mathcal{P} \in \mathsf{ap}(I)\ \mid\ S \cap \mathcal{P} = \emptyset\}} \mathcal{P}$$

Comme $\mathsf{ap}(I) = \mathsf{ap}(\sqrt{I})$, on a immédiatement l'égalité annoncée. $\lhd$

Notations I.5 Soit $\mathbf{A}$ un anneau commutatif unitaire nœthérien et I un idéal de $\mathbf{A}$. Soit $\Pi = (\mathcal{Q}_1, \ldots, \mathcal{Q}_l)$ une décomposition primaire minimale de I. On rappelle qu'un idéal premier $\mathcal{P} \in \mathcal{V}(I)$ (i.e. contenant I) est dit :

(*i*) *associé à* I s'il existe $\mathcal{Q} \in \Pi$ tel que $\mathcal{P} = \sqrt{\mathcal{Q}}$,

(*ii*) *minimal de* I si $(\forall \mathcal{P}' \in \mathcal{V}(I))\ \ I \subseteq \mathcal{P}' \subseteq \mathcal{P} \implies \mathcal{P}' = \mathcal{P}$.

On désignera dans la suite par $\mathsf{ap}(I)$ (resp. $\mathsf{mp}(I)$) l'ensemble des idéaux premiers associés à I (resp. minimaux de I). On rappelle (voir [van91]) que l'on a :

$$|\,\mathsf{mp}(I)\,| < \infty \quad \text{et} \quad \mathsf{mp}(I) \subseteq \mathsf{ap}(I)$$

On rappelle qu'un idéal primaire $\mathcal{Q} \in \Pi$ est dit :

(1) *composante immergée de* I s'il existe $\mathcal{P} \in \mathsf{ap}(I)$ tel que $\mathcal{P} \subseteq \sqrt{\mathcal{Q}}$ et $\mathcal{P} \neq \sqrt{\mathcal{Q}}$.

(2) *composante isolée de* I si elle n'est pas une composante immergée de I.

Une sous-famille de $\mathcal{C} \subseteq \Pi$ est dite *composant isolé de* I si :

$$(\forall \mathcal{Q} \in \mathcal{C})\ (\forall \mathcal{Q}' \in \Pi \setminus \mathcal{C})\ \sqrt{\mathcal{Q}'} \not\subseteq \sqrt{\mathcal{Q}}$$

Pour un composant isolé $\mathcal{C} \subseteq \Pi$, par abus de langage, l'intersection des éléments de $\mathcal{C}$ est également appelée *composant isolé de* I.

Proposition I.25 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire nœthérien,* I *un idéal de* $\mathbf{A}$ *et* S *une partie multiplicativement stable de* $\mathbf{A}$*. Alors* $S^{-1}I$ *est un composant isolé de* I*.*

Preuve. $\triangleright$ Soit $\Pi = (\mathcal{Q}_1, \ldots, \mathcal{Q}_l)$ une décomposition primaire de I. D'après le théorème I.9, le saturé $S^{-1}I$ de I par S est l'intersection des idéaux primaires $\mathcal{Q}_i$ qui ne rencontrent pas S. Il est clair qu'une composante primaire de Π rencontrant S ne peut être contenu dans un des $\mathcal{Q}_i$ précédents. Par suite, $S^{-1}I$ est bien un composant isolé de I. $\triangleleft$

Proposition I.26 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire nœthérien,* I *un idéal de* $\mathbf{A}$ *et* $\mathcal{Q}_0$ *une composante primaire de* I*. Alors on a :*

$$\mathcal{Q}_0 \text{ isolée} \iff \left(\mathbf{A} \setminus \sqrt{\mathcal{Q}_0}\right)^{-1} I = \mathcal{Q}_0$$

Preuve. $\triangleright$ Soit Π une décomposition primaire minimale de I. On pose $S = \mathbf{A}\setminus\sqrt{\mathcal{Q}_0}$. On vérifie immédiatement que S est multiplicativement stable. Nous avons les équivalences suivantes :

$$\begin{array}{rcl} \mathcal{Q}_0 \text{ isolée} & \iff & (\forall \mathcal{P} \in \mathsf{ap}(I))\ \ \mathcal{P} \neq \sqrt{\mathcal{Q}_0} \implies \mathcal{P} \not\subseteq \sqrt{\mathcal{Q}_0} \\ & \iff & (\forall \mathcal{P} \in \mathsf{ap}(I))\ \ \mathcal{P} \neq \sqrt{\mathcal{Q}_0} \implies \mathcal{P} \cap S \neq \emptyset \\ & \iff & \{\mathcal{Q} \in \Pi \mid \sqrt{\mathcal{Q}} \cap S = \emptyset\} = \{\mathcal{Q}_0\} \\ & \iff & S^{-1}I = \mathcal{Q}_0 \end{array}$$

à l'aide du théorème I.9. $\triangleleft$

Théorème I.11 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire nœthérien,* I *un idéal de* $\mathbf{A}$ *et* Π *une décomposition primaire minimale de* I*. Soit* $\mathcal{C} \subseteq \Pi$*. Alors les propositions suivantes sont équivalentes :*

(i) $\mathcal{C}$ *est composant isolé de* I

(ii) *il existe une partie* S *multiplicativement stable de* $\mathbf{A}$ *telle que* $S^{-1}I = \mathcal{C}$*.*

Preuve. $\triangleright$ La proposition I.25 précédente fournit l'implication $(ii) \implies (i)$. La preuve de la réciproque est analogue à la preuve de la proposition I.26. Supposons donc que $\mathcal{C}$ soit un composant isolé de I. On pose alors :

$$S \;=\; \cap_{\mathcal{Q}\in\mathcal{C}}(\mathbf{A} \setminus \sqrt{\mathcal{Q}})$$

On vérifie immédiatement que S est multiplicativement stable. Rappelons maintenant le *lemme d'évitement* suivant : si un idéal premier est contenu dans une réunion finie d'idéaux premiers, alors il est contenu dans l'un deux. Par suite, on a les équivalences suivantes :

$$\begin{array}{rcl} \mathcal{C} \text{ isolée} & \iff & (\forall \mathcal{Q} \in \mathcal{C})\ (\forall \mathcal{Q}' \in \Pi \setminus \mathcal{C})\ \sqrt{\mathcal{Q}'} \not\subseteq \sqrt{\mathcal{Q}} \\ & \iff & (\forall \mathcal{Q}' \in \Pi \setminus \mathcal{C})\ \sqrt{\mathcal{Q}'} \not\subseteq \cup_{\mathcal{Q}\in\mathcal{C}} \sqrt{\mathcal{Q}} \\ & \iff & (\forall \mathcal{Q}' \in \Pi \setminus \mathcal{C})\ \sqrt{\mathcal{Q}'} \cap S \neq \emptyset \\ & \iff & \{\mathcal{Q} \in \Pi \mid \sqrt{\mathcal{Q}} \cap S = \emptyset\} = \mathcal{C} \\ & \iff & S^{-1}I = \mathcal{C} \end{array}$$

à l'aide du théorème I.9. $\triangleleft$

Proposition I.27 *Soit $I = \langle f_1, \ldots, f_s \rangle$ un idéal de $\mathbf{P}_n$, $h \in \mathbf{P}_n$ et y une nouvelle variable. Alors, on a :*

$$I : h^\infty \;=\; \langle f_1, \ldots, f_s, 1 - yh \rangle \cap \mathbf{P}_n$$

Preuve. $\triangleright$ La preuve de l'égalité annoncée se fait en utilisant des techniques déjà éprouvées dans la théorème I.6 et le théorème I.5. Soit tout d'abord $g \in I : h^\infty$. Il existe donc $m \in \mathbb{N}$ tel que : $h^m g \in I$, et, par suite, il existe des polynômes $p_1, \ldots, p_s$ de $\mathbf{P}_n$ tel que : $h^m g = \sum_i p_i f_i$. Or, on a :

$$\begin{aligned} g &= y^m h^m g + (1 - y^m h^m) g \\ &= \textstyle\sum_i p_i f_i y^m + (1 - yh)(1 + yh + \cdots + y^{m-1} h^{m-1}) g \end{aligned}$$

Comme $g \in \mathbf{P}_n$, on a bien : $g \in \langle f_1, \ldots, f_s, 1 - yh \rangle \cap \mathbf{P}_n$. Inversement, supposons que g appartienne à ce dernier idéal. Alors, il existe des polynômes $q_1, \ldots, q_s, q_0 \in \mathbf{k}[x_1, \ldots, x_n, y]$ tels que $g = \sum_i q_i f_i + q_0(1 - yh)$. En remplaçant y par $1/h$, puis en multipliant cette égalité par h^m où m est le degré maximum en y des polynômes q_i, on obtient $h^m g \in I$, c'est à dire $g \in I : h^\infty$. $\triangleleft$

Remarque I.17 Avec la proposition I.27 précédente, il vient :

$$h \in \sqrt{I} \iff I : h^\infty = \mathbf{P}_n$$

à l'aide du théorème I.6. Ce dernier résultat se déduit également de la proposition I.23.

Théorème I.12 *Soient $I, J \subseteq \mathbf{P}_n$ deux idéaux. Soit G un système de générateurs de J et h leur produit. Le corps $\mathbf{K}$ sur lequel sont prises les variétés étant algébriquement clos, on a :*

$$\mathbf{V}(I : h^\infty) \;=\; \overline{\mathbf{V}(I) \setminus \mathbf{V}(J)}$$

Preuve. $\triangleright$ On a clairement $\overline{\mathbf{V}(I) \setminus \mathbf{V}(J)} = \overline{\mathbf{V}(\sqrt{I}) \setminus \mathbf{V}(h)}$. l'aide du théorème I.8, on tire $\overline{\mathbf{V}(I) \setminus \mathbf{V}(J)} = \mathbf{V}(\sqrt{I} : h)$. Or, d'après la proposition I.20, on a $\mathbf{V}(\sqrt{I} : h) = \mathbf{V}(\sqrt{I} : h^\infty)$. Enfin, avec la proposition I.24, il vient $\mathbf{V}(\sqrt{I} : h^\infty) = \mathbf{V}(\sqrt{I : h^\infty}) = \mathbf{V}(I : h^\infty)$. $\triangleleft$

Exemple I.4 Nous reprenons notre exemple favori (où le corps de base est $\mathbb{Q}$ le corps des nombres rationnels) c'est-à-dire

$$I = \langle x^2 + y + z - 1, x + y^2 + z - 1, x + y + z^2 - 1 \rangle$$

Rappelons qu'une base de Gröbner de I pour l'ordre lexicographique avec $z < y < x$ est $G = \{x + y + z^2 - 1, y^2 - y - z^2 + z, 2z^2 y + z^4 - z^2, z^6 - 4z^4 + 4z^3 - z^2\}$. La décomposition primaire minimale de I est $\{\mathcal{Q}_1, \ldots, \mathcal{Q}_4\}$ avec :

$$\begin{aligned} \mathcal{Q}_1 &= \{-x + z,\ -y + z,\ z^2 + 2\ z - 1\} \\ \mathcal{Q}_2 &= \{x + z - 1,\ -y + z,\ z^2\} \\ \mathcal{Q}_3 &= \{-x + z,\ y + z - 1,\ z^2\} \\ \mathcal{Q}_4 &= \{x + z - 1,\ y + z - 1,\ z^2 - 2\ z + 1\} \end{aligned}$$

Considérons la partie multiplicativement stable S_1 engendrée par z. Une base de Gröbner (toujours pour le même ordre) de ${S_1}^{-1}I$ est :

$${S_1}^{-1}I = \left\{2\ x + z^2 - 1,\ 2\ y + z^2 - 1,\ z^4 - 4\ z^2 + 4\ z - 1\right\}$$

On vérifie que la décomposition primaire minimale de ${S_1}^{-1}I$ est $\{\mathcal{Q}_1, \mathcal{Q}_4\}$. Ainsi, la *saturation* par S_1 de I a *fait grossir* cet idéal en éliminant ces composantes primaires qui rencontraient S_1. Ou, du point de vue des variétés affines, on a éliminé les solutions pour lesquelles on avait $z = 0$.

Considérons la partie multiplicativement stable S_2 engendrée par le polynôme $h = (z-1)\ z\ (z^2 + 2\ z - 1)$. Notons que le polynôme g_4 univarié en z de la base lexicographique de I ci-dessus se factorise en irréductibles comme suit :

$$g_4 = (z-1)^2\ z^2\ \left(z^2 + 2\ z - 1\right)$$

Ainsi h est en fait la partie sans carrés de g_4. Par suite, $h \in \sqrt{\langle g_4 \rangle}$ et donc $h \in \sqrt{I}$. l'aide de la proposition I.23 on en tire immédiatement ${S_2}^{-1}I = \mathbf{P}_3$.

Exemple I.5 Avec pour corps de base $\mathbf{k} = \mathbb{Q}$, on considère l'idéal I suivant donné par sa base lexicographique (avec $y < x$) :

$$I = \langle (y^2-2)\ (x^2 - 2\ y\ x + 3),\ (y^2-3)\ (y^2-2) \rangle$$

Une décomposition primaire minimale de I est $\{\mathcal{Q}_1, \mathcal{Q}_2\}$ avec :

$$\mathcal{Q}_1 = \left\{y^2 - 2\right\} \quad \text{et} \quad \mathcal{Q}_2 = \left\{y^2 - 3,\ x^2 - 2\ y\ x + 3\right\}$$

On vérifie :

$$h = y^2 - 2 \quad \Longrightarrow \quad I : h^\infty = \mathcal{Q}_2$$

De même :

$$h = y^2 - 3 \quad \Longrightarrow \quad I : h^\infty = \mathcal{Q}_1$$

Notons que dans les deux cas on a exactement $\mathbf{V}(I : h^\infty) = \mathbf{V}(I) \setminus \mathbf{V}(h)$. On considère maintenant $h = y - x$. Comme $h \in \sqrt{\mathcal{Q}_2}$ et $h \notin \sqrt{\mathcal{Q}_1}$ on obtient naturellement $I : h^\infty = \mathcal{Q}_1$. Cependant, dans ce cas $\mathbf{V}(I : h^\infty) \neq \mathbf{V}(I) \setminus \mathbf{V}(h)$. En effet, $(\sqrt{2}, \sqrt{2}) \in \mathbf{V}(I : h^\infty)$ mais $(\sqrt{2}, \sqrt{2}) \notin \mathbf{V}(I) \setminus \mathbf{V}(h)$. Par suite, $\mathbf{V}(I) \setminus \mathbf{V}(h)$ n'est pas ici une variété affine et son passage à la fermeture lui rajoute le point manquant.

Chapitre II

Scindages d'idéaux et pgcd de polynômes univariés

Introduction

Reprenons l'exemple de l'introduction du premier chapitre, à savoir le système :

$$\left\{\begin{array}{rcl} x^2+y+z-1 & = & 0 \\ x+y^2+z-1 & = & 0 \\ x+y+z^2-1 & = & 0 \end{array}\right.$$

Nous avons vu qu'une base de Gröbner pour l'ordre lexicographique induit par l'ordre $z < y < x$ sur les variables des polynômes ci-dessus est : $\{g_1, g_2, g_3, g_4\}$ où :

$$\begin{array}{rcl} g_1 & = & x+y+z^2-1 \\ g_2 & = & y^2-y-z^2+z \\ g_3 & = & 2z^2y+z^4-z^2 \\ g_4 & = & z^6-4z^4+4z^3-z^2 \end{array}$$

Continuons la résolution du système initial à partir de la base de Gröbner précédente. On cherche à ne plus avoir qu'un seul polynôme dont la variable principale est y. Sous réserve que z^2 (et par suite z) ne soit pas nul, l'équation $g_3 = 0$ fournit une expression de y, à savoir $y = \frac{1-z^2}{2}$. Par conséquent, on doit donc distinguer les cas $z = 0$ et $z \neq 0$. Comme nous l'avons vu dans l'introduction du premier chapitre, ceci conduit à réécrire le système

$$g_1 = g_2 = g_3 = g_4 = 0$$

comme suit :

$$\left\{\begin{array}{rcl} x+y-1 & = & 0 \\ y^2-y+z & = & 0 \\ z^2 & = & 0 \end{array}\right. \quad \text{ou} \quad \left\{\begin{array}{rl} x+y+z^2-1 = & 0 \\ y^2-y-z^2+z = & 0 \\ 2y+z^2-1 = & 0 \\ z^4-4z^2+4z-1 \;= & 0 \end{array}\right.$$

Le premier système ainsi obtenu permet d'obtenir facilement les triplets (z, y, x) solutions. Qu'en est-il du second ? Injectons dans sa deuxième équation l'expression $y = \frac{1-z^2}{2}$. On obtient :

$$(\frac{1-z^2}{2})^2 - \frac{1-z^2}{2} - z^2 + z = 0$$

C'est-à-dire :

$$1 - 2z^2 + z^4 - 2 + 2z^2 - 4z^2 + 4z = 0$$

ce qui est en fait la dernière équation de ce second système. Autrement dit, il peut simplement s'écrire :

$$\left\{\begin{array}{rcl} x + y + z^2 - 1 = & 0 \\ 2y + z^2 - 1 = & 0 \\ z^4 - 4z^2 + 4z - 1 & = & 0 \end{array}\right.$$

De ce système on déduit facilement les triplets (z, y, x) solutions.

Cet exemple amène au moins deux questions ?

1. comment ce type de *distinction de cas* peut-il être généré automatiquement par un algorithme ?
2. comment formaliser mathématiquement ce type de *distinction de cas* ?

Une réponse à la première question est donnée par Daniel Lazard et apparaît dans [DDD85]. Il s'agit de la remarque suivante. Soit $\mathbf{k}$ un corps et p un polynôme de $\mathbf{k}[x]$ de degré positif et sans carrés (mais non nécessairement irréductible). Soit $\mathbf{A}$ l'extension simple séparable de $\mathbf{k}$ par p, c'est-à-dire l'anneau $\mathbf{k}[x]/\langle p\rangle$. Soit $\alpha \in \mathbf{A}$. L'élément α est-il une unité de $\mathbf{A}$? Soit $q \in \mathbf{k}[x]$ un représentant de α et g le pgcd des polynômes p et q. Puisque $\mathbf{k}[x]$ est principal, il existe u et v dans $\mathbf{k}[x]$ tel que $up + vq = g$. Si $g = 1$, alors α est inversible dans $\mathbf{A}$ et un représentant de son inverse est u. Si g est de degré non nul, alors il existe $f \in \mathbf{k}[x]$, également de degré positif, et tel que $p = fg$. Soit ζ une racine de p. Comme p est sans facteurs carrés, ou bien $f(\zeta) = 0$ et $g(\zeta) \neq 0$, ou bien $f(\zeta) \neq 0$ et $g(\zeta) = 0$. Ainsi, ou bien $f(\zeta) = 0$ et q est inversible dans $\mathbf{k}[x]/\langle f\rangle$, ou bien $g(\zeta) = 0$ et q est nul dans $\mathbf{k}[x]/\langle g\rangle$. Comme $\mathbf{k}[x]/\langle p\rangle$ est isomorphe au produit (direct) des anneaux $\mathbf{k}[x]/\langle f\rangle$ et $\mathbf{k}[x]/\langle g\rangle$ (définition II.1, proposition II.7) on s'aperçoit qu'en *passant dans une structure-produit* on a pu répondre à notre question sur *chacun des facteurs*.

Cette idée a depuis largement été développée, notamment sur un plan pratique tout du moins, par Dominique Duval et Teresa Gomez Diaz en particulier ([Duv87, Duv94, Gom92]). Elle a débouché sur des logiciels sophistiqués ([Duv87, Gom92]) générant automatiquement *ces factorisations* et produisant ainsi des résultats sous forme de distinctions de cas. De plus, le logiciel de Teresa Gomez Diaz ([Gom92]) permet de résoudre de façon élégante de nombreux problèmes dont le cadre va bien au-delà de la résolution de systèmes d'équations algébriques.

La plupart des algorithmes présentés dans cette thèse sont fondés sur l'idée de Daniel Lazard, exposée plus haut. Cependant, notre approche diffère de celle de Dominique Duval et Teresa Gomez Diaz sur au moins un point : la formalisation mathématique de la notion de distinction de cas. Pour ces auteurs, le calcul algébrique à effectuer et les éventuelles distinctions de cas qu'il va générer, sont deux mondes à part. Dominique Duval a du reste replacé ce second aspect dans une théorie en dehors de l'algèbre commutative, *la théorie des esquisses* ([DR94a, DR94b]).

Nous développons dans ce chapitre un autre point de vue : nous traitons les distinctions de cas dans un calcul algébrique avec des outils de l'algèbre commutative. En effet, le fait que l'on puisse travailler dans une structure-produit fait partie de la nature algébrique du problème. De plus, comme nous le verrons dans le chapitre 5, la gestion efficace des distinctions de cas est un problème algébrique.

Le but de ce chapitre est de décrire le cadre algébrique que nos algorithmes utilisent lorsqu'ils génèrent des distinctions de cas. Voici le plan de ce chapitre. Dans la première section, nous rappelons la notion de *produit d'anneaux* et donnons quelques propositions et théorèmes sur le calcul dans ce type de structure. Dans la seconde section, nous introduisons et étudions la notion de *scindage d'un idéal*. Il s'agit d'une formalisation possible de ces *distinctions de cas* lorsque l'on calcule modulo un idéal. Dans la troisième section, nous étudions une notion de *pgcd de polynômes univariés à coefficients dans un anneau quelconque*. Sur cette notion reposera les méthodes de résolution de systèmes d'équations algébriques que nous traiterons dans les chapitres suivants. Comme dans l'exemple ci-dessus, le calcul de ces pgcd, qui se fait généralement modulo un idéal, peut générer le scindage de cet idéal ; on travaille alors dans une structure-produit. Ceci explique le lien entre les trois notions présentées dans ce chapitre. Le calcul de ces pgcd sera fondé sur une propriété (prop. II.26) de l'algorithme des sous-résultants apparaissant dans [MR95].

1 Produit d'anneaux

Après avoir précisé quelques notations (notations II.1 et II.2) nous revoyons la notion de produit d'anneaux (définition II.1, propositions II.1 et II.2). On rappelle alors une caractérisation classique de la structure de produit d'anneaux à l'aide des idéaux (proposition II.3). La preuve de celle-ci fait ressortir l'intérêt d'un autre concept que nous révisons aussi (définition II.2 et proposition II.4) : celui d'idempotent. À l'aide des idempotents on obtient une autre caractérisation de la structure de produit d'anneaux (proposition II.5). On donne aussi deux autres caractérisations en termes d'idéaux correspondant à ce que l'on rencontrera en pratique, c'est-à-dire dans nos algorithmes (proposition II.6 et proposition II.7). On rappelle ensuite comment tout anneau nœthérien se factorise en un produit (fini) d'anneaux nœthériens indécomposables (théorème II.1).

Nous passons en suite à des résultats *non classiques* et directement liés à nos préoccupations algorithmiques. Si certains de ces résultats sont probablement connus sous d'autres formes, nous ne les avons pas rencontrés dans notre littérature. Le problème central est le suivant. En pratique, si nos anneaux possèdent une structure de produit d'anneaux, celle-ci n'est pas explicitement donnée. Comment peut-on alors découvrir une telle factorisation ? La première idée qui vient à l'esprit est de rechercher des diviseurs de zéro. Ceci n'est pas suffisant comme l'illustre la remarque suivante. Soit $\mathbf{A}$ un anneau commutatif unitaire, a et b deux éléments non nuls et non inversibles de $\mathbf{A}$ et I l'idéal de $\mathbf{A}$ engendré par leur produit. L'anneau $\mathbf{A}/I$ n'est en général pas isomorphe au produit des anneaux $\mathbf{A}/\langle a\rangle$ et $\mathbf{A}/\langle b\rangle$ sauf si $\langle a,b\rangle = \mathbf{A}$. Prendre par exemple $\mathbf{A} = \mathbf{k}[x]$ où $\mathbf{k}$ est un corps et $a = b = x$. Dans ce cas, $\mathbf{A}/I$ possède le nilpotent x alors que le produit des anneaux $\mathbf{A}/\langle a\rangle$ et $\mathbf{A}/\langle b\rangle$ est en fait un produit de corps, donc un un anneau réduit (i.e. sans nilpotents) . Notons que si l'on a $a + b = 1$ alors les éléments a et b sont des idempotents. En effet on a $a^2 = a(1 - b) = a$. Inversement si e est un idempotent on a $e(1 - e) = 0$ et $e + (1 - e) = 1$. Bref, Le bon outil pour factoriser un anneau est l'idempotent ! Pour cette raison, nous nous intéresserons aux anneaux réduits où tout élément régulier est inversible (remarque II.3). Dans un tel anneau nœthérien tout diviseur de zéro est un idempotent (proposition II.12). Nous établirons alors le théorème II.2 qui nous a été suggéré par Daniel Lazard : un anneau commutatif nœthérien est réduit et tel que tout élément régulier est inversible si et seulement s'il s'agit d'un produit de corps. Enfin, nous établirons un théorème fondamental (th. II.3) pour la seconde partie (les scindages) réalisant un lien entre le spectre d'un anneau et ses idempotents.

Notations II.1 Soit $\mathbf{A}$ un anneau commutatif unitaire et E une partie non vide de $\mathbf{A}$. On note $\mathbf{Id}_{\mathbf{A}}(E)$ l'idéal de $\mathbf{A}$ engendré par E, $\mathbf{A}/E$ l'anneau-quotient de $\mathbf{A}$ par cet idéal, et $\overline{a}^E$ la classe résiduelle d'un élément a de $\mathbf{A}$ dans $\mathbf{A}/E$. Si E est la partie vide, on pose $\mathbf{A}/E = \mathbf{A}$. On désignera par $|E|$ le cardinal de E. Si $E = \{a_1, \ldots, a_n\}$ on notera $\langle a_1, \ldots, a_n\rangle_{\mathbf{A}}$ (ou $\langle a_1, \ldots, a_n\rangle$ si aucune confusion n'est à craindre) l'idéal $\mathbf{Id}_{\mathbf{A}}(E)$. Notons que si E contient des unités, $\mathbf{Id}_{\mathbf{A}}(E)$ est $\mathbf{A}$. On désignera par $\mathbf{reg}(\mathbf{A})$ l'ensemble multiplicativement stable des éléments réguliers (i.e. qui ne sont pas des diviseurs de zéro) de $\mathbf{A}$ et par $\mathbf{fr}(\mathbf{A})$ l'anneau total des fractions de $\mathbf{A}$ (i.e. l'anneau des fractions à numérateur dans $\mathbf{A}$ et à dénominateur dans $\mathbf{reg}(\mathbf{A})$). Pour $a \in \mathbf{A}$ et $b \in \mathbf{reg}(\mathbf{A})$, s'il existe un élément (nécessairement unique) $c \in \mathbf{A}$ tel que $a = bc$, on notera a exquo b le quotient exact de a par b (i.e. cet élément c). Pour $a \in \mathbf{A}$, on notera $\mathbf{Div}(a)$ l'ensemble des éléments $b \in \mathbf{reg}(\mathbf{A})$ tel qu'il existe $c \in \mathbf{A}$

tel que $a = bc$. Enfin, on désignera par $\mathbf{Un}(\mathbf{A})$ l'ensemble des unités de $\mathbf{A}$, et par $\mathbf{Nil}(\mathbf{A})$ l'ensemble des nilpotents de $\mathbf{A}$.

Définition II.1 *Soient $\mathbf{A}_1, \ldots, \mathbf{A}_n$ des anneaux commutatifs unitaires. On munit le produit cartésien $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$ d'une structure d'anneau, dite* anneau-produit, *(que l'on notera de la même manière) en posant :*

$$(a_1, \ldots, a_n) + (b_1, \ldots, b_n) \;=\; (a_1 + b_1, \ldots, a_n + b_n)$$

et :

$$(a_1, \ldots, a_n)\,(b_1, \ldots, b_n) \;=\; (a_1\, b_1, \ldots, a_n\, b_n)$$

On dira qu'un anneau $\mathbf{A}$ a une structure de produit d'anneaux *s'il existe des anneaux $\mathbf{A}_1, \ldots, \mathbf{A}_n$, distincts de $\mathbf{A}$ et de l'anneau trivial (celui réduit à un élément), avec $n \geq 2$, tel que $\mathbf{A}$ soit isomorphe à l'anneau-produit $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$. On définit de manière analogue la notion de* produit de corps. *Dans les deux cas, il s'agit de produits finis.*

Notations II.2 *Soient $\mathbf{A}_1, \ldots, \mathbf{A}_n$ des anneaux commutatifs unitaires et $\mathbf{A}$ l'anneau-produit $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$. Soit $i \in \{1, \ldots, n\}$ et $a \in \mathbf{A}$. On désignera par $\pi_i(a)$ la composante de a appartenant à $\mathbf{A}_i$. On notera également $\vec{e_i}$ l'élément de $\mathbf{A}$ tel que :*

$$(\forall j \in \{1, \ldots, n\})\; \pi_j(\vec{e_i}) = \delta_{ij} 1_{\mathbf{A}_j}$$

où δ_{ij} est le symbole de Kronecker, c'est-à-dire l'entier appartenant à $\{0, 1\}$ et non nul si et seulement si $i = j$.

Proposition II.1 *Soient $\mathbf{A}_1, \ldots, \mathbf{A}_n$ des anneaux commutatifs unitaires et $\mathbf{A}$ l'anneau-produit $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$. Alors, on a :*

(i) $\mathbf{Un}(\mathbf{A}_1 \times \cdots \times \mathbf{A}_n) \;=\; \mathbf{Un}(\mathbf{A}_1) \times \cdots \times \mathbf{Un}(\mathbf{A}_n)$

(ii) $n \geq 2 \Longrightarrow \mathbf{A}$ *non intègre*

(iii) les idéaux de $\mathbf{A}$ sont de la forme $I_1 \times \cdots \times I_n$ où I_i est un idéal de $\mathbf{A}_i$.

(iv) Un idéal de $\mathbf{A}$ est premier si et seulement s'il existe $i \in \{1, \ldots, n\}$ tel qu'il existe $\mathcal{P}_i$ idéal premier de $\mathbf{A}_i$ tel que

$$I \;=\; \mathbf{A}_1 \times \cdots \times \mathbf{A}_{i-1} \times \mathcal{P}_i \times \mathbf{A}_{i+1} \times \cdots \times \mathbf{A}_n$$

Si de plus $\mathbf{A}$ est un produit de corps, alors, on a :

(v) $\mathbf{Div}(0) = \{a \in \mathbf{A} \;\mid (\; a \neq 0) \text{ et } (\exists i \in \{1, \ldots, n\})\; \pi_i(a) = 0\}$

(vi) $\mathbf{A} = \mathbf{Div}(0) \cup\; \mathbf{Un}(\mathbf{A}) \cup\; \{0\}$

(vii) $\mathbf{A} = \mathbf{fr}(\mathbf{A})$

(viii) $\mathbf{Nil}(\mathbf{A}) = \{0\}$

Preuve. $\triangleright$ Ces propriétés sont triviales à l'exception de *(iii)* et *(iv)* Vérifions *(iii)*. Soit I un idéal de $\mathbf{A}$. On pose $I_i = \vec{e_i}\, I$ (notation II.2). De sorte que $I = I_1 + \ldots + I_n$. Notons que I_i est un idéal du sous-anneau $\{0\} \times \cdots \times \{0\} \times \mathbf{A}_i \times \{0\} \times \cdots \times \{0\}$ de $\mathbf{A}$ et s'identifie donc à un idéal de $\mathbf{A}_i$. Avec cette identification, il vient finalement $I = I_1 \times \ldots \times I_n$. Vérifions

(iv). Notons que les idéaux de la forme annoncée sont clairement des idéaux premiers de $\mathbf{A}$ (directement à partir de la définition). Réciproquement, supposons que I soit premier. Si l'un des I_i (dont le produit cartésien est I) n'est pas un idéal premier de $\mathbf{A}_i$, il est clair que I ne peut être premier. On suppose donc que chaque I_i est un idéal premier de $\mathbf{A}_i$ Supposons maintenant que deux au moins des I_i diffèrent de l'idéal-unité, disons I_1 et I_2. Les éléments $(a_1, 1, a_3, \ldots, a_n)$ et $(1, a_2, a_3, \ldots, a_n)$ n'appartiennent pas à I, mais leur produit appartient à I, absurde. Notons que (vi) et (vii) sont équivalentes, que $\mathbf{A}$ soit un produit de corps ou pas. $\lhd$

Proposition II.2 *Soient $\mathbf{A}_1, \ldots, \mathbf{A}_n$ des anneaux commutatifs unitaires à idéaux principaux et $\mathbf{A}$ l'anneau-produit $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$. Alors $\mathbf{A}$ est à idéaux principaux.*

Preuve. $\rhd$ Soit I un idéal de $\mathbf{A}$. D'après (iii) de la proposition II.1 pour chaque $i \in \{1, \ldots, n\}$ il existe un idéal I_i de $\mathbf{A}_i$ tel que $I = I_1 \times \cdots \times I_n$. Par hypothèse, pour chaque $i \in \{1, \ldots, n\}$ il existe $a_i \in \mathbf{A}_i$ tel que $I_i = \langle a_i \rangle$. Soit $x \in I$. Il existe donc pour chaque $i \in \{1, \ldots, n\}$ un élément $x_i \in I_i$ de sorte que $x = (x_1, \ldots, x_n)$. De plus, pour chaque $i \in \{1, \ldots, n\}$ il existe $b_i \in \mathbf{A}_i$ tel que $x_i = b_i a_i$. D'où $x = (b_1, \ldots, b_n)(a_1, \ldots, a_n)$ et $I = \langle (a_1, \ldots, a_n) \rangle$. $\lhd$

Proposition II.3 *Soit $\mathbf{A}$ un anneau commutatif unitaire. Alors, $\mathbf{A}$ a une structure de produit d'anneaux si et seulement s'il existe des idéaux non triviaux (i.e. distincts de $\mathbf{A}$ et de son idéal nul) $I_1, \ldots, I_n$ de $\mathbf{A}$ dont la somme est $\mathbf{A}$ et 2 à 2 de produit nul.*

Preuve. $\rhd$ Supposons tout d'abord qu'il existe des anneaux $\mathbf{A}_1, \ldots, \mathbf{A}_n$, distincts de $\mathbf{A}$ et de l'anneau trivial, avec $n \geq 2$, et tel que $\mathbf{A}$ soit isomorphe à $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$. Soit I_i l'idéal non trivial de $\mathbf{A}$ constitué par les éléments de $\mathbf{A}$ dont toutes les composantes sont nulles sauf éventuellement celle sur $\mathbf{A}_i$. Alors on vérifie immédiatement que l'on a :

$$I_1 + \cdots + I_n \;=\; \mathbf{A} \quad \text{et} \quad i \neq j \;\Rightarrow I_i I_j \;=\; \langle 0 \rangle_{\mathbf{A}}$$

Réciproquement, soient $I_1, \ldots, I_n$ des idéaux non triviaux de $\mathbf{A}$ avec les propriétés ci-dessus. Il existe $u_i \in I_i$ tel que l'on ait :

$$u_1 + \cdots + u_n \;=\; 1_{\mathbf{A}} \quad \text{et} \quad i \neq j \;\Rightarrow u_i u_j = 0$$

On remarquera que l'on a :

$$u_i{}^2 \;=\; u_i(1 - u_1 - \cdots - u_{i-1} - u_{i+1} - \cdots - u_n) \;=\; u_i$$

Deux des u_i au moins sont non nuls. On peut donc les supposer tous non nuls. Notons $\mathbf{A}_i$ l'idéal $\langle u_i \rangle_{\mathbf{A}}$ (vu comme sous-anneau de $\mathbf{A}$) et ϕ l'application de $\mathbf{A}$ dans l'anneau-produit $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$ définie par $\phi(a) = (u_i a, \ldots, u_n a)$. On vérifie sans peine que $\mathbf{A}_i$ n'est ni $\mathbf{A}$, ni $\langle 0 \rangle_{\mathbf{A}}$ et que ϕ est un isomorphisme. On notera que $\phi^{-1}(a_1, \ldots, a_n) = \phi^{-1}(u_1^2 a_1, \ldots, u_n^2 a_n) = u_1 a_1 + \cdots + u_n a_n$. $\lhd$

Remarque II.1 La proposition classique ([Laf74, SZ67, van91]) précédente suggère l'utilisation des idempotents pour étudier les produits d'anneaux. Nous en rappelons la définition, puis quelques propriétés.

Définition II.2 *Soit $\mathbf{A}$ un anneau commutatif unitaire. Un élément $e \in \mathbf{A}$ est un* idempotent de $\mathbf{A}$ *si $e^2 = e$. L'ensemble des idempotents de $\mathbf{A}$ sera noté* $\mathbf{Idem}(\mathbf{A})$. *Un idempotent e de $\mathbf{A}$ sera dit :*

– non trivial *si* $e \neq 0$ *et* $e \neq 1$,
– décomposable *s'il existe* $f, g \in \mathbf{Idem}(\mathbf{A})$ *non nuls tels que* $e = f + g$,
– indécomposable *s'il n'est pas décomposable.*

Deux idempotents e *et* f *sont dits* orthogonaux *si leur produit est nul. Des idempotents* $e_1, \ldots, e_n$ *de* $\mathbf{A}$ *sont dits* deux à deux orthogonaux *si pour tout* i *et tout* j *de* $\{1, \ldots, n\}$ *le produit* $e_i e_j$ *est nul.*

Remarque II.2 Soit $\mathbf{A}$ un anneau commutatif unitaire. Soient e, f, g trois idempotents de $\mathbf{A}$ tels que $e = f + g$. Alors on a :

$$2fg = (f^2 + 2fg + g^2) - (f + g) = e^2 - e = 0$$

Ainsi f et g sont nécessairement orthogonaux.

Proposition II.4 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire. Alors on a les propriétés suivantes :*

(i) *si* e *et* f *sont des idempotents de* $\mathbf{A}$ *alors* ef, $1-e$ *et* $e+f-ef$ *sont aussi des idempotents de* $\mathbf{A}$, *de plus* ef *et* $1-e$ *sont orthogonaux,*

(ii) *si* $e_1, \ldots, e_n$ *sont des idempotents de* $\mathbf{A}$ *deux à deux orthogonaux, alors leur somme* $e_1 + \cdots + e_n$ *est aussi un idempotent de* $\mathbf{A}$,

(iii) *si* e *est un idempotent non trivial de* $\mathbf{A}$ *alors* e *est un diviseur de zéro* $\mathbf{A}$

(iv) *si* e *est un idempotent décomposable, avec* $e = f + g$, *où* f, g *sont des idempotents non nuls orthogonaux, alors* $1 - e = (1 - e)(1 - f)$ *et* $\langle 1 - e\rangle \neq \langle 1 - f\rangle$

(v) *si* e *et* f *sont deux idempotents indécomposables et distincts alors ils sont orthogonaux.*

Preuve. $\triangleright$ Les propriétés (i) à (iii) sont triviales. Vérifions (iv). Soit donc e un idempotent décomposable de $\mathbf{A}$. Soient alors f et g deux idempotents orthogonaux non nuls tels que $e = f + g$. Il vient :

$$(1 - e)(1 - f) = 1 - e - f + ef = 1 - e - f + (f + g)f = 1 - e$$

Ainsi $(1 - e) \in \langle 1 - f\rangle$. Cependant $1 - e$ et $1 - f$ n'engendrent pas le même idéal. En effet, si c'était le cas il existerait $b \in \mathbf{A}$ tel que $1 - f = b(1 - e)$. Par suite, on aurait :

$$g = e - f = e - ef^2 = e - e(f + g)f = e - e^2 f = e - ef = e(1 - f) = be(1 - e) = 0$$

Vérifions (v). Soient donc g et f deux idempotents indécomposables. On peut écrire :

$$g = g - fg + fg = (1 - f)g + fg$$

D'après (i), les éléments $(1 - f)$ et fg sont des idempotents orthogonaux. Il en va de même de $(1 - f)g$ et de fg. Comme g est indécomposable, on en déduit que $g = gf$ ou $fg = 0$. De même, f étant indécomposable, on obtient $f = fg$ ou $fg = 0$. Comme f et g sont distincts, seul reste le cas $fg = 0$. $\triangleleft$

Proposition II.5 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire. Les assertions suivantes sont équivalentes :*

(i) il existe a et b dans $\mathbf{A}$ *non nuls et tels que :* $ab = 0$ *et* $\langle a, b\rangle = \mathbf{A}$,

(ii) $\mathbf{A}$ *a une structure de produit d'anneaux,*

(iii) $\mathbf{A}$ *possède des idempotents non triviaux.*

Preuve. ▷ On a les implications $(i) \Rightarrow (ii)$ et $(ii) \Rightarrow (iii)$ d'après la proposition II.3 et sa preuve. Soit maintenant $e \in \mathbf{A}$ tel que $e^2 = e$ et $e \notin \{1, 0\}$ On a : $e(1-e) = 0$ et $e + (1-e) = 1$. Ainsi, (iii) entraîne (i). ◁

Proposition II.6 *Soit* $\mathbf{A}$ *un anneau (commutatif, unitaire). Alors,* $\mathbf{A}$ *a une structure de produit d'anneaux si et seulement s'il existe des idéaux non triviaux* $I_1, \ldots, I_n$ *2 à 2 comaximaux et dont le produit est l'idéal nul de* $\mathbf{A}$.

Preuve. ▷ Supposons tout d'abord qu'il existe des anneaux $\mathbf{A}_1, \ldots, \mathbf{A}_n$ distincts de $\mathbf{A}$ et de l'anneau trivial, avec $n \geq 2$, et tel que $\mathbf{A}$ soit isomorphe à $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$. Soit I_i l'idéal non trivial de $\mathbf{A}$ constitué par les éléments de $\mathbf{A}$ dont la composante sur $\mathbf{A}_i$ est nulle. Alors on vérifie immédiatement que l'on a :

$$I_1 \cdots I_n \;=\; \langle 0\rangle_{\mathbf{A}} \quad \text{et} \quad i \neq j \;\Rightarrow I_i + I_j \;=\; \mathbf{A}$$

Réciproquement, soient $I_1, \ldots, I_n$ des idéaux non triviaux de $\mathbf{A}$ avec les propriétés ci-dessus. Notons $\mathbf{A}_i$ l'anneau-quotient $\mathbf{A}/I_i$, et, h_i la surjection canonique de $\mathbf{A}$ dans $\mathbf{A}_i$. Pour $a \in \mathbf{A}$ on pose $h(a) = (h_1(a), \ldots, h_n(a))$. Puisque les I_i sont 2 à 2 comaximaux, on a d'une part la surjectivité de h (c'est le théorème des restes chinois) et d'autre part l'injectivité de h. En effet, sous cette hypothèse, le produit des I_i est égal à leur intersection et, par suite, de $h(a) = 0$ on tire $a = 0$. Bref, h est un isomorphisme. ◁

Proposition II.7 *Soit* $\mathbf{A}$ *un anneau (commutatif, unitaire). Alors,* $\mathbf{A}$ *a une structure de produit de corps si et seulement s'il existe des idéaux maximaux 2 à 2 distincts* $I_1, \ldots, I_n$ *de* $\mathbf{A}$ *dont le produit est égal à l'idéal nul de* $\mathbf{A}$.

Preuve. ▷ Supposons tout d'abord qu'il existe des corps $\mathbf{K}_1, \ldots, \mathbf{K}_n$, avec $n \geq 2$, tel que $\mathbf{A}$ soit isomorphe à $\mathbf{K}_1 \times \cdots \times \mathbf{K}_n$. Soit I_i l'idéal non trivial de $\mathbf{A}$ constitué par les éléments de $\mathbf{A}$ dont la composante sur $\mathbf{K}_i$ est nulle. Notons que $\mathbf{A}/I_i \cong \mathbf{K}_i$ et donc que I_i est maximal. Les idéaux I_i sont 2 à 2 distincts et leur produit est nul. La réciproque résulte de la proposition précédente. Il suffit de remarquer (encore) que deux idéaux maximaux distincts sont comaximaux. ◁

Proposition II.8 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire nœthérien. Alors, tout idempotent de* $\mathbf{A}$ *est somme d'idempotents indécomposables.*

Preuve. ▷ Soit e_0 un idempotent de $\mathbf{A}$. Supposons que e_0 ne soit pas somme d'idempotents indécomposables. En particulier e_0 est décomposable. Il existe donc des idempotents orthogonaux non nuls e_1 et e'_1 dont e_0 est la somme. Rappelons que d'après le point (iv) de la proposition II.4 on a l'inclusion stricte :

$$\langle 1 - e_0\rangle \subset \langle 1 - e_1\rangle$$

L'un des idempotents e_1 et e'_1 ne peut être la somme d'idempotents indécomposables. Supposons qu'il s'agisse de e_1. Il existe encore des idempotents orthogonaux non nuls e_2 et e'_2 dont e_1 est la somme. De plus, on a l'inclusion stricte :

$$\langle 1 - e_1\rangle \subset \langle 1 - e_2\rangle$$

Bref, par récurrence, on construit une suite strictement croissante (pour l'inclusion) d'idéaux de $\mathbf{A}$, ce qui contredit le fait que $\mathbf{A}$ soit nœthérien. $\lhd$

Définition II.3 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire. On dira ici que* $\mathbf{A}$ *est* indécomposable *si* $1_{\mathbf{A}}$ *est un idempotent indécomposable.*

Proposition II.9 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire. Alors* $\mathbf{A}$ *est indécomposable si et seulement s'il ne possède pas de structure de produit d'anneaux.*

Preuve. $\rhd$ Supposons tout d'abord que $\mathbf{A}$ soit indécomposable. Soit e est idempotent de $\mathbf{A}$. On a bien sur $1 = (1-e)+e$ et $(1-e)e = 0$. Comme 1 est indécomposable on a soit $1-e = 0$, soit $e = 0$. Ainsi $\mathbf{A}$ ne possède pas d'idempotents non triviaux. Par suite, avec la proposition II.5, $\mathbf{A}$ ne possède pas de structure de produit d'anneaux. La réciproque est tout aussi aisée. $\lhd$

Théorème II.1 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire nœthérien possèdant au moins un idempotent non trivial. Alors* $\mathbf{A}$ *a une structure de produit d'anneaux* $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$ *tel que pour chaque* $i \in \{1, \ldots, n\}$ *l'anneau* $\mathbf{A}_i$ *est nœthérien et indécomposable.*

Preuve. $\rhd$ D'après la proposition II.8 tout idempotent de $\mathbf{A}$ est somme d'idempotents indécomposables. En particulier , il existe des idempotents indécomposables $e_1, \ldots, e_n$ dont la somme est 1. Si l'un des e_i apparaît k fois dans cette somme, avec $k \geq 2$, on peut remplacer ke_i par e_i . En effet :

$$e_i = 1\, e_i = \Sigma_j\, e_j e_i = k e_i^2 = k e_i$$

Bref, on peut supposer les e_i 2 à 2 distincts. Par suite, à l'aide du point (v) de la proposition II.4 on en déduit que les $e_1, \ldots, e_n$ sont orthogonaux dans leur ensemble. On raisonne alors comme dans la seconde moitié de la preuve de la proposition II.3. On considère les anneaux $\mathbf{A}_i$ définis par $\mathbf{A}_i = \langle e_i \rangle$ et l'isomorphisme de $\mathbf{A}$ dans $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$. qui à un élément a associe le n-uplet $(e_1 a, \ldots, e_n a)$. Notons que chaque e_i est l'élément neutre pour la multiplication dans $\mathbf{A}_i$. Par suite, les $\mathbf{A}_i$ sont indécomposables. Enfin, si l'un des $\mathbf{A}_i$ n'était pas nœthérien, il en irait de même de $\mathbf{A}$, grâce au point (iii) de la proposition II.1. $\lhd$

Remarque II.3 Soit $\mathbf{A}$ un anneau commutatif unitaire. Nous allons étudier les nilpotents non triviaux de $\mathbf{A}$ dans le cas où tout élément de $\mathbf{A}$ est soit nul, soit inversible, soit diviseur de zéro non nilpotent. Autrement dit, dans le cas où $\mathbf{A}$ est un anneau réduit où tout élément régulier de $\mathbf{A}$ est inversible.

Proposition II.10 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire. Les propriétés suivantes sont équivalentes :*

(i) $\mathbf{A}$ *est réduit*

(ii) $(\forall x \in \mathbf{A})\ \ x \neq 0 \implies (\exists \mathcal{P} \in \mathsf{spec}(\mathbf{A}))\ x \notin \mathcal{P}$

Preuve. $\rhd$ En effet, $\mathbf{A}$ réduit signifie $\mathbf{Nil}(\mathbf{A}) = \{0\}$. Or $\mathbf{Nil}(\mathbf{A})$ est l'intersection des idéaux premiers de $\mathbf{A}$ ([SZ67] vol. I). $\lhd$

Proposition II.11 *Soit* $\mathbf{A}$ *un anneau réduit et* $x \in \mathbf{A}$ *non nul. Alors on a :*

$$\langle 0 \rangle : x = \bigcap_{\{\mathcal{P} \in \mathsf{spec}(\mathbf{A}) \ \mid\ x \notin \mathcal{P}\}} \mathcal{P}$$

Preuve. $\triangleright$ Soient x et y dans $\mathbf{A}$. Comme $\mathbf{A}$ est réduit, le seul élément de $\mathbf{A}$ qui appartienne à l'intersection de tous les idéaux premiers de $\mathbf{A}$ est 0. On a par conséquent, les équivalences suivantes :

$$\begin{array}{rcl} y \in \langle 0 \rangle : x & \iff & xy = 0 \\ & \iff & (\forall \mathcal{P} \in \mathsf{spec}(\mathbf{A}))\ xy \in \mathcal{P} \\ & \iff & (\forall \mathcal{P} \in \mathsf{spec}(\mathbf{A}))\ x \notin \mathcal{P} \implies y \in \mathcal{P} \end{array}$$

D'où le résultat. $\triangleleft$

Proposition II.12 *Soit* $\mathbf{A}$ *un anneau réduit nœthérien. Alors on a :*

$$\mathsf{spec}(\mathbf{A}) = \mathsf{ap}(\langle O \rangle)$$

Autrement dit, tout idéal premier distinct de $\mathbf{A}$ *est l'un des idéaux premiers associés à* $\langle O \rangle$*.*

Preuve. $\triangleright$ Notons $\mathcal{P}_1, \ldots, \mathcal{P}_n$ les idéaux premiers associés à $\langle O \rangle$. Puisque $\mathbf{A}$ est réduit, $\langle O \rangle$ est intersection des idéaux $\mathcal{P}_1, \ldots, \mathcal{P}_n$. Soit alors $\mathcal{P}$ un idéal premier de $\mathbf{A}$ distinct de $\mathbf{A}$. Comme tout élément de $\mathbf{A}$ est soit nul, soit inversible, soit diviseur de zéro, l'idéal $\mathcal{P}$ est contenu dans $\mathbf{Div}(0) \cup \{0\}$. Or $\mathbf{A}$ étant nœthérien, $\mathbf{Div}(0)$ est contenu dans la réunion des idéaux $\mathcal{P}_1, \ldots, \mathcal{P}_n$ ([SZ67] vol. I p. 214). Rappelons maintenant le *lemme d'évitement* suivant : si un idéal premier est contenu dans une réunion finie d'idéaux premiers, alors il est contenu dans l'un deux. Par suite, $\mathcal{P}$ est contenu dans l'un des idéaux $\mathcal{P}_1, \ldots, \mathcal{P}_n$. Comme l'idéal premier $\mathcal{P}$ contient (trivialement) l'intersection des $\mathcal{P}_i$, il contient l'un d'eux. Finalement, $\mathcal{P}$ est l'un des $\mathcal{P}_i$. $\triangleleft$

Proposition II.13 *Soit* $\mathbf{A}$ *un anneau nœthérien réduit où tout élément régulier est inversible et soit* x *un élément de* $\mathbf{A}$*. Alors les propositions suivantes sont équivalentes :*

- (i) *x est un diviseur de zéro*
- (ii) *x est un idempotent non trivial*

Preuve. $\triangleright$ L'implication $(ii) \implies (i)$ est immédiate. Vérifions sa réciproque. Nous allons établir :

$$(\forall x \in \mathbf{Div}(0))\ \langle x \rangle + \langle 0 \rangle : x = \mathbf{A}$$

Soit x un diviseur de zéro de $\mathbf{A}$ et $\mathcal{M}$ un idéal maximal contenant $\langle x \rangle + \langle 0 \rangle : x$. D'après la proposition II.11 l'idéal $\mathcal{M}$ contient l'intersection des idéaux premiers distincts de $\mathbf{A}$ et auxquels x n'appartient pas. D'après la proposition II.12 précédente, ces idéaux sont tous maximaux et en nombre fini. L'idéal $\mathcal{M}$ contient donc l'un deux. Comme $x \in \mathcal{M}$, il vient $\mathcal{M} = \mathbf{A}$. Par suite, il existe $y \in \langle 0 \rangle : x$ tel que $1 = x + y$. Ainsi $x^2 = x(1-y) = x$ est un idempotent non trivial de $\mathbf{A}$. $\triangleleft$

Théorème II.2 (Lazard) *Un anneau commutatif unitaire nœthérien est un anneau réduit et tel que tout élément régulier est inversible si et seulement si c'est un produit de corps.*

Preuve. $\triangleright$ Soit $\mathbf{A}$ un anneau commutatif unitaire nœthérien. Supposons tout d'abord que $\mathbf{A}$ soit un anneau réduit où tout élément régulier est inversible. Si ce n'est pas un corps, il suffit d'appliquer la proposition II.12 puis la proposition II.7. Ainsi, si $\mathcal{P}_1, \ldots, \mathcal{P}_n$ désignent les idéaux premiers de $\mathbf{A}$ on a :

$$\mathbf{A} \approx \mathbf{A}/\mathcal{P}_1 \times \cdots \times \mathbf{A}/\mathcal{P}_n$$

Bref, les idéaux $\mathcal{P}_1$ étant maximaux, l'anneau $\mathbf{A}$ est un produit de corps. La réciproque résulte immédiatement des propriétés d'un produit de corps (proposition II.1). $\triangleleft$

Théorème II.3 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire nœthérien. Alors, les propriétés suivantes sont équivalentes :*

(1) *le graphe* G *associé à l'ensemble partiellement ordonné* $(\mathsf{spec}(\mathbf{A}), \subseteq)$ *est connexe*

(2) *l'anneau* $\mathbf{A}$ *ne possède pas d'idempotents non triviaux*

Preuve. $\triangleright$ Montrons tout d'abord que si $\mathbf{A}$ possède des idempotents non triviaux alors le graphe G n'est pas connexe. Soit e un idempotent non trivial de $\mathbf{A}$ et posons $f = 1 - e$. Comme $1 = e+f$ aucun idéal premier de $\mathsf{spec}(\mathbf{A})$ ne contient à la fois e et f. Donc $\mathcal{V}_e \cap \mathcal{V}_f = \emptyset$. Soit $\mathcal{P} \in \mathsf{spec}(\mathbf{A})$. Comme $ef = 0 \in \mathcal{P}$. l'idéal premier $\mathcal{P}$ contient soit e soit f. Par suite, $\mathcal{V}_e$ et $\mathcal{V}_f$ réalisent une partition des sommets de G en deux composantes connexes. Montrons maintenant que si G n'est pas connexe, alors $\mathbf{A}$ possède deux idempotents non triviaux dont la somme est 1. On suppose donc qu'il existe deux idéaux I et J de $\mathbf{A}$ tels que l'on ait :

$$\mathcal{V}(I) \cap \mathcal{V}(J) = \emptyset \quad \text{et} \quad \mathcal{V}(I) \cup \mathcal{V}(J) = \mathsf{spec}(\mathbf{A})$$

Nos idéaux I et J sont donc comaximaux. En effet, un idéal maximal de $\mathbf{A}$ doit contenir l'idéal $I + J$: il ne peut s'agir que de $\mathbf{A}$ lui-même. D'autre part, on a $\mathcal{V}(I \cap J) = \mathsf{spec}(\mathbf{A})$. Par conséquent, tout idéal premier $\mathcal{P}$ de $\mathsf{spec}(\mathbf{A})$ contient le produit IJ. Ainsi pour tout $a \in I$ et tout $b \in J$, le produit ab est nilpotent. Soient alors $a \in I$ et $b \in J$ tel que $1 = a + b$. Soient de plus n et m les plus petit entiers naturels tels que $a^n = b^m = 0$. Avec $a + b = 1$, on vérifie $n \geq 3$ et $m \geq 3$. Puis, à l'aide la formule du binôme on vérifie qu'il existe des entiers p et q tels que $pa^{n-1}b^{m-2} + qa^{n-2}b^{m-1} = 1$. Il suffit d'élever $(a + b)$ à la puissance $m + n - 3$. Notons que ni $pa^{n-1}b^{m-2}$, ni $qa^{n-2}b^{m-1}$ ne peuvent être nuls, sans quoi a et b seraient inversibles. Enfin on a :

$$a^{n-1}b^{m-2}a^{n-2}b^{m-1} = a^{2n-3}b^{2m-3}$$

Comme, $n \geq 3 \implies 2n - 3 \geq n$, le produit ci-dessus est nul. Nous avons donc exhibé deux diviseurs de zéro c et d de produit nul et de somme 1. Ce sont par conséquent des idempotents non triviaux, puisque l'on a par exemple : $c^2 = c(1 - d) = c$. $\triangleleft$

2 Scindage d'un idéal

Dans cette section nous introduisons la notion de scindage d'un idéal (déf. II.4). L'introduction et l'étude de cette notion est justifiée, entre autre, par la proposition II.14. Celle-ci exprime en particulier le fait que scinder un idéal dans un anneau de polynômes signifie décomposer l'ensemble de ses zéros et donc *résoudre* le système d'équations associé, en un sens général. Les scindages d'idéaux modélisent les distinctions de cas dans les calculs algébriques. Un premier exemple est la *distinction* $x = 0$ *ou* $x \neq 0$ que l'on étudie dans la proposition II.15.

Une autre justification de la notion de scindage est la suivante. Il est fréquent que l'on ait à calculer dans un certain anneau $\mathbf{A}$ *modulo un certain ensemble de relations* que l'on peut représenter par un idéal I de $\mathbf{A}$. Autrement dit, on calcule dans $\mathbf{A}/I$. Il peut alors être commode de factoriser cet anneau. Nous savons d'après le théorème II.3 que ceci n'est possible que si le graphe $(\mathsf{spec}(\mathbf{A}/I), \subseteq)$ n'est pas connexe. Cette possibilité se traduit en termes de scindages de I avec la notion de *scindage biunivoque* (déf. II.6). L'étude de ces scindages particuliers constitue l'objet essentiel des résultats de ce chapitre. Il en ressort que le bon cas est celui où I est intersection finie d'idéaux maximaux. Dans ce cas factoriser $\mathbf{A}/I$ permet de diviser un calcul dans $\mathbf{A}/I$ en des calculs distincts dont l'ensemble est équivalent au calcul initial dans $\mathbf{A}/I$. À l'opposé, le mauvais cas est celui où I est primaire sans être premier. Dans ce cas scinder I provoque une duplication du calcul initial dans $\mathbf{A}/I$.

Définition II.4 *Soit* $\mathbf{A}$ *un anneau (commutatif, unitaire) et* I *un idéal de* $\mathbf{A}$. *On appelle* scindage *de* I *tout* n*-uplet* $S = (S_1, \ldots, S_n)$ *d'idéaux de* $\mathbf{A}$ *tel que :*

$$S_1 \cdots S_n \;\subseteq\; I \;\subseteq\; S_1 \cap \cdots \cap S_n$$

On écrira alors :

$$I \longmapsto (S_1, \ldots, S_n)$$

L'entier n *est appelé* taille *du scindage et un scindage de taille* n *est dit* n-aire. *Les idéaux* S_i *sont appelés* composantes *du scindage* S. *Une composante* S_j *est dite* redondante *si*

$$S_1 \cdots S_{j-1} S_{j+1} \cdots S_n \;\subseteq\; I$$

Pour une composante S_j *de* S, *le nombre d'indices* i *compris entre* 1 *et* n *tel que* $S_j = S_i$ *est appelé l'*ordre de multiplicité *de la composante* S_j *dans* S. *Le scindage* S *sera dit* sans composantes multiples, *si toutes ses composantes ont* 1 *pour ordre de multiplicité Il sera dit* propre *si* I *est une partie propre de chacune de ses composantes. Il sera dit* minimal *s'il est propre et si aucune de ses composantes n'est redondante. Enfin, il sera dit* premier *s'il est propre et si aucune de ses composantes n'admet de scindages propres.*

Remarque II.4 Supposons maintenant que $S = (S_1, \ldots, S_n)$ soit un scindage propre de l'idéal I. Alors, il est de taille au moins 2. Si de plus S est minimal, alors aucune de ses composantes n'est $\mathbf{A}$. Notons que $\mathbf{A}$ n'admet pas de scindages propres, tandis que $\langle 0 \rangle_{\mathbf{A}}$ admet des scindages propres si et seulement si $\mathbf{A}$ n'est pas intègre. Notons aussi que de tout scindage on peut *extraire* un scindage minimal du même idéal. Notons enfin qu'un scindage minimal peut avoir des composantes comparables pour l'inclusion. Par exemple, considérer dans $\mathbf{A}[x, y]$ le scindage (minimal et premier) suivant

$$\langle x^2, x + y \rangle \longmapsto (\langle x, y \rangle, \langle x, y \rangle)$$

Notations II.3 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire,* I *un idéal de*$\mathbf{A}$ *et* a *un élément de* $\mathbf{A}$. *On désignera par* $I + a$ *l'idéal engendré par* $I \cup \{a\}$, *c'est-à-dire la somme d'idéaux* $I + \langle a \rangle$.

Proposition II.14 *Soit* $\mathbf{A}$ *un anneau (commutatif, unitaire),* I *un idéal de* $\mathbf{A}$ *et* $(S_1, \ldots, S_n)$ *un scindage de* I. *Alors, on a :*

(i) I *est premier si et seulement s'il n'admet pas de scindages propres.*

(ii) I *est radical si et seulement si tout scindage minimal de* I *est sans composantes multiples*

(iii) Si $\mathbf{A}$ *est un anneau de polynômes sur un corps* $\mathbf{k}$ *et si* $\mathbf{K}$ *est un sur-corps de* $\mathbf{k}$, *alors*

$$\mathbf{V_K}(I) \;=\; \bigcup_{i=1}^{n} \mathbf{V_K}(S_i)$$

Preuve. $\triangleright$ Vérifions (i). Soit $I \longmapsto (S_1, \ldots, S_n)$ un scindage propre. On a donc $n \geq 2$. De plus, pour chaque $i \in \{1, \ldots, n\}$ on peut choisir a_i dans $S_i \setminus I$. Puisque le produit des S_i est contenu dans I on a : $a_1 \cdots a_n \in I$. Par suite I n'est pas premier. Réciproquement, si I n'est pas premier, il existe a et b dans $\mathbf{A} \setminus I$ avec $ab \in I$. On a alors :

$$(I + a)(I + b) \subseteq I \subseteq (I + a) \cap (I + b)$$

Ceci prouve (i). Vérifions (ii). Supposons que $I \longmapsto (S_1, \ldots, S_n)$ soit un scindage minimal avec composantes multiples. Quitte à renuméroter les composantes on peut supposer $S_1 = S_2$. Comme le scindage est minimal, on peut choisir les a_i de sorte que $a = a_2 \cdots a_n \notin I$. Mais $a_2 \in S_1$ et a^2 appartient au produit des composantes $S_1, \ldots, S_n$, donc à I et l'idéal I n'est pas radical. Réciproquement, si I n'est pas radical, alors il existe $a \in \mathbf{A}$ et un entier $p \geq 2$ tel que $a^p \in I$ et $a^{p-1} \notin I$. Ainsi, Le p-uplet d'idéaux de $\mathbf{A}$ tous égaux à $I + \langle a \rangle$ est un scindage minimal de I. Ceci prouve (ii). Enfin (iii) résulte du fait que les variétés affines sur $\mathbf{L}$ du produit et de l'intersection des idéaux S_i sont toutes deux égales à la réunion des variétés affines sur $\mathbf{L}$ de chacun des idéaux S_i. Cette dernière propriété participe à la justification de la notion de scindage. $\triangleleft$

Définition II.5 *Soit* $\mathbf{A}$ *un anneau (commutatif, unitaire) et* I, S_1, S_2 *des idéaux de* $\mathbf{A}$*. On dira que* (S_1, S_2) *est un* scindage de première espèce *de* I *si* (S_1, S_2) *est un scindage de* I *et s'il existe* $p \in \mathbf{A}$ *tel que :*

$$S_1 = I + p \quad \text{et} \quad S_2 = I : p$$

Proposition II.15 *Soit* $\mathbf{A}$ *un anneau commutatif, unitaire,* I *un idéal de* $\mathbf{A}$ *et* $p \in \mathbf{A}$*. Alors, on a :*

(i) $I \longmapsto (I + p, I : p)$

(ii) $I = \sqrt{I}$ et $p \notin I \quad \longrightarrow \quad I \longmapsto (I + p, I : p^\infty)$

(iii) $I = \sqrt{I}$ et $p \notin I \quad \longrightarrow \quad I = \sqrt{(I + p)} \cap (I : p^\infty)$

(iv) I primaire et $p \notin \sqrt{I} \quad \longrightarrow \quad I = (I + p) \cap (I : p)$

Preuve. $\triangleright$ La propriété (i) se vérifie sans peine. La propriété (ii) s'en déduit à l'aide de la proposition I.20. Vérifions (iii). D'après (ii), par *passage au radical* il vient :

$$\sqrt{(I + p)(I : p^\infty)} \subseteq I \subseteq \sqrt{(I + p) \cap (I : p^\infty)}$$

Or, le produit et le quotient de deux idéaux J et K de $\mathbf{A}$ ont même radical ; de plus, ce radical est égal à $\sqrt{J} \cap \sqrt{K}$. Enfin, puisque I est radical, on a aussi : $\sqrt{I : p^\infty} = I : p^\infty$ (prop. I.20). Vérifions (iv). Il suffit de vérifier l'inclusion $(I + p) \cap (I : p) \subseteq I$. Soient alors $x \in I$ et $y \in \mathbf{A}$ tel que $x + yp \in I : p$. On a donc $xp + yp^2 \in I$. Donc $yp^2 \in I$. Or I est primaire. Par suite, si $yp \notin I$, une puissance de p appartient à I. Or, par hypothèse, $p \notin \sqrt{I}$. Donc $x + yp \in I$. $\triangleleft$

Définition II.6 *Soit* $\mathbf{A}$ *un anneau (commutatif, unitaire) et* I, S_1, $\ldots$, S_n *des idéaux de* $\mathbf{A}$ *tel que* $S = (S_1, \ldots, S_n)$ *soit un scindage de* I*. On appelle* homomorphisme associé *au scindage* S *l'homomorphisme d'anneaux noté* h_S *et défini comme suit :*

$$h_S : \begin{array}{ccc} \mathbf{A}/I & \longrightarrow & \mathbf{A}/S_1 \times \cdots \times \mathbf{A}/S_n \\ \overline{x}^I & \longmapsto & (\overline{x}^{S_1}, \ldots, \overline{x}^{S_n}) \end{array}$$

On dira que S est :

(i) univoque *si I est égal à l'intersection des idéaux $S_1, \ldots, S_n$,*

(ii) biunivoque *si les idéaux $S_1, \ldots, S_n$ sont 2 à 2 comaximaux.*

Remarque II.5 Reprenons les notations de la définition II.6 précédente. Rappelons que le produit et l'intersection d'idéaux 2 à 2 comaximaux sont égaux. Par conséquent, si S est biunivoque, S est aussi univoque. De plus, à l'aide de la proposition II.6 p. 54 on a l'isomorphisme :

$$\mathbf{A}/I \cong \mathbf{A}/S_1 \times \cdots \times \mathbf{A}/S_n$$

Notons également que l'on a :

$$S \text{ biunivoque} \implies S \text{ sans composantes multiples}$$

Proposition II.16 *Soit $\mathbf{A}$ un anneau (commutatif, unitaire) et I, S_1, ..., S_n tel que $I \longmapsto S = (S_1, \ldots, S_n)$. Soit h_S le homomorphisme associé au scindage S. Alors on a :*

(i) $\mathrm{K\scriptstyle ER}(h_S) = (\cap_{i=1}^n S_i)/I$

(ii) $\mathrm{I\scriptstyle M}(h_S) = \{(\overline{y_1}^{S_1}, \ldots, \overline{y_n}^{S_n}) \mid i \neq j \implies \overline{y_j}^{S_i+S_j} = \overline{y_i}^{S_i+S_j}\}$

(iii) *h_S surjectif* $\iff (\forall i, j \in \{1, \ldots, n\})\ i \neq j \implies S_i + S_j = \mathbf{A}$

où $\mathrm{K\scriptstyle ER}(h_S)$ et $\mathrm{I\scriptstyle M}(h_S)$ désignent respectivement le noyau et l'image de h_S.

Preuve. $\triangleright$ La propriété (i) est immédiate. Vérifions (ii). Déterminons l'image de h_S. Pour $y_1, \ldots, y_n \in \mathbf{A}$, on cherche $x \in \mathbf{A}$ tel que :

$$h_S(\overline{x}^I) = (\overline{y_1}^{S_1}, \ldots, \overline{y_n}^{S_n})$$

Ceci est possible si et seulement si :

$$\overline{x - y_1}^{S_1} = \overline{x - y_2}^{S_2} = \cdots = \overline{x - y_n}^{S_n} = 0$$

c'est à dire si et seulement si :

$$(\exists (s_1, s_2, \ldots, s_n) \in S_1 \times S_2 \times \cdots \times S_n) \mid y_1 + s_1 = y_2 + s_2 = \cdots = y_n + s_n$$

D'où (ii). La propriété (iii) est en fait le théorème dit *des restes chinois*. D'après (ii), il est clair que si les S_i sont deux à deux comaximaux h_S est surjectif. Réciproquement, pour $i \in \{1, \ldots, n\}$, notons ω_i l'élément du produit d'anneau $\mathbf{A}/S_1 \times \cdots \times \mathbf{A}/S_n$ dont toutes les composantes valent 1 sauf la i-ième qui est nulle. Chaque ω_i a un antécédent u_i dans S_i qui se réduit à 1 modulo S_j pour $j \neq i$. Autrement dit :

$$(\exists (u_1, \ldots, u_n) \in S_1 \times S_2 \times \cdots \times S_n) \mid i \neq j \implies \overline{u_i}^{S_j} = 1$$

Remarquons que cette dernière condition est équivalente à la surjectivité de h_S. En effet :

$$h_S(\overline{(1 - u_1)y_1 + \cdots + (1 - u_n)y_n}^I) = (\overline{y_1}^{S_1}, \ldots, \overline{y_n}^{S_n})$$

De plus cette condition est équivalente à $S_i + S_j = \mathbf{A}$ pour tout i et tout j dans $\{1, \ldots, n\}$ avec $i \neq j$. $\triangleleft$

Proposition II.17 *Soit* $\mathbf{A}$ *un anneau (commutatif, unitaire) et* I, S_1, ..., S_n *tel que* $I \longmapsto S = (S_1, \ldots, S_n)$. *Soit* h_S *le homomorphisme associé au scindage* S. *Alors on a :*

(*i*) S *univoque* $\iff$ h_S *injective*

(*iii*) S *biunivoque* $\iff$ h_S *bijective*

Preuve. $\triangleright$ Les propriétés (*i*) et (*ii*) résultent immédiatement de la proposition II.16 précédente et de la remarque II.5. $\triangleleft$

Proposition II.18 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire,* I *un idéal de* $\mathbf{A}$ *et* $S = (S_1, \ldots, S_n)$ *un scindage de* I. *Alors,* $(\sqrt{S_1}, \ldots, \sqrt{S_n})$ *est un scindage univoque de* $\sqrt{I}$

Preuve. $\triangleright$ Immédiat d'après les propriétés du radical d'un idéal. Voir aussi la preuve de la proposition II.15. $\triangleleft$

Proposition II.19 *Soit* $\mathbf{A}$ *un anneau (commutatif, unitaire), et* I,S_1,S_2,$S_{2,1}$,$S_{2,2}$,S_3 *des idéaux de* $\mathbf{A}$. *Alors, on a :*

(*i*) *Si* $I \longmapsto (S_1, S_2)$ *et* $S_2 \longmapsto (S_{2,1}, S_{2,2})$ *sont des scindages (resp. scindages univoques) (resp. scindages biunivoques) alors* $I \longmapsto (S_1, S_{2,1}, S_{2,2})$ *est un scindage (resp. scindage univoque) (resp. scindage biunivoque)*

(*ii*) *Si* $I \longmapsto (S_1, S_2, S_3)$ *est un scindage univoque (resp. biunivoque) alors* $I \longmapsto (S_1, S_2 \cap S_3)$ *est un scindage univoque (resp. biunivoque).*

(*iii*) *Si* $I \longmapsto (S_1, S_2, S_3)$ *est un scindage biunivoque alors* $I \longmapsto (S_1, S_2S_3)$ *est un scindage biunivoque.*

Preuve. $\triangleright$ Vérifions (*i*). Supposons que $I \longmapsto (S_1, S_2)$ et $S_2 \longmapsto (S_{2,1}, S_{2,2})$ soient des scindages (resp. scindages univoques). Remarquons que l'on a :

$$S_1(S_{2,1}S_{2,2}) \subseteq S_1S_2 \subseteq I \subseteq S_1 \cap S_2 \subseteq S_1 \cap (S_{2,1} \cap S_{2,2})$$

Supposons de plus que les deux scindages binaires soient biunivoques. Alors on a :

$$\mathbf{A} = S_1 + S_2 \subseteq S_1 + S_{2,1} \quad \text{et} \quad \mathbf{A} = S_1 + S_2 \subseteq S_1 + S_{2,1}$$

Ceci prouve (*i*). Vérifions (*ii*). Supposons $I \longmapsto (S_1, S_2, S_3)$ soit un scindage univoque. Alors on a :

$$S_1(S_2 \cap S_3) \subseteq S_1 \cap (S_2 \cap S_3) = S_1 \cap S_2 \cap S_3 = I$$

Donc $I \longmapsto (S_1, S_2 \cap S_3)$ est un scindage univoque. Supposons de plus que $I \longmapsto (S_1, S_2, S_3)$ soit biunivoque. Comme S_1,S_2,S_3 sont 2 à 2 comaximaux, S_1 et $S_2 \cap S_3$ sont comaximaux et $I \longmapsto (S_1, S_2 \cap S_3)$ est biunivoque. Enfin, (*ii*) entraîne (*iii*). $\triangleleft$

Théorème II.4 *Soit* $\mathbf{A}$ *un anneau nœthérien,* I *un idéal de* $\mathbf{A}$ *et* $S = (S_1, \ldots, S_n)$ *un scindage de* I. *Si* S *est propre et biunivoque, alors* S *est minimal.*

Preuve. $\triangleright$ On raisonne par l'absurde. Soit donc $S = (S_1, \ldots, S_n)$ un scindage biunivoque de I et S_j une composante redondante de S. Alors, on a :

$$I = S_1 \cdots S_n \subseteq S_1 \cdots S_{j-1}S_{j+1} \cdots S_n \subseteq I$$

Par conséquent :

$$\mathbf{A}/S_1 \times \cdots \times \mathbf{A}/S_n \cong \mathbf{A}/S_1 \times \cdots \mathbf{A}/S_{j-1} \times \mathbf{A}/S_{j+1} \times \cdots \mathbf{A}/S_n$$

Notons que le membre de droite ci-dessus est un anneau nœthérien, puisqu'il est isomorphe à l'anneau quotient $\mathbf{A}/I$. Or, si $\mathbf{A}$ est nœthérien et si $\mathbf{B}$ est un second anneau non trivial, commutatif et unitaire, il n'existe pas d'isomorphisme de $\mathbf{A}$ dans $\mathbf{A} \times \mathbf{B}$. En effet, supposons qu'il existe un isomorphisme f de $\mathbf{A}$ dans $\mathbf{A} \times \mathbf{B}$. Posons :

$$I_0 = f^{-1}(\{0_{\mathbf{A}}\} \times \mathbf{B})$$

Comme f est injective et comme $\mathbf{B}$ compte au moins deux éléments, I_0 est un idéal de $\mathbf{A}$ qui n'est pas réduit à l'idéal nul de $\mathbf{A}$. Comme f est surjective et comme $\mathbf{A}$ compte au moins deux éléments, I_0 n'est pas $\mathbf{A}$. Posons maintenant :

$$I_1 = f^{-1}(I_0 \times \mathbf{B})$$

Comme f est injective et comme I_0 compte au moins deux éléments dont $0_{\mathbf{A}}$, I_1 est un idéal de $\mathbf{A}$ contenant strictement I_0. Comme f est surjective et comme I_0 n'est pas $\mathbf{A}$, I_1 est strictement contenu dans $\mathbf{A}$. Plus généralement, en posant :

$$I_{i+1} = f^{-1}(I_i \times \mathbf{B})$$

on construit une suite d'idéaux de $\mathbf{A}$ strictement croissante et infinie :

$$\langle 0_{\mathbf{A}} \rangle \subset I_0 \subset I_1 \subset \cdots \subset I_i \subset I_{i+1} \subset \cdots$$

Ceci contredit le fait que $\mathbf{A}$ soit nœthérien. $\lhd$

Théorème II.5 *Soit $\mathbf{A}$ un anneau commutatif unitaire et I un idéal de $\mathbf{A}$. Alors, I admet un scindage biunivoque propre et sans composantes égales à $\mathbf{A}$ si et seulement si $\mathbf{A}/I$ a une structure de produit d'anneaux.*

Preuve. $\rhd$ Soit I un idéal de l'anneau $\mathbf{A}$. Si I admet un scindage biunivoque propre et sans composantes égales à $\mathbf{A}$ alors $\mathbf{A}/I$ a une structure de produit d'anneaux, d'après la remarque II.5. De plus, aucun des facteurs de ce produit n'est égal à $\mathbf{A}$ ou à l'anneau trivial. Réciproquement, si $\mathbf{A}/I$ a une telle structure alors il existe des idéaux non triviaux $J_1, \ldots, J_n$ de $\mathbf{A}/I$ de produit nul et 2 à 2 comaximaux. Par conséquent, il existe des idéaux $S_1, \ldots, S_n$ de $\mathbf{A}$, distincts de $\mathbf{A}$ et contenant strictement I, de produit égal à I et 2 à 2 comaximaux. Bref, I admet un scindage biunivoque propre et sans composantes égales à $\mathbf{A}$. $\lhd$

Remarque II.6 Si un idéal I d'un anneau $\mathbf{A}$ admet un scindage biunivoque premier, c'est à dire en idéaux premiers, alors $\mathbf{A}/I$ est réduit (i.e. sans nilpotents). En effet, il possède dans ce cas une structure de produit d'anneaux intègres. Inversement, avec $\mathbf{A}$ nœthérien, si I est radical, alors I est intersection d'un nombre fini d'idéaux premiers qui en forment alors un scindage univoque. Cependant, si I n'est pas radical, il peut avoir des scindages biunivoques, comme le montre l'exemple suivant dans $\mathbf{K}[x, y]$ où le scindage est en fait une décomposition primaire minimale de l'idéal de départ :

$$\langle x(x+1), (y+x)(y-1) \rangle \longmapsto (\langle x, y \rangle, \langle x, y-1 \rangle, \langle x+1, (y-1)^2 \rangle)$$

Proposition II.20 *Soit $\mathbf{A}$ un anneau commutatif unitaire et I un idéal de $\mathbf{A}$ distinct de $\mathbf{A}$. Soit a et b deux éléments de $\mathbf{A} \setminus I$. Alors on a :*

$$I \longmapsto (I + a, I + b) \iff ab \in I$$

On suppose $ab \in I$ et $I + \langle a, b \rangle = \mathbf{A}$. Soient a',b' dans $\mathbf{A}$ et c dans I tel que $c + aa' + bb' = 1$. Alors on a :

(*i*) $(I+a, I+b)$ *est un scindage biunivoque de* I.

(*ii*) $(\forall y_1, y_2 \in \mathbf{A})$ $h_S^{-1}(\overline{y_1}^{I+a}, \overline{y_2}^{I+b}) \;=\; \overline{bb'y_1 + aa'y_2}^{I}$

Preuve. $\rhd$ Les propriétés annoncées sont faciles à établir. Vérifions par exemple la dernière. Soient $y_1, y_2 \in \mathbf{A}$. Il vient :

$$\begin{aligned} h_S(\overline{bb'y_1 + aa'y_2}^{I}) &= (\overline{bb'y_1}^{I+a}, \overline{aa'y_2}^{I+b}) \\ &= (\overline{(1-c-aa')y_1}^{I+a}, \overline{(1-c-bb')y_2}^{I+b}) \\ &= (\overline{y_1}^{I+a}, \overline{y_2}^{I+b}) \end{aligned}$$

puisque $c \in I+a$ et $c \in I+b$. $\lhd$

Définition II.7 *Soit* $\mathbf{A}$ *un anneau (commutatif, unitaire) et* I, S_1, S_2 *des idéaux de* $\mathbf{A}$. *On dira que* (S_1, S_2) *est un* scindage de deuxième espèce *de* I *si* (S_1, S_2) *est un scindage de* I *et s'il existe* $a, b \in \mathbf{A}$ *tel que :*

$$S_1 = I + a \quad \text{et} \quad S_2 = I + b$$

Plus généralement, pour des idéaux $S_1, \ldots, S_n$ *de* $\mathbf{A}$, *avec* $n \geq 3$, *on dit que* $(S_1, \ldots, S_n)$ *est un scindage de deuxième espèce de* I *s'il existe un idéal* J *de* $\mathbf{A}$ *et deux entiers* i *et* j *de* $\{1, \ldots, n\}$ *avec* $i < j$ *tels que* $(S_1, \ldots, S_{i-1}, S_{i+1}, \ldots, S_{j-1}, S_{j+1}, \ldots, S_n, J)$ *soit un scindage de deuxième espèce de* I *et* (S_i, S_j) *soit un scindage deuxième espèce de* J.

Remarque II.7 Donnons un exemple de scindage de première espèce et un exemple de scindage de deuxième espèce du même idéal. Dans l'anneau $\mathbf{A} = \mathbf{K}[x, y, z]$ où $\mathbf{K}$ est un corps et où $x > y > z$ on considère l'idéal I suivant :

$$I = \langle y^2 + z, zx^2 + (y + yz)x - z \rangle$$

Les idéaux étant décrits à l'aide de leurs bases lexicographiques, on a le scindage de première espèce suivant (avec $p = z$) :

$$I \longmapsto (\langle z, y^2, yx \rangle, I + \langle yx^2 + (y^2 - 1)x - y \rangle)$$

D'autre part, on a le scindage de deuxième espèce :

$$I \longmapsto (\langle y^2 + z, x + y \rangle, \langle y^2 + z, zx + y \rangle)$$

Les méthodes de résolution de systèmes d'équations algébriques que nous aborderons dans cette thèse utilisent toutes des scindages de première espèce et, pour certaines d'entre elles, des scindages de deuxième espèce. Ces méthodes, qui sont *capables de découvrir des scindages de deuxième espèce*, donnent souvent des résultats *plus exploitables* que celles qui ne le peuvent pas, comme nous le verrons plus loin. Les scindages de deuxième espèce sont également un outil théorique commode, comme le montrent l'énoncé suivant et sa preuve.

Théorème II.6 *Soit* $\mathbf{A}$ *un anneau nœthérien et* I *un idéal non premier de* $\mathbf{A}$. *Alors, il existe des idéaux premiers* $P_1, \ldots, P_n$ *tel que* $I \longmapsto (P_1, \ldots, P_n)$ *soit un scindage de deuxième espèce. Si* $\mathbf{A}$ *est artinien et si* I *est radical, alors tout scindage minimal extrait du précédent est biunivoque et* $\mathbf{A}/I$ *est un produit de corps.*

Preuve. $\triangleright$ Puisque I n'est pas premier, I admet un scindage binaire de deuxième espèce. En remplaçant chaque composante qui n'est pas un idéal premier par un scindage binaire de deuxième espèce, on obtient des suites strictement croissantes d'idéaux. Comme $\mathbf{A}$ est nœtherien chacune de ces suites doit être finie et donc l'un de leurs termes doit être un idéal premier. On a ainsi construit un arbre binaire orienté (par l'inclusion stricte) dont toutes les chaînes (et en particulier celles qui sont maximales) sont finies. Cet arbre est donc fini et ses nœuds sont tous des idéaux premiers $P_1, \ldots, P_n$ qui constituent un scindage de deuxième espèce de I, que l'on peut de plus supposer minimal, quitte à supprimer les composantes redondantes. Si $\mathbf{A}$ est artinien alors les idéaux P_i sont maximaux. Si $\mathbf{A}/I$ est réduit alors les composantes du scindage $I \longmapsto (P_1, \ldots, P_n)$ sont 2 à 2 distinctes. Par suite, sous les deux hypothèses, les composantes P_i sont des idéaux 2 à 2 comaximaux et ce scindage est biunivoque. $\triangleleft$

Proposition II.21 *Soit* $\mathbf{A}$ *un anneau (commutatif, unitaire) et* I *un idéal non trivial de* $\mathbf{A}$. *Alors, on a :*

(i) *si tout scindage de deuxième espèce de* I *est biunivoque, alors* I *est radical.*

(ii) *si* I *possède des scindages de deuxième espèce biunivoques, alors* I *n'est pas primaire.*

Preuve. $\triangleright$ Vérifions (i). Supposons tout d'abord que tout scindage fondamental de I est biunivoque. Si I n'est pas radical alors il existe $a \in \mathbf{A}$ et un entier $n \geq 2$ tel que : $a^n \in I$ et $a^{n-1} \notin I$. Le scindage $I \longmapsto (I + \langle a^{n-1} \rangle, I + a)$ étant biunivoque, on a :

$$(I + \langle a^{n-1} \rangle) \; + \; (I + a) \;\; = \;\; I + a \;\; = \;\; \mathbf{A}$$

Par suite :

$$(I + \langle a^{n-1} \rangle)(I + a) \;\; = \;\; I + \langle a^{n-1} \rangle \;\; = \;\; I$$

Ce qui est absurde. Vérifions (ii). Supposons qu'il existe a et b dans $\mathbf{A} \setminus I$ tels que

$$ab \in I \quad \text{et} \quad \langle a, b \rangle \; = \; \mathbf{A}$$

Soient alors a' et b' dans $\mathbf{A}$ tels que : $aa' + bb' = 1$ Si I est primaire, alors on peut supposer que a est nilpotent dans $\mathbf{A}/I$. Par suite bb' est la somme d'une unité et d'un nilpotent dans $\mathbf{A}/I$. C'est donc unité (prop. III.1 p. 75) D'où $0 = \overline{ab}^I = \overline{abb'}^I = \overline{a}^I$. Absurde. $\triangleright$

Remarque II.8 Les réciproques des deux implications de la proposition II.21 sont fausses. Considérer dans $A = \mathbf{k}[x, y, z]$ l'idéal $I = \langle xy, yz, zx \rangle$. Cet idéal est non primaire et radical. Tous les scindages de deuxième espèce de I ne sont pas biunivoques. En effet, soit $a, b \in \mathbf{A} \setminus I$ tel que : $ab \in I$. On peut supposer a et b réduits par rapport à I vue comme une base de Gröbner pour l'ordre du degré. De plus, les trois variables ayant des rôles similaires, on peut en fait supposer $a \in \mathbf{k}[x]$ et $b \in \mathbf{k}[y]$. Comme $ab \in I$, les polynômes a et b (sont de degré non nul et) ne peuvent avoir de termes constants. Pour que $I + a$ et $I + b$ soient maximaux pour l'inclusion avec les contraintes précédentes, on choisira $a = x$ et $b = y$. Mais

$$(I + \langle x \rangle) \; + \; (I + \langle y \rangle) \;\; = \;\; \langle x, y \rangle \neq \mathbf{A}$$

En fait, si l'on suppose $\mathbf{A}$ nœtherien et I de dimension zéro, alors la réciproque de la seconde implication de la proposition II.21 devient vraie.

Proposition II.22 *Soit* $\mathbf{A}$ *un anneau nœthérien et* I *un idéal non trivial de* $\mathbf{A}$*. Si* I *est de dimension zéro et non primaire, alors toute décomposition primaire minimale de* I *est un scindage biunivoque de* I*.*

Preuve. $\triangleright$ Soit $I \longmapsto Q = (Q_1, \ldots, Q_n)$ une décomposition primaire minimale de l'idéal I supposé non primaire. Posons $P_i = \sqrt{Q_i}$. Puisque Q est minimale les P_i sont 2 à 2 distincts. L'idéal I étant de dimension 0, les P_i sont alors maximaux, ils sont aussi 2 à 2 comaximaux. Comme une puissance de P_i est contenue dans Q_i, les Q_i sont aussi 2 à 2 comaximaux et Q est un scindage biunivoque de I. $\triangleleft$

3 Pgcd de polynômes univariés

Notations II.4 Soit $\mathbf{A}$ un anneau commutatif unitaire. Soient $p, q \in \mathbf{A}[X]$ avec $p \neq 0$. On notera $\mathsf{lc}(p)$ le coefficient du monôme de plus haut degré de p et $\mathsf{deg}(p)$ ce degré. Pour $\mathsf{deg}(p) > 0$, on désignera par $\mathsf{prem}(q,p)$ et $\mathsf{pquo}(q,p)$ le pseudo-reste et le pseudo-quotient de q par p. Rappelons que ces polynômes sont définis par :

(*i*) $\delta = \mathsf{max}(0, \mathsf{deg}(q) - \mathsf{deg}(p) + 1)$.

(*ii*) $\mathsf{lc}(p)^{\delta} q = \mathsf{pquo}(q,p)\, p \,+\, \mathsf{prem}(q,p)$

(*iii*) $\mathsf{prem}(q,p) \neq 0 \implies \mathsf{deg}(\mathsf{prem}(q,p), X) < \mathsf{deg}(p)$

Définition II.8 *Soit* $\mathbf{A}$ *un anneau commutatif, unitaire et* p *un polynôme non nul de* $\mathbf{A}[X]$*. On dira que* p *est* quasi-unitaire *(resp.* unitaire*) si* $\mathsf{lc}(p) \in \mathbf{reg}(\mathbf{A})$ *(si* $\mathsf{lc}(p) \in \mathbf{Un}(A)$*).*

Définition II.9 ([MR95]) *Soient* $\mathbf{A}$ *un anneau commutatif unitaire et* p_1, p_2, g *dans* $\mathbf{A}[X]$*. On dira que* g *est un pgcd de* p_1 *et* p_2 *si l'on a :*

$$\langle p_1, p_2 \rangle_{\mathbf{fr}(\mathbf{A})[X]} \;=\; \langle g \rangle_{\mathbf{fr}(\mathbf{A})[X]}$$

Remarque II.9 Si $\mathbf{A}$ est un corps, on retrouve la *notion habituelle* de pgcd dans $\mathbf{A}[X]$. Si $\mathbf{A}$ est factoriel, g est un pgcd au *sens usuel* si et seulement si g est un pgcd au sens de la définition précédente qui divise p_1 et p_2 dans $\mathbf{A}[X]$. Si $\mathbf{fr}(\mathbf{A})[X]$ n'est pas principal, p_1 et p_2 ne possèdent pas nécessairement de pgcd au sens de la définition précédente. Le choix de cette définition a pour but de donner un cadre général aux algorithmes de calcul de pgcd qui fournissent une *relation de Bezout* et un *pseudo-diviseur*. Le prototype d'un tel algorithme est celui dit *des sous-résultants*. De plus, cette définition sera justifiée par les propositions II.24 et II.25.

Proposition II.23 *Soient* $\mathbf{A}$ *un anneau commutatif unitaire et* p_1, p_2, g, g' *dans* $\mathbf{A}[X]$*. Supposons que* g *et* g' *soient des pgcd de* p_1 *et* p_2*. Si* g *et* g' *sont quasi-unitaire alors il existe* $e, e' \in \mathbf{reg}(\mathbf{A})$ *tels que :*

$$e\, g \;=\; e'\, g'$$

Preuve. $\triangleright$ En effet, puisque g et g' sont des pgcd de p_1 et p_2, on a :

$$\langle g \rangle_{\mathbf{fr}(\mathbf{A})[X]} = \langle g' \rangle_{\mathbf{fr}(\mathbf{A})[X]}$$

Il existe donc f et f' dans $\mathbf{fr}(\mathbf{A})[X]$ tel que :

$$g' = f\,g \text{ et } g = f'\,g'$$

Comme les coefficients dominants de g et g' sont dans $\mathbf{reg}(\mathbf{A})$ on en déduit :

$$\mathsf{deg}(g') \geq \mathsf{deg}(g) \text{ et } \mathsf{deg}(g) \geq \mathsf{deg}(g')$$

Ainsi, les polynômes g et g' ont le même degré. Par suite, f et f' sont des unités de $\mathbf{fr}(\mathbf{A})$, c'est-à-dire des fractions à numérateur et dénominateur dans $\mathbf{reg}(\mathbf{A})$. D'où le résultat. $\triangleleft$

Remarque II.10 On suppose que l'on dispose d'une application ϕ de $\mathbf{A}[X] \times \mathbf{A}[X]$ dans $\mathbf{A}[X]$ tel que pour tout p_1 et tout p_2 non nuls de $\mathbf{A}[X]$ il existe une suite finie $r_0, r_1, \ldots, r_n, r_{n+1}$ d'éléments de $\mathbf{A}[X]$ tel que l'on ait :

(i) $r_0 = p_1$ et $r_1 = p_2$

(ii) $(\forall i \leq n)\ \phi(r_{i-1}, r_i) = r_{i+1}$

(ii) $(\forall i \leq n)\ \langle r_{i-1}, r_i \rangle_{\mathbf{fr}(\mathbf{A})[X]} = \langle r_i, r_{i+1} \rangle_{\mathbf{fr}(\mathbf{A})[X]}$

(iv) $(\forall i \leq n)\ r_i \neq 0$ et $r_{n+1} = 0$

La suite $(r_0, r_1, \ldots, r_n, r_{n+1})$ est dite *suite des restes* de p_1 et p_2 calculée par ϕ. Il est clair que r_n est un pgcd de p_1 et p_2. Lorsque $\mathbf{A}$ est un corps, on peut choisir pour ϕ le reste dans la division euclidienne, c'est l'algorithme d'Euclide. Lorsque $\mathbf{A}$ est intègre, on peut choisir pour ϕ le reste dans la pseudo-division (notation II.4), si l'on ne souhaite pas travailler dans $\mathbf{fr}(\mathbf{A})[X]$. Mais alors la croissance en longueur des coefficients des polynômes r_i peut être exponentielle. On obtient une croissance linéaire de ces coefficients, en *modifiant* ϕ de la manière suivante. On pose $f_i = \mathsf{lc}(r_i)$ et $\delta_i = \mathsf{deg}(r_i) - \mathsf{deg}(r_{i+1})$ pour $0 \leq i \leq n$. De plus, on introduit deux suites $(\psi_i)_{i \geq 2}$ et $(\beta_i)_{i \geq 2}$ dans $\mathbf{reg}(\mathbf{A})$ données par :

$$\psi_2 = -1 \text{ et } \psi_i = (-f_{i-2})^{\delta_{i-2}} {\psi_{i-1}}^{1-\delta_{i-3}}$$

et :

$$\beta_2 = (-1)^{\delta_0+1} \text{ et } \beta_i = -f_{i-2} {\psi_i}^{\delta_{i-2}}$$

On définit alors $\phi(r_{i-1}, r_i)$ par :

$$f_i^{\delta_{i-1}+1}\, r_{i-1} = q_{i+1}\, r_i + \beta_{i+1}\, r_{i+1}$$

où q_{i+1} et $\beta_{i+1}\, r_{i+1}$ sont respectivement le quotient et le reste de le pseudo-division de r_{i-1} par r_i. On démontre que chaque ψ_i et r_{i+1} appartiennent bien à $\mathbf{A}$ et $\mathbf{A}[X]$ respectivement. C'est l'algorithme dit des *sous-résultants* ([Loo82]). Une propriété importante de cet algorithme est la suivante. Soit $\mathbf{B}$ un second anneau intègre et F un homomorphisme d'anneaux de $\mathbf{A}$ dans $\mathbf{B}$. Soit $({r'}_0, {r'}_1, \ldots, {r'}_m, {r'}_{m+1})$ la suite des restes de $F(p_1)$ et $F(p_2)$ calculée par l'algorithme des sous-résultants dans $\mathbf{B}[X]$. Alors, $F(r_i)$ est proportionnel à l'un des ${r'}_j$ ou est nul. C'est la propriété de *spécialisation* de l'algorithme des sous-résultants.

Proposition II.24 *Soit* $\mathbf{A}$ *un anneau (commutatif, unitaire) tel que* $\mathbf{A}$ *soit isomorphe à l'anneau-produit* $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$*. Soient* p*,* q *des éléments de* $\mathbf{A}[X]$*. Soit* h_i *la projection* $\mathbf{A}$ *dans* $\mathbf{A}_i$ *et* f_i *l'homomorphisme de* $\mathbf{A}[X]$ *dans* $\mathbf{A}_i[X]$ *prolongeant* h_i *en posant* $f_i(X) = X$*. Alors, on a :*

(i) *si pour chaque* i *les polynômes* $f_i(p)$ *et* $f_i(q)$ *possèdent des pgcd dans* $\mathbf{A}_i[X]$ *alors* p *et* q *possèdent des pgcd dans* $\mathbf{A}[X]$*,*

(ii) *si pour chaque* i *les polynômes* $f_i(p)$ *et* $f_i(q)$ *possèdent des pgcd quasi-unitaires tous de même degré dans* $\mathbf{A}_i[X]$ *alors* p *et* q *possèdent des pgcd quasi-unitaires dans* $\mathbf{A}[X]$*,*

(iii) *si tous les* $\mathbf{A}_i$ *sont intègres, tout couple* (p, q) *de polynômes de* $\mathbf{A}[X]$ *possèdent des pgcd.*

(iv) *si les* $\mathbf{A}_i$ *sont intègres et nœthériens, alors tous les idéaux de* $\mathbf{fr}(\mathbf{A})[X]$ *sont principaux.*

Preuve. $\triangleright$ De (i) on déduit (iii), puis de (iii) on déduit (iv). Vérifions (i) et (ii). Pour $a \in \mathbf{A}$ on pose $h(a) = (h_1(a), \ldots, h_n(a))$ D'après la preuve de la proposition II.6, h est un isomorphisme. On considère l'application f de $\mathbf{A}[X]$ dans $\mathbf{A}_1[X] \times \cdots \times \mathbf{A}_n[X]$ muni de la structure d'anneau produit et définie par $f(r) = (f_1(r), \ldots, f_n(r))$. On vérifie que f est un isomorphisme. On pose maintenant $p_i = f_i(p)$ et $q_i = f_i(q)$, et on suppose qu'il existe g_i dans $\mathbf{A}_i[X]$ qui soit un pgcd de p_i et q_i c'est à dire tel que :

$$\langle p_i, q_i \rangle_{\mathbf{fr}(\mathbf{A}_i)[X]} \;=\; \langle g_i \rangle_{\mathbf{fr}(\mathbf{A}_i)[X]}$$

Il existe donc des polynômes u_i, v_i, ${p'}_i$ et ${q'}_i$ dans $\mathbf{A}_i[X]$ et des nombres α_i, β_i et γ_i dans $\mathbf{reg}(\mathbf{A}_i)$ le tout tel que l'on ait :

$$u_i p_i + v_i q_i = \alpha_i g_i \quad \text{et} \quad \beta_i p_i = {p'}_i g_i \quad \text{et} \quad \gamma_i q_i = {q'}_i g_i$$

En posant $g = f^{-1}(g_1, \ldots, g_n)$, il vient :

$$h^{-1}(\alpha_1, \ldots, \alpha_n) g = f^{-1}(u_1, \ldots, u_n) p + f^{-1}(v_1, \ldots, v_n) q$$

De même :

$$h^{-1}(\beta_1, \ldots, \beta_n) p = f^{-1}({p'}_1, \ldots, {p'}_n) g \quad \text{et} \quad h^{-1}(\gamma_1, \ldots, \gamma_n) q = f^{-1}({q'}_1, \ldots, {q'}_n) g$$

Remarquons que, pour $a \in \mathbf{A}$, si pour $1 \le i \le n$ on a : $h_i(a) \in \mathbf{reg}(\mathbf{A}_i)$ alors on doit avoir : $a \in \mathbf{reg}(\mathbf{A})$. Finalement, g d'une part, et, p et q d'autre part, engendrent bien le même idéal dans $\mathbf{fr}(\mathbf{A})[X]$; de plus, si les polynômes g_i sont quasi-unitaires tous de même degré, il en va de même du polynôme g. $\triangleleft$

Proposition II.25 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire et* I *un idéal de* $\mathbf{A}$ *intersection finie d'idéaux maximaux. Alors, tout couple de polynômes de* $(\mathbf{A}/I)[X]$ *admet des pgcd. Ceci est en particulier le cas si* $\mathbf{A}$ *est un anneau de polynômes sur un corps* $\mathbf{k}$ *et si* I *est un idéal radical de dimension zéro.*

Preuve. $\triangleright$ En effet, d'après le théorème II.6, l'anneau $\mathbf{A}/I$ est (un corps ou) un produit de corps. On conclut alors avec la proposition II.24 $\triangleleft$

Définition II.10 ([MR95]) *Soit* $\mathbf{A}$ *un anneau intègre,* f *une application de* $\mathbf{reg}(\mathbf{A})$ *dans* $\mathbf{reg}(\mathbf{A})$ *et* p_1, p_2 *deux polynômes de* $\mathbf{A}[X]$*. On considère l'algorithme suivant, appelé* New-SubResGcd *défini comme suit, la syntaxe étant celle du langage* AXIOM*([JS92]) :*

```
NewSubResGcd(p1,p2, [f :(reg(A) ↦ reg(A))] ) ==
  if deg(p2) > deg(p1) then (p1,p2) := (p2,p1)
  δ := deg(p1) - deg(p2)
  ψ' : A := -1 ; β' :A := −1^(δ+1)
  [p2 := f(lc(p2))*p2]
  α' :A := lc(p2)
  while (deg(p2) > 0) repeat
     p3 := prem(p1,p2) exquo β'
     [p3 := f(lc(p3))*p3]
     if δ > 0 then
        ψ' := (−α')^δ exquo ψ'^(δ−1)
     (p1 , p2) := (p2 , p3)
     if (deg(p2) > 0) then
        δ := deg(p1) - deg (p2)
        β' := -α' ψ'^δ
        α' := lc(p2)
  if p2 = 0 then return p1 else return p2
```

Remarque II.11 L'unique différence avec l'algorithme original de calcul de pgcd fondé sur les sous-résultants ([Loo82]) réside en le fait que chaque polynôme de la séquence, sauf le premier polynôme, est multiplié par une constante obtenue en appliquant f à son coefficient dominant. Notons que si $f(x) = 1$ alors la séquence calculée est exactement celle de l'algorithme des sous-résultants. De même si l'on choisit $f(x) = 1/x$ et si $\mathbf{A} = \mathbf{fr}(\mathbf{A})$, alors on retrouve l'algorithme d'Euclide dans $\mathbf{fr}(\mathbf{A})[X]$ où tous les polynômes de la séquence, sauf le premier, ont 1 pour coefficient dominant.

Proposition II.26 ([MR95]) *Soit* $\mathbf{A}$ *un anneau intègre. Pour toute application* $f : \mathbf{reg}(\mathbf{A}) \to \mathbf{reg}(\mathbf{A})$*, l'algorithme NewSubResGcd se termine et calcule un pgcd des polynômes* d'entrée p_1 *et* p_2 *au sens de la définition II.9.*

Preuve. $\triangleright$ La proposition suivante exprime le fait que l'algorithme se déroule dans $\mathbf{A}[X]$. Plus précisément, toutes les divisions dans $\mathbf{A}$ sont exactes. D'autre part, il est clair que l'algorithme se termine et calcule bien un pgcd au sens de la définition II.9. En effet, la remarque II.10 s'applique à la séquence de polynômes calculée par l'algorithme *NewSubResGcd.* $\triangleleft$

Proposition II.27 *Soient* F'_k, α'_k, β'_k *and* ψ'_k *les termes calculés par l'algorithme NewSubResGcd et définis pour* $k \geq 3$ *comme suit :* $F'_1 = p_1$, $\lambda_2 = f(\mathsf{lc}(p_2))$, $F'_2 = \lambda_2 p_2$, ${\alpha'}_2 = \mathsf{lc}(F'_2)$, $\delta_{k-2} = \mathsf{deg}(F'_{k-2}) - \mathsf{deg}(F'_{k-1})$, $\psi'_3 = -1$, $\beta'_3 = (-1)^{\delta_1+1}$, $\lambda_k = f(\mathsf{lc}(\mathsf{prem}(F'_{k-2}, F'_{k-1})))$, $\lambda_k \mathsf{prem}(F'_{k-2}, F'_{k-1}) = \beta'_k F'_k$, $\psi'_{k+1} = \dfrac{(-\alpha'_{k-1})^{\delta_{k-2}}}{{\psi'_k}^{\delta_{k-2}-1}}$, $\beta'_{k+1} = -\alpha'_{k-1} {\psi'_{k+1}}^{\delta_{k-1}}$, *et* ${\alpha'}_k = \mathsf{lc}(F'_k)$. *Alors, les polynômes* F'_k *appartiennent à* $\mathbf{A}[X]$ *et les nombres* ψ'_k *à* $\mathbf{A}$.

Preuve. ▷ Soient $F_1, F_2, \ldots, F_i, \ldots$ les polynômes calculés par l'algorithme (original) des sous-résultants et α_i le coefficient dominant de F_i. On pose aussi : $\beta_3 = (-1)^{\delta_1+1}$ et $\beta_{i+1} = -\alpha_{i-1}\psi_{i+1}^{\delta_{i-1}}$ où $\psi_3 = -1$ et $\psi_{i+1} = \dfrac{(-\alpha_{i-1})^{\delta_{i-2}}}{\psi_i^{\delta_{i-2}-1}}$. Les β'_i et ψ'_i vérifient les mêmes relations de récurrence où α'_i (le coefficient dominant de F'_i) remplace α_i. Remarquons tout d'abord que les polynômes F_i et F'_i sont associés lorsqu'ils sont vus comme des polynômes de $\mathbf{fr}(\mathbf{A})[X]$. Examinons les premiers pas de l'alhorithme. Nous avons $F'_2 = \lambda_2 F_2$, $\alpha'_2 = \lambda_2\alpha_2$, $\psi'_3 = \psi_3$, $\beta'_3 = \beta_3$. On peut écrire : $\mathsf{prem}(F'_1, F'_2) = \mathsf{prem}(F_1, \lambda_2 F_2) = \lambda_2^{\delta_1+1}\mathsf{prem}(F_1, F_2)$ Et par suite :

$$F'_3 = \lambda_3\lambda_2^{\delta_1+1}F_3$$

puisque $\beta'_3 = \beta_3$. Regardons maintenant les choses d'une façon générale. Posons :

$$F'_i = \mu_i F_i$$

En travaillant a priori dans le corps des fractions de $\mathbf{A}$, nous avons :

$$\begin{aligned}
\lambda_{i+2}\mathsf{prem}(F'_i, F'_{i+1}) &= \beta'_{i+2}F'_{i+2} \\
&= \beta'_{i+2}\mu_{i+2}F_{i+2} \\
&= \lambda_{i+2}\mathsf{prem}(\mu_i F_i, \mu_{i+1}F_{i+1}) \\
&= \lambda_{i+2}\mu_i\mu_{i+1}^{\delta_i+1}\beta_{i+2}F_{i+2}
\end{aligned}$$

Et par suite :

$$\beta'_{i+2}\mu_{i+2} = \lambda_{i+2}\mu_i\mu_{i+1}^{\delta_i+1}\beta_{i+2}$$

Faisant intervenir les ψ au lieu des β et remarquant que $\alpha'_i = \mu_i\alpha_i$, nous sommes conduits à :

$$\mu_{i+2}{\psi'}_{i+2}^{\delta_i} = \lambda_{i+2}\mu_{i+1}^{\delta_i+1}\psi_{i+2}^{\delta_i} \tag{II.1}$$

Que l'on réécrit aussi comme suit :

$$\frac{\mu_{i+2}}{\mu_{i+1}} = \lambda_{i+2}\left[\frac{\mu_{i+1}\psi_{i+2}}{{\psi'}_{i+2}}\right]^{\delta_i}. \tag{II.2}$$

D'autre part, du fait de la relation de récurrence sur les ψ_i, nous avons :

$$\begin{aligned}
\frac{\mu_{i+1}\psi_{i+2}}{{\psi'}_{i+2}} &= \mu_{i+1}\left[\frac{\alpha_i}{\alpha'_i}\right]^{\delta_{i-1}}\left[\frac{{\psi'}_{i+1}}{\psi_{i+1}}\right]^{\delta_{i-1}-1} \\
&= \left[\frac{{\psi'}_{i+1}}{\psi_{i+1}}\right]^{\delta_{i-1}-1}\frac{\mu_{i+1}}{\mu_i^{\delta_{i-1}}}
\end{aligned} \tag{II.3}$$

En effectuant la substitution $i \to i-1$ dans (II.2) nous trouvons :

$$\left[\frac{\psi_{i+1}}{{\psi'}_{i+1}}\right]^{\delta_{i-1}} = \frac{\lambda_{i+1}\mu_i^{\delta_{i-1}+1}}{\mu_{i+1}}$$

équation que l'on injecte dans (II.3) pour obtenir :

$$\frac{\mu_{i+1}\psi_{i+2}}{\psi'_{i+2}} = \frac{\lambda_{i+1}\mu_i^{\delta_{i-1}+1}}{\mu_{i+1}}\mu_{i+1}\left[\frac{\psi_{i+1}}{\mu_i\psi'_{i+1}}\right]^{\delta_{i-1}} = \lambda_{i+1}\frac{\mu_i\psi_{i+1}}{\psi'_{i+1}}$$

Posons :

$$u_i = \frac{\mu_i\psi_{i+1}}{\psi'_{i+1}}$$

Puisque $\mu_2 = u_2 = \lambda_2$, il vient :

$$u_i = \prod_{j=2}^{j=i} \lambda_j$$

Pour établir que les F'_i restent dans $\mathbf{A}[X]$ il suffit de vérifier que les μ_i appartiennent à $\mathbf{A}$. En utilisant la valeur de u_i induite par (II.2), nous obtenons :

$$\frac{\mu_{i+2}}{\mu_{i+1}} = \lambda_{i+2}{u_{i+1}}^{\delta_i} \tag{II.4}$$

Par suite, μ_i reste dans $\mathbf{A}$ et est divisible par u_i. Finalement, puisque $\psi_i \in \mathbf{A}$ nous en déduisons que les ψ'_i restent aussi dans $\mathbf{A}$ ce qui termine la preuve. $\triangleleft$

Chapitre III

Ensembles triangulaires et tours d'extensions simples

Introduction

Soit $\mathbf{k}$ un corps et $\mathbf{K}$ un sur-corps de $\mathbf{k}$. Lorsque l'on résout dans $\mathbf{K}$ un système d'équations algébriques donné par une famille de polynômes à coefficients dans $\mathbf{k}$, on est souvent amené à introduire des *corps intermédiaires* entre $\mathbf{k}$ et $\mathbf{K}$ pour décrire les solutions de ce système. En pratique, et en particulier dans les algorithmes présentés dans cette thèse, ces corps intermédiaires n'apparaissent pas explicitement et il est souvent commode de travailler avec des sur-anneaux de $\mathbf{k}$ non nécessairement intègres. Ces sur-anneaux sont construits en faisant jouer aux indéterminées du système, l'une après l'autre, un rôle particulier. Cette idée est précisée dans la notion de *tours d'extensions simples*. En outre, ces sur-anneaux sont représentés par des ensembles de polynômes, les *ensembles triangulaires* indiquant le rôle de chacune des indéterminées du système. Pour illustrer cette remarque, commençons par un exemple très simple, encore une fois tiré de notre système d'équations favori, celui de l'introduction des deux chapitres précédents. Nous avons vu dans l'introduction du second chapitre qu'en distinguant les cas $z = 0$ et $z \neq 0$, les solutions (z, y, x) de ce système pouvaient s'écrire :

$$\left\{ \begin{array}{rcl} z^2 & = & 0 \\ y^2 - y + z & = & 0 \\ x + y - 1 & = & 0 \end{array} \right. \quad \text{ou} \quad \left\{ \begin{array}{rcl} z^4 - 4z^2 + 4z - 1 & = & 0 \\ 2y + z^2 - 1 & = & 0 \\ 2x + z^2 - 1 & = & 0 \end{array} \right.$$

Cette décomposition est du reste le résultat brut produit par l'un des algorithmes que nous étudierons plus loin. Examinons le second système produit. Que peut-on dire de ses solutions ? A quels ensembles de nombres appartiennent les z, y et x solutions de ce système ? Ils appartiennent bien sr au corps $\mathbb{C}$ des nombres complexes. Peut-on les situer dans un plus petit corps intermédiaire entre $\mathbb{Q}$, le corps des coefficients des polynômes, et $\mathbb{C}$? Oui, mais cela suppose de factoriser le polynôme définissant z en facteurs irréductibles. Ceci est aisé sur cet exemple, mais peut l'être beaucoup moins sur d'autres exemples. Cette factorisation faite, il faut ensuite factoriser le polynôme définissant y sur chacun des corps définis pas la factorisation du polynôme en z, et ainsi de suite. Ceci est immédiat sur cet exemple, mais peut s'avérer fort long sur d'autres exemples. De plus, ces factorisations successives peuvent allonger considérablement la taille de l'ensemble des solutions. Voici un exemple. Le corps des coefficients étant toujours le corps $\mathbb{Q}$ des nombres rationnels, on considère le système suivant :

$$\begin{aligned} a+b+c+d+e &= 0 \\ ab+bc+cd+de+ea &= 0 \\ abc+bcd+cde+dea+eab &= 0 \\ abcd+bcde+cdea+deab+eabc &= 0 \\ abcde &= 1 \end{aligned}$$

La décomposition produite sur ce nouveau système par le même algorithme que celui mentionné plus haut est :

$$\begin{aligned} T_1 &= \{e^5-1,\ d-e,\ c-e,\ b^2+3\ e\ b+e^2,\ a+b+3\ e\} \\ T_2 &= \{e^5-1,\ d-e,\ c^2+3\ e\ c+e^2,\ b+c+3\ e,\ a-e\} \\ T_3 &= \{e^5-1,\ d^6+4\ e\ d^5+5\ e^2\ d^4+5\ e^3\ d^3+5\ e^4\ d^2+4\ d+e, \\ &\quad 5\ c+8\ e\ d^5+30\ e^2\ d^4+30\ e^3\ d^3+25\ e^4\ d^2+30\ d+22\ e, \\ &\quad 5\ b-2\ e\ d^5-10\ e^2\ d^4-15\ e^3\ d^3-10\ e^4\ d^2-10\ d-8\ e, \\ &\quad 5\ a-6\ e\ d^5-20\ e^2\ d^4-15\ e^3\ d^3-15\ e^4\ d^2-15\ d-9\ e\} \\ T_4 &= \{e^{10}+123\ e^5+1,\ 55\ d^2+(-2\ e^6-233\ e)\ d-8\ e^7-987\ e^2, \\ &\quad 55\ c+e^6+144\ e,\ 55\ b+e^6+144\ e,\ 55\ a+55\ d-2\ e^6-233\ e\} \end{aligned}$$

où chaque T_i doit être compris comme l'ensemble des membres de gauche d'un système d'équations dont les membres de droite sont tous nuls. Les solutions du système initial sont alors la réunion des solutions des T_i. Autrement dit, si F désigne l'ensemble des polynômes définissant le système initial on a :

$$\mathbf{V}(F) = \cup_{i=1}^{4} \mathbf{V}(T_i)$$

Le temps de calcul de cette décomposition est de 20 secondes dans nos programmes écrits dans la version 2.0 du logiciel AXIOM ([JS92]) et exécutés sur une machine SPARC 10. Supposons maintenant que l'on souhaite passer à la forme factorisée mentionnée plus haut. Pour chaque T_i, on calcule donc une factorisation en facteurs irréductibles dans $\mathbb{Q}[e]$ du polynôme en e. Ainsi pour T_1, le polynôme en e se factorise comme suit :

$$(e-1)\left(e^4+e^3+e^2+e+1\right)$$

Notons que l'on factorise dans $\mathbb{Q}[e]$ et que l'on ne va pas jusqu'à la résolution en facteurs linéaires ! Chaque facteur f d'une telle décomposition définit une extension simple de $\mathbb{Q}$, à savoir $\mathbb{Q}[e]/\langle f\rangle$, qui est un corps. Dans chaque $(\mathbb{Q}[e]/\langle f\rangle)[d]$ on peut ensuite factoriser en facteurs irréductibles le polynôme en d de T_i. Chacun de ces s facteurs irréductibles permet de définir une extension simple du corps $\mathbb{Q}[e]/\langle f\rangle$. On continue ce processus avec les variables c, b et a. et l'on obtient la décomposition suivante en 400 secondes dans les mêmes conditions que précédemment :

$$
\begin{array}{rcl}
S_1 & = & \{e-1,\ d-1,\ c-1,\ b^2+3\ b+1,\ a+b+3\} \\
S_2 & = & \{e^4+e^3+e^2+e+1,\ d-e,\ c-e,\ b+e^2+2\ e+1,\ a-e^2+e-1\} \\
S_3 & = & \{e^4+e^3+e^2+e+1,\ d-e,\ c-e,\ b-e^2+e-1,\ a+e^2+2\ e+1\} \\
S_4 & = & \{e-1,\ d-1,\ c^2+3\ c+1,\ b+c+3,\ a-1\} \\
S_5 & = & \{e^4+e^3+e^2+e+1,\ d-e,\ c+e^2+2\ e+1,\ b-e^2+e-1,\ a-e\} \\
S_6 & = & \{e^4+e^3+e^2+e+1,\ d-e,\ c-e^2+e-1,\ b+e^2+2\ e+1,\ a-e\} \\
S_7 & = & \{e-1,\ d^2+3\ d+1,\ c+d+3,\ b-1,\ a-1\} \\
S_8 & = & \{e-1,\ d^4+d^3+d^2+d+1,\ c-d^2,\ b-d^3,\ a+d^3+d^2+d+1\} \\
S_9 & = & \{e^4+e^3+e^2+e+1,\ d-1,\ c+e^3+e^2+e+1,\ b-e^3,\ a-e^2\} \\
S_{10} & = & \{e^4+e^3+e^2+e+1,\ d-e^2+e-1,\ c+e^2+2\ e+1,\ b-e,\ a-e\} \\
S_{11} & = & \{e^4+e^3+e^2+e+1,\ d-e^3,\ c-1,\ b-e^2,\ a+e^3+e^2+e+1\} \\
S_{12} & = & \{e^4+e^3+e^2+e+1,\ d+e^3+e^2+e+1,\ c-e^2,\ b-1,\ a-e^3\} \\
S_{13} & = & \{e^4+e^3+e^2+e+1,\ d-e^2,\ c-e^3,\ b+e^3+e^2+e+1,\ a-1\} \\
S_{14} & = & \{e^4+e^3+e^2+e+1,\ d+e^2+2\ e+1,\ c-e^2+e-1,\ b-e,\ a-e\} \\
S_{15} & = & \{e^2+3\ e+1,\ d-1,\ c-1,\ b-1,\ a+e+3\} \\
S_{16} & = & \{e^2+3\ e+1,\ d+e+3,\ c-1,\ b-1,\ a-1\} \\
S_{17} & = & \{e^4-4\ e^3+6\ e^2+e+1,\ 11\ d+3\ e^3-13\ e^2+26\ e-2, \\
 & & 11\ c+3\ e^3-13\ e^2+26\ e-2,\ 11\ b+3\ e^3-13\ e^2+26\ e-2, \\
 & & 11\ a-9\ e^3+39\ e^2-67\ e+6\} \\
S_{18} & = & \{e^4-4\ e^3+6\ e^2+e+1,\ 11\ d-9\ e^3+39\ e^2-67\ e+6, \\
 & & 11\ c+3\ e^3-13\ e^2+26\ e-2,\ 11\ b+3\ e^3-13\ e^2+26\ e-2, \\
 & & 11\ a+3\ e^3-13\ e^2+26\ e-2\} \\
S_{19} & = & \{e^4+e^3+6\ e^2-4\ e+1,\ 11\ d-9\ e^3-15\ e^2-64\ e-3, \\
 & & 11\ c+3\ e^3+5\ e^2+25\ e+1,\ 11\ b+3\ e^3+5\ e^2+25\ e+1, \\
 & & 11\ a+3\ e^3+5\ e^2+25\ e+1\} \\
S_{20} & = & \{e^4+e^3+6\ e^2-4\ e+1,\ 11\ d+3\ e^3+5\ e^2+25\ e+1, \\
 & & 11\ c+3\ e^3+5\ e^2+25\ e+1,\ 11\ b+3\ e^3+5\ e^2+25\ e+1, \\
 & & 11\ a-9\ e^3-15\ e^2-64\ e-3\}
\end{array}
$$

Cette décomposition doit être comprise de manière analogue à la précédente. Autrement dit on a :

$$\mathbf{V}(F) = \cup_{i=1}^{20} \mathbf{V}(S_i)$$

La décomposition à l'aide des ensembles T_i est donc plus concise et plus rapidement calculable que la décomposition à l'aide des ensembles S_i. Cependant les ensembles S_i ont des propriétés que n'ont pas les ensembles T_i. En effet, chacun des S_i définit un corps par le processus décrit plus haut. De plus, chacun des S_i engendre un idéal premier dans $\mathbb{Q}[e, d, c, b, a]$. Par suite, les S_i constituent un scindage minimal du radical de l'idéal engendré par F dans $\mathbb{Q}[e, d, c, b, a]$. La décomposition de $\mathbf{V}(F)$ à l'aide des $\mathbf{V}(S_i)$ est donc canonique en ce sens. Se pose alors le problème général suivant. Peut-on trouver un sous-ensemble $\mathcal{T}$ de $\mathbf{P}_n$ tel que pour toute famille finie $F \subseteq \mathbf{P}_n$ il existe $T_1, \ldots, T_n \in \mathcal{T}$ avec des propriétés raisonnables tel que :

$$\mathbf{V}(F) = \cup_{i=1}^{n} \mathbf{V}(T_i)$$

cette décomposition étant canonique en un certain sens, calculable en un temps acceptable et suffisamment claire et concise pour être utile ? Ce chapitre et les suivants donneront des éléments de réponse à cette question. Une première remarque, qui sera illustrée dans le sixième chapitre, est que l'on ne peut exiger des T_i qu'ils définissent des idéaux premiers.

Ainsi, contrairement à ce qui est fait dans les cours d'algèbre élémentaire, les extensions simples de notre corps de base $\mathbf{k}$ ne seront pas nécessairement des corps. Nous commençons donc ce chapitre par quelques remarques élémentaires mais fondamentales dans notre cas. Ces résultats sont des exercices routiniers d'algèbre. Nous en donnons (sommairement) les preuves, car nous ne connaissons pas d'ouvrages classiques contenant ces résultats. La seconde section est consacrée aux *tours d'extensions simples*. Le cas particulier des *tours d'extensions simples séparables* nous permet de retrouver la notion de produit de corps largement étudiée dans le second chapitre : en effet, chaque tour d'extensions simples séparables est isomorphe à un produit de corps. En pratique, les tours d'extensions simples sont données par des ensembles particuliers de polynômes multivariés, les *ensembles triangulaires*. Ces ensembles triangulaires et les notions s'y rattachant sont regroupés dans la troisième section. Se pose ensuite le problème suivant : étant donné un ensemble triangulaire, représente-t-il une tour d'extensions simples ? Et si oui, que peut-on dire de cette tour d'extensions simples ? On s'intéresse alors aux ensembles triangulaires introduits par Michael Kalkbrener, les *chaînes régulières* ([Kal91, Kal93]) puis à ceux introduits par Daniel Lazard ([Laz91a]). Dans la cinquième section on étudie la représentation et le calcul des *quantités algébrico-transcendantes vivant* dans nos tours d'extensions simples. La sixième et dernière section est dédiée aux systèmes quasi-algébriques. Plus précisément, on s'intéresse à la description de leurs zéros à l'aide des ensembles triangulaires : on rappelle différents théorèmes de décomposition des zéros d'un s.q.a en zéros réguliers d'ensembles triangulaires. Nous commençons par étudier ces zéros réguliers et établissons différents résultats relatifs aux chaînes régulières et aux ensembles de Lazard. L'outil-clef est la notion de saturé d'un ensemble triangulaire (déf. III.19). En effet, d'une part le saturé d'un ensemble triangulaire a pour variété affine associée la clôture de Zariski de ses zéros réguliers (th. III.3). D'autre part, la tour d'extensions simples associée à un ensemble triangulaire régulier T est isomorphe à l'anneau total des fractions de l'anneau-quotient de l'anneau des polynômes par le saturé de T (prop. III.18). Ce dernier (et nouveau) résultat a de nombreuses conséquences, tant algorithmiques (voir la section sur la normalisation d'un polynôme par un ensemble de Lazard dans le quatrième chapitre) que théoriques (th. III.5, III.6 et III.10 prop. III.19). En particulier, on introduit une notion équivalente à celle des chaînes régulières (les *ensembles réguliers*, déf. III.12) sans référence aux zéros ou aux idéaux premiers associés à un idéal. Cette notion, *plus intrinsèque* est bien adaptée à notre notion de pgcd (définition II.9) et aux algorithmes que nous exposerons dans les chapitres suivants. Philippe Aubry, qui a beaucoup travaillé sur les chaînes régulières de Michael Kalkbrener, a contribué à la mise en place de cette dernière section. Je l'en remercie vivement.

1 Idéaux de $\mathbf{A}[X]$ engendrés par un idéal de $\mathbf{A}$

Cette section rassemble des résultats techniques sur les idéaux de $\mathbf{A}[X]$ engendrés par un idéal d'un anneau commutatif unitaire $\mathbf{A}$. Pour cela on commence par étudier les unités et les diviseurs de zéro de $\mathbf{A}[X]$ (prop. III.2). Puis, pour un idéal I de $\mathbf{A}$, on étudie le transfert des propriétés de I en tant qu'idéal de $\mathbf{A}$ vers l'idéal de $\mathbf{A}[X]$ que I engendre dans cet anneau (prop. III.4).

Notations III.1 Dans toute cette section $\mathbf{A}$ sera un anneau commutatif unitaire. Pour un idéal I de $\mathbf{A}$, on désignera par $I[X]$ l'idéal de $\mathbf{A}[X]$ engendré par I dans cet anneau. Enfin, pour $p = \sum_{i=0}^{d} c_i X^i$ dans $\mathbf{A}[X]$ on désignera par $\mathsf{content}(p)$ l'idéal $\langle c_0, \ldots, c_d \rangle_{\mathbf{A}}$ c'est-à-dire l'idéal de $\mathbf{A}$ engendré par les coefficients de p.

Proposition III.1 *Soient u et x deux éléments de $\mathbf{A}$. Si u est une unité et si x est nilpotent, alors $u + x$ est une unité de $\mathbf{A}$.*

Preuve. $\triangleright$ Soit $m \in \mathbb{N}$ tel que $x^m = 0$. On a alors :

$$(u+x)\left(\frac{1}{u} - \frac{x}{u^2} + \frac{x^2}{u^3} + \cdots + (-1)^{m-1}\frac{x^{m-1}}{u^m}\right) = 1$$

ce qui prouve que $u + x$ est une unité de $\mathbf{A}$. $\triangleleft$

Proposition III.2 *On note $\mathbf{B}$ l'anneau de polynômes $\mathbf{A}[X]$. Soit $f = \sum_0^n a_i X^i$ un polynôme de $\mathbf{B}$ avec $a_i \in \mathbf{A}$ et $a_n \neq 0$. Alors on a :*

(i) $(f \in \mathbf{Un}(\mathbf{B})) \iff ((a_0 \in \mathbf{Un}(\mathbf{A}))$ et $(a_1, \ldots, a_n \in \mathbf{Nil}(\mathbf{A})))$

(ii) $(f \in \mathbf{Nil}(\mathbf{B})) \iff (a_0, \ldots, a_n \in \mathbf{Nil}(\mathbf{A}))$

(iii) $(f \in \mathbf{Div}(0)) \iff (\exists a \in \mathbf{A} \mid (a \neq 0)$ et $(af = 0))$

Preuve. $\triangleright$ Vérifions (i). Cette propriété est immédiate si $n = 0$. Supposons maintenant $n > 0$ et vérifions tout d'abord l'implication directe. On suppose donc qu'il existe $g \in \mathbf{A}[X]$ tel que $fg = 1$. On pose $g = \sum_0^m b_j X^j$. Quitte à introduire des b_j nuls, on peut supposer : $m \geq n$. On a alors :

$$\begin{cases} a_n b_m & = & 0 \\ a_n b_{m-1} + a_{n-1} b_m & = & 0 \\ a_n b_{m-2} + a_{n-1} b_{m-1} + a_{n-2} b_m & = & 0 \\ & \vdots & \\ a_n b_0 + \cdots + a_0 b_n & = & 0 \end{cases}$$

En multipliant la seconde équation par a_n on tire des deux premières équations $a_n^2 b_{m-1} = 0$. Puis, en multipliant la troisième équation par a_n^2 on obtient $a_n^3 b_{m-2} = 0$. Par récurrence, il vient $a_n^{m+1} b_0 = 0$. Or on a $a_0 b_0 = 1$. Par suite $a_n^{m+1} = 0$ et donc $a_n \in \mathbf{Nil}(\mathbf{A})$. À l'aide de la proposition III.1 on en déduit que $f - a_n X^n = \sum_0^{n-1} a_i X^i$ est une unité. On montre ainsi que tous les a_i pour $1 \leq i \leq n$ sont nilpotents. Enfin, a_0 est une unité puisque l'on a $a_0 b_0 = 1$. La réciproque est immédiate à l'aide de la proposition III.1.

Vérifions (ii). Si f est nilpotent, alors $1 + Xf$ est une unité, d'après la proposition III.1. Par suite, avec (i), on en tire $a_0, \ldots, a_n$ nilpotents. La réciproque s'obtient en élevant f à une puissance assez grande.

Vérifions (*iii*). Supposons tout d'abord qu'il existe g non nul dans $\mathbf{A}[X]$ tel que $fg = 0$. On choisit g de degré minimal avec cette propriété. On pose $g = \sum_0^m b_j X^j$ avec $b_m \neq 0$. Soit $r \in \mathbb{N}$ tel que $0 \leq r \leq n$. On a :

$$\left(\sum_r^n a_i X^i\right) g \;=\; -\left(\sum_0^{r-1} a_i X^i\right) g$$

dont le degré est inférieur ou égal à $r - 1 + m$. Posons $h = \left(\sum_r^n a_i X^{i-r}\right) g$. Supposons que h ne soit pas nul. Alors, son degré est inférieur strictement à celui de g. Or on a :

$$\left(\sum_r^n a_i X^{i-r}\right) g\, f = 0$$

C'est-à-dire :

$$hf = 0$$

Comme g est de degré minimal avec le fait d'appartenir à $0 : \langle f \rangle$, le polynôme h doit être nul. On a donc :

$$\left(\sum_r^n a_i X^{i-r}\right) g = 0$$

Pour $r = n$ on en déduit : $a_n g = 0$. Puis, pour avec $r = n - 1, \ldots, r = 0$, il vient :

$$a_{n-1} g = a_{n-2} g = \cdots = a_0 g = 0$$

Par suite, tous les produits $a_i b_j$ sont nuls. On en déduit la conclusion. La réciproque est immédiate. $\lhd$

Proposition III.3 *On suppose* $\mathbf{A}$ *nœthérien. Soit* I *un idéal de* $\mathbf{A}$ *et* p *un polynôme de* $\mathbf{A}[X]$*. Alors, on a :*

$$p \in I[X] \iff \mathsf{content}(p) \subseteq I$$

Preuve. $\rhd$ Soit $\{a_1, \ldots, a_l\}$ un système de générateurs de I dans $\mathbf{A}$ et posons $p = \sum_{i=0}^d c_i X^i$. Si $p \in I[X]$, il existe des polynômes $q_1, \ldots, q_l$ de $\mathbf{A}[X]$ tels que :

$$p = \sum_{j=1}^{l} q_j a_j$$

On peut supposer chaque q_j de degré d_j tel que : $d_j \leq d$ et poser $q_j = \sum_{i=0}^d b_{j,i} X^i$. Par suite, pour $0 \leq i \leq d$, on a :

$$c_i = \sum_{j=1}^{l} b_{j,i} a_j$$

ce qui prouve :

$$\langle c_0, \ldots, c_d \rangle_{\mathbf{A}} \subseteq I$$

L'implication réciproque est immédiate. $\lhd$

Proposition III.4 *On suppose* $\mathbf{A}$ *nœthérien. Pour des idéaux* I, I_1, I_2 *de* $\mathbf{A}$ *et un élément* h *de* $\mathbf{A}$*, on a :*

(1) $I[X] \cap \mathbf{A} \;=\; I$

(2) $(I_1 \cap I_2)[X] \;=\; I_1[X] \cap I_2[X]$

(3) $\mathbf{A}[X]/(I[X]) \simeq (\mathbf{A}/I)[X]$

(4) *I premier $\Longleftrightarrow$ $I[X]$ premier*

(5) *I radical $\Longrightarrow$ $I[X]$ radical*

(6) $\sqrt{I[X]} = \sqrt{I}[X]$

(7) *I radical* $\Longrightarrow (I : h^\infty)[X] = (I[X]) : h^\infty$

(8) *I primaire $\Longleftrightarrow$ $I[X]$ primaire*

(9) $(I : h^\infty)[X] = (I[X]) : h^\infty$

Preuve. $\triangleright$ Les propriétés (1) et (2) résultent de la proposition III.3 par *simple jeu de réécriture*. La propriété (3) s'obtient en considérant l'homomorphisme surjectif d'anneaux ϕ de $\mathbf{A}[X]$ dans $(\mathbf{A}/I)[X]$ défini de la manière suivante :

(*i*) $\phi(X) = X$

(*ii*) $\forall a \in \mathbf{A}\ \phi(a) = \overline{a}^I$

En utilisant la proposition III.3, on vérifie que ϕ a pour noyau $I[X]$ ce qui prouve (3).

Pour vérifier (4), on utilise (3) afin d'établir les équivalences suivantes : I premier $\Longleftrightarrow$ $\mathbf{A}/I$ intègre $\Longleftrightarrow$ $(\mathbf{A}/I)[X]$ intègre $\Longleftrightarrow$ $\mathbf{A}[X]/(I[X])$ intègre $\Longleftrightarrow$ $I[X]$ premier.

Pour vérifier (5), l'anneau $\mathbf{A}$ (et par conséquent l'anneau $\mathbf{A}[X]$) étant nœthérien, on écrit que l'idéal radical I est intersection finie des idéaux premiers de $\mathsf{ap}(I)$. On en déduit, à l'aide des propriétés (2) et (4), que $I[X]$ est intersection finie d'idéaux premiers (donc radicaux) de $\mathbf{A}[X]$. Bref, $I[X]$ est un idéal radical de $\mathbf{A}[X]$.

Vérifions (6). On a $I \subseteq \sqrt{I}$. D'où $I[X] \subseteq \sqrt{I}[X]$. Par suite, à l'aide de (5), il vient $\sqrt{I[X]} \subseteq \sqrt{I}[X]$. Inversement, dans l'anneau $\mathbf{A}[X]$ on a $I \subseteq I[X]$. On en déduit dans $\mathbf{A}[X]$ l'inclusion d'ensembles : $\sqrt{I} \subseteq \sqrt{I[X]}$. D'où par passage aux idéaux engendrés : $\sqrt{I}[X] \subseteq \sqrt{I[X]}$.

Vérifions (7). On suppose I radical. D'après le théorème I.9, dans l'anneau $\mathbf{A}$, on a :

$$I : h^\infty = \bigcap_{\{\mathcal{P} \in \mathsf{ap}(I)\,\mid\, h \notin \mathcal{P}\}} \mathcal{P}$$

D'où, avec (2) :

$$(I : h^\infty)[X] = \bigcap_{\{\mathcal{P} \in \mathsf{ap}(I)\,\mid\, h \notin \mathcal{P}\}} \mathcal{P}[X]$$

D'autre part, d'après (2), on a :

$$I[X] = \bigcap_{\{\mathcal{P} \in \mathsf{ap}(I)\}} \mathcal{P}[X]$$

D'après (4), la décomposition ci-dessus est la décomposition primaire minimale de $I[X]$ dans $\mathbf{A}[X]$. Par suite, en utilisant le théorème I.9 dans $\mathbf{A}[X]$, il vient :

$$I[X] : h^\infty = \bigcap_{\{\mathcal{P} \in \mathsf{ap}(I)\,\mid\, h \notin \mathcal{P}[X]\}} \mathcal{P}[X]$$

Or, puisque $h \in \mathbf{A}$, on a :

$$h \in \mathcal{P}[X] \iff h \in \mathcal{P}$$

Bref, on a bien :

$$(I : h^\infty)[X] = I[X] : h^\infty$$

Vérifions (8). Montrons tout d'abord que si I est primaire dans $\mathbf{A}$ alors $I[X]$ est primaire dans $\mathbf{A}[X]$. Grâce à (3), il suffit d'établir que tout diviseur de zéro dans $(\mathbf{A}/I)[X]$ est nilpotent. Soit $p = \sum_{i=0}^{d} c_i X^i \in \mathbf{A}[X]$ dont la classe résiduelle dans $(\mathbf{A}/I)[X]$ est un diviseur de zéro. Alors, d'après le point (*iii*) de la proposition III.2 il existe $a \in \mathbf{A}$ tel que

$$\overline{a}^I \neq 0 \text{ et } ap \in I[X]$$

Comme I est primaire, on en déduit que chaque coefficient c_i de f a une puissance dans I. Ainsi, chaque $\overline{c_i}^I$ est un nilpotent dans $\mathbf{A}/I$. Enfin, à l'aide du point (*ii*) de la proposition III.2 on en tire que p est bien un nilpotent de $(\mathbf{A}/I)[X]$. Réciproquement, supposons que $I[X]$ soit primaire dans $\mathbf{A}[X]$ et vérifions que I est primaire dans $\mathbf{A}$. Soient alors a et b dans $\mathbf{A}$ avec $ab \in I$ et $b \notin I$. Puisque l'on a $I[X] \cap \mathbf{A} = I$, nous avons aussi : $ab \in I[X]$ et $b \notin I[X]$. On en déduit la conclusion.

Vérifions (9). Soit $S = (\mathcal{Q}_1, \ldots, \mathcal{Q}_s)$ une décomposition primaire de I. D'après le théorème I.9 on a :

$$I : h^\infty = \bigcap_{\{\mathcal{Q} \in S \mid h \notin \sqrt{\mathcal{Q}}\}} \mathcal{Q}$$

On raisonne alors comme dans la preuve de (7) en utilisant (8) au lieu de (6). $\lhd$

2 Tours d'extensions simples

On généralise, aux polynômes univariés à coefficients dans un anneau commutatif unitaire quelconque, les notions usuelles de *polynôme primitif*, *polynôme sans carrés*, *partie primitive* et *partie sans carrés* (def. III.1 et III.2). Avec la notion précédente de *sans carrés*, nous allons établir un théorème fondamental (th. III.1) pour la résolution des systèmes d'équations algébriques par les algorithmes développés dans les deux prochains chapitres de cette thèse. Le procédé de résolution des systèmes d'équations algébriques présenté dans le cinquième chapitre est basé sur une méthode proposée par Lazard dans [Laz91a] où ce théorème fondamental apparaît sans preuve. Nous pourrions faire apparaître ce théorème comme une conséquence des résultats classiques sur les anneaux nœthériens via le théorème II.2 p. 56. Nous avons préféré en donner ici une preuve directe s'appuyant uniquement sur les propriétés de $\mathbf{k}[X]$ où $\mathbf{k}$ est un corps. Puis, dans le cas où l'anneau $\mathbf{A}$ est un corps ou un produit de corps, la proposition III.5 donne une condition suffisante pour pouvoir calculer une partie sans carrés d'un polynôme de $\mathbf{A}[X]$. En s'appuyant sur les deux résultats précédents, on peut alors donner la définition des *tours d'extensions simples* et, en particulier, celle des *tours d'extensions simples séparables* (def. III.3). On termine cette section par l'une des remarques essentielles pour la suite : toute tour d'extensions simples séparables est un produit de corps (th. III.2).

Définition III.1 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire de caractéristique nulle. Soit* p *un polynôme non nul dans* $\mathbf{A}[X]$*. On désigne par* $\frac{\partial p}{\partial X}$ *le polynôme-dérivé de* p*. On dira ici que* p *est*

(i) primitif *si l'idéal de* $\mathbf{A}$ *engendré par ses coefficients est l'idéal-unité, autrement dit, si l'on a :*

$$\mathsf{content}(p) \;=\; \mathbf{A}$$

(ii) sans carrés *si l'idéal de* $\mathbf{fr}(\mathbf{A})[X]$ *engendré par* p *et son polynôme-dérivé est l'idéal-unité, autrement dit, si l'on a :*

$$\langle p, \tfrac{\partial p}{\partial X}\rangle_{\mathbf{fr}(\mathbf{A})[X]} \;=\; \mathbf{A}$$

Définition III.2 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire de caractéristique nulle. Soient* p *et* q *deux polynômes non nuls de* $\mathbf{A}[X]$*. On dira ici que* q *est une :*

(i) partie primitive *de* p *si* q *est primitif et si l'on a :*

$$\langle p\rangle_{\mathbf{fr}(\mathbf{A})[X]} \;=\; \langle q\rangle_{\mathbf{fr}(\mathbf{A})[X]}$$

(ii) partie sans carrés *de* p *si* q *est sans carrés et si l'on a :*

$$\sqrt{\langle p\rangle_{\mathbf{fr}(\mathbf{A})[X]}} \;=\; \langle q\rangle_{\mathbf{fr}(\mathbf{A})[X]}$$

Théorème III.1 *Soit* $\mathbf{A}$ *un anneau commutatif unitaire de caractéristique nulle. Soit* p *dans* $\mathbf{A}[X]$ *de degré non nul. On suppose que* $\mathbf{A}$ *est muni d'une structure de corps ou de produit de corps et que* p *est unitaire et sans carrés (au sens précédent). Alors on a :*

(i) *l'idéal engendré par* p *dans* $\mathbf{A}[X]$ *est radical.*

(ii) *les anneaux* $\mathbf{fr}(\mathbf{A}[X])$ *et* $\mathbf{A}[X]/\langle p\rangle$ *sont chacun munis d'une structure de corps ou de produit de corps.*

Preuve. $\triangleright$ Vérifions la propriété (i). Rappelons que pour un corps ou un produit de corps $\mathbf{A}$ on a :

$$\mathbf{fr}(\mathbf{A}) = \mathbf{A}$$

On note $\frac{\partial p}{\partial X}$ le polynôme-dérivé de p. On suppose donc :

$$\langle p, \tfrac{\partial p}{\partial X}\rangle_{\mathbf{A}[X]} = \mathbf{A}[X]$$

La propriété (i) est claire si $\mathbf{A}$ est un corps. En effet, si $\mathbf{A}$ est un corps alors $\mathbf{A}[X]$ est principal. Par suite p et $\frac{\partial p}{\partial X}$ n'ont pas d'autres diviseurs communs que les éléments non nuls de $\mathbf{A}$. Par conséquent, $\mathbf{A}[X]$ étant aussi factoriel, le polynôme p est, à une unité près, produit d'éléments irréductibles 2 à 2 distincts. Donc, $\langle p\rangle_{\mathbf{A}[X]}$ est intersection d'idéaux premiers, et $\langle p\rangle_{\mathbf{A}[X]}$ est radical. Supposons maintenant que $\mathbf{A}$ soit un produit de corps $\mathbf{k}_1 \times \cdots \times \mathbf{k}_n$ et notons $\pi_i f$ la $i-ième$ composante d'un élément f de $\mathbf{A}[X]$. Considérons maintenant l'idéal R_i de $\mathbf{A}[X]$ engendré par

$$\mathbf{k}_1[X] \times \cdots \times \mathbf{k}_{i-1}[X] \times \{\pi_i p\} \times \mathbf{k}_{i+1}[X] \times \cdots \times \mathbf{k}_n[X]$$

On vérifie sans peine que chaque R_i est radical. De plus, les idéaux R_i sont 2 à 2 comaximaux D'autre part, on a clairement :

$$\langle p \rangle_{\mathbf{A}[X]} = \prod_1^n R_i$$

Par suite, il vient :

$$\langle p \rangle_{\mathbf{A}[X]} = \cap_1^n R_i$$

Ainsi $\langle p \rangle_{\mathbf{A}[X]}$ est intersection finie d'idéaux radicaux, et donc radical, ce qui prouve (i). Nous réutiliserons plus loin cette décomposition de $\langle p \rangle_{\mathbf{A}[X]}$.

Vérifions (ii). supposons tout d'abord que $\mathbf{A}$ soit un corps. Alors, il est clair que $\mathbf{fr}(\mathbf{A}[X])$ est aussi un corps. Vérifions qu'il va de même de $\mathbf{A}[X]/\langle p \rangle$. Comme nous l'avons vu plus haut, l'idéal $\langle p \rangle_{\mathbf{A}[X]}$ est intersection d'idéaux premiers. Comme $\mathbf{A}[X]$ est principal ces idéaux sont en fait maximaux. Par suite, à l'aide de la proposition II.7, $\mathbf{A}[X]/\langle p \rangle$ est un produit de corps

On suppose désormais que $\mathbf{A}$ est un produit de corps $\mathbf{k}_1 \times \cdots \times \mathbf{k}_n$. Vérifions tout d'abord que $\mathbf{fr}(\mathbf{A}[X])$ est un produit de corps. Nous devons identifier les éléments réguliers (i.e. non diviseurs de zéro) de $\mathbf{A}[X]$. D'après la proposition III.2 les polynômes f de $\mathbf{A}[X]$ sont des diviseurs de zéro s'il existe un diviseur de zéro a de $\mathbf{A}$ tel que : $af = 0$. Autrement dit, ces polynômes f de $\mathbf{A}[X]$ sont ceux tel qu'il existe $i \in \{1, \ldots, n\}$ tel que $\pi_i f = 0$. Par suite, on a :

$$\mathbf{reg}(\mathbf{A}[X]) = \mathbf{reg}(\mathbf{k}_1[X]) \times \cdots \times \mathbf{reg}(\mathbf{k}_n[X])$$

On en déduit l'isomorphisme :

$$\mathbf{fr}(\mathbf{A}[X]) \;\simeq\; \mathbf{k}_1(X) \times \cdots \times \mathbf{k}_n(X)$$

Vérifions maintenant que $\mathbf{A}[X]/\langle p \rangle$ est un produit de corps. On vérifie tout d'abord que pour des anneaux C et D et un élément (c, d) de leur produit on a l'isomorphisme :

$$C \times D/\langle (c, d) \rangle \;\simeq\; \mathbf{C}/\langle c \rangle \times \mathbf{D}/\langle d \rangle$$

Par conséquent :

$$\mathbf{A}[X]/\langle p \rangle \;\simeq\; \mathbf{k}_1[X]/\langle \pi_1 p \rangle \times \cdots \times \mathbf{k}_n[X]/\langle \pi_n p \rangle$$

Si l'un des facteurs du produit d'anneaux précédent n'était pas réduit (i.e. possédait des éléments nilpotents) alors il en irait de même pour $\mathbf{A}[X]/\langle p \rangle$. Or $\langle p \rangle$ est radical d'après (i). Par suite, chaque $\langle \pi_i p \rangle_{\mathbf{k}_i[X]}$ est radical. D'après le cas où $\mathbf{A}$ est un corps, cas traité précédemment, on en tire que chaque $\mathbf{k}_1[X]/\langle \pi_i p \rangle$ est un produit de corps. Finalement, $\mathbf{A}[X]/\langle p \rangle$ est bien un produit de corps. Ceci termine la preuve de ce théorème. $\triangleleft$

Remarque III.1 Avec les notations et hypothèses du théorème III.1, remarquons que dans l'anneau $\mathbf{A}[X]/\langle p \rangle$ tout élément régulier est soit nul, soit inversible. En effet, cet anneau est un produit de corps.

Proposition III.5 *Soit* $\mathbf{A}$ *un anneau (commutatif, unitaire) de caractéristique nulle tel que* $\mathbf{fr}(\mathbf{A})$ *soit un corps ou un produit de corps. Soit* $p = \sum_{i=0}^{d} c_i X^i$ *non nul dans* $\mathbf{A}[X]$. *Notons* $\frac{\partial p}{\partial X}$ *son polynôme-dérivé. Supposons qu'il existe* $c \in \mathbf{A}$ *tel que :*

$$\langle c \rangle_{\mathbf{A}} = \langle c_0, \ldots, c_d \rangle_{\mathbf{A}}$$

Soit enfin $g \in \mathbf{A}[X]$ *tel que :*

$$\langle p, \tfrac{\partial p}{\partial X} \rangle_{\mathbf{fr}(\mathbf{A})[X]} \;=\; \langle g \rangle_{\mathbf{fr}(\mathbf{A})[X]}$$

Alors on a :

(i) *si* $c \in \mathbf{reg}(\mathbf{A})$ *alors* p exquo c *est une partie primitive de* p

(ii) *si* $\mathsf{lc}(g) \in \mathbf{reg}(\mathbf{A})$ *alors* $\mathsf{pquo}(p, g)$ *est une partie sans carrés de* p *(où* $\mathsf{pquo}(p, g)$ *désigne le pseudo-quotient de* p *par* g, *voir notation II.4 p. 65).*

Preuve. $\triangleright$ La propriété (i) est triviale si $\mathbf{fr}(\mathbf{A})$ est un corps. Elle s'étend au cas où $\mathbf{fr}(\mathbf{A})$ est un produit de corps, en raisonnant composante à composante. En effet, l'hypothèse $c \in \mathbf{reg}(\mathbf{A})$ signifie que c n'a pas de composante nulle sur $\mathbf{fr}(\mathbf{A})$.

Vérifions maintenant la propriété (ii). Supposons tout d'abord que $\mathbf{fr}(\mathbf{A})$ soit un corps que l'on note $\mathbf{k}$. Soit $\mathbf{K}$ un sur-corps de $\mathbf{k}$ où p se factorise en produit de facteurs linéaires sous une forme :

$$p = e\, \Pi_{i=1}^{n} (X - a_i)^{n_i}$$

où e appartient à $\mathbf{k}$ et les a_i appartiennent à $\mathbf{K}$. On vérifie que le polynôme :

$$h = \Pi_{i=1}^{n} (X - a_i)^{n_i - 1}$$

engendre le même idéal que la paire $\{p, \frac{\partial p}{\partial X}\}$ dans $\mathbf{k}[X]$. Par suite h et g sont associés dans $\mathbf{k}[X]$. On peut donc supposer : $g = eh$. D'où :

$$\mathsf{pquo}(p, g) = \Pi_{i=1}^{n} (X - a_i)$$

On vérifie alors sans peine que $\mathsf{pquo}(p, g)$ est une partie sans carrés de p et l'on en déduit la propriété (ii). Si $\mathbf{fr}(\mathbf{A})$ est un produit de corps, l'hypothèse $\mathsf{lc}(g) \in \mathbf{reg}(\mathbf{A})$ permet de raisonner composante à composante comme dans la preuve du théorème III.1. $\triangleleft$

Définition III.3 *Soient* $\mathbf{k}$ *un corps,* $\mathbf{K}$ *sa clôture algébrique et et* $\mathbf{A}_0, \ldots, \mathbf{A}_n$ *des anneaux commutatifs unitaires tels que*

$$\mathbf{k} = \mathbf{A}_0 \subseteq \mathbf{A}_1 \subseteq \cdots \subseteq \mathbf{A}_n$$

Soient $X_1, \ldots, X_n$ *des variables algébriquement libres dans leur ensemble sur* $\mathbf{K}$. *On dira ici que la séquence* $(\mathbf{A}_0, \ldots, \mathbf{A}_n)$ *est une* tour d'extensions simples du corps $\mathbf{k}$ *si pour chaque* $i \in \{1, \ldots, n\}$ *l'une des deux conditions suivantes est réalisée :*

(i) $\mathbf{A}_i = \mathbf{fr}(\mathbf{A}_{i-1}[X_i])$

(ii) *il existe* $p_i \in \mathbf{A}_{i-1}[X_i]$ *de degré positif et unitaire tel que :*

$$\mathbf{A}_i = \mathbf{fr}(\ \mathbf{A}_{i-1}[X_i]/\langle p_i \rangle)$$

Soit i entre 1 et n. Si la condition (i) est réalisée, on écrira : $\mathbf{A}_{i-1} \stackrel{\bullet(X_i)}{\longrightarrow} \mathbf{A}_i$. *Si la condition (ii) est réalisée, on écrira :* $\mathbf{A}_{i-1} \stackrel{p_i(x_i)}{\longrightarrow} \mathbf{A}_i$.

La séquence $(\mathbf{A}_0, \ldots, \mathbf{A}_n)$ *sera dite* tour algébrique d'extensions simples du corps $\mathbf{k}$ *si pour chaque* $i \in \{1, \ldots, n\}$ *seule la condition (ii) est réalisée.*

La séquence $(\mathbf{A}_0, \ldots, \mathbf{A}_n)$ *sera dite* tour d'extensions simples séparables du corps $\mathbf{k}$ *si pour chaque* $i \in \{1, \ldots, n\}$ *la condition (i) précédente ou la condition (iii) suivante est réalisée :*

(iii) *il existe* $p_i \in \mathbf{A}_{i-1}[X_i]$ *de degré positif, unitaire et sans carrés tel que :*

$$\mathbf{A}_i = \mathbf{A}_{i-1}[X_i]/\langle p_i \rangle$$

Enfin, la séquence $(\mathbf{A}_0, \ldots, \mathbf{A}_n)$ *sera dite* tour algébrique d'extensions simples séparables du corps $\mathbf{k}$ *si pour chaque* $i \in \{1, \ldots, n\}$ *seule la condition (iii) est réalisée.*

Remarque III.2 Nous devons vérifier que la terminologie introduite dans la définition III.3 est cohérente. Plus précisément, il faut vérifier qu'une tour d'extensions simples séparables du corps $\mathbf{k}$ est une tour d'extensions simples de $\mathbf{k}$. Ceci résulte de la proposition suivante et du fait que si un anneau $\mathbf{A}$ est un corps ou un produit de corps alors on a $\mathbf{A} = \mathbf{fr}(\mathbf{A})$.

Théorème III.2 *Soit* $\mathbf{k}$ *un corps et* $(\mathbf{A}_0, \ldots, \mathbf{A}_n)$ *une tour d'extensions simples séparables du corps* $\mathbf{A}_0 = \mathbf{k}$. *Alors, pour* $1 \leq i \leq n$, *l'anneau* $\mathbf{A}_i$ *est un corps ou un produit de corps.*

Preuve. $\triangleright$ Ceci résulte immédiatement du théorème III.1. $\triangleleft$

3 Ensembles triangulaires

Cette section rassemble les définitions et notations liées à la *vision univariée et récursive* des polynômes multivariés. Sa lecture n'est donc peut-être pas très captivante, mais on a préféré rassembler en une seule section toutes ces notions, afin qu'il soit commode de s'y reporter.

Notations III.2 Dans toute la suite $\mathbf{R}$ désignera un anneau intègre de corps des fractions $\mathbf{k}$. Le corps $\mathbf{K}$ dénote toujours (notation I.1 p. 19) une extension de $\mathbf{k}$. On rappelle que n est un entier positif et que V est un ensemble de n variables que l'on suppose ordonnées comme suit :

$$x_1 < x_2 < \cdots < x_n$$

Pour $i \in \{1, \ldots, n\}$, on désigne par $\mathbf{R}_i = \mathbf{R}[x_1, \ldots, x_i]$ et $\mathbf{P}_i = \mathbf{k}[x_1, \ldots, x_i]$ les anneaux des polynômes en i variables à coefficients dans $\mathbf{R}$ et $\mathbf{k}$ respectivement. Nous posons aussi $\mathbf{R}_0 = \mathbf{R}$ et $\mathbf{P}_0 = \mathbf{k}$. Soit $E \subseteq \mathbf{R}_n$ et soient $p, q \in \mathbf{R}_n$, avec $p \neq 0$ et $q \notin \mathbf{R}$. Pour $v \in V$, on dénote par $\mathsf{deg}(p, v)$ le degré de p vu comme polynôme univarié en v. On désigne par $\mathsf{var}(E)$ l'ensemble des variables $v \in V$ pour lesquelles il existe $r \in E$ avec $r \neq 0$ et $\mathsf{deg}(r, v) > 0$. Si $E = \{r\}$ nous écrirons simplement $\mathsf{var}(r)$ au lieu $\mathsf{var}(E)$. Enfin, si $E \not\subseteq \mathbf{R}$, on dénote par $\mathsf{mvar}(E)$ le plus grand élément de $\mathsf{var}(E)$.

Définition III.4 *Soit* $q \in \mathbf{R}_n$, *avec* $q \notin \mathbf{R}$. *On appelle :*

– variable principale *de q, dénotée par* $\mathsf{mvar}(q)$, *le plus grand élément de* $\mathsf{var}(q)$.

- initial *de* q*, dénoté par* $\mathsf{init}(q)$*, le coefficient dominant de* q *vu comme un polynôme univarié en* $\mathsf{mvar}(q)$.
- degré principal *de* q*, dénoté par* $\mathsf{mdeg}(q)$*, le degré* $\mathsf{deg}(q, \mathsf{mvar}(q))$
- monôme principal *de* q*, dénoté par* $\mathsf{mmon}(q)$*, le monôme dominant de* q *vu comme un polynôme univarié en* $\mathsf{mvar}(q)$ *; on a donc* $\mathsf{mmon}(q) = \mathsf{mvar}(q)^{\mathsf{mdeg}(q)}$
- tête *de* q*, dénotée par* $\mathsf{head}(q)$*, le produit* $\mathsf{init}(q)\mathsf{mmon}(q)$*, c'est-à-dire le terme dominant de* q *vu comme un polynôme univarié en* $\mathsf{mvar}(q)$
- queue *de* q*, dénotée par* $\mathsf{tail}(q)$ *le polynôme* $q - \mathsf{head}(q)$.

Notations III.3 Soit $E \subseteq \mathbf{R}_n$. On désigne par $\mathsf{algVar}(E)$ l'ensemble des variables $v \in V$ pour lesquelles il existe $r \in E$ avec $r \notin \mathbf{R}$ tel que $\mathsf{mvar}(r) = v$. Soit $v \in V$. On désigne par E_v^-, E_v et E_v^+ l'ensemble des polynômes non constants $r \in E$ tels que $\mathsf{mvar}(r) < v$, $\mathsf{mvar}(r) = v$ et $\mathsf{mvar}(r) > v$ respectivement. Si $E_v = \{r\}$ on écrira simplement $E_v = r$.

Définition III.5 *Soient* $p, q \in \mathbf{R}_n$*, avec* $p \neq 0$ *et* $q \notin \mathbf{R}$*. Suivant [Laz92b] on définit récursivement un ordre partiel dans* $\mathbf{R}_n$ *en posant* $p \prec q$ *si*

(*i*) $p \in \mathbf{R}$ *ou si*

(*ii*) $\mathsf{mvar}(p) < \mathsf{mvar}(q)$ *ou si*

(*iii*) ($\mathsf{mvar}(p) = \mathsf{mvar}(q)$ *et* $\mathsf{mdeg}(p) < \mathsf{mdeg}(q)$) *ou si*

(*iv*) ($\mathsf{mvar}(p) = \mathsf{mvar}(q)$ *et* $\mathsf{mdeg}(p) = \mathsf{mdeg}(q)$ *et* $\mathsf{init}(p) \prec \mathsf{init}(q)$) *ou si*

(*v*) $\mathsf{init}(p) \not\prec \mathsf{init}(q)$ *et* $\mathsf{init}(q) \not\prec \mathsf{init}(p)$ *et* $\mathsf{tail}(p) \prec \mathsf{tail}(q)$

On dira que p *est* inférieur *(resp.* supérieur*) à* q pour l'ordre de Ritt et Wu avec le raffinement de Lazard *si l'on a* $p \prec q$ *(resp.* $q \prec p$*). On dira que* p *et* q *sont incomparables pour l'ordre de Ritt et Wu avec le raffinement de Lazard, et l'on écrira* $p \sim_{rwl} q$*, si ni* $p \prec q$*, ni* $q \prec p$ *ne sont vrais.*

Dans la suite, on écrira également $p <_{rwl} q$ *(resp.* $p >_{rwl} q$*) au lieu de* $p \prec q$ *(resp.* $q \prec p$*). Notons que l'on a* $p <_{rwl} q$ *(resp.* $p >_{rwl} q$*) (resp.* $p \sim_{rwl} q$*) si l'algorithme suivant retourne* **true** *(resp.* **false***) (resp.* **"failed"***). Sa syntaxe est celle du langage* AXIOM *([JS92]).*

```
compare(a :Rn,b :Rn) : Union(Boolean,"failed") ==
    (b ∈ R) and (a ∈ R) => "failed"
    (b ∈ R) => false
    (a ∈ R) => true
    mvar(a) < mvar(b) => true
    mvar(a) > mvar(b) => false
    mdeg(a) < mdeg(b) => true
    mdeg(a) > mdeg(b) => false
    c := compare(init(a), init(b))
    c case Boolean => c
    return compare(tail(a), tail(b))
```

Définition III.6 *Une partie* T *de* $\mathbf{R}_n$ *est appelée* ensemble triangulaire de $\mathbf{R}_n$ *si chaque polynôme de* T *n'appartient pas à* $\mathbf{R}$ *et si pour tout* p *et tout* q *de* T *avec* $p \neq q$ *on a* $\mathsf{mvar}(p) \neq \mathsf{mvar}(q)$*. On dira alors que* $v \in V$ *est* algébrique par rapport à *l'ensemble triangulaire* T *si*

$v \in \mathsf{algVar}(T)$ *et* transcendante par rapport à T *sinon. Un ensemble triangulaire* T *sera dit* algébrique *si toutes les variables de* V *sont algébriques par rapport à* T. *On appellera* degré de T *le* $n-$*uplet de* $\mathbb{N}^n$ *noté* $\mathsf{deg}(T)$ *et défini par* $\mathsf{deg}(T) = (d_1, \ldots, d_n)$ *où* d_i *est nul si* $x_i \notin \mathsf{algVar}(T)$ *et est égal à* $\mathsf{mdeg}(\ T_{x_i})$ *sinon.*

Définition III.7 *Soit* $p \in \mathbf{R}_n$. *Soit* $\mathsf{iter}(p)$ *le sous-ensemble de* $\mathbf{R}_n$ *défini récursivement comme suit :*

$$\text{si } p \in \mathbf{R} \text{ alors } \mathsf{iter}(p) = \emptyset \text{ sinon } \mathsf{iter}(p) = \{p\} \cup \mathsf{iter}(\mathsf{init}(p))$$

Alors $\mathsf{iter}(p)$ *est un ensemble triangulaire de* $\mathbf{R}_n$ *dont les éléments sont appelés les* initiaux itérés *de* p.

Définition III.8 *Soit* $E \subseteq \mathbf{R}_n$ *et soient* $p, q \in \mathbf{R}_n$, *avec* $p \neq 0$ *et* $q \notin \mathbf{R}$. *On dira que :*

- p *est* réduit par rapport à q *et l'on écrira* $\mathsf{red?}(p, q)$ *si l'on a* $\mathsf{deg}(p, \mathsf{mvar}(q)) < \mathsf{mdeg}(q)$,
- p *est* réduit par rapport à E *et l'on écrira* $\mathsf{red?}(p, E)$ *si l'on a* $\mathsf{red?}(p, r)$ *pour tout* $r \in E$,
- p *est* normalisé par rapport à q *et l'on écrira* $\mathsf{normalized?}(p, q)$ *si aucun des initiaux itérés de* p *n'a* $\mathsf{mvar}(q)$ *pour variable principale,*
- p *est* normalisé par rapport à E *et l'on écrira* $\mathsf{normalized?}(p, E)$ *si l'on a* $\mathsf{normalized?}(p, r)$ *pour tout* $r \in E$,
- p *est* initialement réduit par rapport à q *et l'on écrira* $\mathsf{iRed?}(p, q)$ *si* p *est normalisé par rapport à* q *ou si l'on a* $\mathsf{red?}(\ \mathsf{iter}(p)_v, q)$ *où* v *est* $\mathsf{mvar}(q)$,
- p *est* initialement réduit par rapport à E *et l'on écrira* $\mathsf{iRed?}(p, E)$ *si l'on a* $\mathsf{iRed?}(p, r)$ *pour tout* $r \in E$.

Définition III.9 *Un ensemble triangulaire* T *de* $\mathbf{R}_n$ *est dit* réduit *ou* auto-réduit (*resp.* initialement réduit) (*resp.* normalisé) *si pour chaque* $t \in T$, *en désignant par* v *la variable principale de* t, *on a* $\mathsf{red?}(t,\ T_v^-)$ (*resp.* $\mathsf{iRed?}(t,\ T_v^-)$) (*resp.* $\mathsf{normalized?}(t,\ T_v^-)$).

Proposition III.6 *Soit* $T \subseteq \mathbf{R}_n$ *un ensemble triangulaire algébrique. Alors* T *est normalisé si et seulement si l'on a* $\mathsf{init}(t) \in \mathbf{R}$ *pour chaque* $t \in T$.

Notations III.4 Soit $v \in V$ et Soient $p, q \in \mathbf{R}_n$, avec $p \neq 0$ et $\mathsf{deg}(q, v) > 0$. On désigne par $\mathsf{prem}(p, q, v)$ et $\mathsf{pquo}(p, q, v)$ le pseudo-reste et le pseudo-quotient de p par q , ces polynômes étant vus comme univariés en v. Si $v = \mathsf{mvar}(q)$, on écrira $\mathsf{prem}(p, q)$ et $\mathsf{pquo}(p, q)$ au lieu de $\mathsf{prem}(p, q, v)$ et $\mathsf{pquo}(p, q, v)$ respectivement.

Soit $T \subseteq \mathbf{R}_n$ un ensemble triangulaire et E une partie quelconque de $\mathbf{R}_n$. On définit le polynôme de $\mathbf{R}_n$ noté $\mathsf{prem}(p, T)$ de la manière suivante :

$$\text{si } \ T = \emptyset \ \text{ alors } \ \mathsf{prem}(p, T) = p \ \text{ sinon } \ \mathsf{prem}(p, T) = \mathsf{prem}(\mathsf{prem}(p,\ T_v),\ T_v^-)$$

où v est $\mathsf{mvar}(T)$. Puis, on dénote par $\mathsf{prem}(E, T)$ le sous-ensemble de $\mathbf{R}_n$ dont les éléments sont les polynômes $\mathsf{prem}(r, T)$ pour r décrivant E. Si $T = \{q\}$ on écrit simplement $\mathsf{prem}(E, q)$ simplement au lieu de $\mathsf{prem}(E, T)$. Enfin, on désignera par $\mathsf{red}_{\mapsto 0}(T)$ l'ensemble des polynômes de $\mathbf{R}_n$ défini par :

$$\mathsf{red}_{\mapsto 0}(T) = \{p \in \mathbf{P}_n \ \mid \ \mathsf{prem}(p, T) = 0\}$$

Proposition III.7 *Soit $p \in \mathbf{R}_n$ et $T = \{t_1, \ldots, t_m\}$ un ensemble triangulaire non vide de $\mathbf{R}_n$. On note i_j l'initial de t_j. Alors, il existe des entiers naturels $e_1, \ldots, e_m$ et des polynômes $q_1, \ldots, q_m$ de $\mathbf{R}_n$ tels que*

$$i_1^{e_1} \cdots i_m^{e_m} p \; = \; q_1 t_1 + \cdots + q_m t_m + \mathsf{prem}(p, T)$$

Cette propriété, dont la preuve est immédiate, est connue sous le nom de remainder formula *([Wu84a]).*

Définition III.10 *Soit $T \subseteq \mathbf{R}_n$ un ensemble triangulaire non vide et S la partie multiplicativement stable de $\mathbf{R}_n$ engendré par les initiaux de polynômes de T. On appelle* saturé de T dans $\mathbf{P}_n$ *l'idéal de $\mathbf{P}_n$ noté $\mathsf{sat}(T)$ et défini par :*

$$\mathsf{sat}(T) = S^{-1}\langle T \rangle$$

où $\langle T \rangle$ désigne l'idéal que T engendre dans $\mathbf{P}_n$. Pour $T = \emptyset$ on convient $\mathsf{sat}(T) = \langle 0 \rangle$.

Proposition III.8 *Pour tout ensemble triangulaire $T \subseteq \mathbf{R}_n$ on a :*

$$\mathsf{red}_{\mapsto 0}(T) \subseteq \mathsf{sat}(T)$$

L'ensemble $\mathsf{red}_{\mapsto 0}(T)$ n'est pas nécessairement un idéal de $\mathbf{P}_n$ et l'inclusion précédente n'est généralement pas une égalité.

Preuve. $\triangleright$ L'inclusion résulte immédiatement de la proposition III.7 précédente. Donnons maintenant un contre-exemple illustrant le fait que $\mathsf{red}_{\mapsto 0}(T)$ n'est pas nécessairement un idéal de $\mathbf{P}_n$. Supposons $n \geq 2$ et posons $T = \{x_1(x_1 - 1), x_1 x_2 - 1\}$. Une base de Gröbner lexicographique de $\mathsf{sat}(T_4)$ est $\{x_1 - 1, x_2 - 1\}$. Poons $p = (x_1 - 1)(x_1 x_2 - 1)$ et $q = -(x_1 - 1)x_1 x_2$. On vérifie $\mathsf{prem}(p, T) = \mathsf{prem}(q, T) = 0$. Cependant $p + q = 1 - x_1$ et $(p + q) \notin \mathsf{red}_{\mapsto 0}(T)$. Ainsi $\mathsf{red}_{\mapsto 0}(T)$ n'est pas un idéal et $\mathsf{red}_{\mapsto 0}(T) \neq \mathsf{sat}(T)$. $\triangleleft$

Remarque III.3 Nous avons déjà rencontré la pseudo-division entre polynômes univariés dans l'algorithme des sous-résultants (remarque II.10 p. 66). Elle intervient aussi dans de nombreux algorithmes de résolution de systèmes d'équations algébriques ([Kal93, Laz91a, Wan93b, Wu87]). Dans ce cadre, il s'agit généralement du calcul du pseudo-reste $\mathsf{prem}(p, T)$ d'un polynôme p par rapport à un ensemble triangulaire auto-réduit T dans $\mathbf{R}_n$. Ce calcul se fait souvent en appliquant la définition-même de $\mathsf{prem}(p, T)$ (notation III.4). Mais alors on calcule en fait un multiple d'un représentant de $\overline{p}^{\langle T \rangle}$ comme le montre la proposition III.7. Pour diminuer la taille des coefficients de ce multiple, on peut remplacer la pseudo-division par ce que l'on appelle parfois la pseudo-division paresseuse définie ci-après.

Soient $p, q \in \mathbf{R}_n$ tel que $q \notin \mathbf{R}$. Soit cd une application de $\mathbf{reg}(\mathbf{R}_n) \times \mathbf{reg}(\mathbf{R}_n) \longrightarrow \mathbf{reg}(\mathbf{R}_n)$ calculant un diviseur commun de ses arguments, donc tel que

$$\mathsf{cd}(a, b) = c \quad \Longrightarrow \quad (\exists a', b' \in \mathbf{reg}(\mathbf{R}_n)) \;\; a = a'c \;\; \text{et} \;\; b = b'c$$

On note $\mathsf{lazyPrem}(p, q)$ et $\mathsf{lazyPquo}(p, q)$ les polynômes R et Q respectivement obtenus en effectuant la *pseudo-division paresseuse* de p par q selon l'algorithme suivant :

lazyPseudoDivision (p, q) ==
$Q := 0$
$R := p$
$v := \mathsf{mvar}(q)$
$d := \mathsf{mdeg}(q)$
$\delta := \mathsf{deg}(R, v) - d$
while $\delta \geq 0$ **repeat**
$c := \mathsf{cd}(\mathsf{init}(q), \mathsf{lc}(R, v))$
$T := (\mathsf{lc}(R, v)\ \mathsf{exquo}\ c)\, v^{\delta}$
$Q := (\mathsf{init}(q)\ \mathsf{exquo}\ c)\, Q + T$
$R := (\mathsf{init}(q)\ \mathsf{exquo}\ c)\, R - T\, q$
$\delta := \mathsf{deg}(R, v) - d$
return $[Q, R]$

où $\mathsf{lc}(R, v)$ désigne le coefficient dominant de R vu comme polynôme univarié en v. Soit S la partie multiplicativement stable de $\mathbf{R}_n$ engendrée par les diviseurs de l'initial de q. Les polynômes $\mathsf{lazyPrem}(p, q)$ et $\mathsf{lazyPquo}(p, q)$ satisfont alors aux propriétés suivantes :

(*i*) $(\exists s \in S) \quad s\, p = q\, \mathsf{lazyPquo}(p, q) + \mathsf{lazyPrem}(p, q)$

(*ii*) $\mathsf{lazyPrem}(p, q) \neq 0 \implies \mathsf{deg}(\mathsf{lazyPrem}(p, q), \mathsf{mvar}(q)) < \mathsf{mdeg}(q)$

Pour $p \in \mathbf{R}_n$ et T un ensemble triangulaire de $\mathbf{R}_n$, on définit $\mathsf{lazyPrem}(p, T)$ de manière analogue à $\mathsf{prem}(p, T)$. La proposition III.7 est encore valable en y remplaçant $\mathsf{prem}(p, T)$ par $\mathsf{lazyPrem}(p, T)$ et le produit $i_1^{e_1} \cdots i_m^{e_m}$ par un produit de diviseurs des initiaux des polynômes de T.

4 Tours d'extensions simples et ensembles triangulaires

Les deux remarques suivantes introduisent la problématique étudiée dans cette section.

Remarque III.4 Soient $\mathbf{R}$ un anneau intègre, $\mathbf{k}$ son corps des fractions, et $\mathbf{K}$ la clôture algébrique de $\mathbf{k}$. Soient $\mathbf{A}_0, \ldots, \mathbf{A}_n$ des anneaux commutatifs unitaires tels que :

$$\mathbf{k} = \mathbf{A}_0 \subseteq \mathbf{A}_1 \subseteq \cdots \subseteq \mathbf{A}_n$$

On suppose que les variables $x_1, \ldots, x_n$ forment une partie algébriquement libre sur $\mathbf{K}$. On suppose que $(\mathbf{A}_0, \ldots, \mathbf{A}_n)$ est une tour d'extensions simples du corps $\mathbf{A}_0 = \mathbf{k}$ (déf. III.3). Comme d'habitude, pour $i \in \{1, \ldots, n\}$, on désigne respectivement par $\mathbf{R}_i$ et $\mathbf{P}_i$ les anneaux $\mathbf{R}[x_1, \ldots, x_i]$ et $\mathbf{k}[x_1, \ldots, x_i]$. De plus, on pose $\mathbf{P}_0 = \mathbf{k}$ et $\mathbf{R}_0 = \mathbf{R}$. Pour $i \in \{1, \ldots, n\}$, on considère l'homomorphisme H_i de $\mathbf{A}_{i-1}[x_i]$ dans $\mathbf{A}_i$ défini de la façon suivante :

(*i*) si $\mathbf{A}_{i-1} \overset{\bullet(x_i)}{\longrightarrow} \mathbf{A}_i$ alors H_i est l'injection canonique $q \longrightarrow \frac{q}{1}$ de $\mathbf{A}_{i-1}[x_i]$ dans $\mathbf{fr}(\mathbf{A}_{i-1}[x_i])$

(*ii*) si $\mathbf{A}_{i-1} \overset{p_i(x_i)}{\longrightarrow} \mathbf{A}_i$ alors H_i est l'homomorphisme surjectif canonique $q \longrightarrow \overline{q}^{\langle p_i \rangle}$ de $\mathbf{A}_{i-1}[x_i]$ dans $\mathbf{A}_{i-1}[x_i]/\langle p_i \rangle$ suivi de l'injection canonique $\alpha \longrightarrow \frac{\alpha}{1}$ du précédent anneau dans $\mathbf{fr}(\mathbf{A}_{i-1}[x_i]/\langle p_i \rangle)$

Pour $i \in \{1, \ldots, n\}$ on considère alors l'homomorphisme F_i de $\mathbf{P}_{i+1}$ dans $\mathbf{A}_i[x_{i+1}]$ tel que :

(i) $F_i(x_{i+1}) = x_{i+1}$

(ii) $(\forall p \in \mathbf{P}_i)$ $F_i(p) = \begin{cases} H_1(p) & \text{si } i = 1 \\ H_i(F_{i-1}(p)) & \text{si } i > 1 \end{cases}$

De façon imagée, l'homomorphisme F_i interprète les variables $x_1, \ldots, x_i$ des polynômes de $\mathbf{P}_{i+1}$ en tant que quantités algébriques ou transcendantes suivant la nature du passage d'un $\mathbf{A}_{j-1}$ à $\mathbf{A}_j$, alors que x_{i+1} reste une variable pour F_i. Pour chaque $1 \leq i \leq n$, on vérifie :

$$\mathbf{A}_{i-1} \overset{p_i(x_i)}{\longrightarrow} \mathbf{A}_i \quad \Longrightarrow \quad (\exists P_i \in \mathbf{R}_i \setminus \mathbf{R}_{i-1}) \ \langle F_{i-1}(P_i) \rangle_{\mathbf{A}_{i-1}[x_i]} = \langle p_i \rangle_{\mathbf{A}_{i-1}[x_i]}$$

En effet, on peut *chasser* les éventuels dénominateurs des coefficients de p_i sans changer l'idéal qu'il engendre dans $\mathbf{A}_{i-1}[x_i]$. Les polynômes P_i ainsi construits constituent un ensemble triangulaire de $\mathbf{R}_n$.

Remarque III.5 Soit T un ensemble triangulaire de $\mathbf{R}_n$. On pose $\mathbf{A}_0 = \mathbf{k}$. On cherche à définir un processus réciproque à celui décrit dans la remarque III.4 précédente.

On définit tout d'abord l'anneau $\mathbf{A}_1$, extension simple de $\mathbf{A}_0$, et l'homomorphisme F_1 de $\mathbf{P}_2$ dans $\mathbf{A}_1[x_2]$ comme suit. Si $x_1 \notin \mathsf{algVar}(T)$ on pose :

$$\mathbf{A}_1 = \mathbf{fr}(\mathbf{A}_0[x_1]) \text{ et } (\forall p \in \mathbf{P}_1) \ F_1(p) = \frac{p}{1}$$

sinon on pose :

$$\mathbf{A}_1 = \mathbf{fr}(\ \mathbf{A}_0[x_1]/\langle\ T_{x_1} \rangle) \text{ et } (\forall p \in \mathbf{P}_1) \ F_1(p) = \frac{\overline{p}^{\langle\ T_{x_1} \rangle}}{1}$$

Puis, dans tous les cas on définit : $F_1(x_2) = x_2$. On note alors x_1 l'image de x_1 par F_1. On doit maintenant distinguer deux cas. Si $x_2 \notin \mathsf{algVar}(T)$ on peut poser : $\mathbf{A}_2 = \mathbf{fr}(\mathbf{A}_1[x_2])$. Sinon, pour définir une extension algébrique $\mathbf{A}_2$ de $\mathbf{A}_1$ à l'aide de T_{x_2}, on doit bien sr avoir : $\mathsf{deg}(F_1(\ T_{x_2})) > 0$. Mais, plus généralement, la condition $\mathsf{deg}(F_i(\ T_{x_{i+1}})) > 0$ si $x_{i+1} \in \mathsf{algVar}(T)$ n'est pas suffisante pour assurer une extension algébrique de $\mathbf{A}_i$ à $\mathbf{A}_{i+1}$ à l'aide de $T_{x_{i+1}}$. Considérons pour cela l'exemple suivant où $\mathbf{k}$ est le corps $\mathbb{Q}$ des nombres rationnels :

$$T = \{x_1^2 - 2,\ x_2^2 - 2x_1x_2 + 2,\ (x_1 - x_2)x_3^2 + 1\}$$

On a $F_1(\ T_{x_2}) = (x_2 - x_1)^2$ et par suite $F_2(\ T_{x_3})$ n'est pas unitaire.

Une condition nécessaire et suffisante pour que $F_i(\ T_{x_{i+1}})$ soit unitaire (et, par suite, pour que l'on puisse construire l'extension précédente) sera donnée dans la proposition III.18. Elle reposera entre autre sur le résultat classique suivant ([SZ67] vol.I p.214). Si I est un idéal d'un anneau nœthérien $\mathbf{A}$, pour tout élément $x \in \mathbf{A}$ on a :

$$\overline{x}^I \in \mathbf{Div}(0)_{\mathbf{A}/I} \quad \Longleftrightarrow \quad (\exists \mathcal{P} \in \mathsf{ap}(I))\ x \in \mathcal{P}$$

D'autre part, elle reposera sur la notion de *saturé d'un ensemble triangulaire* (déf. III.10). Le saturé d'un ensemble triangulaire sera étudié dans la sixième section de ce chapitre. L'une des motivations de ce développement sera l'étude de la notion de *chaîne régulière* de à Michael Kalkbrener ([Kal93]) et présentée plus loin dans cette section (définition III.11).

Nous établirons du reste que les chaînes régulières et les ensembles triangulaires décrivant les tours d'extensions simples sont en fait les mêmes objets (théorème III.6).

Une condition suffisante et facile à tester, assurant $F_i(T_{x_{i+1}})$ unitaire (et par suite permettant de construire l'extension précédente) est donnée dans la proposition III.9 ci-après et repose sur la notion *d'ensemble triangulaire normalisé* (déf. III.9) due à Daniel Lazard ([Laz91a]). Il résultera alors de cette proposition et du théorème III.6 que les ensembles triangulaires normalisés sont des chaînes régulières, la réciproque étant fausse.

Proposition III.9 *Soit* $(\mathbf{A}_0, \ldots, \mathbf{A}_i)$ *une tour d'extensions simples de* $\mathbf{k}$. *Soit* F_i *l'homomorphisme de* $\mathbf{P}_{i+1}$ *dans* $\mathbf{A}_i[x_{i+1}]$ *et* T *l'ensemble triangulaire de* $\mathbf{P}_i$ *définis dans la remarque III.4. Soit* $T_{x_{i+1}}$ *un polynôme de* $\mathbf{P}_{i+1}$ *de degré strictement positif en* x_{i+1}. *Si* $T_{x_{i+1}}$ *est normalisé par rapport à* T, *alors le polynôme* $F_i(T_{x_{i+1}})$ *est unitaire.*

Preuve. $\rhd$ Remarquons tout d'abord que si un élément x d'un anneau $\mathbf{A}_j$ (pour $0 \leq j < i$) est inversible dans $\mathbf{A}_j$ alors son image *canonique* $H_{j+1}(x)$ dans $\mathbf{A}_{j+1}$ est encore inversible. Ceci étant dit, nous devons vérifier que $F_i(\mathsf{init}(T_{x_{i+1}}))$ est inversible dans $\mathbf{A}_i$. On distingue deux cas. Tout d'abord le cas où : $\mathsf{init}(T_{x_{i+1}}) \in \mathbf{k}$ pour lequel on a clairement $F_i(\mathsf{init}(T_{x_{i+1}}))$ inversible, à l'aide de la remarque précédente. Passons au cas contraire et posons :

$$x_j = \mathsf{mvar}(\mathsf{init}(T_{x_{i+1}}))$$

Notons que l'on peut supposer que $\mathsf{init}(\mathsf{init}(T_{x_{i+1}}))$ est inversible dans $\mathbf{A}_{j-1}$ (en raisonnant par récurrence sur le nombre d'éléments de $\mathsf{iter}(T_{x_{i+1}})$). Par suite, d'après la proposition III.2, $\mathsf{init}(T_{x_{i+1}})$ ne peut être un diviseur de zéro de $\mathbf{A}_{j-1}[x_j]$. Par construction (définition III.3) $\mathsf{init}(T_{x_{i+1}})$ est donc inversible dans $\mathbf{A}_j$. On en déduit le résultat. $\lhd$

Notations III.5 Soit $i \in \{1, \ldots, n\}$. Soit I un idéal de $\mathbf{P}_i$. On pose alors :

$$\mathcal{K}(I) = \{\mathbf{fr}(\mathbf{P}_n/\mathcal{P}) \mid \mathcal{P} \in \mathsf{ap}(I)\}$$

Notons que les éléments de $\mathcal{K}$(I) sont des corps. Soit $\mathcal{P} \in \mathsf{ap}(I)$. On pose $\kappa = \mathbf{fr}(\mathbf{P}_n/\mathcal{P})$. Soit $f \in \mathbf{P}_{i+1}$. On désignera par $\overline{\overline{f}}^{\mathcal{P}}$ ou par $\overline{f}^{\kappa}$ l'image *canonique* de f dans $\kappa[x_{i+1}]$.

Définition III.11 *Soit* $0 \leq i \leq n$ *et* T *un ensemble triangulaire de* $\mathbf{P}_i$ *et* $\mathsf{Rep}_i(T)$ *un idéal de* $\mathbf{P}_i$. *On dit que* T *est une* chaîne régulière *de* $\mathbf{P}_i$ *dont* $\mathsf{Rep}_i(T)$ *est la* représentation, *si l'une des assertions suivantes est vraie, où* $\mathcal{K}_i(T)$ *dénote* $\mathcal{K}(\mathsf{Rep}_i(T))$:

(*i*) $i = 0$, $T = \emptyset$, $\mathsf{Rep}_0(T) = \{0\}$,

(*i*) $i > 0$ $T_{x_i}^-$ *chaîne régulière de* $\mathbf{P}_{i-1}$ *de représentation* $\mathsf{Rep}_{i-1}(T)$ *et :*

$$\begin{cases} x_i \notin \mathsf{algVar}(T) \\ f \in \mathsf{Rep}_i(T) \iff \forall \kappa \in \mathcal{K}_{i-1}(T_{x_i}^-) \ \ \overline{f}^{\kappa} = 0 \end{cases}$$

ou

$$\begin{cases} x_i \in \mathsf{algVar}(T) \\ (\forall \kappa \in \mathcal{K}_{i-1}(T_{x_i}^-)) \ \ \overline{\mathsf{init}(T_{x_i})}^{\kappa} \neq 0 \\ f \in \mathsf{Rep}_i(T) \iff \forall \kappa \in \mathcal{K}_{i-1}(T_{x_i}^-) \ \ \overline{f}^{\kappa} \in \sqrt{\langle \overline{T_{x_i}}^{\kappa} \rangle} \end{cases}$$

Proposition III.10 *Soit $1 \leq i \leq n$ et T une chaîne régulière de $\mathbf{P}_i$.*
(1) *si $x_i \notin \mathsf{algVar}(T)$ alors :*

$$\mathsf{Rep}_i(T) \quad = \quad \sqrt{\mathsf{Rep}_{i-1}(T)}[x_i]$$

(2) *si $x_i \in \mathsf{algVar}(T)$ alors :*

$$\mathsf{Rep}_i(T) \quad = \quad \{f \in \mathbf{P}_i \quad | \quad (\exists m \in \mathbb{N}) \ \mathsf{prem}(f^m,\ T_{x_i}) \in \mathsf{Rep}_i(\ T_{x_i}^-)\}$$

Preuve. $\triangleright$ Vérifions tout d'abord (1). On suppose donc : $x_i \notin \mathsf{algVar}(T)$. Soit $f \in \mathbf{P}_i$ et soit $\mathcal{P} \in \mathsf{ap}(\mathsf{Rep}_{i-1}(T))$. Remarquons que l'on a :

$$\overline{\overline{f}}^{\mathcal{P}} = 0 \quad \Longleftrightarrow \quad \mathsf{content}(f) \subseteq \mathcal{P}$$

D'où :

$$f \in \mathsf{Rep}_i(T) \quad \Longleftrightarrow \quad (\forall \mathcal{P} \in \mathsf{ap}(\mathsf{Rep}_{i-1}(T)))\ \mathsf{content}(f) \subseteq \mathcal{P}$$

C'est-à-dire :

$$f \in \mathsf{Rep}_i(T) \quad \Longleftrightarrow \quad \mathsf{content}(f) \subseteq \sqrt{\mathsf{Rep}_{i-1}(T)}$$

Par suite, à l'aide de la proposition III.3, il vient finalement :

$$f \in \mathsf{Rep}_i(T) \quad \Longleftrightarrow \quad f \in \sqrt{\mathsf{Rep}_{i-1}(T)}[x_i]$$

Vérifions maintenant (2). On suppose donc : $x_i \in \mathsf{algVar}(T)$. Posons $t = \ T_{x_i}$. Soit $f \in \mathbf{P}_i$ et soit $\mathcal{P} \in \mathsf{Rep}_{i-1}(\ T_{x_i}^-)$. La condition :

$$\overline{\overline{f}}^{\mathcal{P}} \in \sqrt{\langle \overline{\overline{t}}^{\mathcal{P}} \rangle_{\mathbf{fr}(\mathbf{P}_{i-1}/\mathcal{P})[x_i]}}$$

se réécrit :

$$(\exists m \in \mathbb{N}) \quad | \quad \overline{\overline{f^m}}^{\mathcal{P}} \in \langle \overline{\overline{t}}^{\mathcal{P}} \rangle_{\mathbf{fr}(\mathbf{P}_{i-1}/\mathcal{P})[x_i]}$$

Pour $m \in \mathbb{N}$ posons :

$$r_m = \mathsf{prem}(f^m, t) \ \text{ et } \ h = \mathsf{init}(t)$$

Notons q_m l'élément de $\mathbf{P}_i$ et e_m l'entier naturel tel que :

$$h^{e_m} f^m = q_m t + r_m \quad \text{et} \quad r_m \neq 0 \implies \mathsf{deg}(r_m, x_i) < \mathsf{mdeg}(t)$$

Comme T est une chaîne régulière, l'initial h de $t = \ T_{x_i}$ ne peut appartenir à $\mathcal{P}$. Par conséquent, notre condition se réécrit encore :

$$(\exists m \in \mathbb{N}) \quad | \quad \overline{\overline{r_m}}^{\mathcal{P}} \in \langle \overline{\overline{t}}^{\mathcal{P}} \rangle_{\mathbf{fr}(\mathbf{P}_{i-1}/\mathcal{P})[x_i]}$$

Du fait de la relation $\mathsf{deg}(r_m, x_i) < \mathsf{mdeg}(t)$, elle se réécrit enfin :

$$(\exists m \in \mathbb{N}) \quad | \quad \mathsf{content}(r_m) \subseteq \mathcal{P}$$

Par suite, on a :

$$f \in \mathsf{Rep}_i(T) \quad \Longleftrightarrow \quad (\exists M \in \mathbb{N}) \;\mid\; (\forall \mathcal{P} \in \mathsf{Rep}_{i-1}(\, T^-_{x_i})) \;\; \mathsf{content}(r_M) \subseteq \mathcal{P}$$

C'est-à-dire :

$$f \in \mathsf{Rep}_i(T) \quad \Longleftrightarrow \quad (\exists M \in \mathbb{N}) \;\mid\; \mathsf{content}(r_M) \subseteq \sqrt{\mathsf{Rep}_{i-1}(\, T^-_{x_i})}$$

Par suite, à l'aide de la proposition III.3, il vient :

$$f \in \mathsf{Rep}_i(T) \quad \Longleftrightarrow \quad (\exists M \in \mathbb{N}) \;\mid\; r_M \in \sqrt{\mathsf{Rep}_{i-1}(\, T^-_{x_i})}[x_i]$$

D'où, à l'aide de la propriété (1)

$$f \in \mathsf{Rep}_i(T) \quad \Longleftrightarrow \quad (\exists M \in \mathbb{N}) \;\mid\; r_M \in \mathsf{Rep}_i(\, T^-_{x_i})$$

On obtient ainsi la formule (2) désirée. $\triangleleft$

Remarque III.6 Comme nous le verrons plus loin, les chaînes régulières permettent de *construire* des tours d'extensions simples. Nous devons pour cela préciser ce processus de construction que nous avons initié dans la remarque III.5. C'est l'objet de la définition III.12.

Notations III.6 Soient $\mathbf{A}_0, \mathbf{A}_1, \ldots, \mathbf{A}_n$ des anneaux commutatifs unitaires. tels que $\mathbf{k} = \mathbf{A}_0$ et tels que, pour $i \in \{1, \ldots, n\}$, l'anneau $\mathbf{A}_{i-1}$ soit un sous-anneau de $\mathbf{A}_i$. Soit x_{n+1} une variable supplémentaire. Posons $\mathbf{P}_{n+1} = \mathbf{k}[x_1, \ldots, x_{n+1}]$. Soit F_0 l'application-identité de $\mathbf{P}_1$. Enfin, pour $i \in \{1, \ldots, n\}$, soit F_i un homomorphisme d'algèbres surjectif de $\mathbf{P}_{i+1}$ dans $\mathbf{A}_i[x_{i+1}]$ tel que $F_i(x_{i+1}) = x_{i+1}$.

Définition III.12 *Soit T un ensemble triangulaire de $\mathbf{P}_i$. l'ensemble triangulaire T est un* ensemble régulier *de $\mathbf{P}_i$ dont la* tour d'extensions simples associée *est $(\mathbf{A}_0, \ldots, \mathbf{A}_i)$ et dont* l'interprétation *(ou* homomorphisme associé*) est $(F_0, \ldots, F_i)$, si l'une des conditions suivantes est vraie :*

(1) *$i = 0$ et T est vide*

(2) *$i > 0$, l'ensemble $T^-_{x_i}$ est un ensemble régulier de $\mathbf{P}_{i-1}$ de tour d'extensions simples associée $(\mathbf{A}_0, \ldots, \mathbf{A}_{i-1})$ et d'interprétation $(F_0, \ldots, F_{i-1})$ tel que l'une des conditions suivantes soit vraie :*

   (*a*) *$x_i \notin \mathsf{algVar}(T)$, et pour tout $p \in \mathbf{P}_i$ l'élément $F_i(p)$ est l'image canonique de $F_{i-1}(p)$ dans $\mathbf{A}_i$, où*
   $$\mathbf{A}_i = \mathbf{fr}(\mathbf{A}_{i-1}[x_i])$$

   (*b*) *$x_i \in \mathsf{algVar}(T)$, l'élément $F_{i-1}(\mathsf{init}(\, T_{x_i}))$ de $\mathbf{A}_{i-1}$ est une unité, et pour tout $p \in \mathbf{P}_i$ l'élément $F_i(p)$ est l'image canonique de $\overline{F_{i-1}(p)}^{\langle t_i \rangle}$ dans $\mathbf{A}_i$ où $t_i = F_{i-1}(\, T_{x_i})$ et*
   $$\mathbf{A}_i = \mathbf{fr}(\mathbf{A}_{i-1}[x_i]/\langle t_i \rangle)$$

Définition III.13 *Avec les hypothèses et notations de la définition III.12 précédente. On suppose que T est un ensemble triangulaire régulier de $\mathbf{R}_n$ de tours d'extensions simples associé $(\mathbf{A}_0, \ldots, \mathbf{A}_n)$ et d'interprétation (quenceF0n). l'ensemble T est dit :*

(i) sans carrés *si pour* $1 \leq i \leq n$ *on a : si* $x_i \in \mathsf{algVar}(T)$ *alors le polynôme* $F_{i-1}(\,T_{x_i})$ *de* $\mathbf{A}_{i-1}[x_i]$ *est sans carrés,*

(ii) primitif *si pour* $1 \leq i \leq n$ *on a : si* $x_i \in \mathsf{algVar}(T)$ *alors pour* $1 \leq j < i$ *les coefficients du polynôme* T_{x_i} *vu comme polynôme multivarié de* $\mathbf{A}_j[x_{j+1}, \cdots, x_i]$ *engendre l'idéal unité de* $\mathbf{A}_j$*.*

Proposition III.11 *Tout ensemble triangulaire normalisé de* $\mathbf{R}_n$ *est un ensemble régulier.*

Remarque III.7 La proposition III.11 précédente légitime la définition suivante. D'autre part, nous établirons dans la section suivante que tout ensemble triangulaire normalisé est une chaîne régulière (la réciproque étant fausse). Ceci résultera immédiatement du fait que tout ensemble triangulaire est régulier si et seulement si c'est une chaîne régulière (théorème III.6).

Définition III.14 *Soit* T *un ensemble triangulaire de* $\mathbf{R}_n$*. On dit que* T *satisfait les* axiomes de Lazard *([Laz91a]) ou que* T *est un* ensemble de Lazard *si* T *est normalisé, sans carrés et primitif.*

Remarque III.8 Tout ensemble triangulaire normalisé de $\mathbf{R}_n$ n'est pas nécessairement une base de Gröbner de $\langle T\rangle$ pour l'ordre lexicographique induit par l'ordre sur les variables $x_1 < \cdots < x_n$. Considérer l'exemple suivant avec $x > y > z$. L'ensemble triangulaire :

$$\left\{z\ x^2 + (z-1)\ x,\ (z+1)\ y + z\right\}$$

a pour base de Gröbner pour l'ordre lexicographique :

$$\left\{y\ x^2 + (2\ y+1)\ x,\ z\ x^2 + (z-1)\ x,\ (z+1)\ y + z\right\}$$

Remarque III.9 Soit $\mathbf{K}$ une clôture algébrique de $\mathbf{k}$ (le corps des fractions de $\mathbf{R}$) de degré de transcendance au moins n sur $\mathbf{k}$. Soit T un ensemble triangulaire normalisé de $\mathbf{R}_n$. On pose $x = \mathsf{algVar}(T)$ et $Y = \mathsf{var}(T)\backslash x$. On peut se demander si T est une base de Gröbner pour l'ordre lexicographique induit par l'ordre sur x de l'idéal qu'il engendre dans $\mathbf{fr}(\mathbf{k}[Y])[x]$. La réponse est non comme le montre l'exemple suivant. On pose $T = \{x_1^3 + 1, (x_2 + x_1)x_3^2 + 1\}$. La base de Gröbner associée à T dans les conditions précédentes est :

$$\{x_1^3 + 1, x_3^2 + \frac{1}{x_2^3 - 1}x_1^2 - \frac{x_2}{x_2^3 - 1}x_1 + \frac{x_2^2}{x_2^3 - 1}\}$$

5 Réduction et image modulaire d'un polynôme par un ensemble triangulaire

Notations III.7 Pour un module $\mathbf{M}$ sur $\mathbf{A}$ et une partie multiplicativement stable S de $\mathbf{reg}(\mathbf{A})$, on notera par $\mathrm{S}^{-1}\mathbf{M}$ le $\mathbf{A}$-module des fractions à numérateur dans $\mathbf{M}$ et dénominateur dans S. Chaque élément l de $\mathrm{S}^{-1}\mathbf{M}$ peut être représenté de façon non nécessairement unique sous une forme am/s avec $a \in \mathbf{A}$, $m \in \mathbf{M}$ et $s \in S$. On écrira alors $\mathsf{numPart}(l) = a$, $\mathsf{polPart}(l) = m$, $\mathsf{numer}(l) = am$ et $\mathsf{denom}(l) = s$. On suppose désormais que $\mathbf{A}$ est un anneau intègre. Si $\mathbf{M}$ est un anneau de polynômes à variables dans un ensemble V et à coefficients dans $\mathbf{A}$, alors deux valeurs différentes de $\mathsf{polPart}(l)$ engendrent le même idéal dans l'anneau des polynômes à variables dans V et à coefficients dans le corps des fractions de $\mathbf{A}$. De plus si $\mathbf{A}$ est factoriel et si S est $\mathbf{reg}(\mathbf{A})$, alors a, m et s peuvent être choisis de façon unique.

Remarque III.10 Nous avons vu que les ensembles triangulaires fournissent une représentation des tours d'extensions simples. Ceci conduit naturellement au problème suivant : pour $a \in \mathbf{R}_n$ et T un ensemble triangulaire de $\mathbf{R}_n$ déterminer un représentant *aussi simple que possible* de $\overline{a}^{\langle T\rangle}$. Notons que même si les polynômes que nous manipulerons dans nos algorithmes sont *moralement* dans $\mathbf{P}_n$, nous travaillerons effectivement dans $\mathbf{R}_n$, c'est-à-dire avec des coefficients entiers plutôt que fractionnaires. Ceci ayant pour objectif de minimiser l'utilisation explicite de fractions. Celles-ci ne peuvent cependant pas être complètement évitées. Nous reviendrons au fur et à mesure sur ce problème. Commençons par le cas où T ne contient qu'un polynôme b. On cherche $r \in \mathbf{R}_n$ tel que :

$$\overline{a}^{\langle T\rangle} = \overline{r}^{\langle T\rangle} \text{ et } \mathsf{deg}(r, \mathsf{mvar}(b)) < \mathsf{mdeg}(b)$$

Cependant si b n'est pas unitaire, ce problème n'a en général pas de solution. Prendre $\mathbf{R} = \mathbf{Z}$, $n = 2$, $a = (X_2 + 1)^{100}$ et $b = X_1 a + 1$. On souhaiterait alors écrire $a \equiv \frac{-1}{X_1} \bmod b$. Il faut donc rendre inversible l'initial de b et travailler dans la tour d'extensions simples associée à T, et donc dans $\mathbf{fr}(\ \mathbf{R}[X_1]\ /\langle b\rangle)[X_2]$. Plus généralement posons $T = \{t_1, \ldots, t_m\}$. On note i_j l'initial de t_j. On utilise la pseudo-division paresseuse avec pour opération de diviseur commun cd la fonction constante égale à 1. Alors, d'après la proposition III.7, il existe des entiers naturels $e_1, \ldots, e_m$ tels que

$$\overline{i_1^{e_1} \cdots i_m^{e_m} a}^{\langle T\rangle} = \overline{\mathsf{lazyPrem}(a, T)}^{\langle T\rangle}$$

Par suite, si chaque i_j est inversible modulo T alors on peut expliciter $\overline{a}^{\langle T\rangle}$. En particulier, si T est normalisé, il est représenté par la quantité $\mathsf{modulo}(a, T)$ définie par :

$$\mathsf{modulo}(a, T) = \mathsf{lazyPrem}(a, T)/(I_1^{e_1} \cdots I_m^{e_m})$$

Plus précisément, soit S_T le sous-ensemble multiplicativement stable de $\mathbf{R}_n$ engendré par les initiaux de T et 1, et soit $\mathbf{M}_T$ le sous-module sur $\mathbf{R}$ de $\mathbf{R}_n$ constitué par les éléments de $\mathbf{R}_n$ réduits par rapport à T. Alors, l'application $a \longrightarrow \mathsf{modulo}(a, T)$, définie ci-dessus, est à valeurs dans $\mathrm{S}_T{}^{-1}\ \mathbf{M}_T$. Nous donnons maintenant l'algorithme que nous utilisons pour calculer $\mathsf{modulo}(a, T)$. Il s'exprime en fonction de l'opération $(a, t) \mapsto \mathsf{modulo}(a, t)$ de $\mathbf{R}_n \times \mathbf{R}_n$ dans $\mathbf{reg}(\mathbf{R}_n)^{-1}\mathbf{R}_n$ définie plus haut à l'aide de $\mathsf{lazyPrem}(a, T)$ et de telle sorte que pour $t \notin \mathbf{R}$ on ait $\mathsf{modulo}(a, t) = \mathsf{modulo}(a, \{t\})$.

Proposition III.12 *Soit T un ensemble triangulaire de $\mathbf{R}_n$. On dénote par S_T le sous-ensemble multiplicativement stable de $\mathbf{R}_n$ engendré par 1 et les initiaux des polynômes de T. Pour $p \notin \mathbf{R}$ on pose $v = \mathsf{mvar}(p)$, $i_p = \mathsf{init}(p)$, $d_p = \mathsf{mdeg}(p)$ et $t_p = \mathsf{tail}(p)$. Alors l'élément $\mathsf{modulo}(p, T)$ de $S_T{}^{-1}\mathbf{R}_n$ peut être défini en itérant les règles suivantes :*

(1) $T = \emptyset$ *or* $p \in \mathbf{R} \implies \mathsf{modulo}(p, T) = \frac{p}{1}$

(2) $\mathsf{red?}(p,\ T_v) \implies \mathsf{modulo}(p, T) = \mathsf{modulo}(i_p,\ T_v^-)\frac{v^{d_p}}{1} + \mathsf{modulo}(t_p, T)$

(3) $\mathsf{modulo}(v^{d_p}, T_v) = \frac{r}{s}$ *et* $\mathsf{modulo}(i_p, T_v^-) = \frac{r'}{s'} \Longrightarrow \mathsf{modulo}(p, T) = \frac{\mathsf{modulo}(rr', T)}{ss'} + \mathsf{modulo}(t_p, T)$

Par conséquent, pour tout $p \in \mathbf{R}_n$ *il existe* $r \in \mathbf{R}_n$ *et* $s \in S_T$ *tel que* $\mathsf{modulo}(p, T) = \frac{r}{s}$ *où* $\mathsf{red?}(r, T)$ *et* $sp - r \in \langle T \rangle_{\mathbf{R}_n}$. *De plus* $\mathsf{modulo}(p, T)$ *peut-être calculé par l'algorithme suivant :*

```
modulo(p, T) ==
    empty?(T) or p ∈ R => p/1
    v := mvar(T)
    empty?( T_v^- ) => modulo(p, T_v)
    v := mvar(p)
    q := 0/1
    if not empty?( T_v^- )
        then
            while positive?(deg(p, v)) repeat
                q := q + modulo(init(p), T_v^- ) * mmon(p)/ 1
                p := tail(p)
    q := q + modulo(p, T_v^- )
    if v ∈ algVar(T)
        then
            q := numPart(q) * (modulo(polPart(q), T_v))/denom(q)
            if not empty?( T_v^- )
                then
                    p := polPart(q)
                    r := numPart(q)
                    s := denom(q)
                    q := 0/1
                    while positive?(deg(p, v)) repeat
                        q := q + modulo(init(p), T_v^- ) * mmon(p)/ 1
                        p := tail(p)
                    q := r * (q + modulo(p, T_v^- )) / s
    return q
```

Remarque III.11 Notons que les fractions ci-dessus peuvent ne pas être maintenues en permanence sous forme irréductible. Nous reviendrons plus loin sur ces aspects techniques.

Remarque III.12 On suppose maintenant que pour chaque $t \in T$ on a $\mathsf{init}(t) \in \mathbf{R}$. Ainsi, T est une base de Gröbner pour l'ordre lexicographique de l'idéal que T engendre dans $\mathbf{P}_n$ et un polynôme $p \in \mathbf{R}_n$ est réduit par rapport à T au sens des bases de Gröbner si et seulement s'il est réduit par rapport à T au de la définition III.8. Dans ce cas, et si $\mathbf{R}$ est un anneau à pgcd, il nous est apparu expérimentalement intéressant de remplacer le calcul de $\mathsf{modulo}(a, T)$ par celui de $\mathsf{rem}(a, T)$, opération que nous décrivons maintenant. Elle procède comme le calcul de la forme normale d'un polynôme par rapport à une base de Gröbner (déf. I.5 p. 22). Nous donnerons tout d'abord un algorithme pour l'opération $a \mapsto \mathsf{headRem}(a, T)$ de $\mathbf{R}_n$ dans $\mathbf{reg}(\mathbf{R})^{-1}\mathbf{R}_n$ qui produit une fraction a'/r où $\mathsf{lm}(a)$ est réduit par rapport à T et tel que : $(r\,a\, - \,a') \in \langle T \rangle$. Puis nous donnerons un algorithme pour l'opération $a \mapsto \mathsf{rem}(a, T)$ de $\mathbf{R}_n$

dans $\mathbf{reg}(\mathbf{R})^{-1}\mathbf{M}_T$ qui produit une fraction cb/r où b est un polynôme primitif réduit par rapport à T et où c et r sont des éléments de $\mathbf{R}$ sans diviseurs communs, le tout tel que : $(r\,a\;-\;c\,b) \in \langle T \rangle$.

```
headRem(a, T) ==
    empty?(T) => a/1
    r : R := 1
    l1 := sort(<_rwl, [t ∈ T])
    l2 := l1
    while (not zero? a) and (not empty? l2) repeat
        b := first(l2)
        m := lm(a)
        if not (d := deg(m, mvar(b)) − mdeg(b) < 0)
            then
                α := lc(a)
                β := lc(b)
                g := gcd(α, β)
                α := α exquo g
                β := β exquo g
                a := β reductum(a) − α mvar(b)^d reductum(b)
                r := rβ
                l2 := l1
            else
                l2 := rest(l2)
    g := gcd(a, r)
    return (a exquo g)/(r exquo g)

rem(a, T) ==
    empty?(T) => a/1
    r, s : R
    a/r := headRem(a, T)
    zero? a => 0/1
    b : R_n := lm(a)
    c : R := lc(a)
    while (not zero? a := reductum(a)) repeat
        a/s := headRem(a, T)
        r := sr
        α := lc(a)
        g := gcd(c, α)
        b := s (c exquo g) b + (α exquo g) lm(a)
        c := g
    return(c, b, r)
```

Remarque III.13 Continuons notre étude de la *réduction* d'un polynôme $p \in \mathbf{R}_n$ par rapport à un ensemble triangulaire T de $\mathbf{R}_n$. Désormais, nous ne cherchons plus à déterminer un représentant *aussi simple que possible* de $\overline{p}^{\langle T \rangle}$ mais simplement des informations sur $\overline{p}^{\langle T \rangle}$ comme par exemple sa variable principale, ou son degré en celle-ci. Nous nous intéressons aux notions de *réduction en tête* ([Wu87]) et *réduction au sens des initiaux itérés* introduite dans [Laz91b] et déjà rappelé précédemment (déf. III.8 p. 84). Nous révisons ces deux notions dans la définition suivante.

Définition III.15 *Soit $p \in \mathbf{R}_n$ et T un ensemble triangulaire de $\mathbf{R}_n$. On dira que p est* réduit en tête par rapport à T *si* $\mathsf{head}(p)$ *est réduit par rapport à T. On dira que p est* initiallement réduit par rapport à T *si les polynômes de* $\mathsf{iter}(p)$ *sont réduits par rapport à T.*

Remarque III.14 Avec les notations de la définition précédente, soit encore S_T le sous-ensemble multiplicativement stable de $\mathbf{R}_n$ engendré par 1 et les initiaux des polynômes de T. Nous allons maintenant décrire des algorithmes *produisant* à partir de p et de T des polynômes notés $\mathsf{headReduce}(p, T)$ et $\mathsf{iRed}(p, T)$ respectivement réduit en tête et initiallement réduit par rapport à T et tels qu'il existe e et f dans S_T tels que :

$$(\mathsf{headReduce}(p, T) - ep) \in \mathbf{Id}(T) \text{ et } (\mathsf{iRed}(p, T) - fp) \in \mathbf{Id}(T)$$

Pour cela, étant donné $t \in \mathbf{R}_n$, avec $t \notin \mathbf{R}$, on se donne tout d'abord des opérations $p \mapsto \mathsf{headReduce}(p, t)$ et $p \mapsto \mathsf{iRed}(p, t)$ tel que l'on ait :

$$\mathsf{headReduce}(p, t) = \mathsf{headReduce}(p, \{t\}) \text{ et } \mathsf{iRed}(p, t) = \mathsf{iRed}(p, \{t\})$$

Nous définissons la seconde dans la proposition suivante.

Proposition III.13 *Soit T un ensemble triangulaire de $\mathbf{R}_n$. On dénote par S_T le sous-ensemble multiplicativement stable de $\mathbf{R}_n$ engendré par 1 et les initiaux des polynômes de T. Pour $p \notin \mathbf{R}$ on pose $v = \mathsf{mvar}(p)$, $i_p = \mathsf{init}(p)$, $d_p = \mathsf{mdeg}(p)$ et $t_p = \mathsf{tail}(p)$. L' élément $\mathsf{iRed}(p, T)$ de $\mathbf{R}_n$ peut être défini en itérant les règles suivantes :*

(a) $\mathsf{iRed?}(p, T) \implies \mathsf{iRed}(p, T) = p$

(b) $\mathsf{modulo}(p,\ T_v) = \frac{r}{s} \implies \mathsf{iRed}(p, T) = \mathsf{iRed}(r, T)$

(c) $\mathsf{iRed}(i_p, T_v^-) = r$ *et* $\mathsf{modulo}(s\, i_p\ -\ r,\ T_v^-) = 0 \implies \mathsf{iRed}(p, T) = \mathsf{iRed}(rv^{d_p} + st_p, T)$

Par conséquent, pour chaque $p \in \mathbf{R}_n$ il existe $r \in \mathbf{R}_n$ et $s \in S$ tels que $\mathsf{iRed}(p, T) = r$ *avec* $\mathsf{iRed?}(r, T)$ *et $sp - r \in \langle T \rangle_{\mathbf{R}_n}$. De plus* $\mathsf{iRed}(p, T)$ *peut-être calculé par l'algorithme ci-contre :*

```
iRed(a, b) ==
    b ∈ R => error
    a ∈ R => a
    mvar(a) = mvar(b) => lazyPrem(a, b)
    i_a := init(a)
    m_a := mmon(a)
    t_a := tail(a)
    while (i_a ∉ R) and (mvar(i_a) ≥ mvar(b)) repeat
        if (mvar(i_a) = mvar(b)) and (mdeg(i_a) ≥ mdeg(b))
            then
                i_a/e := modulo(i_a, b)
                t_a := e t_a
        if (i_a ∉ R)
            then
                t_a := tail(i_a) m_a + t_a
                m_a := mmon(i_a) m_a
                i_a := init(i_a)
    return (i_a m_a + t_a)
```

Définition III.16 *On appelle* prédicat de réduction *dans* $\mathbf{R}_n$ *toute relation binaire* redOp ? *dans* $\mathbf{R}_n$ *telle que pour tout couple* (p, t) *de polynômes on ait*

(i) *si p n'est pas initialement réduit par rapport à $\{t\}$ alors* redOp?(p, t) *est faux*

(ii) *si p est (complètement) réduit par rapport à $\{t\}$ alors* redOp?(p, t) *est vrai*

Si redOp ? *est un prédicat de réduction, alors pour $p \in \mathbf{R}_n$ et T un ensemble triangulaire de $\mathbf{R}_n$, on désigne par* redOp?(p, T) *la conjonction des* redOp?(p, t) *pour t décrivant T.*

On appelle opération de réduction *associée au prédicat de réduction* redOp ? *toute opération* $(p, t) \mapsto \mathsf{redOp}(p, t)$ *de* $\mathbf{R}_n \times \mathbf{R}_n$ *dans* $\mathbf{R}_n$ *tel que :*

(i) *il existe $e \in S_T$ tel que* $(\mathsf{redOp}(p, t) - ep) \in \langle t \rangle$

(ii) $\mathsf{redOp}(p, t) = p \Longrightarrow \mathsf{redOp?}(p, t)$

(iii) $\mathsf{redOp}(\mathsf{redOp}(p, t), t) = \mathsf{redOp}(p, t)$

Remarque III.15 Soit donc redOp ? un prédicat de réduction dans $\mathbf{R}_n$ et redOp une opération de réduction associée à redOp ?. On donne maintenant un algorithme qui pour $p \in \mathbf{R}_n$ et T est un ensemble triangulaire de $\mathbf{R}_n$, retourne un polynôme $r \in \mathbf{R}_n$ tel que

(i) il existe $e \in S_T$ tel que : $(r - ep) \in \mathbf{Id}(T)$

(ii) redOp?(r, T) est vrai

```
reduce(p, T, redOp, redOp ?) ==
    (p ∈ R) => p
    empty?(T) => p
    l_1 := sort(<_rwl, [t ∈ T])
    l_2 := l_1
    while (p ∉ R) and (not empty ? l_2) repeat
        t := first(l_2)
```

```
        if redOp?(p, t)
            then
                l2 := rest(l2)
            else
                p := redOp(p, T)
                l2 := l1
    return(p)
```

Nous terminons cette remarque par une notation. On désignera par :

$$(L, T) \mapsto \mathsf{rewriteSet}(L, T, \mathsf{redOp}, \mathsf{redOp}\,?)$$

l'opération qui pour une famille L de polynômes de $\mathbf{R}_n$ et un ensemble triangulaire T de $\mathbf{R}_n$ retourne une famille R de polynômes de $\mathbf{R}_n$ telle que :

(i) s'il existe $p \in L$ tel que $\mathsf{reduce}(p, T, \mathsf{redOp}, \mathsf{redOp}\,?) \in \mathbf{reg}(\mathbf{R})$ alors $R = \{1\}$

(ii) $R = \{\mathsf{reduce}(p, T, \mathsf{redOp}, \mathsf{redOp}\,?) \neq 0 \mid p \in L\}$ sinon

Nous utiliserons cette opération dans la description de certains algorithmes.

Définition III.17 *Soit T un ensemble triangulaire de $\mathbf{R}_n$ et $(p, t) \mapsto \mathsf{redOp?}(p, t)$ un prédicat de réduction d'un polynôme par rapport à un autre. On dira que T est auto-réduit au sens de ce prédicat si l'on a :*

$$(\forall p \in T)\ \ \mathsf{redOp?}(p,\ T^-_{\mathsf{mvar}(p)})$$

Remarque III.16 En pratique et dans la section 5 du chapitre 6 consacrée à la méthode de Wu, la notion de réduction utilisée sera soit la réduction complète, soit la réduction de tête, soit la réduction des initiaux itérés.

6 Systèmes quasi-algébriques

L'objectif de cette section est de commencer à répondre à la question posée dans l'introduction de ce chapitre. À l'aide des ensembles triangulaires, étudiés dans les précédentes sections, nous allons rappeler différents théorèmes donnant des décompositions des variétés affines à l'aide de ces ensembles. Ces théorèmes recevront des preuves constructives dans les prochains chapitres. Rappelons, en effet, que l'un de nos buts est d'étudier des algorithmes calculant de telles décompositions. Par conséquent, il nous faut préciser ce que nous entendons par *décomposer une variété affine algorithmiquement*. D'autre part, il nous faut étudier les *solutions* que fournissent les ensembles triangulaires.

Il ressort déjà de ce qui a été dit (et il ressortira encore plus encore de ce qui sera dit plus loin) que les *bons* ensembles triangulaires doivent être réguliers (déf. III.12 p. 90). Comme on le verra avec le théorème III.6 p. 106, les ensembles triangulaires réguliers et les chaînes régulières de Michael Kalkbrener sont les mêmes objets. Par conséquent, les *solutions* décrites par un ensemble triangulaire doivent au moins ne pas annuler les initiaux des polynômes qui le composent. Cette conclusion, un peu rapide pour l'instant, sera justifiée, entre autre, par le théorème III.4 p. 103. En fait, cette propriété est le point de départ de la définition originale des chaînes régulières ([Kal91]). Elle est également, sous une forme plus faible, l'un des points essentiels de la décomposition de Wu Wen-Tsün ([Wu87], théorème III.7).

Ainsi, résoudre avec des ensembles triangulaires contraint à utiliser des systèmes quasi-algébriques (déf. III.18). Nous commençons cette section par l'étude de ces systèmes. Puis, dans la remarque III.18, nous précisons ce que nous entedrons par résoudre de tels systèmes. Nous passons en suite à l'étude des *zéros réguliers* des ensembles triangulaires. Cette étude est liée aux propriétés des ensembles triangulaires réguliers. En effet, d'une part le *saturé* d'un ensemble triangulaire (déf. III.19) a pour variété affine associée la clôture de Zariski de ses zéros réguliers (th. III.3). D'autre part, la tour d'extensions simples associée à un ensemble triangulaire régulier T est isomorphe à l'anneau total des fractions de l'anneau-quotient de l'anneau des polynômes par le saturé de T (prop. III.18). Ce dernier (et nouveau) résultat a de nombreuses conséquences, tant algorithmiques (voir la section sur la normalisation d'un polynôme par un ensemble de Lazard dans le cinquième chapitre) que théoriques (th. III.5 et III.6, prop. III.19).

Après cette étude, nous rappelons différents théorèmes de *résolution* des systèmes quasi-algébriques à l'aide de zéros réguliers d'ensembles triangulaires. Suivant l'ordre chronologique, nous commençons par le théorème de Wu Wen-Tsün, mentionné plus haut (th. III.7). Sa preuve constructive sera donnée p. 230. Cette preuve est basée sur la notion d'ensemble caractéristique ([Rit32, Rit66]). Une étude détaillée de cette notion est exposée dans [ALM97]. On y présente les liens entre les ensembles caractéristiques et les bases de Gröbner. L'un de ses liens est donné par le th. III.8 dû à Lazard. Nous généralisons ce résultat dans [ALM97]. Un autre résultat important concernant les ensembles caractéristiques et apparaissant dans [ALM97] est le théorème III.10. On y donne trois autres formulations équivalentes de la notion d'ensemble régulier.

Un théorème de Dongming Wang ([Wan93b]) que nous rappelerons dans le chpitre 6 (th. VI.1) donne une décomposition analogue à celle de Wu. Cependant, sa preuve constructive (p. 219) débouche sur un algorithme assez différent et expérimentalement plus efficace (voir la comparaison du sixième chapitre). Cependant les méthodes associées à ces décompositions ne permettent pas toujours de détecter les systèmes quasi-algébriques inconsistants ou d'éliminer, dans une décomposition, les systèmes *résolus* dont les zéros sont contenus dans ceux d'un autre système résolu de la décomposition. Enfin, une même méthode, pour un même système de départ, peut donner, dans des implantations différentes, des décompositions différentes. En exigeant des propriétés supplémentaires sur les ensembles triangulaires, Daniel Lazard a proposé une méthode permettant de résoudre ces problèmes ([Laz91a]). L'objet du cinquième chapitre de cette thèse est de montrer, qu'après s'être donné des outils algébriques adéquats, cette méthode pouvait effectivement être mise en œuvre et que cette méthode est au moins aussi efficace que celles de Wu, Wang ou Kalkbrener. Le théorème III.11 décrit les propriétés des décompositions qu'elle calcule.

Les théorèmes de Wu, Wang et Lazard que nous venons d'évoquer ont en commun le fait qu'ils permettent de décomposer une variété affine en zéros réguliers d'ensembles (ou systèmes) triangulaires. L'ensemble des zéros réguliers d'un ensemble triangulaire n'est pas nécessairement une variété affine (rem. I.13 p. 37). Par suite, étant donnée une décomposition $(W_1, \ldots, W_l)$ en zéros réguliers d'une variété affine V, la famille $(\overline{W}_1, \ldots, \overline{W}_l)$ est une autre décomposition de V, cette fois-ci en clôtures de zéros réguliers. La méthode de Michael Kalkbrener fournit directement des décompositions de cette seconde nature (p. 222).

Définition III.18 *On appelle* système quasi-algébrique *(s.q.a. en abrégé) dans* $\mathbf{P}_n$ *tout couple* $(\{f_1, \ldots, f_s\}, \{h_1, \ldots, h_t\})$ *de deux ensembles de polynômes de* $\mathbf{P}_n$ *que l'on présente généralement comme suit :*

$$\left\{ \begin{array}{lcl} f_1 & = & 0 \\ & \vdots & \\ f_s & = & 0 \\ h_1 & \neq & 0 \\ & \vdots & \\ h_t & \neq & 0 \end{array} \right.$$

Notons Σ *un tel système. Par abus de langage, les polynômes* $f_1, \ldots, f_s$ *(resp.* $h_1, \ldots, h_t$*) sont appelés les équations (resp. inéquations) de* Σ.

Soit $\mathbf{K}$ *un sur-corps de* $\mathbf{k}$. *On appelle* zéro *de* Σ *dans* $\mathbf{K}$ *tout élément du sous ensenble de* $\mathbf{A}^n(\mathbf{K})$ *noté* $\mathbf{Z}_{\mathbf{K}}(\Sigma)$ *et défini par :*

$$\mathbf{Z}_{\mathbf{K}}(\Sigma) \ = \ \mathbf{V}_{\mathbf{K}}(f_1, \ldots, f_s) \setminus \mathbf{V}_{\mathbf{K}}(h)$$

où h *désigne le produit des polynômes* $h_1, \ldots, h_t$.

Le système quasi-algébrique Σ *sera dit* contradictoire *si :*

$$(\exists j \in \{1, \ldots, t\}) \ \mid \ h_j \in \langle f_1, \ldots, f_s \rangle_{\mathbf{P}_n}$$

Le système quasi-algébrique Σ *sera dit* consistant *si*

$$\mathbf{Z}_{\mathbf{K}}(\Sigma) \ \neq \ \emptyset$$

Dans le cas contraire, il sera dit inconsistant.

Enfin, le sytème quasi-algébrique Σ *sera appelé* système triangulaire *si* $\{ f_1, \ldots, f_s \}$ *est un ensenble triangulaire de* $\mathbf{P}_n$.

Proposition III.14 *Avec les notations de la définition précédente, et en supposant que* $\mathbf{K}$ *est une clôture algébrique de* $\mathbf{k}$*, le système* Σ *est inconsistant si et seulement si on a :*

$$h \in \sqrt{\langle f_1, \ldots, f_s \rangle_{\mathbf{P}_n}}$$

Preuve. $\triangleright$ En effet, Σ inconsistant signifie :

$$\mathbf{V_K}(\sqrt{\langle f_1, \ldots, f_s \rangle}) \setminus \mathbf{V_K}(h) = \emptyset$$

D'où, par passage à la clôture dans l'égalité précédente, à l'aide du théorème I.8 et du théorème des zéros faible (proposition I.7)

$$\sqrt{\langle f_1, \ldots, f_s \rangle} : \langle h \rangle = \mathbf{P}_n$$

On en tire la conclusion à l'aide du point (*iii*) de la proposition I.17. $\triangleleft$

Remarque III.17 Si un s.q.a. Σ est contradictoire, alors il est inconsistant. La réciproque est fausse. Considérer le système $(\{x^2\}, \{x\})$.

Notations III.8 Dans la suite, on désignera par $\mathcal{S}$ l'ensemble des s.q.a. dans $\mathbf{P}_n$.

Remarque III.18 Notre but est de donner une description de $\mathbf{Z_K}(\Sigma)$ par un procédé algorithmique. Pour cela, on entendra (pour le moment) par *méthode de résolution dans* $\mathbf{K}$ des s.q.a. la donnée :

1. d'une famille $\mathcal{R}$ de s.q.a. dits *systèmes résolus*
2. d'une application **Algo** : $\mathcal{S} \longmapsto \mathcal{R}$ telle que :
   (a) $\mathbf{Z_K}(\Sigma) = \mathbf{Z_K}(\mathbf{Algo}(\Sigma))$
   (b) $(\forall\, \Sigma \in \mathcal{R})\ \mathbf{Algo}(\Sigma) = (\Sigma)$

On pourra éventuellement exiger les propriétés supplémentaires comme les propriétés suivantes :

1. $\mathbf{Z_K}(\Sigma) = \mathbf{A}^n(\mathbf{K}) \Longrightarrow \mathbf{Algo}(\Sigma) = (\{0\}, \emptyset)$
2. $\mathbf{Z_K}(\Sigma) = \emptyset \Longrightarrow \mathbf{Algo}(\Sigma) = (\{1\}, \emptyset)$

Pour cela **Algo** pourra entre autre mettre en œuvre des règles comme :

1. ôter de (Σ) toute équation $0 = 0$
2. ajouter à (Σ) des combinaisons linéaires des équations
3. ôter toute autre équation ou inéquation si $1 = 0$ est une équation de (Σ) ou si $0 \neq 0$ est une inéquation de (Σ)
4. remplacer (Σ) par $(\{1\}, \emptyset)$, si $p = 0$ et $p \neq 0$ apparaissent dans (Σ)

Cependant on voudrait pouvoir aussi utiliser des règles du type :

1. pour des polynômes p et q de $\mathbf{P}_n$, si l'on a $pq = 0$ est une équation de(Σ), alors remplacer Σ par $(\Sigma + \{p = 0\}, \Sigma + \{q = 0\})$
2. pour un polynôme p de $\mathbf{P}_n$, remplacer Σ par $(\Sigma + \{p = 0\}, \Sigma + \{p \neq 0\})$

Nous devons pour cela étendre la notion précédente de résolution. On entendra donc par méthode de résolution des s.q.a. la donnée :

1. d'une famille $\mathcal{R}$ de S.Q.A. dits *systèmes résolus*
2. d'une application **Algo** de $\mathcal{S}$ dans l'ensemble des parties de $\mathcal{R}$ tel que :
   (a) $\mathbf{Z_K}(\Sigma) = \cup\{\mathbf{Z_K}(\xi) \mid \xi \in \mathbf{Algo}(\Sigma)\}$
   (b) $(\forall\, \Sigma \in \mathcal{R})\ \mathbf{Algo}(\Sigma) \ =\ (\{\Sigma\})$

Le théorème de Wu Wen-Tsün (th. III.7) nous apprend que les systèmes de la forme (T, E), où T est un ensemble triangulaire et E l'ensembles de ses initiaux, est une famille de systèmes résolus des s.q.a. sans inéquations. Le théorème de Dongming Wang (th. VI.1) nous apprend que les systèmes de la forme (T, Q), où T est un ensemble triangulaire et Q un ensemble quelconque de polynômes, est une famille de systèmes résolus pour $\mathcal{S}$.

Définition III.19 *Soit* $\mathbf{K}$ *une clôture algébrique de* $\mathbf{k}$ *(le corps des fractions de* $\mathbf{R}$*). Soit* T *un ensemble triangulaire non vide de* $\mathbf{R}_n$ *et* h *le produit des initiaux des polynômes de* T*. On appelle* zéro régulier *de* T *tout point de la partie de* $\mathbf{A}^n\mathbf{(K)}$ *notée* $\mathbf{ZR_K}(T)$ *et égale à l'ensemble des zéros du système quasi-algébrique* $(T, \{h\})$*. On a donc :*

$$\mathbf{ZR_K}(T) = \mathbf{V_K}(T) \setminus \mathbf{V_K}(h)$$

Lorsque T *satisfait les axiomes de Lazard (définition III.14)* $\mathbf{ZR_K}(T)$ *sera également notée* $\mathbf{W}(T)$ *et appelée* quasi-composante de T

Théorème III.3 *Soit* $\mathbf{K}$ *une clôture algébrique de* $\mathbf{k}$*. Pour tout ensemble triangulaire* T *de* $\mathbf{R}_n$ *on a :*

$$\overline{\mathbf{ZR_K}(T)} = \mathbf{V_K}(\mathsf{sat}(T))$$

Preuve. $\triangleright$ Ce théorème est une conséquence immédiate du théorème I.12 p. 44. $\triangleleft$

Remarque III.19 Nous allons maintenant établir que si un ensemble triangulaire T de $\mathbf{R}_n$ est une chaîne régulière (définition III.11) alors sa représentation n'est autre que le radical de son saturé. Avant cela, nous aurons besoin de quelques propositions intéressantes pour elles-mêmes où l'on s'autorisera la confusion (*habituelle*) entre un ensemble triangulaire de $\mathbf{R}_n$ et l'idéal qu'il engendre dans $\mathbf{P}_n$

Proposition III.15 *Soit* $0 \leq i < n$ *et* T *un ensemble triangulaire de* $\mathbf{P}_i$*. Alors on a :*

(i) $\mathsf{sat}(T[x_{i+1}]) \ =\ \mathsf{sat}(T)[x_{i+1}]$

(ii) $\sqrt{\mathsf{sat}(T[x_{i+1}])} \ =\ \sqrt{\mathsf{sat}(T)}[x_{i+1}]$

Preuve. $\triangleright$ La propriété (i) résulte immédiatement du point (9) de la proposition III.4. La propriété (2) résulte de (i) et du point (6) de cette même proposition. $\triangleleft$

Proposition III.16 *Soit* $0 \leq i \leq n$ *et* T *un ensemble triangulaire non vide de* $\mathbf{P}_i$ *tel que :* $x_i \in \mathsf{algVar}(T)$*. Soit* h *l'initial de* T_{x_i}*. Soit enfin* $r \in \mathbf{P}_i$*. On suppose :*

(i) *h n'appartient à aucun des idéaux premiers associés à* $\sqrt{\mathsf{sat}(\ T^-_{x_i})}$

(ii) $\mathsf{deg}(r, x_i) < \mathsf{mdeg}(\ T_{x_i})$

Alors on a :

$$r \in \mathsf{sat}(T) \implies r \in \sqrt{\mathsf{sat}(\ T^-_{x_i})}[x_i]$$

Preuve. $\triangleright$ Supposons tout d'abord que $T^-_{x_i}$ soit vide. Alors :

$$r \in \mathsf{sat}(T) \implies (\exists e \in \mathbb{N}) \mid h^e r \in \langle\, T_{x_i} \rangle$$

Avec l'hypothèse (ii), il vient :

$$r \in \mathsf{sat}(T) \implies r = 0$$

D'autre part, si $T^-_{x_i}$ est vide, on a :

$$\mathsf{sat}(\, T^-_{x_i}) \;=\; \langle 0 \rangle$$

On a donc bien l'implication recherchée.

Supposons maintenant $T^-_{x_i} \neq \emptyset$ et notons h' le produit des initiaux des polynômes de $T^-_{x_i}$. Supposons que l'on ait : $r \in \mathsf{sat}(T)$. Par définition-même de $\mathsf{sat}(T)$ et puisque $T = \{\, T_{x_i}\} \cup T^-_{x_i}$, il existe donc $m \in \mathbb{N}$ et $q \in \mathbf{P}_i$ tel que : $(hh')^m r + q\, T_{x_i} \in \langle\, T^-_{x_i} \rangle$. Soit $\mathcal{P}$ un idéal premier associé à $\sqrt{\mathsf{sat}(\, T^-_{x_i})}$. Du fait de L'hypothèse (i), le polynôme $\overline{\overline{q\, T_{x_i}}}^{\mathcal{P}}$, s'il est non nul, a un degré en x_i strictement supérieur à celui du polynôme $\overline{\overline{(hh')^m r}}^{\mathcal{P}}$. Or ces deux polynômes sont opposés. On a donc : $\overline{\overline{(hh')^m r}}^{\mathcal{P}} = 0$. Grâce à l'hypothèse (ii), on a en fait $h \notin \mathcal{P}$ et donc $\overline{\overline{(h')^m r}}^{\mathcal{P}} = 0$. Par suite, le produit $(h')^m r$ appartient à l'idéal $\mathcal{P}[x_i]$ qui est premier (proposition III.4). Or, d'après le théorème I.9 p. 41, on a aussi : $h' \notin \mathcal{P}$. On en tire : $r \in \mathcal{P}[x_i]$. On en déduit le résultat annoncé. $\triangleleft$

Proposition III.17 *Soit $0 \le i \le n$ et T un ensemble triangulaire non vide de $\mathbf{P}_i$ tel que : $x_i \in \mathsf{algVar}(T)$. Soit h l'initial de T_{x_i}. Soit enfin $f \in \mathbf{P}_i$. On suppose que h n'appartient à aucun des idéaux premiers associés à $\sqrt{\mathsf{sat}(\, T^-_{x_i})}$. Alors on a :*

$$f \in \sqrt{\mathsf{sat}(T)} \iff (\exists m \in \mathbb{N}) \mid \mathsf{prem}(f^m,\; T_{x_i}) \in \sqrt{\mathsf{sat}(\, T^-_{x_i})}[x_i]$$

Preuve. $\triangleright$ Supposons tout d'abord que l'on ait : $f \in \sqrt{\mathsf{sat}(T)}$. Il existe donc $m \in \mathbb{N}$ tel que : $f^m \in \mathsf{sat}(T)$. Posons alors : $r = \mathsf{prem}(f^m,\; T_{x_i})$ et soit $e \in \mathbb{N}$ et $q \in \mathbf{P}_i$ tel que :

$$r = h^e f^m - q\, T_{x_i}$$

On en tire : $r \in \mathsf{sat}(T)$. A l'aide de la proposition III.16 on en déduit la conclusion recherchée, à savoir : $r \in \sqrt{\mathsf{sat}(\, T^-_{x_i})}[x_i]$.

Réciproquement, supposons qu'il existe $m \in \mathbb{N}$ tel que l'on ait :

$$\mathsf{prem}(f^m,\; T_{x_i}) \in \sqrt{\mathsf{sat}(\, T^-_{x_i})}[x_i]$$

Posons : $r = \mathsf{prem}(f^m,\; T_{x_i})$. On peut supposer $T^-_{x_i} \neq \emptyset$ sans quoi le résultat est immédiat. Soit alors h' le produit des initiaux des polynômes de $T^-_{x_i}$. L'hypothèse entraîne :

$$r \in \sqrt{T} : h'^{\infty}$$

Soit $e \in \mathbb{N}$ et $q \in \mathbf{P}_i$ tel que :

$$h^e f^m = q\ T_{x_i} + r$$

Par suite, on a :

$$h^e f^m \in \sqrt{T} : h'^{\infty}$$

C'est-à-dire :

$$f^m \in (\sqrt{T} : h'^{\infty}) : h^{\infty}$$

À l'aide de la proposition I.17 point (iv) p. 38, on en déduit :

$$f^m \in \mathsf{sat}(T)$$

et la proposition est démontrée. $\lhd$

Théorème III.4 (Aubry) *Soit $0 \leq i \leq n$ et T une chaîne régulière de $\mathbf{P}_i$. Alors on a :*

$$\mathsf{Rep}_i(T) \ = \ \sqrt{\mathsf{sat}(T)}$$

Preuve. $\rhd$ On raisonne par récurrence sur i et l'on s'appuie sur la proposition III.10. Pour $i = 0$, la propriété résulte de la convention : $\mathsf{sat}(\emptyset) = \langle 0 \rangle$ (définition III.19). Supposons maintenant $i > 0$. Si $x_i \notin \mathsf{algVar}(T)$, on utilise le point (1) de la proposition III.10, l'hypothèse de récurrence et la proposition III.15.

Supposons désormais $x_i \in \mathsf{algVar}(T)$. D'après le point (2) de la proposition III.10 :

$$\mathsf{Rep}_i(T) \ = \ \{f \in \mathbf{P}_i \ \mid \ (\exists m \in \mathbb{N})\ \mathsf{prem}(f^m,\ T_{x_i}) \in \mathsf{Rep}_i(\ T^-_{x_i})\}$$

Or, d'après ce qui précède, on a :

$$\mathsf{Rep}_i(\ T^-_{x_i}) \ = \ \sqrt{\mathsf{sat}(\ T^-_{x_i}[x_i])}$$

D'où, avec la proposition III.15 :

$$\mathsf{Rep}_i(\ T^-_{x_i}) \ = \ \sqrt{\mathsf{sat}(\ T^-_{x_i})[x_i]}$$

On conclut alors à l'aide de la proposition III.17. $\lhd$

Remarque III.20 Le résultat suivant, d'apparence technique, est **la colonne vertébrale** de la plupart des résultats nouveaux de ce chapitre.

Proposition III.18 *Soit $0 \leq i \leq n$ et T un ensemble triangulaire régulier de $\mathbf{P}_i$, d'interprétation $(F_0, \ldots, F_i)$ et de tour d'extensions simples associé $(\mathbf{A}_0, \ldots, \mathbf{A}_i)$. Alors on a :*

(1) $(\forall p \in \mathbf{P}_i) \quad F_i(p) = 0 \iff p \in \mathsf{sat}(T)$

(2) $\mathbf{A}_i \ \cong\ \mathbf{fr}(\mathbf{P}_i/\mathsf{sat}(T))$

(3) $(\forall p \in \mathbf{P}_i) \quad F_i(p) \in \mathbf{Un}(\mathbf{A}_i) \iff (\forall \mathcal{P} \in \mathsf{ap}(\mathsf{sat}(T)))\ p \notin \mathcal{P}$

Preuve. $\triangleright$ Vérifions (1). On raisonne par récurrence sur i. Si $i = 0$, la propriété est immédiate. Cela résulte de la définition de F_0 (déf. III.12 p. 90) et de celle du saturé d'un ensemble triangulaire vide (déf. III.10 p. 85). Soit maintenant $i > 0$. Si $x_i \notin \mathsf{algVar}(T)$ alors, d'après la définition III.12, on a :

$$F_i(p) = 0 \iff F_{i-1}(p) = 0$$

Si $p \in \mathbf{P}_{i-1}$ alors, d'après l'hypothèse de récurrence $F_{i-1}(p) = 0$ équivaut à $p \in \mathsf{sat}(T_{x_i}^-)$. À l'aide de la proposition III.15 ceci signifie aussi : $p \in \mathsf{sat}(T)$. Si maintenant $p \in \mathbf{P}_i \setminus \mathbf{P}_{i-1}$, alors $F_{i-1}(p) = 0$ signifie que tous les coefficients de p sont nuls par F_{i-1}, c'est-à-dire par hypothèse de récurrence : $p \in \mathsf{sat}(T_{x_i}^-)[x_i]$. À l'aide de la proposition III.15 on en déduit encore l'équivalence de cette dernière condition avec : $p \in \mathsf{sat}(T)$.

On suppose donc désormais : $x_i \in \mathsf{algVar}(T)$. Soit $p \in \mathbf{P}_i$. Posons $r = \mathsf{prem}(p, T_{x_i})$ et $h = \mathsf{init}(T_{x_i})$. On désigne par h' le produit des initiaux des polynômes de $T_{x_i}^-$. Comme $F_i(h) \in \mathbf{Un}(\mathbf{A}_i)$ (puisque $F_{i-1}(h) \in \mathbf{Un}(\mathbf{A}_{i-1})$) et comme $F_i(T_{x_i}) = 0$ on vérifie l'équivalence :

$$F_i(p) = 0 \iff F_i(r) = 0$$

Nous allons maintenant établir :

$$F_i(r) = 0 \iff F_{i-1}(r) = 0$$

Notons qu'il suffit d'établir l'implication directe. Si $F_i(r) = 0$ alors, par définition de F_i, on a :

$$F_{i-1}(r) \in \langle F_{i-1}(T_{x_i}) \rangle$$

Supposons que $F_{i-1}(r)$ ne soit pas nul. Comme $F_{i-1}(T_{x_i})$ est unitaire :

$$\mathsf{deg}(F_{i-1}(r), x_i) \geq \mathsf{deg}(F_{i-1}(T_{x_i}), x_i)$$

Or, comme $\mathsf{deg}(r, x_i) < \mathsf{mdeg}(T_{x_i})$ on a aussi :

$$\mathsf{deg}(F_{i-1}(r), x_i) < \mathsf{deg}(F_{i-1}(T_{x_i}), x_i)$$

Bref, nous avons bien l'équivalence :

$$F_i(p) = 0 \iff F_{i-1}(r) = 0$$

Par hypothèse de récurrence, il vient :

$$F_i(p) = 0 \iff r \in \mathsf{sat}(T_{x_i}^-)[x_i]$$

Par conséquent :

$$F_i(p) = 0 \implies r \in \langle T \rangle : {h'}^\infty$$

De la définition de $\mathsf{prem}(f, T_{x_i})$, on déduit donc :

$$F_i(p) = 0 \implies p \in \mathsf{sat}(T)$$

Réciproquement, si $p \in \mathsf{sat}(T)$, alors il existe $m \in \mathbb{N}$ tel que : $(hh')^m p \in \langle T \rangle$. Or, par définition de F_i, pour tout $t \in \langle T \rangle$, on a : $F_i(t) = 0$. Comme $F_i(hh') \in \mathbf{Un}(\mathbf{A}_i)$ (rappelons que, pour $j < i$, toute unité dans $\mathbf{A}_j$ demeure une unité dans $\mathbf{A}_{j+1}$) on en déduit : $F_i(p) = 0$.

Nous avons donc vérifié (1). Vérifions (2). Si $i = 0$ le résultat est trivial. Supposons $i > 0$. De (1), par hypothèse de récurrence, il vient pour tout $p \in \mathbf{P}_{i-1}$

$$F_{i-1}(p) = 0 \iff p \in \mathsf{sat}(\,T^-_{x_i}).$$

Le noyau de F_{i-1} est donc $\mathsf{sat}(\,T^-_{x_i})[x_i]$ et l'on a :

$$\mathbf{A}_{i-1}[x_i] \;\cong\; \mathbf{P}_i/(\mathsf{sat}(\,T^-_{x_i})[x_i])$$

Si $x_i \notin \mathsf{algVar}(T)$, on sait que $\mathsf{sat}(\,T^-_{x_i})[x_i] = \mathsf{sat}(T)$. De plus, par définition-même des ensembles réguliers, on a $\mathbf{A}_i = \mathbf{fr}(\mathbf{A}_{i-1}[x_i])$. On obtient donc l'isomorphisme désiré. On procède de même dans le cas $x_i \in \mathsf{algVar}(T)$.

Nous avons donc vérifié (2). Vérifions (3). Soit $p \in \mathbf{P}_i$. Par construction de $\mathbf{A}_i$, nous savons que $F_i(p)$ est soit une unité, soit un diviseur de zéro. D'après (2), $F_i(p)$ est un diviseur de zéro dans $\mathbf{A}_i$ si et seulement si $\overline{p}^{\mathsf{sat}(T)}$ est un diviseur de zéro dans $\mathbf{P}_i/\mathsf{sat}(T)$. Il suffit maintenant d'utiliser le résultat classique suivant. Pour tout idéal I d'un anneau nœthérien $\mathbf{A}$, pour tout $x \in \mathbf{A}$, l'élément $\overline{x}^I$ est un diviseur de zéro dans $\mathbf{A}/I$ si et seulement s'il un idéal premier associé à I auquel x appartient. Voir p.214 du volume 1 de [SZ67]. La propriété (3) en découle. $\triangleright$

Remarque III.21 Soit $0 \leq i \leq n$ et T un ensemble triangulaire régulier de $\mathbf{R}_i$, d'homomorphisme associé F_i. Dans la preuve précédente, on a en fait établi :

$$(\forall p \in \mathbf{P}_i) \quad F_i(p) = 0 \iff \mathsf{prem}(p, T) = 0 \iff p \in \mathsf{sat}(T)$$

On en déduit la proposition suivante.

Proposition III.19 *Soit T un ensemble triangulaire régulier de $\mathbf{R}_n$. Alors on a :*

$$\mathsf{red}_{\mapsto 0}(T) \;=\; \mathsf{sat}(T)$$

Théorème III.5 *Soit $T \subseteq \mathbf{R}_n$ un ensemble de Lazard et $(\mathbf{A}_0, \ldots, \mathbf{A}_n)$ sa tour d'extensions simples associée. Alors on a :*

(i) *chaque $\mathbf{A}_i$ est un produit de corps*

(ii) $\mathsf{sat}(T)$ *est un idéal radical*

(iii) *pour chaque $p, q \in \mathbf{A}[x]$ il existe $g \in \mathbf{A}[x]$ tel que g soit un pgcd de p et q*

preuve $\triangleright$ La propriété (i) résulte du théorème III.2 p. 82. La propriété (ii) résulte de (i) et la proposition III.18. En effet celle-ci montre que $\mathbf{A}_n$ est isomorphe à $\mathbf{fr}(\mathbf{P}_n/\mathsf{sat}(T))$. Ce second anneau est donc réduit. La propriété (iii) résulte de (i) et de la proposition II.24. $\triangleleft$

Proposition III.20 *Soit $n \in \mathbb{N}$ et T une chaîne régulière de $\mathbf{P}_n$. On a*

$$\mathsf{Rep}_n(T) \neq \mathbf{P}_n.$$

Preuve. $\triangleright$ Montrons par récurrence sur l'entier n que $1 \notin \mathsf{Rep}_n(T)$. Lorsque $n = 0$, on a par définition $\mathsf{Rep}_0(T) = \{0\}$. Supposons maintenant $n > 0$.

Dans le cas où $x_n \notin \mathsf{algVar}(T)$, on a avec la proposition III.10

$$\mathsf{Rep}_n(T) = \sqrt{\mathsf{Rep}_{n-1}(T)}[x_n]$$

Par récurrence $1 \notin \mathsf{Rep}_{n-1}(T)$, donc $1 \notin \mathsf{Rep}_n(T)$.

Si $x_n \in \mathsf{algVar}(T)$, alors $1 \in \mathsf{Rep}_n(T)$ si et seulement $1 \in \mathsf{Rep}_{n-1}(\ T^-_{x_n})$ d'après la proposition III.17 Le résultat découle donc immédiatement de l'hypothèse de récurrence. $\lhd$

Théorème III.6 *Soit $0 \leq i \leq n$ et T un ensemble triangulaire de $\mathbf{P}_i$. Alors T est une chaîne régulière de $\mathbf{P}_i$ si et seulement si T est un ensemble régulier de $\mathbf{P}_i$.*

Preuve. $\rhd$ On raisonne par récurrence sur i. Si $i = 0$, la propriété est immédiate. Soit maintenant $i > 0$. Si $x_i \notin \mathsf{algVar}(T)$ alors la propriété résulte de l'hypothèse de récurrence.

Supposons donc que : $x_i \in \mathsf{algVar}(T)$. Par hypothèse de récurrence, $T^-_{x_i}$ est une chaîne régulière de $\mathbf{P}_{i-1}$ si et seulement si $T^-_{x_i}$ est un ensemble régulier de $\mathbf{P}_{i-1}$. Supposons donc que $T^-_{x_i}$ est un ensemble régulier de $\mathbf{P}_{i-1}$. Soit alors associé F_{i-1} l'homomorphisme associé à $T^-_{x_i}$ et soit $(\mathbf{A}_0, \ldots, \mathbf{A}_{i-1})$ sa tour d'extensions simples associé. Alors, d'après la définition III.11, T est est une chaîne régulière de $\mathbf{P}_i$ si et seulement si on a :

$$(\forall \mathcal{P} \in \mathsf{ap}(\mathsf{Rep}_{i-1}(\ T^-_{x_i})))\ \ \mathsf{init}(\ T_{x_i}) \notin \mathcal{P}$$

Or, d'après le théorème III.4, on a :

$$\mathsf{Rep}_{i-1}(\ T^-_{x_i}) \ = \ \sqrt{\mathsf{sat}(\ T^-_{x_i})}$$

Par conséquent, d'après le point (2) de la proposition III.18, T est est une chaîne régulière de $\mathbf{P}_i$ si et seulement si on a :

$$F_{i-1}(\mathsf{init}(\ T_{x_i})) \in \mathbf{Un}(\mathbf{A}_{i-1})$$

C'est-à-dire, d'après la définition III.12, si T est un ensemble régulier de $\mathbf{P}_i$. **Ouf!!!** $\rhd$

Proposition III.21 *Soit $\mathbf{K}$ une clôture algébrique de $\mathbf{k}$ (le corps des fractions de $\mathbf{R}$) de degré de transcendance au moins n sur $\mathbf{k}$. Soit T un ensemble triangulaire normalisé de $\mathbf{R}_n$ Alors on a :*

$$\mathbf{ZR}_{\mathbf{K}}(T) \neq \emptyset$$

Preuve. $\rhd$ Résulte des propositions III.11 p. 91, et III.20 ci-dessus, et du théorème III.6 précédent $\lhd$

Remarque III.22 De cette équivalence entre les chaînes régulières et les ensembles triangulaires réguliers, on peut tirer la remarque suivante, qu'illustreront les algorithmes présentés dans les chapitres suivants. Ce qui peut être fait avec les chaînes régulières et les outils classiques de l'algèbre, comme les idéaux premiers associés à un idéal, peut également être fait de *manière plus intrinsèque* avec les tours d'extensions simples, l'outil-clef, comme nous le verrons plus loin, étant la notion de pgcd (définition II.9).

Nous revenons maintenant aux généralités sur la résolution des systèmes d'équations algébriques. Le théorème suivant, dû à Wu Wen Tsün ([Wu87]), exprime le fait que toute variété affine peut se décomposer comme une réunion finie d'ensembles des zéros réguliers d'ensembles triangulaires.

Théorème III.7 (Wu Wen-Tsün) *Soit F une famille de polynômes de $\mathbf{R}_n$ et $\mathbf{K}$ une clôture algébrique de $\mathbf{k}$. Alors, il existe une famille d'ensembles triangulaires $(T_1, \ldots, T_s)$ telle que :*

$$\mathbf{V}_{\mathbf{K}}(F) \;=\; \cup_1^s \, \mathbf{ZR}_{\mathbf{K}}(T_i)$$

Remarque III.23 Nous réénoncerons ce théorème en le précisant (voir théorème VI.2) et nous en donnerons la preuve constructive dûe à Wu Wen Tsün ([Wu87]). Notons que ce théorème est immédiat si F est de dimension zéro et si $\mathbf{K} = \mathbf{k}$. En effet, les idéaux premiers associés à T sont alors maximaux et ces derniers sont triangulaires, puisque de la forme $(x_1 - a_1, \ldots, x_n - a_n)$ où les $a_1, \ldots, a_n$ appartiennent à $\mathbf{K}$.

Nous allons maintenant donner un autre théorème donnant une décomposition en ensembles triangulaires. Sa preuve utilise les bases de Gröbner et la décomposition primaire. De plus, elle repose essentiellement sur le théorème suivant de Lazard ([Laz92b]). Ce théorème exprime, entre autre chose, le fait que, dans le cas d'un idéal premier P, calculer un ensemble caractéristique (au sens de Lazard) de P n'est pas plus *difficile* théoriquement que d'en calculer une base de Gröbner minimale pour l'ordre lexicographique.

Définition III.20 *Soit* $\mathbf{K}$ *une clôture algébrique de* $\mathbf{k}$. *Soit F une famille de polynômes de* $\mathbf{P}_n$. *On appelle* ensemble caractéristique au sens de Lazard *de F tout ensemble triangulaire initiallement réduit (i.e. auto réduit pour la réduction des initiaux itérés) tel que l'on ait :*

$$\mathbf{V}_{\mathbf{K}}(F) = \overline{\mathbf{ZR}_{\mathbf{K}}(T)}$$

Théorème III.8 (Lazard) *Soit P un idéal premier de $\mathbf{P}_n$ et G une base de Gröbner minimale de P pour l'ordre lexicographique. On désigne par $\mathsf{mvar}(G)$ l'ensemble des $v \in V$ tel qu'il existe $g \in G$ tel que : $\mathsf{mvar}(g) = v$. Pour chaque $v \in \mathsf{mvar}(G)$, on désigne par t_v le polynôme de G tel que $\mathsf{mvar}(t_v) = v$ et $\mathsf{lm}(t_v)$ est minimum parmi les polynômes g de G tels que $\mathsf{mvar}(g) = v$. Soit $T = \{t_v \mid v \in \mathsf{mvar}(G)\}$ l'ensemble triangulaire ainsi construit. Alors, T est un ensemble caractéristique (au sens de Lazard) de P.*

Preuve. $\triangleright$ Remarquons tout d'abord que pour tout $g \in G$ le polynôme $\mathsf{init}(g)$ est réduit par rapport à G au sens des bases de Gröbner, puisque G est une base de Gröbner minimale. Par conséquent :

$$(\forall g \in G) \; \mathsf{init}(g) \notin P$$

Soit maintenant $v \in \mathsf{mvar}(G)$. Nous devons vérifier que t_v est initialement réduit par rapport à T_v^-. Soit alors $i \in \mathsf{iter}(t_v)$ un initial itéré non constant de t_v. Supposons que i ne soit pas réduit par rapport à T_v^- et posons $w = \mathsf{mvar}(i)$. Alors $w \in \mathsf{mvar}(G)$. De plus il existe un entier e et des polynômes q et r tel que :

$$\mathsf{init}(t_w)^e \, i \;=\; q \, t_w \,+\, r \quad \text{et} \quad r <_{rwl} t_w$$

Si $r = 0$ alors $\mathsf{init}(t_w)^e \, i \in \mathbf{Id}(T)$. Par suite, $\mathsf{init}(t_w)^e \, i \in P$. Comme P est premier, l'un des initiaux $\mathsf{init}(t_w)$ ou i appartient à P, ce qui est absurde. Si $r \neq 0$, alors en calculant $\mathsf{iRed}(t_v,\, T_v^-)$ on obtient un polynôme t'_v de variable principale v appartenant à $\mathbf{Id}(T)$ et donc à $\mathbf{Id}(G)$ et tel que $\mathsf{lm}(t'_v) <_{lex} \mathsf{lm}(t_v)$. Ceci contredit le fait que G soit une base de Gröbner minimale et la définition de T. Soit maintenant $f \in \mathbf{Id}(G)$ Il existe $g \in G$ tel que $\mathsf{lm}(g)$ divise $\mathsf{lm}(f)$. Or, par construction-même de T, tout $g \in G$ est réductible, au sens des ensembles triangulaires, par T. Par suite, tout $f \in \mathbf{Id}(G)$ est réductible (au sens des ensembles triangulaires) par T. Désignons par h le produit des initiaux En désignant par h le produit des initiaux des polynômes de T, pour chaque $f \in \mathbf{Id}(G)$ il existe donc un polynôme r, et un entier n tel que

$$(h^n f - r) \in \mathbf{Id}(T) \text{ et } \begin{cases} r = 0 & ou \\ r <_{rwl} f & sinon \end{cases}$$

Or $\mathbf{Id}(T) \subseteq \mathbf{Id}(G)$ Donc $r \in \mathbf{Id}(G)$. Si $r \neq 0$, on peut remplacer f par r dans l'argument précédent. Bref, on a finalement :

$$\mathbf{Id}(T) \subseteq \mathbf{Id}(G) \subseteq \mathbf{Id}(T) : h^\infty$$

Ce qui entraîne :

$$\mathbf{V}(\mathbf{Id}(T) : h^\infty) \subseteq \mathbf{V}(G) \subseteq \mathbf{V}(T)$$

A l'aide du théorème III.3 on tire de la première inclusion :

$$\overline{\mathbf{ZR}_{\mathbf{K}}(T)} \subseteq \mathbf{V}(P)$$

Si $\overline{\mathbf{ZR}_{\mathbf{K}}(T)}$ était vide, h appartiendrait à $\sqrt{\mathbf{Id}(T)}$ et par suite à P. Comme P est premier et qu'aucun des initiaux de T ne lui appartient, ceci est absurde. Par suite, $\mathbf{V}(P)$ étant indécomposable, on a :

$$\overline{\mathbf{ZR}_{\mathbf{K}}(T)} = \mathbf{V}(P)$$

et le théorème est démontré. $\lhd$

Corollaire III.1 *Soit P un idéal premier de $\mathbf{P}_n$. Alors il existe un ensemble triangulaire T de $\mathbf{P}_n$ initialement réduit tel que :*

$$P = \mathsf{sat}(T)$$

Théorème III.9 (Lazard) *Soit F une famille de polynômes de $\mathbf{R}_n$ et $\mathbf{K}$ une clôture algébrique de $\mathbf{k}$. Alors, il existe une famille d'ensembles triangulaires $(C_1, \ldots, C_s)$ initialement réduits et dont les saturés sont des idéaux premiers, le tout tel que :*

$$\mathbf{V}_{\mathbf{K}}(F) = \cup_1^s \overline{\mathbf{ZR}_{\mathbf{K}}(C_i)}$$

Preuve $\rhd$ Il suffit de considérer une décomposition primaire de F et d'appliquer à chacun des idéaux premiers associés à F le corollaire précédent. $\lhd$

Remarque III.24 Nous étudions maintenant une autre notion d'ensemble caractéristique qui nous permet de retrouver les ensembles triangulaires réguliers. Il s'agit de la notion classique dûe à Ritt ([Rit32, Rit66]). Elle repose sur l'ordre original de Ritt et Wu (sans le raffinement de Lazard déf. VI.4 p. 227) sur les ensembles triangulaires que nous rappelons maintenant. Dans [Rit32], la notions d'ensemble caractéristique est définie pour les ensembles triangulaires réduits. Nous l'étendons au cas des *ensembles triangulaires fins*, notion introduite par Wang dans [Wan93b], généralisant celle d'ensemble triangulaire initialement réduit.

Nous commençons donc par cette notion (déf. III.21. Nous passons ensuite à l'ordre de Ritt et Wu sur les les ensembles triangulaires fins. (déf. III.22) puis à la notion d'ensemble caractéristique (au sens de Ritt) d'un idéal. Pour une étude détaillée des ensembles caractéristiques au sens de Ritt voir [Rit66, ALM97]. Nous pouvons alors exposer le théorème III.10 donnant trois caractérisations équivalentes d'un ensemble régulier, et donc d'une chaîne régulière.

Définition III.21 *Un ensemble triangulaire* $T\mathbf{P}_n$ *est dit* fin *si pour tout* $v \in \mathsf{algVar}(T)$ *on a* $\mathsf{prem}(\mathsf{init}(\ T_v),\ T_v^-) \neq 0$.

Définition III.22 *Soient* p, q *deux polynômes de* $\mathbf{R}_n$*. On dira* p *est* inférieur à q pour l'ordre de Ritt et Wu *et l'on écrira* $p \prec_{rw} q$ *si l'une des conditions suivantes est réalisée*

(*i*) $p \in \mathbf{R}$ et $q \notin \mathbf{R}$

(*ii*) $p, q \notin \mathbf{R}$ et $\mathsf{mvar}(p) < \mathsf{mvar}(q)$

(*iii*) $p, q \notin \mathbf{R}$ et $\mathsf{mvar}(p) = \mathsf{mvar}(q)$ et $\mathsf{mdeg}(p) < \mathsf{mdeg}(q)$

On dira que p *est* supérieur à q *(pour ce même ordre) et l'on écrira* $p \succ_{rw} q$ *si on a :* $q \prec_{rw} p$*. Enfin, on dira que* p *et* q *sont* incomparables *(pour ce même ordre) dans tous les autres cas. Notons que :*

$$p \prec_{rw} q \Longrightarrow \mathsf{red?}(p, q)$$

Soient $T = \{t_1, \ldots, t_l\}$ *et* $S = \{s_1, \ldots, s_k\}$ *deux ensembles triangulaires fins de* $\mathbf{P}_n$*. On dira que* T *est* inférieur à S pour l'ordre de Ritt et Wu *et l'on écrira* $T \prec_{rw} S$ *si l'une des conditions suivantes est réalisée*

(*i*) *il existe un entier* $1 \leq i \leq \min(k, l)$ *tel que :*

$$(\forall j \in \{1, \ldots, i-1\})\ t_j \sim_{rw} s_j \quad \text{et} \quad t_i <_{rw} s_i$$

(*ii*) $l > k$ et $(\forall j \in \{1, \ldots, k\})\ t_j \sim_{rw} s_j$

On dira que T *est* supérieur à S *(pour ce même ordre) et l'on écrira* $T \succ_{rw} S$ *si on a :* $S \prec_{rw} T$*. Enfin, on dira que* S *et* T *sont* incomparables *(pour ce même ordre) dans tous les autres cas c'est-à-dire si l'on a :*

$$l = k \quad \text{et} \quad (\forall j \in \{1, \ldots, k\})\ t_j \sim_{rw} s_j$$

Définition III.23 *Soit* I *un idéal de* $\mathbf{P}_n$*. On appelle* ensemble caractéristique au sens de Ritt *de* I *tout ensemble triangulaire fin* T *minimal pour l'ordre de Ritt et Wu parmi les ensembles triangulaires fins contenus dans* I*.*

Théorème III.10 (Aubry, Lazard, Moreno Maza) *Soit* T *un ensemble triangulaire fin de* $\mathbf{P}_n$*. Alors les propositions suivantes sont équivalentes :*

(*i*) T *est régulier*

(*ii*) $\mathsf{red}_{\mapsto 0}(T) = \mathsf{sat}(T)$

(*iii*) T *est un ensemble caractéristique au sens de Ritt de* $\mathsf{sat}(T)$

Preuve $\triangleright$ Nous avons déjà rencontré l'implication $(i) \Longrightarrow (ii)$ (prop. III.19). Vérifions la contra-posée de l'implication $(ii) \Longrightarrow (iii)$. Si T n'est pas minimal parmi les ensembles triangulaires initialement réduits contenus dans $\mathsf{sat}(T)$, c'est qu'il existe un polynôme $p \in \mathsf{sat}(T)$ non réductible à 0 par T. Notons que la réciproque est tout aussi claire. On a donc l'équivalence $(ii) \Longleftrightarrow (iii)$. Vérifions la contra-posée de l'implication $(iii) \Longrightarrow (i)$. Avec nos notations habituelles, on suppose donc qu'il existe $i \in \{1, \ldots, n\}$ tel que $F_{i-1}(\mathsf{init}(T_{x_i}))$ divise zéro dans $\mathbf{A}_{i-1}$. On peut supposer $T_{x_i}^-$ régulier. Par conséquent, d'après la proposition III.18

$$\mathbf{A}_{i-1} \approx \mathbf{fr}(\mathbf{P}_{i-1}/\mathsf{sat}(T_{x_i}^-))$$

Il existe donc un polynôme $p \in \mathbf{P}_{i-1}$ n'appartenant pas à $\mathsf{sat}(T_{x_i}^-)$ tel que :

$$\mathsf{init}(T_{x_i})p \in \mathsf{sat}(T_{x_i}^-)$$

Par conséquent,

$$p \in \mathsf{sat}(T)$$

D'autre part, par hypothèse de récurrence, le théorème est vrai pour (i-1). Comme $T_{x_i}^-$ est régulier de $\mathbf{P}_{i-1}$:

$$p \notin \mathsf{sat}(T_{x_i}^-) \iff p \notin \mathsf{red}_{\mapsto 0}(T_{x_i}^-)$$

Bref, il existe un polynôme $p \in \mathbf{P}_{i-1}$ appartenant à $\mathsf{sat}(T)$ et non réductible à 0 par T. $\triangleleft$

Remarque III.25 Nous allons maintenant énoncer un troisième théorème dû à Lazard (apparaissant sans démonstration dans [Laz91a]) de décomposition d'une variété affine à l'aide d'ensembles triangulaires. Nous donnerons une preuve constructive dans le chpitre 5. Avant cela nous avons besoin d'une définition.

Définition III.24 (Lazard) *Un idéal I de $\mathbf{P}_n$ est dit* quasi-premier *s'il existe un ensemble triangulaire T de $\mathbf{P}_n$ satisfaisant les axiomes de Lazard (définition III.14) tel que*

$$I = \mathsf{sat}(T)$$

Une partie W de $\mathbf{A}^n(\mathbf{K})$ est dite quasi-composante *s'il existe un ensemble triangulaire T de $\mathbf{P}_n$ satisfaisant les axiomes de Lazard tel que*

$$W = \mathbf{W}(T)$$

Soit W une partie de $\mathbf{A}^n(\mathbf{K})$ et $S_1, \ldots, S_l$ des ensembles triangulaires satisfaisant les axiomes de Lazard. On dira que $(S_1, \ldots, S_l)$ est une décomposition de Lazard de W *si l'on a :*

$$W \subseteq \cup_1^l \mathbf{W}(S_i) \subseteq \overline{W}$$

Si $(S_1, \ldots, S_l)$ est décomposition de Lazard de W, celle-ci sera dite sans redondances, *si pour tout $j \in \{1, \ldots, l\}$ la famille $(S_1, \ldots, S_{j-1}, S_{j+1}, \ldots, S_l)$ n'est pas une décomposition de Lazard de W.*

Soit de plus T un ensemble triangulaire satisfaisant les axiomes de Lazard. On dira que $(S_1, \ldots, S_l)$ est une décomposition de Lazard de T *si $(S_1, \ldots, S_l)$ est une décomposition de Lazard de $\mathbf{W}(T)$.*

Proposition III.22 *Avec les notations et hypothèses de la définition précédente. On suppose que* $(S_1, \ldots, S_l)$ *est une décomposition de Lazard de* W*. Si* W *est un fermé* $\mathcal{F}_{\mathbf{k}}(\mathbf{A}^n(\mathbf{K}))$ *(donc une variété affine sur* $\mathbf{k}$*) alors on a :*

$$W \ = \ \cup_1^l \, \mathbf{W}(S_i)$$

Si $(S_1, \ldots, S_l)$ *est un scindage de Lazard de* T *on a :*

$$\mathsf{sat}(T) \ = \ \cap_1^l \, \mathsf{sat}(S_i)$$

Preuve $\triangleright$ Le premier point résulte directement de la définition III.24. Le second résulte du premier par passage à la clôture. $\triangleleft$

Théorème III.11 (Lazard) *Soit* F *une famille de polynômes de* $\mathbf{R}_n$ *et* $\mathbf{K}$ *une clôture algébrique de* $\mathbf{k}$*. Alors, il existe une famille d'ensembles triangulaires* $(T_1, \ldots, T_s)$ *satisfaisant les axiomes de Lazard telle que :*

$$\mathbf{V}_{\mathbf{K}}(F) \ = \ \cup_1^s \, \mathbf{W}(T_i)$$

et donnant une décomposition de Lazard sans redondances de $\mathbf{V}_{\mathbf{K}}(F)$

Remarque III.26 Notons que d'une part, contrairement à la décomposition de Wu Wen-Tsün donnée par le théorème III.7, les ensembles triangulaires T_i du théorème précédent vérifient nécessairement

$$\overline{\mathbf{ZR}_{\mathbf{K}}(T_i)} \neq \emptyset$$

Cela résulte de la preuve de la proposition III.21. Ils ont d'autres propriétés, sur lesquelles nous reviendrons, et résultant du fait que chaque $\mathbf{P}_n/\mathsf{sat}(T_i)$ définit un produit de corps. D'autre part, contrairement à la décomposition en ensembles caractéristiques d'idéaux premiers donnée par le théorème III.9 tout zéro de $\mathbf{V}_{\mathbf{K}}(F)$ est un zéro régulier de l'un des T_i.

Nous terminons cette section en revenant à son problème initial, à savoir la résolution des systèmes quasi-algébriques.

Définition III.25 *Soit* Σ *un s.q.a. dans* $\mathbf{P}_n$*. Soit* I *l'idéal engendré par les équations de* Σ *et* h *le produit des inéquations de* Σ*. On appelle* saturé *de* Σ *l'idéal de* $\mathbf{P}_n$ *noté* $\mathsf{sat}(\Sigma)$ *et défini par :*

$$\mathsf{sat}(\Sigma) \ = \ I : h^{\infty}$$

Théorème III.12 *Soit* $\mathbf{K}$ *une clôture algébrique de* $\mathbf{k}$*. Pour tout s.q.a* Σ *dans* $\mathbf{P}_n$ *on a :*

$$\overline{\mathbf{ZR}_{\mathbf{K}}(\Sigma)} \ = \ \mathbf{V}_{\mathbf{K}}(\mathsf{sat}(\Sigma))$$

De plus, il existe une famille d'ensembles triangulaires $(T_1, \ldots, T_s)$ *telle que :*

$$\overline{\mathbf{ZR}_{\mathbf{K}}(\Sigma)} \ = \ \cup_1^s \mathbf{ZR}_{\mathbf{K}}(T_i)$$

Preuve. $\triangleright$ En effet, la preuve du théorème III.3 reste valable en ne supposant pas que T est un ensemble triangulaire. Ceci nous donne le premier point du théorème. Le second point résulte du premier et du théorème III.7. $\triangleleft$

Chapitre IV

Pgcd au-dessus d'une tour algébrique d'extensions simples

1 Présentation

1.1 Introduction

Les calculs de pgcd de polynômes univariés à coefficients dans une tour algébrique d'extensions simples apparaissent dans divers problèmes en algèbre effective : manipulation de nombres algébriques, résolution sous forme d'ensembles triangulaires de systèmes d'équations algébriques. ... Dans ce chapitre, nous présentons un nouvel algorithme de calcul de ces pgcd. Il est fondé sur la généralisation de l'algorithme des sous-résultants présenté dans la chapitre précédent dans la proposition II.26. Cette nouvelle méthode a permis la résolution de problèmes qui ne pouvaient être traités par les techniques utilisées jusqu'alors. L'une des raisons principales de cette avancée est la suivante. Lorsque l'on calcule des pgcd de polynômes univariés *génériques* à l'aide de l'algorithme classique des sous-résultants, avant de *spécialiser* le résultat avec les relations algébriques décrivant l'anneau des coefficients, nous sommes conduits à travailler avec des polynômes multivariés. Par conséquent, les pseudo-divisions accroissent dramatiquement la taille des coefficients intermédiaires et les divisions exactes à effectuer (qui sont des divisions entre polynômes multivariés) sont coûteuses. Notre idée est de garder le coefficient dominant de chaque polynôme intermédiaire dans le corps de base. Ainsi, nous limitons la croissance des coefficients et rendons plus aisées les divisions exactes qui apparaissent dans l'algorithme des sous-résultants. Pour se faire, nous avons besoin de tester l'inversibilité des coefficients dominants. Ceci conduit à utiliser notre algorithme de façon récursive, en appliquant un *processus à la D5* ([DDD85, Duv87]). Nous allons maintenant présenter le problème d'une façon un peu plus formelle, avant d'en donner un énoncé complet et aussi claire que possible dans la seconde moitié de cette section.

Soit $\mathbf{R}$ un anneau intègre (un anneau commutatif, unitaire et sans diviseurs de zéro), $\mathbf{K}$ son corps des fractions et $\mathbf{K}[X_1]$ l'anneau des polynômes univariés à coefficients dans $\mathbf{K}$. Soit $p_1 \in \mathbf{K}[X_1]$ un polynôme sans carrés de degré positif. Soit $\mathbf{A}_1$ l'anneau-quotient $\mathbf{K}[X_1]/\langle p_1 \rangle$. Comment tester l'inversibilité d'un élément q de $\mathbf{A}_1$? Ceci peut être fait en interprétant q comme un élément de $\mathbf{K}[X_1]$ et en calculant un pgcd g de p et q dans $\mathbf{K}[X_1]$. On obtient alors une *factorisation* $p_1 = fg$ et à une décomposition de $\mathbf{K}[X_1]/\langle p_1 \rangle$ comme isomorphe au produit des anneaux $\mathbf{K}[X_1]/\langle f \rangle$ et $\mathbf{K}[X_1]/\langle g \rangle$. Sur ce produit, la première composante de q est inversible, tandis que la seconde est nulle. Ainsi, sans avoir eu besoin de supposer p_1 irréductible, nous avons pu répondre. C'est sur cette remarque de D. Lazard qu'est fondé le système D5 ([DDD85, Duv87]) de manipulation de nombres algébriques.

Nous venons de voir que le problème de l'inversibilité dans $\mathbf{K}[X_1]/\langle p_1 \rangle$ nous a conduit à un calcul de pgcd dans $\mathbf{K}[X_1]$ ainsi qu'à un processus de *discussion*. Ces notions de pgcd et de discussion abordées dans le second chapitre apparaissent dans beaucoup de méthodes de résolution des systèmes d'équations algébriques ([Kal91, Laz91a, Laz92a]). Le problème du calcul de pgcd de polynômes univariés à coefficients dans une tour algébrique d'extensions simples séparables (chapitre suivant) est résolu en utilisant la propriété de spécialisation de l'algorithme des sous-résultants (propriété rappelée dans la remarque II.10 du chapitre précédent, voir aussi [Loo82]) par une méthode décrite dans [Laz91a]. Nous allons la replacer, dans la section suivante, dans le cadre qui nous intéresse dans ce chapitre, à savoir celui des tours algébriques d'extensions simples. Cependant une première implantation de l'algorithme lextriangular de D. Lazard, algorithme que nous exposerons plus loin ([Laz92a]), utilisant cette méthode conduisait à une saturation de la mémoire de l'ordre de 400 méga-octets pour certains problèmes. Ceci fut la motivation initiale pour le développement d'un nouvel algorithme de pgcd, avec lequel nous traitons facilement aujourd'hui ces mêmes problèmes.

Dans la seconde partie de cette section nous allons présenter de façon plus précise le problème du pgcd qui nous intéresse dans ce chapitre. Dans la section suivante nous allons exposer les algorithmes résolvant ce problème. Ces algorithmes sont, à des détails de syntaxe près, nos programmes écrits dans le langage du système de calcul formel AXIOM ([JS92]). Ainsi, nous espérons qu'ils puissent être facilement adaptés dans un autre langage. Dans la dernière section de ce chapitre nous montrons comment cette nouvelle approche de calcul de pgcd permet une implantation efficace de l'algorithme lextriangular. En particulier, elle a permis une résolution automatique complète sous forme d'ensembles triangulaires du *système-test cyclic-7* ([Fau94]) et de retrouver les résultats exposés dans [BF91]. Ce travail commun avec Renaud Rioboo a été exposé à Paris lors du onzième congrès AAECC en 1995 ([MR95]).

1.2 Exposé du problème

Supposons que nous ayons introduit des polynômes non constants $p_1, \ldots, p_n$ et des anneaux $\mathbf{A}_1, \ldots, \mathbf{A}_n$ de telle sorte que :

$$p_i \in \mathbf{A}_{i-1}[X_i] \quad \text{avec :} \quad \mathsf{lc}(p_i) \in \mathbf{A}_0 = \mathbf{K} \quad \text{et :} \quad \mathbf{A}_i = \mathbf{A}_{i-1}[X_i]/\langle p_i \rangle$$

Remarquons que les anneaux $\mathbf{A}_i$ ne sont pas nécessairement intègres. D'autre part les anneaux $\mathbf{A}_i$ sont des $\mathbf{K}$-algèbres de type fini et chaque élément d'un anneau $\mathbf{A}_i$ est soit une unité, soit un diviseur de zéro Par suite, les anneaux $\mathbf{A}_1, \ldots, \mathbf{A}_n$ forment une tour algébrique d'extensions simples de $\mathbf{K}$ au sens de la définition III.3. Posons $\mathbf{P}_i = \mathbf{K}[X_1, \ldots, X_i]$ pour $1 \leq i \leq n+1$ et $\mathbf{P}_0 = \mathbf{K}$. Pour $1 \leq i \leq n$ on considère l'homomorphisme surjectif canonique H_i de $\mathbf{A}_{i-1}[X_i]$ dans $\mathbf{A}_i$ et l'homomorphisme surjectif F_i de $\mathbf{P}_{i+1}$ vers $\mathbf{A}_i[X_{i+1}]$ défini par

(i) $F_i(X_{i+1}) \; = \; X_{i+1}$

(ii) $(\forall p \in \mathbf{P}_i)$ $F_i(p) = \begin{cases} H_1(p) & \text{if } i = 1 \\ H_i(F_{i-1}(p)) & \text{if } i > 1 \end{cases}$

Soient $\widehat{p}_1, \ldots, \widehat{p}_n$ des polynômes de $\mathbf{P}_n$ *représentant* les p_i, c'est à dire tels que $\widehat{p}_i \in \mathbf{P}_i \backslash \mathbf{P}_{i-1}$ et $F_i(\widehat{p}_i) = p_i$ (nous pouvons supposer que le coefficient dominant de $\widehat{p}_i$, vu comme un polynôme univarié en X_i, appartient à $\mathbf{K}$). Soit I l'idéal engendré par les polynômes $\widehat{p}_1, \ldots, \widehat{p}_n$ dans $\mathbf{P}_n$. L'anneau $\mathbf{P}_n/I$ est isomorphe à $\mathbf{A}_n$.

Soient a et b dans $\mathbf{A}_n[X_{n+1}]$ supposés posséder un pgcd au sens de la définition II.9. Rappelons que d'après la proposition II.25 si I est radical, alors l'anneau $\mathbf{A}_n$ est un produit de corps, et a et b possèdent des pgcd. Soient $\widehat{a}$ et $\widehat{b}$ des représentants dans $\mathbf{P}_n[X_{n+1}]$ de a et b respectivement (i.e. $F_n(\widehat{a}) = a$ and $F_n(\widehat{b}) = b$). La méthode classique pour calculer un tel pgcd consiste dans un premier temps à calculer la suite des restes $(\widehat{r}_0, \widehat{r}_1, \ldots, \widehat{r}_m, \widehat{r}_{m+1})$ fournie par l'algorithme des sous-résultants appliqué à $\widehat{a}$ et $\widehat{b}$ dans $\mathbf{P}_n[X_{n+1}]$. Puis, dans un second temps, on applique la propriété de spécialisation de cette algorithme (voir la remarque II.10).

Soit alors j le plus grand entier tel que $F_n(\widehat{c}_j) \neq 0$, où $\widehat{c}_j$ désigne le coefficient dominant de $\widehat{r}_j$. Posons $r_j = F_n(\widehat{r}_j)$. D'après la propriété de spécialisation nous savons que $r_k = 0$ pour $k > j$. On détermine ensuite si $F_n(\widehat{c}_j)$ est inversible dans $\mathbf{A}_n$. Si $\widehat{c}_j \in \mathbf{K}$ alors ceci est le cas et r_j est un pgcd quasi-unitaire de a et b. Sinon, il existe un entier l tel que $\widehat{c}_j \in \mathbf{P}_l \setminus \mathbf{P}_{l-1}$. On calcule alors un pgcd de $F_{l-1}(\widehat{c}_j)$ et de p_l dans $\mathbf{A}_{l-1}[X_l]$ en appliquant récursivement le processus que nous sommes en train de décrire. Si ce pgcd p'_l est de degré nul (en X_l) nous savons que $\widehat{c}_j$ est inversible dans $\mathbf{A}_n$ et que r_j est un pgcd de a et b. Sinon, nous avons trouvé une factorisation $p_l = p'_l p''_l$ dans $\mathbf{A}_{l-1}[X_l]$ et soit $\widehat{p}'_l$ et $\widehat{p}''_l$ dans $\mathbf{P}_l$ tels que $p'_l = F_{l-1}(\widehat{p}'_l)$ et $p''_l = F_{l-1}(\widehat{p}''_l)$. On considère alors les idéaux I' et I'' de $\mathbf{P}_n$ obtenus à partir de I en remplaçant respectivement $\widehat{p}_l$ par $\widehat{p}'_l$ et $\widehat{p}''_l$. Comme nous l'avons fait pour $\mathbf{A}_n$, on définit les séquences $\mathbf{A}'_l, \ldots, \mathbf{A}'_n$ et $\mathbf{A}''_l, \ldots, \mathbf{A}''_n$ avec les homomorphismes surjectifs associés F'_n et F''_n respectivement. Ainsi, on identifie d'une part $\mathbf{A}'_n$ avec $\mathbf{P}_n/I'$ et d'autre part $\mathbf{A}''_n$ avec $\mathbf{P}_n/I''$. Dans la tour $\mathbf{P}_n/I''$, $F''_n(\widehat{c}_j)$ est inversible et $F''_n(\widehat{r}_j)$ est un pgcd de $F''_n(\widehat{a})$ et $F''_n(\widehat{b})$. Par contre, dans la tour $\mathbf{P}_n/I'$ nous avons $F''_n(\widehat{r}_j) = 0$, et un pgcd doit être recherché en repartant de la suite des restes $(\widehat{r}_0, \widehat{r}_1, \ldots, \widehat{r}_{j-1})$ (en considérant le dernier reste dont le coefficient dominant ne se spécialise pas à zéro).

En plus des problèmes techniques que présentent les gestions de la discussion et de la récursion dans cette méthode, un facteur limite sérieusement son utilisation : la croissance des polynômes (multivariés) $\widehat{r}_j$ rendant fort coûteuses les opérations comme la division. Ceci conduit à l'idée de remplacer les calculs dans $\mathbf{P}_n$ par des calculs dans $\mathbf{A}_n$.

2 Quasi-inverses et pgcd au-dessus d'une tour algébrique d'extensions simples

2.1 Notion de quasi-inverse

Définition IV.1 *Soit* $\mathbf{R}$ *un anneau intègre et* $\mathbf{A}$ *un sur-anneau de* $\mathbf{R}$ *sans* $\mathbf{R}$*-torsion, i.e. :* $\mathbf{reg}(\mathbf{R}) \subseteq \mathbf{reg}(\mathbf{A})$. *Soit* $x \in \mathbf{A}$. *On dira qu'un couple* $(y, r) \in \mathbf{A} \times \mathbf{R}$ *est un* quasi-inverse relativement à $\mathbf{R}$ de x *si l'on a les conditions suivantes :*

(i) $xy = r$

(*ii*) $x \neq 0 \Longrightarrow y \neq 0$

Remarque IV.1 Soit $x \in \mathbf{A}$ et (y, r) un quasi-inverse (relativement à $\mathbf{R}$) de x. On a :

$$x \notin \mathbf{reg}(\mathbf{A}) \Longleftrightarrow r = 0.$$

En effet, si r est nul, x est un diviseur de zéro de $\mathbf{A}$. Réciproquement, soit $z \in \mathbf{A}, z \neq 0$ tel que $zx = 0$. On a :

$$xyz = zr = 0$$

Comme $\mathbf{A}$ est sans $\mathbf{R}$-torsion, si $r \neq 0$ alors $z = 0$. Contradiction.

Remarque IV.2 Par suite, si $x \in \mathbf{reg}(\mathbf{A})$ alors $\frac{y}{r}$ est l'inverse de x dans $\mathbf{fr}(\mathbf{A})$. Si $x \notin \mathbf{reg}(\mathbf{A})$, pour quasi-inverse de x, on peut prendre un couple $(y, 0)$, où y est un diviseur de zéro conjoint de x.

Remarque IV.3 Dans le cas d'une tour algébrique d'extensions simples du corps des fractions d'un anneau intègre, nous allons montrer comment le calcul d'un quasi-inverse et celui d'un pgcd (décrit dans l'introduction) se *résolvent mutuellement.*

2.2 Spécifications des algorithmes

Nous avons deux types d'algorithmes. D'une part, ceux qui *cherchent à calculer* un pgcd ou un quasi-inverse, modulo un certain ensemble triangulaire T - nous précisons ci-dessous cette terminologie - et qui éventuellement retournent un scindage de T. Les noms des opérations correspondantes sont suffixées par $\mathsf{ElseSplit}$. D'autre part, ceux qui calculent un pgcd ou un quasi-inverse, modulo un certain ensemble triangulaire à l'aide des algorithmes précédents et rendent un résultat sous la forme d'une *discussion.*

Un nombre algébrique α intervenant au cours des calculs est *codé* par un polynôme p de $\mathbf{R}_n$ et un ensemble triangulaire T modulo lequel p est réduit et définissant la tour où *vit* α. Les calculs effectués par les algorithmes se font donc sur des polynômes de $\mathbf{R}_n$ (et non pas de $\mathbf{P}_n$, afin de limiter les manipulations de fractions) De plus, ces algorithmes sont implantables tel quel en AXIOM à des détails de syntaxe près. Pour les décrire nous donnons tout d'abord leurs spécifications d'entrée, puis celles de sortie. Pour un algorithme $\mathsf{algorithmeQuiCalcule}$ travaillant sur des quantités $a, b, T, \ldots$ nous notons $H_{aqc}(a, b, T, \ldots)$ les spécifications d'entrée et $S_{aqc}(a, b, T, \ldots)$ celles de sortie

Notations IV.1 Dans toute cette section T est un e.t.a.r. (ensemble triangulaire auto-réduit). On pose $\mathsf{var}(T) = \{v_1, \ldots, v_m\}$, $\mathbf{B} = \mathbf{K}[v_1, \ldots, v_m]$ et $\mathbf{A} = \mathbf{B}/T$. Soient $v \in V$, et $p, p_1, p_2 \in \mathbf{R}_n$. On notera $H_{qres}(p, T)$ la condition : $\mathsf{var}(p) \subseteq \mathsf{var}(T)$. Et, on notera $H_{srges}(p_1, p_2, v, T)$ la conjonction des conditions suivantes :

(*i*) $(\forall w \in \mathsf{var}(T))\ w < v$

(*ii*) $\mathsf{deg}(p_1, v) \geq \mathsf{deg}(p_2, v) > 0$

(*iii*) $(\mathsf{var}(p_1) \cup \mathsf{var}(p_2)) \setminus \{v\} \subseteq \mathsf{var}(T)$

(*iv*) $\mathsf{modulo}(p_1, T) = p_1$ et $\mathsf{modulo}(p_2, T) = p_2$

Définition IV.2 *On suppose que $H_{srges}(p_1, p_2, v, T)$ est satisfaite. Soit $g \in \mathbf{R}_n$ réduit par rapport à T et satisfaisant : $\mathsf{var}(g) \subseteq \mathsf{var}(p_1) \cup \mathsf{var}(p_2)$. Ces polynômes s'identifiant à des éléments de $\mathbf{A}[v]$, on dira que g est un* pgcd modulo T *de p_1 et p_2 si g est un pgcd de p_1 et p_2 dans $\mathbf{A}[v]$, au sens de la définition* II.9, *avec $\mathsf{lc}(g) \in \mathbf{reg}(\mathbf{R})$. On suppose que $H_{qres}(T)$ est satisfaite. On dira que $(q, r) \in \mathbf{R}_n \times \mathbf{R}$ est un* quasi-inverse *de p modulo T si q est réduit par rapport à T et si (q, r) est un quasi-inverse de p dans $\mathbf{A} \times \mathbf{R}$, au sens de la définition* IV.1

Les spécifications de sortie des opérations sont les suivantes :

$(S_{qres}(T))$: pour tout $p \in \mathbf{R}_n$, si $H_{qres}(p, T)$ alors $\mathsf{quasiRecipElseSplit}(p, T)$ retourne

- soit $(q, r) \in \mathbf{R}_n \times \mathbf{R}$ un quasi-inverse de p modulo T
- soit deux e.t.a.r. T_1, T_2 tel que $T \longmapsto (T_1, T_2)$ soit un scindage fondamental

$(S_{srges}(T))$: pour tout $(p_1, p_2, v) \in {\mathbf{R}_n}^2 \times V$, si $H_{srges}(p_1, p_2, v, T)$ alors $\mathsf{subResGcdElseSplit}$ $(p_1,\ p_2,\ v\ ,T)$ retourne

- soit $g \in \mathbf{R}_n$ réduit par rapport à T tel que g soit un pgcd modulo T de $p1$ et $p2$
- soit deux e.t.a.r. T_1, T_2 tel que $T \longmapsto (T_1, T_2)$ soit un scindage fondamental

$(S_{esrges}(T))$: pour tout $(p_1, p_2, v) \in {\mathbf{R}_n}^2 \times V$, si $H_{srges}(p_1, p_2, v, T)$ alors $\mathsf{extendedSubResGcdElseSplit}$ (p_1, p_2, v, T) retourne

- soit $(g, f, k) \in \mathbf{R}_n^3$, ces polynômes étant réduits par rapport à T et tels que g soit un pgcd modulo T de $p1$ et $p2$, le tout tel que :

$$\mathsf{modulo}(fp_1 + kp_2, T) = g$$

- soit deux e.t.a.r. T_1, T_2 tel que $T \longmapsto (T_1, T_2)$ soit un scindage fondamental

$(S_{srg}(T))$: pour tout $(p_1, p_2, v) \in {\mathbf{R}_n}^2 \times V$, si $H_{srges}(p_1, p_2, v, T)$ alors $\mathsf{newSubResGcd}$ (p_1, p_2, v, T) retourne une liste $[[g_1, T_1], \ldots, [g_m, T_m]]$ où $g_i \in \mathbf{R}_n$ et où T_i est un e.t.a.r. de $\mathbf{R}_n$, le tout tel que :

- $\mathsf{subResGcdElseSplit}$ (p_1, p_2, v, T_i) retourne g_i
- $T \longmapsto (T_1, \ldots, T_m)$

$(S_c(T))$: pour tout $p \in \mathbf{R}_n$, si $H_{qres}(p, T)$ alors $\mathsf{quasiRecip}(p, T)$ retourne une liste

$$[[[\widehat{p}_1, \parallel p \parallel_1], T_1], \ldots, [\widehat{p}_m, \parallel p \parallel_m], T_m]]$$

où $\widehat{p}_i \in \mathbf{R}_n$, $\parallel p \parallel_i \in \mathbf{R}$, et où T_i est un e.t.a.r. de $\mathbf{R}_n$, le tout tel que :

- $\mathsf{quasiRecipElseSplit}(p, T_i)$ retourne $(\widehat{p}_i, \parallel p \parallel_i)$
- $T \longmapsto (T_1, \ldots, T_m)$

De plus, lorsque T est radical en tant qu'idéal de $\mathbf{P}_n$, tous les scindages produits par ces opérations sont biunivoques.

Enfin, on se donne une opération $\mathsf{split?}$ prenant pour argument n'importe quel résultat produit par les trois premières opérations et retournant *vrai* si et seulement s'il s'agit d'un scindage. La syntaxe des algorithmes est celle du langage du système AXIOM ([JS92]) avec lequel ils ont été mis au point.

2.3 L'algorithme quasiRecipElseSplit

quasiRecipElseSplit(p, T) ==
  p/s := modulo(p, T)
  $p \in \mathbf{R} \hookrightarrow$ **return** $([s, p])$
  u := mvar(p)
  $esrges$:= extendedSubResGcdElseSplit (T_u, p, u, T_u^-)
  if split ?$(esrges)$
  then
    $[T_1, T_2] := esrges$
    return $[T_1 + T_u + T_u^+, T_2 + T_u + T_u^+]$
  else
    $[f, k, g] := esrges$
    $\{$ *nous avons :* modulo$(f\ T_u + k\,p,\ T_u^-) = g$ $\}$ *avec :* init$(g) \in \mathbf{reg}(\mathbf{R})$
    if deg$(g, u) > 0$
    then
      q := pquo(T_u, g)
      return $[\ T_u^- + \{g\} + T_u^+,\ T_u^- + \{q\} + T_u^+]$
    else
      $\{$ *nous avons :* $g \in \mathbf{reg}(\mathbf{R})$ $\}$
      return $[(k\ s),\ g]$

Nous dégageons la remarque importante suivante :

Proposition IV.1 *Supposons que les spécifications $S_{qres}(T)$ et $S_{esrges}(T_u^-)$ soient vérifiées et que* quasiRecipElseSplit(p, T) *ne retourne pas un scindage.*

En posant $(\widehat{p}, \| p \|) =$ quasiRecipElseSplit(p, T), *alors on a :*

(i) modulo$(p, T) = 0 \iff \| p \| = 0$

(ii) modulo$(p, T) \neq 0 \iff \overline{p}^T \in \mathbf{reg}(\mathbf{A})$

Remarque IV.4 La preuve de la remarque précédente est facile. Plus généralement, nos algorithmes induisent une preuve constructive du fait suivant : si T est un idéal radical de $\mathbf{P}_n$ alors tout élément de $\mathbf{A}$ est soit un diviseur de zéro, soit inversible. Pour obtenir ce résultat il faut combiner la proposition suivante (dont la preuve est pour l'instant non constructive) avec les les propositions validant nos algorithmes (qui seront établies plus loin). La preuve de la propositions suivante devient alors constructive, et l'on obtient une preuve constructive du fait précédent. De plus, la proposition suivante montre que le calcul d'un quasi-inverse modulo T revient à celui d'un pgcd modulo un *sous-ensemble triangulaire* de T.

Proposition IV.2 *On suppose que l'idéal engendré par T dans $\mathbf{P}_n$ est radical, que* var$(p) \subseteq$ var(T) *et $\overline{p}^T \in \mathbf{reg}(\mathbf{A})$. Alors, il existe $q \in \mathbf{P}_n$ tel que $\overline{pq}^T = 1$. De plus, le calcul de q se ramène au problème suivant : pour $a, b \in \mathbf{R}_n$ et $w \in$* var(T), *tel que $H_{srges}(a, b, w,\ T_w^-)$ soit satisfaite, trouver $u, v, d \in \mathbf{R}_n$ tels que leurs variables, sauf w, soient algébriques dans T_w^-, d est un pgcd modulo T_w^- de a et b, et* modulo$(au + bv,\ T_w^-) = d$.

Preuve. $\triangleright$ On raisonne par récurrence sur $|\,$deg$(T)\,|$. Si $|\,$deg$(T)\,| = 0$ ou si $p \in \mathbf{reg}(\mathbf{R})$ et on prend $q = p^{-1}$. Supposons $|\,$deg$(T)\,| > 0$ et $p \notin \mathbf{reg}(\mathbf{R})$. Posons $u =$ mvar(p) et

$W = \mathsf{var}(T_u^-)$. Remarquons que $\mathbf{Id}_{\mathbf{K}[W]}(T_u^-)$ est radical. En effet, d'une part $\mathbf{Id}_{\mathbf{P}_n}(T)$ est intersection (finie) d'idéaux premiers de $\mathbf{P}_n$. D'autre part, si P est un idéal premier de $\mathbf{P}_n$, alors $P \cap \mathbf{K}[W]$ est un idéal premier de $\mathbf{K}[W]$. Par suite, T_u^- est intersection (finie) d'idéaux premiers de $\mathbf{K}[W]$. Bref, $\mathbf{Id}_{\mathbf{K}[W]}(T_u^-)$ est radical. Posons $\mathbf{B'} = \mathbf{K}[W]$ et $\mathbf{A'} = \mathbf{B'}/T_u^-$. Comme $\mathbf{Id}_{\mathbf{K}[W]}(T_u^-)$ est radical et de dimension zéro, d'après la proposition II.25, il existe $d \in \mathbf{B'}[u]$ tel que :

$$\langle \overline{p}^{\,T_u^-},\ T_u \rangle_{\mathbf{fr}(\mathbf{A'})[u]} \;=\; \langle \overline{d}^{\,T_u^-} \rangle_{\mathbf{fr}(\mathbf{A'})[u]}$$

Par conséquent, il existe $\alpha, \beta \in \mathbf{reg}(\mathbf{A'})$ et $p', (T_u)' \in \mathbf{B'}[u]$ tel que :

$$\alpha \overline{p}^{\,T_u^-} = \overline{dp'}^{\,T_u^-} \quad \text{et} \quad \beta\, T_u = \overline{d(T_u)'}^{\,T_u^-}$$

Comme $\overline{(T_u)'\alpha p}^{\,T_u^- \cup \{T_u\}} = 0$, on doit avoir $\overline{(T_u)'}^{\,T_u^- \cup \{T_u\}} = 0$, et donc $\mathsf{deg}((T_u)', u) > 0$, sans quoi $\overline{p}^{\,T_u^-}$ serait un diviseur de zéro de $\mathbf{fr}(\mathbf{A'})[u]$ ce qui contredirait l'hypothèse : $\overline{p}^T \in \mathbf{reg}(\mathbf{A})$ (proposition III.2).

Pour cette même raison, on vérifie que : $\mathsf{deg}(d, u) = 0$. Par suite, il existe $f, g \in \mathbf{B'}[u]$ et $\alpha, \beta, \gamma \in \mathbf{reg}((\mathbf{A'}))$ tel que :

$$\overline{\tfrac{f}{\alpha} p + \tfrac{g}{\beta}\, T_u}^{\,T_u^-} \;=\; \gamma$$

Or, par hypothèse de récurrence, il existe $a, b, c \in \mathbf{B'}$ tel que :

$$\alpha \overline{a}^{\,T_u^-} \;=\; \beta \overline{b}^{\,T_u^-} \;=\; \gamma \overline{c}^{\,T_u^-} \;=\; 1$$

Par suite, on a :

$$\overline{c(afp + bg\, T_u)}^{\,T_u^-} \;=\; 1$$

Finalement, il vient

$$\overline{(caf)p}^{\,T} \;=\; 1$$

Le polynôme recherché est $q = caf$, ce qui termine la preuve. $\triangleleft$

Proposition IV.3 *Supposons que les spécifications $S_{qres}(T)$ et $S_{esrges}(T_u^-)$ soient vérifiées, que l'idéal engendré par T dans $\mathbf{P}_n$ soit radical et que* $\mathsf{quasiRecipElseSplit}(p, T)$ *retourne un scindage (T_1, T_2) de T. Alors, les ensembles triangulaires T, T_1, T_2 étant vus comme des idéaux de $\mathbf{P}_n$, ce scindage est biunivoque.*

Preuve. $\triangleright$ Nous reprenons les notations de l'algorithme et W, $\mathbf{B'}$ et $\mathbf{A'}$ comme ci-dessus. On va montrer que :

$$\mathbf{Id}_{\mathbf{P}_n}(T_u^-) + \langle g, q \rangle_{\mathbf{P}_n} \;=\; \mathbf{P}_n$$

D'après la spécification $S_{esrges}(T_u^-)$, il existe $d \in \mathbf{B'}[u] \cap \mathbf{R}_n$ tel que :

$$\langle \overline{g}^{\,T_u^-}, \overline{q}^{\,T_u^-} \rangle_{\mathbf{fr}(\mathbf{A'})[u]} \;=\; \langle \overline{d}^{\,T_u^-} \rangle_{\mathbf{fr}(\mathbf{A'})[u]} \quad \text{et :}$$

$$\mathsf{init}(d) \in \mathbf{reg}(\mathbf{R})$$

Ainsi, il existe $\alpha, \beta \in \mathbf{reg}((\mathbf{A'}))$ et $g', q' \in \mathbf{B'}[u]$ tel que :

$$\alpha \overline{g}^{\,T_u^-} = \overline{dg'}^{\,T_u^-} \text{ et } \beta \overline{q}^{\,T_u^-} = \overline{dq'}^{\,T_u^-}$$

D'après $S_{esrges}(T_u^-)$ et la formule définissant q, les polynômes g et q ont leurs initiaux dans $\mathbf{reg}(\mathbf{R})$. Par suite, les initiaux des polynômes $\overline{g'}^{\,T_u^-}$ et $\overline{q'}^{\,T_u^-}$ sont dans $\mathbf{reg}((\mathbf{A'}))$. Puisque $\mathbf{Id}_{\mathbf{P}_n}(T)$ est radical, on vérifie tout d'abord que l'on a $\mathsf{deg}(g', u) > 0$ et $\mathsf{deg}(q', u) > 0$. De même, si l'on avait $\mathsf{deg}(d, u) > 0$, alors le polynôme $f = g'q'd$ vérifierait $f \notin \mathbf{Id}_{\mathbf{P}_n}(T)$ (en effet $\mathsf{init}(\overline{f}^{\,T_u^-}) \in \mathbf{reg}((\mathbf{A'}))$ et $\mathsf{deg}(f, u) < \mathsf{deg}(T_u, u)$) et $f^2 \in \mathbf{Id}_{\mathbf{P}_n}(T)$. Par suite : $d \in \mathbf{reg}(\mathbf{R})$. En explicitant la *relation de Bezout* donnée par l'égalité des idéaux ci-dessus et en utilisant la proposition IV.2 on en déduit le résultat. $\triangleleft$

2.4 L'algorithme subResGcdElseSplit

Soit $v \in V$ et $p' \in \mathbf{R}_n$ réduit par rapport à T avec :

$$(\forall u \in \mathsf{var}(T))\ u < v \text{ et } (\mathsf{var}(p)) \setminus \{v\} \subseteq \mathsf{var}(T)$$

On considère l'opération normalizeElseSplit définie par :

normalize(p', v, T) ==
  $c := \mathsf{lc}(p', v)$
  $c \in \mathbf{R} \hookrightarrow$ **return** p'
  $qres :=$ quasiRecipElseSplit(c, T)
  split ?$(qres) \hookrightarrow$ **return** $qres$
  $(\widehat{c}, \| c \|) := qres$
  $\mathsf{deg}(p', v) = 0 \hookrightarrow$ **return** $\| c \|$
  return polPart(modulo($\| c \|$ $\mathsf{mvar}(p')^{\mathsf{mdeg}(p')} + \widehat{c}\,\mathsf{tail}(p'), T$))

Proposition IV.4 *On suppose que la spécification $S_{qres}(T)$ est vérifiée et que* normalize(p', v, T) *ne retourne pas un scindage. Posant $p'' =$* normalize(p', v, T)

$$\mathsf{lc}(p'', v) \in \mathbf{R} \quad \text{et} \quad \mathsf{deg}(p', v) = \mathsf{deg}(p'', v)$$

Cela résulte de la proposition IV.1 et l'algorithme s'écrit :

subResGcdElseSplit (p_1, p_2, v, T) ==
  $np2 :=$ normalize(p_2, v, T)
  split ?$(np2) \hookrightarrow$ **return** $np2$
  $p_2 := np2$
  $\delta :=$ mdeg(p_1) - mdeg(p_2)
  $\psi : \mathbf{R} := -1$; $\beta : \mathbf{R} := (-1)^{(\delta+1)}$
  $p_3 :=$ numer(modulo(prem$(p_1, p_2), T$))
  while $(p_3 \neq 0)$ **repeat**
    $\alpha : \mathbf{R} := \mathsf{lc}(p_2, v)$
    $p_3 := p_3$ exquo β

$(p_1,p_2) := (p_2,p_3)$
$np2 := \mathsf{normalize}(p_2, v, T)$
$\mathsf{split\,?}(np2) \hookrightarrow$ **return** $np2$
$p_2 := np2$
if $\mathsf{deg}(p_2, v) > 0$
  then
    if $\delta > 0$ **then**
      $newpsi := -\alpha$
      for i **in** $2..\delta$ **repeat**
        $newpsi := (-\alpha\, newpsi)$ exquo ψ
      $\psi := newpsi$
    $\delta := \mathsf{mdeg}(p_1)$ - $\mathsf{mdeg}(p_2)$
    $\beta := -\alpha\, \psi^\delta$
    $p_3 := \mathsf{numer}(\mathsf{modulo}(\mathsf{prem}(p_1, p_2), T))$
  else
    $p_3 := 0$
return p_2

2.5 L'algorithme extendedSubResGcdElseSplit

Remarque IV.5 L'algorithme extendedSubResGcdElseSplit est la *version étendue* du précédent algorithme. Nous en donnons tout d'abord le *principe*.

Notons a et b les valeurs initiales des polynômes p_1 et p_2. Dans cette version étendue, on effectue au début les initialisations suivantes : $l_1 := p_1$, $l_2 := p_2$, $u_1 := 1/1$, $u_2 := 0/1$, $w_1 := 0/1$, $w_2 := 1/1$, ces variables représentant des éléments de $\mathrm{S}_E{}^{-1}\ \mathbf{M}_T$, de sorte que :

$$\mathsf{modulo}(u_1 a + w_2 b, T) = l_1 \ \text{ et } \ \mathsf{modulo}(u_2 a + w_2 b, T) = l_2$$

Au cours de l'algorithme les variables l_1, l_2, u_1, u_2, w_1 et w_2 sont mises à jour afin de maintenir ces relations, ainsi que : $p_i = \mathsf{numer}(l_i)$. Ainsi, lorsque subResGcdElseSplit (p_1, p_2, v, T) retourne un pgcd modulo T, soit g, de p_1 et p_2, extendedSubResGcdElseSplit (p_1, p_2, v, T) doit retourner de plus f et k dans $\mathbf{R}_n$ réduits par rapport à T et tels que :

$$\mathsf{modulo}(f p_1 + k p_2, T) = g$$

Remarque IV.6 Cet algorithme utilise deux opérations auxiliaires que nous décrivons maintenant. On pose $S = \mathbf{reg}(\mathbf{R})$. On étend l'opération modulo aux éléments de $\mathrm{S}^{-1}\mathbf{R}_n$. Ces derniers peuvent être vus comme des couples constitués d'un élément de $\mathrm{S}^{-1}\mathbf{R}$ et d'un polynôme de $\mathbf{R}_n$. Pour cela, pour $m \in \mathrm{S}^{-1}\mathbf{R}_n$ on pose :

$$\mathsf{modulo}(m, T) = \mathsf{numPart}(m)\ \mathsf{modulo}(\mathsf{polPart}(m), T)\ /\ \mathsf{denom}(m)$$

Soient alors l, u et w dans $\mathrm{S}^{-1}\mathbf{R}_n$. On suppose que :

$$\mathsf{var}(\mathsf{polPart}(l)) \setminus \{v\} \ \subseteq\ \mathsf{var}(T)$$

On considère l'opération $\mathsf{normalize}(l, u, w, v, T)$ définie comme suit :

normalize(l, u, w, v, T) ==
    $c :=$ lc(polPart$(l), v)$
    $c \in \mathbf{R} \hookrightarrow$ **return** $[l, u, w]$
    $ces :=$ quasiRecipElseSplit(c, T)
    split ?$(ces) \hookrightarrow$ **return** ces
    $(\widehat{c}, \| c \|) := ces$
    $(l', u', w') :=$ (modulo$(\widehat{c}\, l, T)$, modulo$(\widehat{c}\, u, T)$, modulo$(\widehat{c}\, w, T)$)
    return $[l', u', w']$

Pour cette opération normalize, nous ajoutons une remarque à la proposition IV.4 :

Proposition IV.5 *Supposons que* normalize(l, u, w, v, T) *ne retourne pas un scindage. Posons alors* $[l', u', w'] =$ normalize(l, u, w, v, T)*. On a :*

(a) lc(polPart$(l'), v) \in \mathbf{R}$ et deg(polPart$(l), v) \neq 0 \Rightarrow$ mdeg(polPart(l')) = mdeg(polPart(l))

(b) modulo$(p_1 u + p_2 w, T) = l \;\Rightarrow\;$ modulo$(p_1 u' + p_2 w', T) = l'$

Preuve. $\triangleright$ En effet, pour (b) on a :

$$\begin{aligned}
\mathrm{modulo}(p_1 u' + p_2 w', T) &= \mathrm{modulo}(\mathrm{modulo}(p_1, T)\,\mathrm{modulo}(\overline{c}\, u, T), T) + \\
&\quad\ \mathrm{modulo}(\mathrm{modulo}(p_2, T)\,\mathrm{modulo}(\overline{c}\, w, T), T) \\
&= \mathrm{modulo}(\overline{c}\, p_1 u, T) + \mathrm{modulo}(\overline{c}\, p_2 w, T) \\
&= \mathrm{modulo}(\overline{c}\,\mathrm{modulo}(p_1 u + p_2 w), T) \ \triangleleft
\end{aligned}$$

On considère d'autre part l'opération clearDenoms (l, u, w) qui met ces 3 fractions au même dénominateur, puis retourne leurs numérateurs. L'algorithme s'écrit alors :

```
extendedSubResGcdElseSplit (p1, p2, v, T) ==
    (l1, l2) := (p1/1, p2/1)
    (u1, u2) := (1/1, 0/1)
    (w1, w2) := (0/1, 1/1)
    luw2 := normalize(l2, u2, w2, v, T)
    split ?(luw2) ↪ return luw2
    [l2, u2, w2] := luw2 ; p2 := numer(l2)
    ψ := -1 ; δ := mdeg(p1) - mdeg(p2) ; β := (-1)**(δ+1)
    [p3, q3] := [prem(p1, p2), pquo(p1, p2)]
    l3 := modulo(p3, T)
    while (l3 ≠ 0) repeat
        α := lc(p2, v) ; λ := α^(δ+1)
        l3 := (numer(l3) exquo β)/denom(l3)
        l4 := modulo(q3, T) ; p3 := numer(l3)
        (s1, s2) := (denom(l1), denom(l2))
        u3 := modulo(λ s1 u1 − l4 s2 u2, T)
        u3 := (numer(u3) exquo β)/denom(u3)
        w3 := modulo(λ s1 w1 − l4 s2 w2, T)
        w3 := (numer(w3) exquo β)/denom(w3)
        (l1, l2) := (l2, l3)
        (p1, p2) := (p2, p3)
        (u1, u2) := (u2, u3)
        (w1, w2) := (w2, w3)
        luw2 := normalize(l2, u2, w2, v, T)
        split ?(luw2) ↪ return luw2
        [l2, u2, w2] := luw2 ; p2 := numer(l2)
        if deg(p2, v) > 0
          then
                if δ > 0  then
                  newpsi := −α
                  for  i  in  2..δ  repeat
                        newpsi := (−α ∗ newpsi) exquo ψ
                  ψ := newpsi
                δ := mdeg(p1) - mdeg(p2)
                β := −α ψ^δ
                [p3, q3] := [prem(p1, p2), pquo(p1, p2)]
                l3 := modulo(p3, T)
          else
                l3 := 0
    return clearDenoms (u2, w2, l2)
```

Proposition IV.6 *Soient g, f, k dans $\mathbf{R}_n$ et réduits par rapport à T. Alors Les propositions suivantes sont equivalentes :*

(*i*) $g =$ subResGcdElseSplit(p_1, p_2, v, T) *et* modulo$(fp_1 + kp_2, T) = g$

(*ii*) $(g, f, k) = \mathsf{extendedSubResGcdElseSplit}(p_1, p_2, v, T)$

Preuve. ▷ Tout d'abord, remarquons que la suite de polynômes $p_1, p_2, p_3, \ldots$ calculée par chacun des algorithmes est la même. Par consèquent, si le résultat de l'un est un scindage, il en est de même de l'autre, et il s'agit du même scindage. Supposons alors qu'aucun ne retourne un scindage. On peut donc poser :

$$g = \mathsf{subResGcdElseSplit}(p_1, p_2, v, T)$$

et :

$$(g, f, k) = \mathsf{extendedSubResGcdElseSplit}(p_1, p_2, v, T)$$

Apres avoir initialisées les variables l_1, l_2, u_1, u_2, w_1, w_2 on a les relations :

$$\mathsf{modulo}(u_1 p_1 + w_2 p_2, T) = l_1 \text{ et } \mathsf{modulo}(u_2 p_1 + w_2 p_2, T) = l_2$$

Il nous suffit d'établir que ces relations sont conservées tout au long du déroulement de l'algorithme. De la proposition IV.5, on déduit que cette conservation a bien lieu après l'instruction $luw_2 := \mathsf{normalize}(l_2, u_2, w_2, v, T)$. Plaçons-nous maintenant au tout début de la boucle **while**. Nous reprenons les notations de l'algorithme. A chaque itération, nous avons :

$$\lambda p_1 = q_3 p_2 + p_3$$

Soit E l'ensemble des initiaux des polynômes de T. La multiplication étant celle de $\mathrm{S}_E{}^{-1}\ \mathbf{M}_T$, après la cinquième instruction de la boucle, nous avons :

$$\lambda p_1 = l_4 p_2 + \beta l_3$$

En effet, p_1 et p_2 sont toujours réduits par rapport à T. Comme $l_1 = p_1/s_1$ et $l_2 = p_2/s_2$, on a aussi :

$$l_3 = (\lambda s_1 l_1 - q_3 s_2 l_2) \text{ exquo } \beta$$

D'autre part, étant donnés μ, ν dans $\mathrm{S}^{-1}\mathbf{R}_n$ et β dans $\mathbf{R}$, on considère l'opération makeLinComMod (μ, ν, β, T) qui retourne l'application ϕ de $\mathrm{S}^{-1}\mathbf{R}_n \times \mathrm{S}^{-1}\mathbf{R}_n$ dans $\mathrm{S}^{-1}\mathbf{R}_n$ définie par

$$\phi(x, y) = (\mathsf{numer}(z) \text{ exquo } \beta)/\mathsf{denom}(z) \text{ avec } z = \mathsf{modulo}(\mu\, x + \nu\, y, T)$$

Posons $\phi = \mathsf{makeLinComMod}(\lambda\, s_1, -l_4\, s_2, \beta, T)$. Par suite, si l'on pose $u_3 = \phi(u_1, u_2)$ et $w_3 = \phi(w_1, w_2)$, il vient :

$$\mathsf{modulo}(u_3 p_1 + w_3 p_2, T) = l_3$$

On en déduit le résultat. ◁

Remarque IV.7 Précisons la manière dont nous gérons nos fractions de $\mathrm{S}_E{}^{-1}\ \mathbf{M}_T$ dans l'implantation de nos algorithmes. Supposons que $\mathbf{R}$ soit un anneau factoriel avec une opération *effective* de pgcd. Alors, en prenant $\mathbf{reg}(\mathbf{R})$ pour S_E, chaque élément l de $\mathrm{S}_E{}^{-1}\ \mathbf{M}_T$ a une forme canonique où $\mathsf{numPart}(l)$ est le contenu de $\mathsf{numer}(l)$ et où $\mathsf{numPart}(l)$ and $\mathsf{denom}(l)$ sont premiers entre eux. Dans l'algorithme extendedSubResGcdElseSplit on gère alors nos fractions de façon paresseuse (voir la section 1.3 du chapitre 6). Ainsi, nos fractions ne sont pas sous forme canonique. En effet, les divisions par β perdraient leur intérêt et pourraient même ne pas être exactes. Nous simplifions uniquement les fractions l_2, u_2, w_2 avant de les mettre au même dénominateur (pour obtenir le résultat de notre algorithme si celui-ci n'est pas un scindage).

2.6 Les algorithmes subResGcd et quasiRecip

Ces algorithmes se déduisent aisément des précédents. Donnons par exemple celui de newSubResGcd.

newSubResGcd (p_1, p_2, v, T) ==
  $toSee := [\,[\,[p_1, p_2], T]\,]$
  $toSave := [\,]$
  while not empty ? $toSee$ **repeat**
    $[[a_j, b_j], T_j]$:= first $toSee$
    $toSee$:= rest $toSee$
    res := subResGcdElseSplit (a_j, b_j, v, T_j)
    if not split ?(res)
      then
        $toSave$:= cons$([res, T_j], toSave)$
      else
        for T **in** res **repeat**
          a_j := polPart(modulo(a_j, T))
          b_j := polPart(modulo(b_j, T))
          if (deg$(a_j, v) = 0$) or (deg$(b_j, v) = 0$)
            then
              $toSave$:= cons$([1, T_j], toSave)$
            else
              if mdeg(a_j) < mdeg(b_j)
                then $(a_j, b_j) := (b_j, a_j)$
              $toSee$:= cons$([\,[a_j, b_j], T], toSee)$
  return $toSave$

2.7 Correction des algorithmes

Nous allons maintenant vérifier que les algorithmes proposés satisfont bien aux spécifications annoncées au début de cette section.

Proposition IV.7 *Les algorithmes* quasiRecipElseSplit, subResGcdElseSplit *et* extendedSubResGcdElseSplit *satisfont respectivement les spécifications* $S_{qres}(T)$*,* $S_{srges}(T)$*, et* $S_{esrges}(T)$ *pour tout* T*.*

Preuve. ▷ Nous avons vu que, lorsque $S_{srges}(T)$ est prouvée, il en va de même de $S_{esrges}(T)$. Il suffit donc de montrer par récurrence sur $|\operatorname{deg}(T)|$ que $S_{qres}(T)$ et $S_{srges}(T)$ sont satisfaites. Soient p, p_1, p_2 dans $\mathbf{R}_n$ et v dans V tel que les hypothèses $H_{srges}(p_1, p_2, v, T)$ et $H_{qres}(p, T)$ soient vérifiées.

Supposons que $|\operatorname{deg}(T)| = 0$, (i.e. T vide). Alors, d'après $H_{qres}(p, T)$, $p \in \mathbf{R}$ et modulo$(p, T) = p/1$. Donc quasiRecipElseSplit(p, T) retourne $(1, p)$, ce qui satisfait $S_{qres}(T)$. D'après $H_{srges}(p_1, p_2, v, T)$, on a :

$$\operatorname{var}(p_1) = \operatorname{var}(p_2) = \{v\}$$

Dans ce cas, subResGcdElseSplit est l'algorithme des sous-résultants habituel, ce qui satisfait $S_{srges}(T)$.

Supposons maintenant $|\,\mathsf{deg}(T)\,| > 0$. Regardons $\mathsf{quasiRecipElseSplit}(p, T)$. On reprend les notations de l'algorithme. On pose donc $p/s = \mathsf{modulo}(p, T)$. Supposons que : $p \notin \mathbf{R}$. On a $|\,\mathsf{deg}(\ T_u^-)\,| < |\,\mathsf{deg}(T)\,|$. On vérifie que les conditions $H_{esrges}(\ T_u, p, u,\ T_u^-)$ sont remplies. Par conséquent, $S_{esrges}(\ T_u^-)$ s'applique. Si $esrges$ est un s.f.e.t.a.r. (scindage fondamental en e.t.a.r.) de T_u, alors $\mathsf{quasiRecipElseSplit}(p, T)$ retourne un s.f.e.t.a.r. de T. Sinon, et si $\mathsf{deg}(g, u) > 0$, on vérifie que l'on obtient aussi un s.f.e.t.a.r. de T. En effet, le polynôme g pseudo-divise T_u modulo T_u^-. Comme $\mathsf{init}(g) \in \mathbf{reg}(\mathbf{R})$, il existe $t \in \mathbf{reg}(\mathbf{R})$ tel que $\mathsf{modulo}(gq,\ T_u^-) = t\ T_u$. De plus on a aussi : $\mathsf{init}(q) \in \mathbf{reg}(\mathbf{R})$. Par conséquent gq appartient à l'idéal de $\mathbf{P}_n$ engendré par T, tandis qu'aucun de ces deux polynôme n'y appartient. Enfin, si $\mathsf{deg}(g, u) = 0$ et si a désigne le polynôme p d'entrée, on a :

$$\mathsf{modulo}(ask, T) = \mathsf{modulo}(pk, T) = g$$

Comme $g \in \mathbf{reg}(\mathbf{R})$, la spécification $S_{qres}(T)$ est finalement bien satisfaite. Déduisons-en que $S_{srges}(T)$ l'est aussi. Regardons $\mathsf{subResGcdElseSplit}\ (p_1, p_2, v, T)$. et reprenons les notations de l'algorithme. Notons que si $\mathsf{normalize}(p_2, v, T)$ retourne un s.f.e.t.a.r. de T (à une étape quelconque de l'algorithme) il en va de même de $\mathsf{subResGcdElseSplit}\ (p_1, p_2, v, T)$. Supposons que cela ne se produise pas. Par hypothèse de récurrence, et, puisque $S_{qres}(T)$ est vérifiée, la proposition IV.1 est vraie et l'on peut alors appliquer la proposition IV.4. Par conséquent, l'instruction $p_2 := np2$ remplace $p2$ par un polynôme ${p'}_2$ de même degré et à initial dans $\mathbf{R}$. De plus, ces polynômes sont associés en tant que polynômes de $\mathbf{fr}(\mathbf{A})[v]$. D'autre part, les initiaux des polynômes de T étant dans $\mathbf{R}$, après chaque instruction initialisant p_3, on a :

$$\langle p_1, p_2 \rangle_{\mathbf{fr}(\mathbf{A})[v]} \ = \ \langle p_2, p_3 \rangle_{\mathbf{fr}(\mathbf{A})[X]}$$

Par suite, le polynôme p_2 retourné est bien un pgcd des polynômes p_1 et p_2 d'entrée, au sens de la définition II.9. Enfin, le fait qu'il soit résultat d'un appel $\mathsf{normalize}(p_2, v, T)$ nous assure qu'il s'agit d'un pgcd modulo T de ces polynômes p_1 et p_2. Ainsi, $S_{srges}(T)$ est satisfaite. Remarquons que si T engendre un idéal radical dans $\mathbf{P}_n$, les scindages éventuels sont biunivoques : ceci résulte de la proposition IV.3. $\triangleleft$

3 Application à la résolution de systèmes de dimension zéro

Soit I un idéal de dimension zéro de $\mathbf{P}_n$ et G une base de Gröbner de I pour l'ordre lexicographique, donnée sous la forme d'une liste triée par monômes de tête croissants. L'algorithme $\mathsf{lextriangular}$ (voir [Laz92a]) retourne un scindage de I dont chaque composante est un ensemble triangulaire. Le principe est le suivant. A une étape quelconque de l'algorithme on a deux listes. La première, *toSave*, est la liste des composantes déjà construites. La seconde, *toSee*, est une liste de triplets. Un triplet est constitué par un ensemble triangulaire T (une composante en construction), la plus grande variable parmi celles de T, soit v, et H la *sous-liste* de G triée comme G et constituée par les polynômes $g \in G$ avec $\mathsf{mvar}(g) > v$.

Soit $t = [T, v, H]$ un tel triplet. Pour ajouter à T un polynôme, on considère le premier polynôme de H, soit p. Soit ip son initial et w sa variable principale. Puisque I est de dimension zéro, ses autres variables sont algébriques par rapport à T. Pour ajouter p à T, on regarde alors si p est quasi-unitaire vu comme polynôme de $(\mathbf{K}[\mathsf{var}(T)]/T)\ [w]$, ce qui revient à déterminer un quasi-inverse de ip modulo T. On obtient donc une *discussion* par l'appel à $\mathsf{quasiRecip}(ip, T)$. Dans les cas où ip est un diviseur de zéro modulo la composante correspondante, on reprend les calculs avec le polynôme de H successeur de p. Dans les autres cas, on multiplie p par l'inverse de ip dans $\mathbf{fr}(\mathbf{K}[\mathsf{var}(T)]/T)$, et on remplace H par les polynômes de H de variable principale supérieure à w. La correction de l'algorithme repose sur le théorème de Gianni et Kalkbrener (voir [Laz92a]) et l'algorithme s'écrit :

```
lextriangular G ==
    p := first G ;  T := {p} ;  H := G \ {p}
    H = ∅  =>  return [T]
    toSee := [[T, X_1, H]] ;  toSave := [ ]
    while not empty ? toSee  repeat
          [T, X_i, H] := first toSee
          toSee := rest toSee
          p := first H
          for [q, r, U] in quasiRecip(init(p), T) repeat
              if r = 0
                then
                      toSee := cons([U, X_i, H \ {p}], toSee)
                else
                      t := r mvar(p)^mdeg(p) + q tail(p)
                      U := cons(t̄^U, U)
                      H' := [g ∈ H | mvar(g) > X_{i+1}]
                      if empty ? H'
                        then
                              toSave := cons(U, toSave)
                        else
                              toSee := cons([U, X_{i+1}, H'], toSee)
    return toSave
```

Dans le sixième chapitre, on donne une dizaine d'exemples d'utilisation de cette algorithme (p. 235, p. 238, p. 253, p. 248, p. 267, p. 282, p. 276, p. 289, p. 293, p. 301, et p 297).

Chapitre V

Pgcd au-dessus d'une tour d'extensions simples séparables

Introduction

Dans ce chapitre nous exposons un algorithme donnant une décomposition d'une variété affine au sens du théorème III.11 p. 111. tant donné un ensemble de polynômes F cette méthode produit donc des ensembles de Lazard $S_1, \ldots, S_l$ tel que :

$$\mathbf{V}(F) \;=\; \cup_1^l \mathbf{W}(S_j)$$

Un procédé non prouvé et relativement compliqué pour obtenir une telle décomposition apparaît dans l'article ([Laz91a]) de Daniel Lazard. Notre construction est cependant basée sur celle de Lazard. Les outils que nous avons développé dans les chapitres chapitres, comme la notion de pgcd (définition II.9 p. 65) et le théorème II.26 p. 68, ainsi que notion de quasi-inverse (définition IV.1 p. 115), donnent, nous semble-t-il une présentation plus claire de la méthode. En particulier, les calculs s'effectuent dans la *bonne structure algébrique* à savoir dans le produit de corps défini par l'ensemble triangulaire courant et non pas avec des polynômes multivariés que l'on spécialise ensuite (voir l'introduction du quatrième chapitre et [Laz91a]).

Deux aspects au moins de cet algorithme rendent son exposé relativement difficile. D'une part, les indéterminées des polynômes sont tantôt vues comme des quantités algébriques ou transcendantes (vivant dans une certaine tour d'extensions simples séparables) tantôt comme des variables. D'autre part, cette construction repose sur une quinzaine d'algorithmes qui s'appellent mutuellement.

Cependant, cet algorithme possède des propriétés intéressantes et originales (voir le chapitre suivant pour une comparaison avec des algorithmes *similaires*). Tout d'abord, chacune des procédures qui le compose résout un problème intéressant pour lui-même. Donnons un exemple. Rappelons (notation I.1 p. 19) que dans toute cette thèse $\mathbf{R}$ est un anneau intègre et $\mathbf{k}$ est un corps des fractions. Dans ce chapitre ce corps sera supposé de caractéristique nulle. De plus, *comme d'habitude*, on désigne par $\mathbf{R}_n$ l'anneau des polynômes à coefficients dans $\mathbf{R}$ et en les variables ordonnées : $X_1 < X_2 < \cdots < X_n$. Soit $T \subseteq \mathbf{R}_n$ un ensemble de Lazard et $p \in \mathbf{R}_n$ un polynôme. Soit $\mathbf{A}_T$ la tour d'extensions simples séparables associée à T (voir la notation V.1 au début de la section suivante). Une question *naturelle* est de savoir si p est inversible ou pas dans $\mathbf{A}_T$. La proposition V.6 de la section suivante fournit un algorithme pour résoudre ce problème.

Une autre propriété importante de cet algorithme est la suivante. Les décompositions produites (i.e. scindages de Lazard d'une variété affine, définition III.24 p. 110) ne comportent pas de quasi-composantes inconsistantes ou redondantes. De plus, la *lecture* d'une décomposition calculée par cet algorithme est plus aisée que celle des décompositions produites par les autres méthodes présentées dans le chapitre suivant. En effet, un point de l'espace où l'on résout est solution du système de départ si et seulement s'il est zéro régulier (définition III.19 p. 101) de l'un des ensembles triangulaires de la décomposition calculée. Donnons un exemple pour illustrer cette propriété. On considère le système suivant, tiré d'un problème de bras manipulateur en robotique. Les variables des polynômes sont ordonnées par : $c_2 > s_2 > c_1 > s_1 > b > a$ et leurs coefficients sont dans $\mathbb{Q}$le corps des nombres rationnels. Ces polynômes sont :

$$\left\{c1\ c2 - s1\ s2 + c1 - a,\ s1\ c2 + c1\ s2 + s1 - b,\ c1^2 + s1^2 - 1,\ c2^2 + s2^2 - 1\right\}$$

Chaque couple (c_i, s_i) est un couple (*cosinus, sinus*) qu'il s'agit d'exprimer en fonction de a et b à l'aide des relations ci-dessus. La solution calculée par notre implantation de cet algorithme est :

```
      2     2   2        3     2        4     2 2    4     2
[ {(4b  + 4a )s1  + (- 4b  - 4a b)s1 + b  + 2a b  + a  - 4a ,
                    2    2             2     2       3    2
   2a c1 + 2b s1 - b  - a , 2a s2 + (2b  + 2a )s1 - b  - a b,
          2    2
   2c2 - b  - a  + 2}
  ,
               2    2                      2
 {a,2s1 - b,4c1  + b  - 4,s2 - b c1,2c2 - b  + 2},
        2     2
 {a,b,c1  + s1  - 1,s2,c2 + 1}]
```

Voici comment il faut la *lire*. Regardons le premier ensemble triangulaire ci-dessus. Sous réserve que $b^2 + a^2$ soit non nul, l'indéterminée s_1 s'obtient en résolvant l'équation du second degré :

$$\left(4\ b^2 + 4\ a^2\right)\ s1^2 + \left(-4\ b^3 - 4\ a^2\ b\right)\ s1 + b^4 + 2\ a^2\ b^2 + a^4 - 4\ a^2 \ = \ 0$$

Sous réserve que a soit non nul, on en déduit c_1 et s_2 à l'aide de :

$$2\ a\ c1 + 2\ b\ s1 - b^2 - a^2 \ = \ 0$$

et :

$$2\ a\ s2 + \left(2\ b^2 + 2\ a^2\right)\ s1 - b^3 - a^2\ b \ = \ 0$$

respectivement, c_2 étant égal à $\frac{b^2+a^2-2}{2}$. Les deux autres ensembles triangulaires fournissent le cas $a = 0$. Du reste, si l'on rajoute cette équation au système de départ, la décomposition obtenue est exactement :

```
                 2    2                        2
 [ {a,2s1 - b,4c1  + b  - 4,s2 - b c1,2c2 - b  + 2},
           2     2
   {a,b,c1  + s1  - 1,s2,c2 + 1}]
```

Qu'en est-il alors du cas $b^2 + a^2 = 0$? Ce cas a en fait été examiné pendant les calculs intermédiaires (voir le théorème V.2 dans la troisième section de ce chapitre) et si l'on rajoute cette équation au système de départ, la décomposition produite est :

```
           2     2
 [ {a,b,c1  + s1  - 1,s2,c2 + 1}]
```

À titre de comparaison donnons le résultat produit par l'algorithme de Michael Kalkbrener (implanté par Philippe Aubry, et décrit dans le sixième chapitre) :

```
         2      2  2         3      2        4     2 2    4     2
  [ {(4b  + 4a )s1  + (- 4b  - 4a b)s1 + b  + 2a b  + a  - 4a ,
                                2    2
     2a c1 + 2b s1 - b  - a , s2 - b c1 + a s1,
                 2
     s1 c2 + b c1  - a s1 c1 + s1 - b}]
```

Il s'agit d'une chaîne régulière C (définition III.11 p. 88). Dans ce cas, un point est solution du système initial s'il appartient à la variété affine associée à l'idéal saturé de C. Par conséquent, les cas $b^2+a^2 = 0$ et $a = 0$ ne sont pas ici *examinés explicitement*. De plus, la relation donnant c_2 est plus compliquée que dans les ensembles de Lazard, les chaînes régulières n'étant en général pas des ensembles triangulaires réduits et normalisés.

Notons, d'autre part, que le procédé de résolution des systèmes d'équations algébriques que fournit l'algorithme de Lazard est incrémental, Ceci, de notre point de vue, lui confère une certaine élégance. Avant de donner le plan de ce chapitre, nous allons détailler la résolution d'un système par cet algorithme, illustrant ainsi cette dernière remarque. On considère maintenant le système suivant (Donati et Traverso) où les variables sont ordonnées par : $x > y > z > t$ et où les coefficients sont dans $\mathbb{Q}$:

$$\left\{x^8 - z,\ x^{10} - t,\ x^{31} - x^6 - x - y\right\}$$

La méthode étant incrémentale, on peut commencer par résoudre le système réduit au polynôme : $x^8 - z$. S'agissant d'un polynôme dont l'initial est constant, on a bien sr :

$$\mathbf{V}(\{x^8 - z\}) \ = \ \mathbf{W}(\{x^8 - z\})$$

Puis, il faut décrire les zéros du second polynôme, à savoir $x^{10} - t$, se trouvant dans la quasi-composante précédente. On trouve le scindage de Lazard suivant :

$$\mathbf{V}(\{x^{10} - t\}) \ \cap \ \mathbf{W}(\{x^8 - z\}) \quad \longmapsto \quad \left(\left\{z^5 - t^4,\ t^3\ x^2 - z^4\right\},\ \{t,\ z,\ x\}\right)$$

Pour cela, on calcule un *pgcd normalisé* (voir la seconde section de ce chapitre) des polynômes $x^8 - z$ et $x^{10} - t$, vus comme polynômes en x et à coefficients dans la tour d'extensions simples $\mathbb{Q}(t, z)$. Ce calcul donne le résultant de ces polynômes à savoir : $(z^5 - t^4)^2$. Cette quantité devant donc être nulle, elle définit une extension algébrique simple séparable (en prenant la partie sans carrés du polynôme précédent) de $\mathbb{Q}(t)$. Notons $\mathbf{A}_{t,z}$ la tour ainsi obtenue. On reprend alors le calcul d'un pgcd des polynômes $x^8 - z$ et $x^{10} - t$ en x et désormais à coefficients dans $\mathbf{A}_{t,z}$. Le nouveau pgcd est : $t^4\ x^2 - t\ z^4$. L'indéterminée t pouvant a priori être nulle, et s'agissant de produire une décomposition en quasi-composantes (où les initiaux des polynômes des ensembles triangulaires correspondant ne peuvent s'annuler) on doit distinguer les cas $t = 0$ et $t \neq 0$. Bref, on obtient le scindage ci-dessus.

On doit maintenant décrire les zéros du troisième polynôme, à savoir $x^{31} - x^6 - x - y$, se trouvant dans l'une ou l'autre des deux quasi-composantes :

$$W_1 = \mathbf{W}(\{t,\ z,\ x\}) \quad \text{et} \quad W_2 = \mathbf{W}(\{z^5 - t^4,\ t^3\ x^2 - z^4\})$$

Le cas de W_1 se résout sans difficulté. On calcule en effet un pgcd des polynômes $x^{31} - x^6 - x - y$ et x en la variable x et à coefficients dans la tour d'extensions algébriques simples associée à $\{t,\ z\}$ ce qui donne y. On *étend* cette tour par $y = 0$. Nos deux polynômes en x sont alors identiques et l'on obtient l'ensemble de Lazard :

$$S_1 = \{t,\ z,\ y,\ x\}$$

Passons à la quasi-composante W_2. On calcule alors un pgcd normalisé des polynômes $x^{31} - x^6 - x - y$ et $t^3\ x^2 - z^4$ vus comme polynômes en la variable x et à coefficients dans la tour $\mathbf{A}_{t,z}$. On obtient le résultant suivant :

$$t^{93}\ y^2 + 2\ t^{92}\ z^2\ y + \left(-t^{96} + 2\ t^{93} + t^{91} - t^{90}\right)\ z^4$$

On souhaite alors construire l'extension algébrique simple de $\mathbf{A}_{t,z}$ par ce dernier polynôme. Ceci conduit encore une fois à distinguer les cas $t = 0$ et $t \neq 0$. Le cas $t = 0$ produit la quasi-composante $\mathbf{W}(\{x, t,\ z,\ x\})$. Le cas $t \neq 0$ produit l'ensemble de Lazard :

$$\left\{z^5 - t^4,\ t^3\ y^2 + 2\ t^2\ z^2\ y + \left(-t^6 + 2\ t^3 + t - 1\right)\ z^4\right\}$$

Notons $\mathbf{A}_{y,t,z}$ la tour associée. On calcule maintenant un pgcd normalisé des polynômes $x^{31} - x^6 - x - y$ et $t^3\ x^2 - z^4$ vus comme polynômes en la variable x et à coefficients dans la tour $\mathbf{A}_{y,t,z}$. On obtient le pgcd suivant :

$$\left(t^{93} - t^{90}\right)\ x - t^{90}\ y - t^{89}\ z^2$$

On souhaite construire l'extension algébrique simple de $\mathbf{A}_{y,t,z}$ par ce dernier polynôme. On distingue les cas $t^4 - t = 0$ et $t^4 - t \neq 0$. Le cas $t^4 - t = 0$ produit l'ensemble de Lazard :

$$S_2 \ = \ \left\{t^3 - 1,\ z^5 - t,\ y + t^2\ z^2,\ x^2 - z^4\right\}$$

Notons que l'on ne retombe pas ici sur le cas $t = 0$ puisque t divise l'initial d'un polynôme définissant $\mathbf{A}_{y,t,z}$. Quant au cas $t^4 - t \neq 0$, il produit l'ensemble de Lazard :

$$S_3 \ = \ \left\{z^5 - t^4,\ t^3\ y^2 + 2\ t^2\ z^2\ y + \left(-t^6 + 2\ t^3 + t - 1\right)\ z^4,\ \left(t^4 - t\right)\ x - t\ y - z^2\right\}$$

Finalement, le résultat calculé par notre procédé est le scindage de Lazard (S_1, S_2, S_3).

Nous donnons maintenant le plan de ce chapitre. Dans la première section, nous étudions la *normalisation d'un polynôme par un ensemble de Lazard*. Plus précisément, étant donné un polynôme p et un ensemble triangulaire T satisfaisant les axiomes de Lazard, on donne une opération $\mathsf{normalize}(p, T)$ (proposition V.6) produisant un scindage de Lazard $(S_1, \ldots, S_l)$ de T de sorte que, si $\mathbf{A}_{S_i}$ désigne la tour d'extensions simples séparables associée à S_i, l'image de p par l'homomorphisme associé à S_i dans $\mathbf{A}_{S_i}$ (définition III.12 p. 90) est soit nulle, soit inversible dans $\mathbf{A}_{S_i}$. Pour décrire l'opération $\mathsf{normalize}(p, T)$ on supposera que l'on dispose d'une opération $\mathsf{quasiRecipElseSplit}(p, T)$ qui, si p est inversible dans la tour d'extensions simples séparables associée à T, calcule alors son inverse, si p est nul dans cette tour retourne 0 et produit un scindage de Lazard de T dans les autres cas. Un algorithme pour cette dernière opération sera donné dans la troisième section (théorème V.2).

Les scindages de Lazard produits par les différents algorithmes présentés dans ce chapitre peuvent générer des décompositions avec redondances (voir définition III.24 p. 110). La résolution du système de Donati et Traverso ci-dessus a mis en évidence ce phénomène avec l'ensemble $\{t,\ z,\ y,\ x\}$ qui est *apparu* plusieurs fois au cours des calculs. Nous traiterons ce problème en deux temps. Tout d'abord, à partir de l'opération $\mathsf{normalize}(p, T)$, pour un couple (T, U) d'ensembles de Lazard, on déduira (proposition V.7) une opération booléenne $\mathsf{containedInComponent?}(T, U)$ qui retourne vrai si et seulement si :

$$\mathbf{W}(T) \subseteq \mathbf{V}(U)$$

Puis, dans la troisième section, en modifiant cette dernière opération, on donnera une stratégie pour produire des scindages de Lazard sans redondances d'une variété affine.

Dans la seconde section, on décrit un algorithme calculant un pgcd de deux polynômes univariés à coefficients dans une tour d'extensions simples séparables (théorème V.1). Le principe est analogue à celui des tours algébriques d'extensions simples : grâce à l'opération de normalisation vue ci-dessus, on utilise le théorème II.26 p. 68 *comme si l'anneau des coefficients de nos deux polynômes était un corps*. Cependant, *l'opération de quasi-inversion* $\mathsf{quasiRecipElseSplit}$ à laquelle fait appel *l'opération de normalisation* $\mathsf{normalizeElseSplit}$ est ici plus compliquée. Détaillons la raison essentielle de cette différence.

Dans le cas *purement algébrique*, c'est-à-dire celui du chapitre précédent, on scinde des idéaux de dimension zéro à l'aide de l'unique règle (voir proposition II.20 p. 62) :

$$ab \in I \implies I \longmapsto (I + \langle a \rangle, I + \langle b \rangle)$$

Dans le cas *algébrico-transcendant*, c'est-à-dire celui de ce chapitre, on scinde des quasi-composantes d'ensembles de Lazard, ce qui revient à scinder des idéaux quasi-premiers de dimension quelconque (i.e. les idéaux saturés de nos ensembles de Lazard) à l'aide de la règle précédente et de la règle (voir proposition II.15 p. 59) :

$$p \notin I \implies I \longmapsto (I + p, I : p)$$

Les scindages du type $I \longmapsto (I + \langle a \rangle, I + \langle b \rangle)$ sont produits par l'algorithme $\mathsf{quasiRecipElseSplit}$ (voir théorème V.2) comme dans le second chapitre. Quant aux scindages du type $I \longmapsto (I + p, I : p)$, ils sont produits par l'algorithme $\mathsf{augment}$ (théorème V.2) qui n'a pas d'équivalent dans le cas purement algébrique. La raison en est la suivante. Supposons que l'on souhaite *étendre* un ensemble de Lazard T avec un polynôme p tel que : $\mathsf{mvar}(p) > \mathsf{mvar}(T)$. Dans le cas purement algébrique, il suffit de réduire p par rapport à T. Dans le cas algébrico-transcendant, la conservation de la propriété *être un ensemble de Lazard* entraine des calculs supplémentaires pouvant engendrer des scindages. En particulier, et même si p est réduit et normalisé par rapport à T, on doit examiner le cas où son initial s'annule sur $\mathbf{W}(T)$. Rappelons que dans le cas purement algébrique, les polynômes normalisés par rapport à un ensemble triangulaire sont à initiaux dans $\mathbf{R}$.

Revenons maintenant à l'algorithme de pgcd évoqué plus haut (théorème V.1) Une autre différence importante avec le cas purement algébrique est la suivante. Donnons quelques notations. Soient p_1, p_2 deux polynômes de $\mathbf{R}_n$, non-constants, et de même variable principale $v = X_i$ pour $1 \leq i \leq n$. Soit $T \subseteq \mathbf{R}_{i-1}$ un ensemble de Lazard tel que p_1 et p_2 soient initialement réduits par rapport à T. Notons $\mathbf{A}_T$ la tour d'extensions simples séparables associée à T (notation V.1). Supposons que l'on souhaite décrire les zéros communs de p_1 et p_2 se trouvant dans $\mathbf{W}(T)$. Supposons de plus que l'opération de pgcd $\mathsf{subResNormalizedGcdElseSplit}$ (p_1, p_2, v, T) (proposition V.8) ne retourne pas un scindage de Lazard de T mais un polynôme r non constant et de degré nul en v. Notons que r est alors normalisé par rapport à T. Cela ne signifie pas que $\mathbf{V}(\langle p_1, p_2 \rangle) \cap \mathbf{W}(T)$ soit vide mais qu'une relation sur une indéterminée (la variable principale de r) non algébrique dans T a été découverte. Le polynôme r doit donc être le résultant des polynômes p_1 et p_2, ces trois polynômes étant vus dans $\mathbf{A}_T[v]$. Pour illustrer cette situation, on peut reprendre le début de la résolution du système de Donati et Traverso ci-dessus. Notons que dans le cas purement algébrique, cette situation ne se présente pas. En effet, dans ce cas, si r est de degré nul en v, alors r est constant. Finalement, dans le cas algébrico-transcendant l'opération $\mathsf{subResNormalizedGcdElseSplit}$ doit être une opération de *pgcd-non-trivial-sinon-résultant-sinon-scindage* !

A partir de cet algorithme de pgcd, nous donnerons un algorithme de *partie primitive* (proposition V.10) et un algorithme de *partie sans carrés* (proposition V.9) (au sens de la définition III.1 p. 79) d'un polynôme univarié à coefficients dans une tour d'extensions simples séparables.

Dans la troisième section, un algorithme appelé $\mathsf{intersect}$ (théorème V.2) donnera une décomposition (au sens de la définition III.24) des zéros d'un polynôme dans une quasi-composante. Nous en déduirons d'une part un algorithme appelé solve produisant une décomposition d'une variété affine au sens du théorème III.11 p. 111. Nous en déduirons d'autre part l'algorithme $\mathsf{quasiRecipElseSplit}$ calculant soit un quasi-inverse d'un polynôme modulo une tour d'extensions simples séparables, soit un scindage de l'ensemble triangulaire définissant cette tour.

Nous n'avons pas encore rédigé la totalité des preuves de ces algorithmes. Cependant, ces algorithmes ont été intensivement testés (voir le sixième chapitre) et leurs résultats vérifiés. Nous illustrerons et commenterons les algorithmes non prouvés. D'autre part, si les remarques V.24, V.22, V.23, V.26, et la proposition V.12, ne constituent pas une preuve des algorithmes proposés ici, elles établissent cependant la preuve de la terminaison et de la correction de l'algorithme original de Daniel Lazard ([Laz91a]).

1 Normalisation d'un polynôme par un ensemble de Lazard

Notations V.1 Pour un polynôme p de $\mathbf{R}_n$ et un ensemble triangulaire T de $\mathbf{R}_n$ on rappelle que $\mathsf{normalized?}(p, T)$ désigne la proposition : *p est normalisé par rapport à T.*

Supposons maintenant que T soit un ensemble régulier (définition III.12 p. 90). Soit $(\mathbf{A}_1, \ldots, \mathbf{A}_n)$ sa tour d'extensions simples associée. On désignera par $\mathbf{A}_T$ l'extension simple $\mathbf{A}_n$ et, par abus de langage, cet anneau sera souvent appelé *la tour d'extensions simples associée à T*. Soit de plus F_n l'homomorphisme associé à T. Rappelons (proposition III.18 p. 103) que F_n est un homomorphisme d'algèbres de $\mathbf{R}_n[X]$ dans $\mathbf{A}_T[X]$ tel que $F_n(X) = X$ et dont le noyau est $\mathsf{sat}(T)[X]$. Dans la suite, on désignera par $\overline{p}^{\widehat{T}}$ vl'image de p par F_n. On pose d'autre part :

$$\mathsf{reduce}(p, T) \ = \mathsf{numer}(\mathsf{modulo}(p, T))$$

où l'opération modulo est celle définie dans la remarque III.10 p. 92. Rappelons que le dénominateur de $\mathsf{modulo}(p, T)$ est un produit d'initiaux de polynômes de T. Le polynôme $\mathsf{reduce}(p, T)$ de $\mathbf{R}_n$ est, à une unité près (en fait un produit d'initiaux de T) dans $\mathbf{A}_T$ un représentant de $\overline{p}^{\widehat{T}}$ réduit par rapport à T.

Dans ce chapitre, chaque opération, dont le nom sera suffixé par $\mathsf{ElseSplit}$, retournera un objet dont le type (au sens informatique du terme) sera soit un certain type dépendant de l'opération, soit le type *scindage de Lazard*. Des minuscules grecques dénoteront les variables recevant les résultats de ces opérations. On se donne une opération booléenne $\mathsf{split?}$ qui, appliquée à une telle variable, retourne vrai si et seulement s'il s'agit d'un scindage de Lazard.

Remarque V.1 Soient $T, S_1, \ldots, S_l \subseteq \mathbf{R}_n$ des ensembles de Lazard. Rappelons la définition III.24. L'ensemble $\{S_1, \ldots, S_l\}$ est un scindage de Lazard de T si l'on a :

$$\mathbf{W}(T) \ \subseteq \ \cup_1^l \mathbf{W}(S_i) \ \subseteq \ \overline{\mathbf{W}(T)}$$

Par conséquent, d'après la proposition III.22 p. 111 on a :

$$\mathsf{sat}(T) \ = \ \cap_1^l \mathsf{sat}(S_i)$$

Définition V.1 *Soient $T, S_1, \ldots, S_l \subseteq \mathbf{R}_n$ des ensembles de Lazard. On dira que $\{S_1, \ldots, S_l\}$ est un* scindage propre de Lazard *de T si $\{S_1, \ldots, S_l\}$ un scindage de Lazard de T et si aucun des idéaux $\mathsf{sat}(S_i)$ n'est égal à $\mathsf{sat}(T)$.*

Remarque V.2 Soit $T \subseteq \mathbf{R}_n$ un ensemble de Lazard. D'après la définition précédente, T n'admet pas de scindage propre de Lazard si et seulement si $\mathsf{sat}(T)$ est un idéal irréductible. Reprenons l'exemple de Donati et Traverso que nous avons traité dans l'introduction de ce chapitre. Avec les notations de cet exemple $\{S_1, S_2, S_3\}$ est un scindage de Lazard de S_3 qui n'est pas un scindage propre de Lazard. Notons que l'idéal engendré par les polynômes du système de départ est premier (renverser l'ordre des variables pour s'en convaincre) et par conséquent irréductible. Les idéaux de dimension zéro $\mathsf{sat}(S_1)$ et $\mathsf{sat}(S_2)$ doivent donc contenir l'idéal de dimension un $\mathsf{sat}(S_3)$. Les deux propositions suivantes vont nous permettre d'approfondir l'étude des scindages de Lazard.

Proposition V.1 *Soient T et U deux ensembles triangulaires de $\mathbf{R}_n$ satisfaisant les axiomes de Lazard. Alors, on a :*

$$\begin{array}{rlcl}
(i) & \sqrt{\mathsf{sat}(T)} & = & \mathsf{sat}(T) \\
(ii) & \mathbf{W}(T) \subseteq \overline{\mathbf{W}(U)} & \iff & \overline{\mathbf{W}(T)} \subseteq \overline{\mathbf{W}(U)} \\
(iii) & \mathbf{W}(T) \subseteq \overline{\mathbf{W}(U)} & \iff & \mathsf{sat}(U) \subseteq \mathsf{sat}(T) \\
(iv) & \mathbf{W}(T) \subseteq \overline{\mathbf{W}(U)} & \Longrightarrow & |\,U\,| \leq |\,T\,|
\end{array}$$

Preuve $\triangleright$ Vérifions (i). Soit $\mathbf{A}_T$ la tour d'extensions simples séparables associée à T. D'après la proposition III.18 p. 103 les anneaux $\mathbf{A}_T$ et $\mathbf{fr}(\mathbf{R}_n/\mathsf{sat}(T))$ ont les mêmes diviseurs de zéro. Or, d'après le théorème III.5 p. 105 l'anneau $\mathbf{A}_T$ est un produit de corps. Par conséquent, il n'admet pas de nilpotents. Donc, l'idéal $\mathsf{sat}(T)$ est radical.

Le point (ii) résulte des axiomes d'une fermeture. Rappelons que si E est un ensemble et $X \longmapsto \overline{X}$ est une application de 2^E, l'ensemble des parties de E, dans 2^E, cette application est une *fermeture* dans E si et seulement si on a :

(a) $(\forall X \subseteq E)\ X \subseteq \overline{X}$

(b) $(\forall X \subseteq E)\ \overline{\overline{X}} = \overline{X}$

(c) $(\forall X, Y \subseteq E)\ X \subseteq Y \implies \overline{X} \subseteq \overline{Y}$

Vérifions la proposition (iii). Notons tout d'abord que, grâce à la proposition (ii) précédente, et, au théorème III.3 p. 101, on a l'équivalence :

$$\mathbf{W}(T) \subseteq \overline{\mathbf{W}(U)} \quad \iff \quad \mathbf{V}_{\mathbf{K}}(\mathsf{sat}(T)) \subseteq \mathbf{V}_{\mathbf{K}}(\mathsf{sat}(U))$$

D'autre part, d'après le point (i), les idéaux $\mathsf{sat}(T)$ et $\mathsf{sat}(U)$ sont radicaux. Par suite, à l'aide du point (ii) de la proposition I.1 p. 20 et du théorème des zéros fort (proposition I.9 p. 32) on en déduit :

$$\mathbf{W}(T) \subseteq \overline{\mathbf{W}(U)} \quad \iff \quad \mathsf{sat}(U) \subseteq \mathsf{sat}(T)$$

Vérifions la proposition (iv). Notons tout d'abord que, de ce qui précède, on tire :

$$\mathbf{W}(T) \subseteq \overline{\mathbf{W}(U)} \quad \Longrightarrow \quad \mathsf{dim}(T) \leq \mathsf{dim}(U)$$

D'après le théorème III.6 p. 106, les ensembles de Lazard T et U sont des chaînes régulières (définition III.11 p. 88) dont les représentations respectives sont les idéaux $\mathsf{sat}(T)$ et $\mathsf{sat}(U)$ (théorème III.4 p. 103) puisque ces idéaux sont radicaux. Or, d'après [Kal93], si C est une chaîne régulière de $\mathbf{R}_n$, la dimension de sa représentation, soit $\mathsf{dim}(\mathsf{Rep}(C))$, est donnée par :

$$\mathsf{dim}(\mathsf{Rep}(C)) \;=\; n - |\,C\,|$$

On en déduit (iv). $\triangleleft$

Proposition V.2 *Soit $T \subseteq \mathbf{R}_n$ un ensemble de Lazard. Les conditions suivantes sont équivalentes :*

(i) *T n'admet pas de scindage propre de Lazard*

(ii) $\mathsf{sat}(T)$ *est un idéal premier*

(iii) $\mathbf{A}_T$ *est un corps*

Preuve $\triangleright$ Notons tout d'abord que les conditions (ii) et (iii) sont équivalentes d'après la proposition III.18 p. 103. Supposons maintenant que la condition (i) soit vraie. Alors, $\mathsf{sat}(T)$ est un idéal irréductible. Or, dans un anneau nœthérien, tout idéal irréductible est primaire. Comme, d'après le point (i) de la proposition V.1 l'idéal $\mathsf{sat}(T)$ est radical, cet idéal est en fait premier et l'on a (ii). La réciproque est immédiate puisque tout idéal premier est irréductible. $\triangleleft$

Remarque V.3 Soit $T, S_1, \ldots, S_l \subseteq \mathbf{R}_n$ des ensembles de Lazard. Supposons que $\mathsf{sat}(T)$ soit premier et que $\{S_1, \ldots, S_l\}$ soit un scindage de Lazard de T. D'après ce qui précède, il existe $i \in \{1, \ldots, l\}$ tel que $\mathsf{sat}(S_i) = \mathsf{sat}(T)$. Alors pour chaque $j \neq i$ l'idéal $\mathsf{sat}(S_j)$ contient strictement $\mathsf{sat}(T)$ et est de dimension strictement inférieure (puisque $\mathsf{sat}(S_i)$ est premier). Les idéaux premiers associés aux idéaux $\mathsf{sat}(S_j)$ pour $j \neq i$ sont alors des composantes immergées de $\mathsf{sat}(T)$.

Cette situation n'est pas incompatible pour autant avec le fait que $\{S_1, \ldots, S_l\}$ soit une décomposition de Lazard sans redondances d'une certaine variété. Pour s'en convaincre, reprendre l'exemple de Donati et Traverso dans l'introduction de ce chapitre.

Remarque V.4 Avec les notions et résultats du début de cette section, nous allons donner une première présentation de la *normalisation d'un polynôme par un ensemble de Lazard*. Le problème que l'on souhaite résoudre est le suivant : étant donné un polynôme $p \in \mathbf{R}_n$ et un ensemble de Lazard $T \subseteq \mathbf{R}_n$, déterminer si la quantité $\overline{p}^{\widehat{T}}$ est inversible dans $\mathbf{A}_T$.

L'anneau $\mathbf{A}_T$ étant un produit de corps, on peut espérer une réponse plus précise qu'un *oui* ou qu'un *non*. En fait, on cherche une réponse de la forme : sur tel facteur du produit de corps p est nul, sur tel autre il est inversible. Plus précisément, on cherche un scindage de Lazard $\{S_1, \ldots, S_l\}$ de T de sorte que, si $\mathbf{A}_{S_i}$ désigne la tour d'extensions simples séparables associée à S_i, la quantité $\overline{p}^{\widehat{S_i}}$ est soit nulle, soit inversible dans $\mathbf{A}_{S_i}$. Voici une première solution qui a le mérite d'être simple, mais qui n'est pas intéressante en pratique comme nous le verrons après.

Proposition V.3 *Supposons que, pour tout idéal $I \subseteq \mathbf{R}_n$, on dispose d'une opération booléenne* $\mathsf{prime?}(I)$ *qui retourne* vrai *si et seulement si I est premier. Supposons de plus que, pour tout ensemble de Lazard $T \subseteq \mathbf{R}_n$ tel que* $\mathsf{sat}(T)$ *n'est pas premier, on dipose d'une opération* $\mathsf{properSplit}(T)$ *qui retourne un scindage propre de T.*

Alors l'opération $\mathsf{theoricalNormalize}(p, T)$ *définie ci-après résout le problème posé dans la remarque V.4 précédente.*

```
theoricalNormalize(p, T) ==
    R := {T}
    S := { }
    for T in R repeat
```

```
        if prime?(sat(T))
            then
                S := S ∪ {T}
            else
                R := R ∪ properSplit(T)
    return S
```

Preuve $\triangleright$ Remarquons tout d'abord qu'à la fin de chaque itération de la boucle **while** l'ensemble $R \cup S$ est un scindage de Lazard de l'ensemble T initial. En effet, on vérifie sans peine que si $\{T_1, \ldots, T_l\}$ est un scindage de Lazard de T et $\{S_1, \ldots, S_m\}$ est un scindage de Lazard de l'un des T_i alors $\{T_1, \ldots, T_{i-1}, S_1, \ldots, S_m, T_{i+1}, \ldots, T_l\}$ est encore un scindage de Lazard de T.

Remarquons ensuite que la boucle **while** se termine en un nombre fini d'itérations. En effet, pour chaque ensemble de Lazard T que l'on scinde l'idéal $\mathsf{sat}(T)$ est remplacé par une intersection d'idéaux qui le contiennent strictement. Or $\mathbf{R}_n$ est nœthérien.

Remarquons pour finir que lorsque la boucle **while** se termine l'ensemble R est vide et l'ensemble S ne contient que des ensembles de Lazard T tel que $\mathbf{A}_T$ est un corps. Par conséquent, pour chacun de ces ensembles T, la quantité $\widehat{\overline{p}^T}$ est soit nulle, soit inversible. $\triangleleft$

Remarque V.5 L'algorithme de la proposition V.3 est inintéressant en pratique pour au moins deux raisons. La première est son coût puisqu'il nécessite d'écrire $\mathsf{sat}(T)$ comme une intersection d'idéaux premiers. La seconde est le *manque de concision* du résultat retourné, en particulier, si p est non nul sur chacun des facteurs du produit de corps $\mathbf{A}_T$!

Nous allons donner une solution plus paresseuse donc moins coûteuse et retournant des résultats plus concis. Comme nos algorithmes ne testent pas la primalité des idéaux saturés associés aux ensembles de Lazard, ils peuvent a priori *scinder* des idéaux premiers (en faisant apparaître des composantes immergées). C'est en fait le cas comme nous l'avons déjà vu avec l'exemple de Donati et Traverso dans l'introduction de ce chapitre. Pour assurer la terminaison de nos algorithmes nous aurons alors besoin de la notion suivante.

Définition V.2 *Soient T et S deux ensembles triangulaires de $\mathbf{R}_n$. On dira que T est* moins algébrique *(resp.* plus algébrique*) que S et l'on écrira $T \prec_{alg} S$ (resp. $T \succ_{alg} S$) si l'algorithme suivant retourne* **true** *(resp.* **false***). Dans les autres cas, on écrira : $T \sim_{alg} S$.*

```
compare(T, S) : Union(Boolean,"failed") ==
    T = ∅ and S ≠ ∅ => return true
    T ≠ ∅ and S = ∅ => return false
    (T = ∅) and (S = ∅) => return "failed"
    mmon(first(S)) <_rwl mmon(first(T)) => return true
    mmon(first(T)) <_rwl mmon(first(S)) => return false
    return compare(rest(T), rest(S))
```

où l'ordre $<_{rwl}$ (définition III.5 p. 83) et les opérations mmon *(notation I.1 p. 19)* first *et* rest *(notation III.3 p. 83) ont été introduites dans les trois premiers chapitres.*

Remarque V.6 Notons tout d'abord qu'il n'existe pas de suites d'ensembles triangulaires strictement croissantes pour l'ordre $\prec_{alg}$. En effet, pour des ensembles triangulaires T et S de $\mathbf{R}_n$ la condition $T \prec_{alg} S$ entraîne :

– soit $|T| <|S|$,
– soit il existe $s \in S$ et $t \in T$ tel que : $\mathsf{mmon}(s) <_{rwl} \mathsf{mmon}(t)$.

Or, il n'existe pas de suites strictement décroissantes dans l'ensemble des monômes à une variable $\mathbf{R}_n$ pour l'ordre $<_{rwl}$.

Remarque V.7 Nous allons maintenant commencer l'exposé de nos algorithmes. Le premier que nous décrirons est celui de l'opération normalize présentée dans l'introduction de ce chapitre. Comme tous nos algorithmes s'appellent mutuellement (directement ou indirectement) nous devons provisoirement admettre que nous disposons de l'un d'eux. Ceci est l'objet de la proposition V.4 suivante. On en déduira alors la proposition V.5 ci-après sur laquelle repose l'opération normalize (proposition V.6).

Proposition V.4 *Soit $T \subseteq \mathbf{R}_n$ un ensemble de Lazard et $p \in \mathbf{R}_n$ un polynôme initialement réduit par rapport à T tel que : $p \notin \mathbf{R}$ et $\mathsf{mvar}(p) \in \mathsf{algVar}(T)$. Alors il existe un algorithme implantant l'opération quasiRecipElseSplit(p, T) qui retourne :*

(i) *soit un couple (q, r) de polynômes de $\mathbf{R}_n$ initialement réduits par rapport à T tel que :*

(a) $r \notin \mathbf{R} \quad \Longrightarrow \quad \mathsf{mvar}(r) \notin \mathsf{algVar}(T)$

(b) $r \notin \mathbf{R} \quad \Longrightarrow \quad \mathsf{mvar}(r) < \mathsf{mvar}(p)$

(c) $\overline{pq - r}^{\widehat{T}} \;=\; 0$

(d) $\overline{r}^{\widehat{T}} = 0 \quad \Longrightarrow \quad \overline{p}^{\widehat{T}} = 0$

(ii) *soit un scindage $\{S_1, \ldots, S_l\}$ de Lazard de T avec $l \geq 2$ et $T \prec_{alg} S_i$ pour $1 \leq i \leq l$.*

Remarque V.8 Avec les hypothèses et notations de la proposition V.4 notons $\mathbf{A}_T$ la tour d'extensions simples séparables associée à T. Remarquons que si quasiRecipElseSplit(p, T) retourne un couple (q, r) tel que $\overline{r}^{\widehat{T}}$ soit inversible dans $\mathbf{A}_T$ alors $\overline{p}^{\widehat{T}}$ est également inversible dans $\mathbf{A}_T$. Ceci sera en particulier le cas si r est non nul et normalisé par rapport à T. L'algorithme donné plus loin implantant l'opération quasiRecipElseSplit(p, T), lorsqu'un couple (q, r) doit être retourné, produira un polynôme r normalisé par rapport à T. Cependant, pour construire l'opération normalizeElseSplit (proposition V.5) à partir de l'opération quasiRecipElseSplit(p, T), il suffit que cette dernière produise un polynôme r *moins algébrique* que p au sens des conditions (a) et (b) ci-dessus.

La condition (d) ci-dessus exprime, quant à elle, le fait que $\overline{p}^{\widehat{T}}$ et $\overline{q}^{\widehat{T}}$ ne peuvent être des diviseurs de zéro conjoints (i.e. leur produit ne peut être nul que si l'une de ces quantités est nulle). En fait, l'algorithme donné plus loin implantant l'opération quasiRecipElseSplit(p, T), lorsque $\overline{p}^{\widehat{T}}$ est un diviseur de zéro, retournera un scindage de Lazard T.

Proposition V.5 *Soit $T \subseteq \mathbf{R}_n$ un ensemble de Lazard et $p \in \mathbf{R}_n$ un polynôme initialement réduit par rapport à T. Alors l'opération* $\mathsf{normalizeElseSplit}(p, T)$ *définie ci-après retourne :*

(*i*) *soit $n \in \mathbf{R}_n$ un polynôme normalisé par rapport à T tel qu'il existe $q \in \mathbf{R}_n$ tel que :*

  (*a*) $\overline{qp - n}^{\widehat{T}} = 0$

  (*b*) $\overline{p}^{\widehat{T}} = 0 \iff \overline{n}^{\widehat{T}} = 0$

(*ii*) *soit un scindage $\{S_1, \ldots, S_l\}$ de Lazard de T avec $l \geq 2$ et $T \prec_{alg} S_i$ pour $1 \leq i \leq l$.*

```
normalizeElseSplit(p, T) ==
    T = ∅ => return p
    p ∈ R => return p
    i_p := init(p)
    m_p := mmon(p)
    t_p := tail(p)
    while (i_p ∉ R) repeat
        if algebraic?(mvar(i_p), T)
            then
                σ := quasiRecipElseSplit(i_p, T)
                split?(σ) => return σ
                (q, r) := σ
                i_p := r
                t_p := q t_p
        if (i_p ∉ R)
            then
                t_p := tail(i_p) m_p + t_p
                m_p := mmon(i_p) m_p
                i_p := init(i_p)
    return i_p m_p + t_p
```

Preuve ▷ Remarquons tout d'abord que l'opération $\mathsf{normalizeElseSplit}(p, T)$ retourne un scindage de Lazard de T si et seulement si l'un des appels $\mathsf{quasiRecipElseSplit}(i_p, T)$ retourne un tel scindage. Supposons désormais que l'opération $\mathsf{normalizeElseSplit}(p, T)$ ne retourne pas un scindage de Lazard de T et vérifions que $\mathsf{normalizeElseSplit}(p, T)$ retourne un polynôme n de $\mathbf{R}_n$ normalisé par rapport à T tel qu'il existe un polynôme q de $\mathbf{R}_n$ tel que :

$$(\overline{p}^{\widehat{T}} = 0 \iff \overline{n}^{\widehat{T}} = 0) \quad \text{et} \quad \overline{qp - n}^{\widehat{T}} = 0$$

Si T est vide ou si p appartient à $\mathbf{R}$ on a clairement le résultat avec $n = p$ et $q = 1$. Remarquons maintenant qu'avant d'entrer dans la boucle **while** on a :

$$p = i_p m_p + t_p$$

Notons que si p est normalisé par rapport à T, cette égalité reste vraie à la fin de chaque itération de la boucle **while**. Par conséquent, si p est normalisé par rapport à T, le polynôme retourné par $\mathsf{normalizeElseSplit}(p, T)$ est p lui-même et le résultat annoncé est encore vérifié avec $n = p$ et $q = 1$. Pour les cas que nous venons d'aborder depuis le début de cette preuve l'équivalence : $\overline{p}^{\widehat{T}} = 0 \iff \overline{n}^{\widehat{T}} = 0$ est immédiate.

Reste donc à traiter la cas où p ou l'un de ses initiaux itérés a une variable principale algébrique par rapport à T. A chaque fois que cela se produit, c'est-à-dire lorsque la variable principale de i_p est algébrique par rapport à T, la quantité $\overline{i_p m_p + t_p}^{\widehat{T}}$ est multipliée par $\overline{q}^{\widehat{T}}$ définie par :

$$\mathsf{quasiRecipElseSplit}(i_p, T) = (q, r)$$

et l'on a :

$$\overline{q(i_p m_p + t_p)}^{\widehat{T}} = \overline{r m_p + q t_p}^{\widehat{T}}$$

à l'aide du point (c) de la proposition V.4. D'après le point (d) de cette même proposition, on a l'implication :

$$\overline{i_p}^{\widehat{T}} \neq 0 \implies \overline{r}^{\widehat{T}} \neq 0$$

Comme le polynôme r est initialement réduit, on a ainsi vérifié que le polynôme i_p est non nul et initialement réduit par rapport à T tout au long de l'algorithme, sauf si i_p était nul avant d'entrer dans la boucle **while**.

De plus, d'après le point (b) de la proposition V.4, le polynôme r est soit dans $\mathbf{R}$, soit de variable principale strictement inférieure à cette de i_p. Enfin, d'après le point (a) de cette même proposition, si r n'appartient pas à $\mathbf{R}$, la variable principale de r n'est pas algébrique par rapport à T.

Par suite, on quitte la boucle **while**. De plus, au sortir de cette boucle, le polynôme :

$$n = i_p m_p + t_p$$

est normalisé par rapport à T et vérifie :

$$(\exists q \in \mathbf{R}_n) \ \overline{pq}^{\widehat{T}} = \overline{n}^{\widehat{T}}$$

où p désigne le polynôme *de départ*. Enfin, on a bien l'équivalence :

$$\overline{p}^{\widehat{T}} = 0 \iff \overline{n}^{\widehat{T}} = 0$$

puisque i_p est non nul et initialement réduit par rapport à T tout au long de l'algorithme, sauf si i_p était nul (et par suite p) avant d'entrer dans la boucle **while**. $\triangleleft$

Proposition V.6 *Soit T un ensemble triangulaire de $\mathbf{R}_n$ satisfaisant les axiomes de Lazard et p un polynôme de $\mathbf{R}_n$. Alors l'opération* $\mathsf{normalize}(p, T)$ *suivante retourne un ensemble de paires $\{[n_1, S_1], \ldots, [n_l, S_l]\}$ où les n_i sont des polynômes de $\mathbf{R}_n$ et les S_i sont des ensembles triangulaires satisfaisant les axiomes de Lazard le tout tel que :*

(i) $\mathsf{normalizeElseSplit}(\mathsf{iRed}(p, S_i), S_i) = n_i$

(ii) *$\{S_1, \ldots, S_l\}$ est un scindage de Lazard de T avec $T \prec_{alg} S_i$ pour $1 \leq i \leq l$ si $l \geq 2$.*

De plus, pour $1 \leq i \leq l$, si $\mathbf{A}_{S_i}$ désigne la tour d'extensions simples séparables associée à S_i, alors on a :

(iii) $n_i \neq 0 \iff \overline{p}^{\widehat{S_i}} \in \mathbf{Un}(\mathbf{A}_{S_i})$

(iv) $n_i = 0 \iff p \in \mathsf{sat}(S_i)$

```
normalize(p, T) ==
    R := {[p, T]}
    S := { }
    for [q, U] in R repeat
        q := iRed(q, U)
        if split?((ν :=normalizeElseSplit(q, U)))
            then
                R := R ∪ {[q, U] for U ∈ ν}
            else
                S := S ∪ {[ν, U]}
    return S
```

Preuve ▷ Les propriétés (i) et (ii) résultent immédiatement de la proposition V.5. Notons que la boucle **for** se termine grâce à la remarque V.6. Vérifions maintenant les propriétés (iii) et (iv). Soit i tel que $1 \leq i \leq l$. Comme n_i est normalisé par rapport à S_i, grâce à la proposition III.9 p. 88 on a :

$$n_i \neq 0 \iff \overline{n_i}^{\widehat{S_i}} \in \mathbf{Un}(\mathbf{A}_{S_i})$$

d'où l'on tire :

$$n_i = 0 \iff \overline{n_i}^{\widehat{S_i}} = 0$$

Par suite, d'après le point (b) de l'alinéa (i) de la proposition V.5, on a :

$$n_i = 0 \iff \overline{\mathsf{iRed}(p, S_i)}^{\widehat{S_i}} = 0$$

et par conséquent, avec la définition III.14 p. 95 :

$$n_i = 0 \implies p \in \mathsf{sat}(S_i)$$

D'autre part, d'après le point (a) de l'alinéa (i) de la proposition V.5 :

$$n_i \neq 0 \implies \overline{\mathsf{iRed}(p, S_i)}^{\widehat{S_i}} \in \mathbf{Un}(\mathbf{A}_{S_i})$$

et par conséquent :

$$n_i \neq 0 \implies \overline{p}^{\widehat{S_i}} \in \mathbf{Un}(\mathbf{A}_{S_i})$$

On en déduit finalement les équivalences données dans les points (iii) et (iv) grâce à la proposition III.18 p. 103. En effet, cette proposition établit l'incompatibilité des assertions $p \in \mathsf{sat}(S_i)$ et $\overline{p}^{\widehat{S_i}} \in \mathbf{Un}(\mathbf{A}_{S_i})$ (qui ne sont pas pour autant la négation l'une de l'autre). ◁

Remarque V.9 On considère maintenant le système suivant (Liu et Lorenz) où les variables sont ordonnées par : $x > y > z > t > a$ et où les coefficients sont dans $\mathbb{Q}$:

$$\{-x + (z - t)\ y + a,\ -z\ x - y + t\ z + a,\ t\ x - t\ y - z + a,\ (y - z)\ x - t + a\}$$

À partir de ce système, nous allons donner quelques exemples de normalisation d'un polynôme par rapport à un ensemble de Lazard. Les résultats reproduits ci-après sont ceux fournis par nos programmes en AXIOM. Ils apparaîtront dans ce texte dans une police de `machine à écrire`.

On remarque tout d'abord que l'on peut *éliminer* x, l'idéal engendré par les polynômes ci-dessus ayant aussi pour système de générateurs :

```
                          2
[x + (- z + t)y - a, (t z - t  - t)y - z + a t + a,
   2
 (z  - t z + 1)y + (- t + a)z - a,
        2       2
 (z - t)y  + (- z  + t z + a)y - a z - t + a]
```

Considérons le sous-système suivant :

```
        2                        2
[(t z - t  - t)y - z + a t + a,(z  - t z + 1)y + (- t + a)z - a]
```

Comme nous le verrons plus loin, une décomposition de Lazard de la variété affine associée à ce sous-système est :

```
    3       2          2     3    2                 2
[ {z  + (- t  - t - a)z  + (t  + t  - a t + 1)z + a t  - a,
     2           2       2
   (t  + 2t)y + z  + (- t  - a + 1)z - 2a t - a}
  ,
        2                                            2
 {t + 2,z  + (- a - 3)z + 3a,(8a + 8)y + (- a + 1)z + a  + 7a},
           2          2
 {t,z - a,(a  + 1)y + a  - a}, {a + 1,t + 2,z - 3,4y + 1}]
```

On note T_1, T_2, T_3, T_4 les quatre ensembles de Lazard ci-dessus dans l'ordre où ils sont donnés. Nous allons *normaliser* le polynôme :

$$p \ = \ (z - t)\ y^2 + \left(-z^2 + t\ z + a\right)\ y - a\ z - t + a$$

par rapport à chacun des ensembles T_i. Nous commençons par le dernier à savoir :

$$T_4 \ = \ \{a + 1,\ t + 2,\ z - 3,\ 4\ y + 1\}$$

Puisque nous avons éliminé x, toutes les variables sont algébriques par rapport à T_4 et le polynôme p doit *se normaliser en une constante.* Comme T_4 n'est constitué que de polynômes de degré un, cette normalisation s'obtient par simple substitution. La valeur de cette constante n'importe pas en pratique. La seule chose qui compte est de savoir si elle est nulle (voir la proposition V.6). Pour cette raison, notre programme retourne :

```
[[val= 1,tower= {a + 1,t + 2,z - 3,4y + 1}]]
```

Nous passons à la normalisation de p par rapport à :

$$T_3 \; = \; \left\{t,\; z-a,\; \left(a^2+1\right)\; y+a^2-a\right\}$$

La variable a n'est pas algébrique dans T_3 et les trois polynômes de T_3 sont de degré un (en leur variable principale). Il suffit donc de réduire p par rapport à T_3. Notons que :

$$\overline{p}^{\widehat{T_3}} \; = \; \frac{1}{\left(a^2+1\right)^2}\; \left(-2\; a^4+a^3+a\right)$$

Pour cette raison normalize(p, T_3) retourne :

```
           4    3                           2           2
  [[val= 2a  - a  - a,tower= {t,z - a,(a  + 1)y + a  - a}]]
```

Nous passons à la normalisation de p par rapport à :

$$T_2 \; = \; \left\{t+2,\; z^2+(-a-3)\; z+3\; a,\; (8\; a+8)\; y+(-a+1)\; z+a^2+7\; a\right\}$$

Comme le polynôme en z dans T_2 est de degré 2, réduire ne suffit pas pour normaliser. On a du reste :

$$\overline{p}^{\widehat{T_2}} \; = \; \frac{1}{64\; (a+1)^2}\; \left(\left(3\; a^3+145\; a^2+293\; a+135\right)\; z-3\; a^4-17\; a^3+91\; a^2+185\; a+128\right)$$

Cependant le numérateur de $\overline{p}^{\widehat{T_2}}$ est inversible dans $\mathbf{A}_{T_2}$ et d'inverse (calculé par quasiRecipElseSplit) :

```
           3       2                        3       2
  [num= (3a  + 145a  + 293a + 135)z - 137a  - 819a  - 1199a - 533,

    den =
             7         6          5           4           3           2
        384a  + 2176a  - 63296a  - 335232a  - 644096a  - 612480a  - 306752a
      +
        - 68224]
```

En prenant la partie primitive du dénominateur de la fraction ci-dessus, normalize(p, T_2) retourne :

```
             7       6        5         4          3         2
   [[val= 6a  + 34a  - 989a  - 5238a  - 10064a  - 9570a  - 4793a - 1066,
                     2                                               2
     tower= {t + 2,z  + (- a - 3)z + 3a,(8a + 8)y + (- a + 1)z + a  + 7a}]]
```

Le cas T_1 est analogue à celui de T_2. Réduire p par T_1 ne suffit pas puisque :

$$\mathsf{head}(\mathsf{iRed}(p, T_1)) \; = \; \left(-t^3-2\; t^2+(-2\; a-3)\; t-2\right)\; z^2$$

et normalize(p, T_1) retourne :

$$\begin{aligned}[[\ \mathrm{val} = {} & 4t^{15} + (-8a + 28)t^{14} + (4a^2 - 50a + 82)t^{13} + (16a^2 - 132a + 144)t^{12} \\ & + (6a^3 + 32a^2 - 226a + 188)t^{11} + (20a^3 + 86a^2 - 322a + 188)t^{10} \\ & + (-2a^4 + 9a^3 + 178a^2 - 375a + 138)t^9 \\ & + (-12a^4 - 14a^3 + 222a^2 - 356a + 92)t^8 \\ & + (-a^5 - 26a^4 - 10a^3 + 244a^2 - 297a + 62)t^7 \\ & + (-4a^5 - 22a^4 - 32a^3 + 208a^2 - 192a + 42)t^6 \\ & + (-3a^5 + 2a^4 - 25a^3 + 119a^2 - 102a + 33)t^5 \\ & + (4a^5 + 18a^4 - 5a^3 + 60a^2 - 57a + 12)t^4 \\ & + (4a^5 + 16a^4 - 8a^3 + 28a^2 - 12a + 4)t^3 + (8a^4 - 4a^3 - 4a)t^2 \\ , \\ \mathrm{tower} = {} & \{z^3 + (-t^2 - t - a)z^2 + (t^3 + t^2 - a\,t + 1)z + a\,t^2 - a, \\ & (t^2 + 2t)y + z^2 + (-t^2 - a + 1)z - 2a\,t - a\}\ ]]\end{aligned}$$

Ces calculs montrent en particulier que p est inversible par rapport à chacune des tours associées à T_1, T_2, T_3, T_4. Nous verrons plus loin ce que l'on peut déduire de ces calculs du point de vue de la résolution du système initial.

Remarque V.10 Considèrons le système suivant que Dominique Duval nous a communiqué :

$$\begin{aligned}[& y^5 - x\,y^4 + (x^2 - x - 1)y^3 + (-x^3 + x^2 + x)y^2 + (-x^3 + x)y + x^4 - x^2, \\ & -y^4 + (2x - 1)y^3 + (-3x^2 + 2x + 1)y^2 + (-3x^2 + 1)y + 4x^3 - 2x, \\ & 5y^4 - 4x\,y^3 + (3x^2 - 3x - 3)y^2 + (-2x^3 + 2x^2 + 2x)y - x^3 + x]\end{aligned}$$

La première équation est une courbe (constiuée d'une droite, d'un cercle et d'une parabole) dont on cherche les points singuliers à l'aide de ce système. Nous vérifierons plus loin qu'une décomposition de Lazard du sous-système constitué par les deux premiers polynômes est :

2 2 3 2

```
[{x  + x - 1,y  - x},{x,y  - y},{2x  - 1,y - x},{x - 1,y - 1}]
```

On note respectivement T_1, T_2, T_3, T_4 les quatre ensembles de Lazard. Comme chacun des T_i engendre un idéal de dimension zéro (i.e. toutes les variables sont algébriques) le normalisé d'un polynôme de $\mathbb{Q}[x, y]$ par un T_i est une constante. Par suite, en normalisant le troisième polynôme par rapport à chaque T_i, nous allons donc *finir* de résoudre le système initial.

Par rapport à $T_1 = \{x^2 + x - 1,\ y^2 - x\}$:

```
                    2           2
[[val= 0,tower= {x   + x - 1,y   - x}]]
```

Par rapport à $T_2 = \{x,\ y^3 - y\}$:

```
                      2
[[val= 1,tower= {x,y   - 1}],[val= 0,tower= {x,y}]]
```

Par rapport à $T_3 = \{2\,x^2 - 1,\ y - x\}$:

```
                     2
[[val= 0,tower= {2x   - 1,y - x}]]
```

Enfin, par rapport à $T_4 = \{x - 1,\ y - 1\}$:

```
[[val= 0,tower= {x - 1,y - 1}]]
```

Finalement, une décomposition de Lazard des points singuliers de notre courbe est :

$$\Big\{\Big\{x^2 + x - 1,\ y^2 - x\Big\},\ \Big\{2\,x^2 - 1,\ y - x\Big\},\ \{x,\ y\},\ \{x - 1,\ y - 1\}\Big\}$$

Proposition V.7 *Soient T et U deux ensembles triangulaires de $\mathbf{R}_n$ satisfaisant les axiomes de Lazard. Alors, l'opération booléenne* containedInComponent?(T, U) *définie ci-après retourne* vrai *si et seulement si :*

$$\mathbf{W}(T) \subseteq \mathbf{V}(U)$$

containedInComponent?(T, U) ==
    $|T| < |U|$ => **false**
    $R := \{[U, T]\}$
    for $[U, T]$ **in** R **repeat**
        $U = \emptyset$ => **iterate**
        $p :=$ first(U)
        $q :=$ iRed(p, T)
        $\nu :=$ normalizeElseSplit(q, T)
        not split?(ν) =>
                $(\nu \neq 0)$ => **return false**
                $R := R \cup \{[$rest$(U), T]\}$
        $R := R \cup \{[U, S]$ **for** $S \in \nu\}$
    return true

Preuve $\triangleright$ Remarquons tout d'abord qu'en raisonnant comme dans la preuve de la proposition V.1. on a l'équivalence :

$$\mathbf{W}(T) \subseteq \mathbf{V}(U) \quad \Longleftrightarrow \quad U \subseteq \mathsf{sat}(T)$$

Introduisons maintenant quelques notations. Dans la suite, nous désignerons par T_1 et U_1 les valeurs initiales des ensembles de Lazard T et U lors de l'appel à containedInComponent ?, par R_i la valeur de l'ensemble R au début de la *i-ième* itération de la boucle **for** et par $[U_i, T_i]$ l'élément de l'ensemble R_i que l'on considère dans cette *i-ième* itération.

Remarquons ensuite qu'il est facile, à l'aide du point (ii) de la proposition V.5, d'établir qu'il n'y a qu'un nombre fini d'itérations de la boucle **for**. Soit N ce nombre. Pour un entier i tel que $1 \leq \imath \leq N$, notons C_i la conjonction des $\mathbf{W}(T) \subseteq \mathbf{V}(U)$ pour $[U, T]$ décrivant R_i. La validité de l'algorithme est alors équivalente à :

$$(\forall i \in \{1, \ldots, N-1\}) \ C_i \quad \Longleftrightarrow \quad C_{i+1}$$

Soit donc i tel que $1 \leq i \leq N$ et $[U_i, T_i]$ l'élément de l'ensemble R_i que l'on considère. Notons que lorsque $i = N$, soit l'ensemble R_i se réduit à un singleton, soit la variable booléenne false est retournée. Plaçons-nous à la première instruction de la boucle **for**.

- Si U_i est vide, par convention, U_i s'identifie à l'idéal trivial $\langle 0 \rangle$. D'où, dans ce cas :

  $$U_i \subseteq \mathsf{sat}(T_i)$$

  Par conséquent, si $i = N$, le résultat retourné est vrai ce qui est correcte d'après la valeur de *l'invariant de boucle* C_i. Si $i < N$, on a bien conservé l'équivalence :

  $$C_i \quad \Longleftrightarrow \quad C_{i+1}$$

- Si U_i n'est pas vide, en reprenant les notations de l'algorithme, supposons tout d'abord que normalizeElseSplit(q, T_i) retourne un scindage $\{S_1, \ldots, S_l\}$ de Lazard de T. On a donc :

  $$\mathsf{sat}(T) \ = \ \cap_1^l \ \mathsf{sat}(S_i)$$

  Par conséquent, on a l'équivalence :

  $$U_i \subseteq \mathsf{sat}(T) \quad \Longleftrightarrow \quad (\forall i \in \{1, \ldots, l\}) \ U_i \subseteq \mathsf{sat}(S_i)$$

  On a nécessairement dans ce cas $i < N$, et nous venons de vérifier l'équivalence :

  $$C_i \quad \Longleftrightarrow \quad C_{i+1}$$

Supposons maintenant que normalizeElseSplit(q, T_i) retourne un polynôme n.

- Si $n \neq 0$, alors d'après le point (iv) de la proposition V.6 on a : $p \notin \mathsf{sat}(T_i)$. Donc : $U_i \not\subseteq \mathsf{sat}(S_i)$. De plus, on a : $i = N$ et le résultat retourné est faux. Bref, grâce à l'invariant de boucle C_i, l'algorithme est encore correcte dans ce cas.
- Supposons pour finir que l'on ait : $n = 0$. Ceci montre que : $p \in \mathsf{sat}(T_i)$. On a nécessairement dans ce cas $i < N$, et nous venons de vérifier l'équivalence :

  $$C_i \quad \Longleftrightarrow \quad C_{i+1}$$

$\triangleleft$

2 Pgcd normalisé

Proposition V.8 *Soient p_1, p_2 deux polynômes de $\mathbf{R}_n$, non-constants, et de même variable principale $v = X_i$ pour $1 \leq i \leq n$. Soit $T \subseteq \mathbf{R}_{i-1}$ un ensemble de Lazard tel que p_1 et p_2 soient initialement réduits par rapport à T.*

Alors l'opération subResNormalizedGcdElseSplit (p_1, p_2, v, T)*, définie page suivante, retourne :*

(a) *soit un polynôme $g \in \mathbf{R}_n$ non constant, normalisé par rapport à T, de variable principale v et tel que :*

$$\langle \overline{p_1}^{\widehat{T}}, \overline{p_2}^{\widehat{T}} \rangle_{\mathbf{A}_T[v]} = \langle \overline{g}^{\widehat{T}} \rangle_{\mathbf{A}_T[v]}$$

(b) *soit un polynôme $r \in \mathbf{R}_n$ non nul, normalisé par rapport à T, constant ou de variable principale strictement inférieure à v et tel que $\overline{r}^{\widehat{T}}$ soit le résultant de $\overline{p_1}^{\widehat{T}}$ et $\overline{p_2}^{\widehat{T}}$ dans $\mathbf{A}_T$*

(c) *soit un scindage $\{S_1, \ldots, S_l\}$ de Lazard de T avec $l \geq 2$ et $T \prec_{alg} S_i$ pour $1 \leq i \leq l$.*

Remarque V.11 Avec les hypothèses et notations de la proposition V.8. Lorsque subResNormalizedGcdElseSplit (p_1, p_2, v, T) retourne un polynôme g satisfaisant (a), le polynôme $\overline{g}^{\widehat{T}}$ est un pgcd de $\overline{p_1}^{\widehat{T}}$ et $\overline{p_2}^{\widehat{T}}$ dans $\mathbf{A}_T[v]$ au sens de la définition II.9 p. 65.

Dans la suite, nous utiliserons également une *version étendue* (c'est-à-dire avec calcul des coefficients de Bezout) de calcul d'un pgcd normalisé de deux polynômes modulo un ensemble de Lazard. Cette opération, bâptisée extendedSubResGcdElseSplit (p_1, p_2, v, T), retournera :

(d) soit des polynômes f,h et g dans $\mathbf{R}_n$ initialement réduits par rapport à T, avec g normalisé par rapport à T, et tels que :

$$\overline{f p_1 + h p_2}^{\widehat{T}} = \overline{g}^{\widehat{T}}$$

(e) soit un scindage $\{S_1, \ldots, S_l\}$ de Lazard de T avec $l \geq 2$ et $T \prec_{alg} S_i$ pour $1 \leq i \leq l$.

Cette opération sera pratiquement identique à l'algorithme extendedSubResGcdElseSplit du cas des tours algébriques d'extension simples (voir proposition IV.6 p. 123). Les seules différences sont les suivantes :

- l'opération normalize (voir proposition IV.5 p. 122) du cas des tours algébriques d'extensions simples doit être remplacée par une version étendue de l'opération normalizeElseSplit de la proposition V.5
- les α, β, ψ, ainsi que les dénominateurs, sont des polynômes au lieu d'être dans $\mathbf{R}$.

Notons qu'alors, si cette version étendue ne retourne pas un scindage, et si le pgcd calculé est de degré nul en la variable v, il ne s'agit pas du résultant r de la proposition précédente, mais d'un sous-multiple de celui-ci. En effet, pour la version étendue du pgcd, on n'a pas besoin de calculer le résultant, s'il n'y a pas de pgcd non trivial puisqu'on utilise le pgcd étendu pour savoir si une quantité algébrique est inversible. Peu importe que le numérateur de l'éventuel inverse soit exactement le résultant.

Algorithme et Preuve ▷ Nous donnons maintenant un algorithme pour l'opération subResNormalizedGcdElseSplit (p_1, p_2, T). Comme cet algorithme est un peu long et technique nous avons introduit deux sous-procédures appelées terminate et update. Ces sous-procédures sont définies à la fin de cette preuve Une présentation *grossière* de leurs spécifications est donnée juste après l'algorithme ci-dessous. Dans cet algorithme, certaines instructions ont été encadrées. Notons que si T est vide, ces instructions encadrées *ne font rien* : plus précisément, les polynômes p_i sont inchangés par ces instructions.

subResNormalizedGcdElseSplit (p_1, p_2, v, T) ==
    if mdeg$(p_1) <$ mdeg(p_2)
        then
            if odd ?(mdeg(p_1)) **then** $p_1 := -p_1$
            $(p_1, p_2) := (p_2, p_1)$
    $\boxed{\nu := \mathsf{normalizeElseSplit}(p_1, T)}$
    $\boxed{\mathsf{split?}(\nu) \Rightarrow \textbf{return } \nu}$
    $\boxed{p_1 := \mathsf{reduce}(\nu, T)}$
    $\boxed{\nu := \mathsf{normalizeElseSplit}(p_2, T)}$
    $\boxed{\mathsf{split?}(\nu) \Rightarrow \textbf{return } \nu}$
    $\boxed{p_2 := \mathsf{reduce}(\nu, T)}$
    $p_3 :=$ prem(p_1, p_2)
    $\boxed{p_3 := \mathsf{reduce}(p_3, T)}$
    $\delta :=$ mdeg$(p_1) -$ mdeg(p_2)
    $\psi := 1$
    while $p_3 \neq 0$ **repeat**
        $(p_1, p_2) := (p_2, p_3)$
        if odd ?(δ) **then** $p_2 := -p_2$
        $\boxed{\nu := \mathsf{normalizeElseSplit}(p_2, T)}$
        $\boxed{\mathsf{split?}(\nu) \Rightarrow \textbf{return } \nu}$
        $\boxed{p_2 := \mathsf{reduce}(\nu, T)}$
        if deg$(p_2, v) > 0$
            then
                $p_3 :=$ prem(p_1, p_2)
                $\boxed{p_3 := \mathsf{reduce}(p_3, T)}$
            else
                $p_3 := 0$
        $(\psi, \delta, \beta) :=$ update (p_1, p_2, ψ, δ)
        $p_3 := p_3$ exquo β
    terminate (p_2, v, δ, ψ)

L'opération terminate détermine si le polynôme retourné par cet algorithme satisfait la condition (a) ou la condition (b), tandis que l'opération update calcule les nouvelles valeurs des quantités ψ, δ et β à partir des quantités p_1, p_2, ψ et δ.

Remarquons tout d'abord que si l'on élimine de cet algorithme les lignes encadrées il s'agit ni plus ni moins du calcul du résultant de p_1 et p_2 :

- vus comme polynômes de $\mathbf{R}_{i-1}[X_i]$
- à l'aide de la méthode des sous-résultants
- et avec retour du dernier polynôme p_2 non nul (au lieu de 0) si le résultant est nul

Pour la notion de *résultant* voir la définition I.9 p. 25. Pour la notion de *sous-résultant* voir [Loo82]. Pour une présentation des algorithmes de calcul de pgcd ou de résultant fondé sur les sous-résultant voir [DST93]. Pour un exposé détaillé de ces algorithmes voir [Loo82]. La remarque II.10 p. 66 ne présente que l'algorithme du calcul du pgcd fondé sur les sous-résultant. Cependant celui du calcul du résultant est semblable. Enfin, pour un exposé clair et concis de toutes ces notions et algorithmes voir [Val95].

Rétablissons maintenant ces lignes encadrées. Nous remarquons maintenant que les différents polynômes calculés par l'algorithme sont *maintenus* dans l'anneau $\mathbf{A}_T[v]$ par les instructions $p_i := \mathsf{reduce}(\ldots, T)$. Par suite, les quantités ψ et β (par lesquelles on divise d'autres quantités) appartiennent à $\mathbf{A}_T$. En effet, elles sont obtenues à partir d'initiaux des polynômes p_i (voir la procédure update). Par conséquent la *réduction complète* :

$$p_3 := \mathsf{reduce}(p_3, T)$$

est impérative si l'on souhaite que la division :

$$p_3 := p_3 \ \mathsf{exquo}\ \beta$$

soit exacte. Voir la remarque V.12.

Nous remarquons ensuite que ces différents polynômes calculés dans l'algorithme sont tous normalisés (en tant que polynômes de $\mathbf{R}_n$). Par conséquent, ils sont tous unitaires en tant que polynômes de $\mathbf{A}_T[v]$ (proposition III.9). Par suite, chaque polynôme a le même degré sur chacun des facteurs du produit de corps $\mathbf{A}_T$. En particulier, après les deux normalisations se trouvant avant la boucle **while** (si celles-ci n'ont pas produit de scindage) les polynômes p_1 et p_2 de départ ont conservé leur variable principale v et leurs degrés. En effet, d'après la proposition V.5 les initiaux des polynômes p_1 et p_2 étant non nuls dans $\mathbf{A}_T$ (puisque p_1 et p_2 sont initialement réduits par rapport à T) leurs *normalisés* sont aussi non nuls (car inversibles dans $\mathbf{A}_T$).

Intéressons-nous plus particulièrement au résultat de l'algorithme ci-dessus lorsqu'il ne retourne pas un scindage. En effet, si notre algorithme retourne un scindage, la condition (c) est immédiatement satisfaite grâce à la proposition V.5. Le polynôme p_2 passé à l'opération terminate est le dernier pseudo-reste non nul calculé par l'algorithme ci-dessus. On distingue deux cas.

- Soit $\mathsf{deg}(g, v) > 0$ et le polynôme g retourné par terminate (p_2, v, δ, ψ) vérifie bien la condition (a) de l'énoncé de la proposition d'après la remarque II.10
- Soit $\mathsf{deg}(g, v) = 0$ et le polynôme r retourné par terminate (p_2, v, δ, ψ) vérifie bien la condition (b). En effet, on a les deux sous-cas suivants.
  - L'anneau $\mathbf{A}_T$ est un corps. Alors, notre algorithme est celui du calcul du résultant des polynômes $\overline{p_1}^{\widehat{T}}$ et $\overline{p_2}^{\widehat{T}}$ fondé sur la méthode des sous-résultants ([Loo82]) et *modifié à la façon* de l'algorithme *NewSubResGcd* de la proposition II.26 p. 68.

– L'anneau $\mathbf{A}_T$ est un produit de corps $\mathbf{K}_1 \times \cdots \times \mathbf{K}_s$. Du fait de la normalisation, chaque polynôme p_i a le même degré sur chacun des $\mathbf{K}_j$ et notre algorithme effectue simultanément ses calculs sur chacun des corps $\mathbf{K}_j$. En particulier, le polynôme p_2 passé à la procédure terminate est non nul, normalisé par rapport à T et de degré nul en v. Il est donc non nul et de de degré nul en v sur chacun des $\mathbf{K}_j$. Or, d'après le sous-cas précédent, notre algorithme est correcte sur un corps. Par conséquent, le projeté sur chaque $\mathbf{K}_j$ du polynôme retourné par terminate satisfait (b). Finalement, du fait des règles de calcul dans un produit de corps, l'algorithme de l'opération subResNormalizedGcdElseSplit (p_1, p_2, v, T) vérifie aussi (b) dans ce sous-cas.

Nous donnons maintenant les définitions des opérations terminate et update.

```
terminate (p2, v, δ, ψ) ==
    g := p2
    deg(g, v) > 0 => return g
    r := g
    if δ > 1
        then
            for i in 2···δ repeat
                r := r g exquo ψ
    return r

update (p1, p2, ψ, δ) ==
    α := init(p1)
    if δ > 1
        then
            ψ' := α
            for i in 2···δ repeat
                ψ' := α ψ' exquo ψ
            ψ := ψ'
        else
            ψ := 1
    δ := mdeg(p1) − mdeg(p2)
    β := ψ^δ α
    return (ψ, δ, β)
```

Voir aussi les preuves des propositions II.24 p. 67 et III.1 p. 79 pour des exemples détaillés d'argumentation du genre : *ça marche sur un corps, donc ça marche aussi sur un produit de corps.* ◁

Remarque V.12 On peut être dérouté à première vue par cet algorithme. D'une part, il essaye de tirer partie des techniques fondées sur les sous-résultants (généralement employées pour des polynômes univariés à coefficients dans un anneau intègre que l'on n'a pas envie de plonger dans son corps des fractions). D'autre part, il fait tout pour se ramener à des polynômes univariés à coefficients dans un corps. Il va même jusqu'à utiliser des fractions (au travers de l'opération modulo) !

Tout d'abord rappelons que le but de cet algorithme est de calculer un pgcd de deux polynômes univariés à coefficients dans une tour d'extensions simples séparables. De plus, lorsque ces deux polynômes n'ont pas de pgcd non trivial, on souhaite connaître leur résultant. À ce double titre, une méthode fondée sur les sous-résultants s'impose.

Or, pour utiliser une telle méthode (parmi celles bien connues), il faut se ramener à des polynômes à coefficients dans un anneau intègre. Ceci est rendu possible grâce au fait qu'une tour d'extensions simples séparables soit un produit de corps.

Enfin, nos polynômes sont dans nos programmes informatiques des éléments de $\mathbf{R}_n$ qu'il faut en permanence interprétés comme des éléments de $\mathbf{A}_T[v]$. En effet, cela est indispensable pour que les *divisions exactes par* β tombent juste. Quant au fait que les β (et les ψ) soient maintenus dans $\mathbf{A}_T$ cela nous est dicté par la normalisation. Cette dernière nous garantit le degré de chacun des polynômes sur chaque facteur d'un produit de corps. Elle permet aussi d'avoir des initiaux plus simples et par voie de conséquence des β et des ψ plus simples. En particulier, dans le cas purement algébrique, la normalisation permet de maintenir les initiaux dans $\mathbf{R}$ (voir le chapitre précédent).

Bref, il faut réduire nos polynômes et cette réduction doit être complète. On ne peut se contenter, par exemple, d'une réduction au sens des initiaux itérés (définition III.15 p. 95). Un bon moyen pour effectuer cette réduction complète est l'utilisation de l'opération modulo qui, si elle utilise (*paresseusement*) quelques fractions, limite, dans ses calculs intermédiaires, le grossissement des polynômes. En effet, elle interprète en permanence les coefficients des polynômes comme éléments d'une certaine tour d'extensions simples (voir la remarque III.10 p. 92) ce que ne fait pas la réduction complète par pseudo-divisions successives suivant les variables décroissantes (voir la définition II.4 p. 65). Cette seconde méthode peut donc propager dans ses calculs intermédiaires des quantités *nulles apparemment non nulles.*

Remarque V.13 L'algorithme précédent a résolu notre problème de calcul de pgcd sous réserve de *jeter* les quasi-composantes contenues dans $\overline{\mathbf{W}(T)}$ sur lesquelles s'annulaient l'un des initiaux des polynômes sous-résultants (apparaissant dans l'algorithme). Nous devons donc produire un algorithme *traitant* toutes les quasi-composantes d'un scindage de Lazard de $\mathbf{W}(T)$. En d'autres termes, de même que nous sommes passés de l'opération normalizeElseSplit à l'opération normalize, il nous faut passer de subResNormalizedGcdElseSplit à une opération subResNormalizedGcd. Or, on peut dire que c'est là que *les ennuis commencent.*

En effet, soit $\{S_1, \ldots, S_l\}$ un scindage de Lazard de T apparaissant dans l'algorithme précédent. Les polynômes p_1 et p_2 *de départ* peuvent ne plus être initialement réduits par rapport à l'un des S_j. Pire ! Une fois initialement réduits leurs variables principales peuvent devenir algébriques dans l'un des S_j.

Théorème V.1 *Soient p_1, p_2 deux polynômes de $\mathbf{R}_n$, non-constants, et de même variable principale $v = X_i$ pour $1 \leq i \leq n$. Soit $T \subseteq \mathbf{R}_{i-1}$ un ensemble de Lazard tel que p_1 et p_2 soient initialement réduits par rapport à T. Alors l'opération* subResNormalizedGcd *(p_1, p_2, v, T) définie ci-après retourne un ensemble $E = \{[g_i, S_i], \ldots, [g_l, S_l]\}$ où les g_i sont des polynômes de $\mathbf{R}_i$ et les S_i sont des ensembles de Lazard de $\mathbf{R}_{i-1}$ tel que :*

(i) *g_i est normalisé par rapport à S_i pour $1 \leq i \leq l$*

(ii) *pour chaque $[g, S]$ de E si $\mathbf{A}_S$ désigne la tour associée à S alors on a :*

$$\langle \overline{p_1}^{\widehat{S}}, \overline{p_2}^{\widehat{S}} \rangle_{\mathbf{A}_S[v]} \;=\; \langle \overline{g}^{\widehat{S}} \rangle_{\mathbf{A}_S[v]}$$

(iii) *pour chaque $[g, S]$ de E si $g \in \mathbf{R}$ ou si $\mathsf{mvar}(g) < v$ et si $\overline{p_1}^{\widehat{S}}$ et $\overline{p_2}^{\widehat{S}}$ sont de degré non nul en v alors :*

$$\mathsf{res}(\overline{p_1}^{\widehat{S}}, \overline{p_2}^{\widehat{S}}, v) = \overline{g}^{\widehat{S}}$$

(iv) *$\{S_1, \ldots, S_l\}$ est un scindage de Lazard de T avec $T \prec_{alg} S_i$ pour $1 \leq i \leq l$ si $l \geq 2$.*

Algorithme et Preuve $\triangleright$ Pour décrire notre algorithme nous introduisons deux sous-procédures de même nom : normalizedGcd

- La première s'applique sur un triplet (p_1, p_2, T) où les p_i sont des polynômes de $\mathbf{R}_{i-1}[X_i]$ de degré éventuellement nul en X_i et où T est un ensemble de Lazard de $\mathbf{R}_{i-1}$. Ses spécifications de sortie sont les mêmes que celles de l'opération subResNormalizedGcd (p_1, p_2, v, T).
- La seconde s'applique à un couple $(\{p_1, \ldots, p_s\}, T)$ où les p_i sont des polynômes de $\mathbf{R}_{i-1}[X_i]$ de degré éventuellement nul en X_i et où T est un ensemble de Lazard de $\mathbf{R}_{i-1}$. Elle retourne aussi un ensemble de couples $E = \{[g_i, S_i], \ldots, [g_l, S_l]\}$ où les g_i sont des polynômes de $\mathbf{R}_i$ et les S_i sont des ensembles de Lazard de $\mathbf{R}_{i-1}$ satisfaisant les points (i) et (iv) des spécifications de sortie des deux opérations précédentes ainsi que le point (v) suivant :

  (v) pour chaque $[g, S]$ de E si $\mathbf{A}_S$ désigne la tour associée à S alors on a :

$$\langle \overline{p_1}^{\widehat{S}}, \ldots, \overline{p_s}^{\widehat{S}} \rangle_{\mathbf{A}_S[v]} \;=\; \langle \overline{g}^{\widehat{S}} \rangle_{\mathbf{A}_S[v]}$$

subResNormalizedGcd (p_1, p_2, v, T) ==
 $\sigma :=$ subResNormalizedGcdElseSplit (p_1, p_2, v, T)
 not split?(σ) => **return** $\{[\sigma, T]\}$
 $S := \{\}$
 for $T \in \sigma$ **repeat**
  $S := S \cup$ normalizedGcd (p_1, p_2, T)
 return S

```
normalizedGcd (p1, p2, T) ==
    p1 := iRed(p1, T)
    p1 = 0 => return normalize(p2, T)
    p2 := iRed(p2, T)
    p2 = 0 => return normalize(p1, T)
    (p1 ∈ R) or (p2 ∈ R) => return {[1, T]}
    (v1, v2) := (mvar(p1), mvar(p2))
    if v1 ∈ algVar(T)
        then
            (p1, p2) := (p2, p1)
            if not normalized?(p1, T)
                then
                    ν := normalize(p1, T)
                    S := {}
                    for [p1, T] ∈ ν repeat
                        S := S ∪ normalizedGcd (p1, p2, T)
                    return S
    v1 = v2 => return subResNormalizedGcd (p1, p2, v1, T)
    v1 < v2 => return normalizedGcd ({p1} ∪ coefficients(p2, v2), T)
    normalizedGcd (coefficients(p1, v1) ∪ {p2}, T)
```

```
normalizedGcd (F ⊆ Rn, T) ==
    R := {[F, T]}
    S := {}
    for [F = {p1, ..., ps}, T] ∈ R repeat
        if s = 1
            then
                S := S ∪ normalize(p1, T)
            else
                σ := normalizedGcd (p1, p2, T)
                for [g, T] ∈ σ repeat
                    if g ∈ R
                        then
                            S := S ∪ {[g, T]}
                        else
                            R := R ∪ {[{g, p3, ..., ps}, T]}
    return S
```

On raisonne par récurrence sur i. Notons que si $i = 1$ alors T est vide, il n'y a pas de scindage possible, et l'opération subResNormalizedGcd(p_1, p_2, v, T) satisfait bien les points (i) à (iv) grâce à la proposition V.8 précédente.

Nous supposons donc $i > 1$. Du fait de la remarque remarque V.6, on peut raisonner par induction sur T à l'aide de l'ordre $\prec_{alg}$. Plus précisément, par induction sur T, nous allons vérifier que chacune des opérations

- $(p_1, p_2, v, T) \longmapsto \mathsf{subResNormalizedGcd}(p_1, p_2, v, T)$
- $(p_1, p_2, T) \longmapsto \mathsf{normalizedGcd}(p_1, p_2, T)$
- $(\{p_1, \ldots, p_s\}, T) \longmapsto \mathsf{normalizedGcd}(p_1, \ldots, p_s\}, T)$

satisfait ces spécifications.

Nous devons donc tout d'abord regarder ce qui se passe si chacune d'entre elles ne scinde pas T. Puis, en utilisant l'hypothèse d'induction, nous examinerons les cas où il y a scindage.

Commençons donc par les cas où T n'est pas scindé.

- Si subResNormalizedGcdElseSplit (p_1, p_2, v, T) ne scinde pas T, alors subResNormalizedGcd (p_1, p_2, v, T) satisfait bien les points (i) à (iv) grâce à la proposition V.8 précédente.
- Passons à la première des deux opérations normalizedGcd. On distingue les sous-cas suivants.
  - Si l'un des polynômes p_1 ou p_2 se réduit à une constante par rapport à T, que T soit scindé ou pas, les points (i) à (iv) sont vérifiés grâce à la proposition V.6.
  - Une fois initialement réduits, les polynômes p_1 et p_2 sont non-constants et de variables principales respectives v_1 et v_2. Notons que p_1 et p_2 engendrent toujours le même idéal dans $\mathbf{A}_T$. On distingue les sous-sous-cas suivants.
    - Si v_1 est algébrique par rapport à T et que v_2 ne l'est pas, on va appeler l'opération normalizedGcd à deux arguments avec des polynômes de variables principales inférieures à X_i et on conclut par récurrence (puisque l'on est ramené tôt ou tard à appeler soit subResNormalizedGcd soit normalize).
    - Si v_1 et v_2 sont algébriques par rapport à T, le polynôme p_2 va être normalisé et l'on est ramené au sous-sous-cas précédent. Notons que le couple (p_1, p_2) engendrent toujours le même idéal avant et après la normalisation de p_2.
    - Si v_1 et v_2 sont égales et non algébriques, alors subResNormalizedGcd (p_1, p_2, v, T) est appelé, ce qui est un cas que nous avons déjà traité. Notons que c'est le seul cas où l'opération normalizedGcd à deux arguments peut éventuellement retourner un résultant des *polynômes en v de départ.*
    - Si v_1 est non algébrique par rapport à T et différente de v_2 on conclut encore par récurrence.
- Passons à la seconde des deux opérations normalizedGcd. Que T soit scindé ou pas, on est ramené à utiliser l'opération normalizedGcd à trois arguments, cas que nous avons déjà examiné.

On suppose maintenant que T est scindé.

- Si subResNormalizedGcdElseSplit (p_1, p_2, v, T) retourne un scindage, alors on conclut en utilisant l'hypothèse d'induction pour l'opération normalizedGcd à trois arguments.
- On regarde maintenant l'opération normalizedGcd à trois arguments. On distingue les sous-cas suivants
  - Le cas où l'un des polynômes p_1 ou p_2 se réduit à une constante par rapport à T a déjà vu plus haut.

  - Si la valeur retournée provient de l'une des trois dernières instructions, on conclut en utilisant soit le cas précédent, soit l'hypothèse de récurrence.
  - Si la valeur retournée provient de l'instruction **return** S, il faut utiliser les propriétés de l'opération normalize et l'hypothèse d'induction.
- Enfin, le cas de l'opération normalizedGcd est immédiat

On peut avoir l'impression à la lecture de ce qui précède de n'avoir pas *montré grand chose* alors que nos trois algorithmes ont l'air *assez compliqué*. Cela tient au fait que les deux opérations normalizedGcd ne servent essentiellement qu'à faire ce qu'il faut pour pouvoir rappeler l'opération subResNormalizedGcd dans les conditions de la proposition V.8 Nous allons illustrer ces opérations. $\triangleleft$

Remarque V.14 Reprenons l'exemple des points singuliers de la courbe traité dans la remarque V.10. On pose :

$$p_1 = y^5 - x\ y^4 + \left(x^2 - x - 1\right)\ y^3 + \left(-x^3 + x^2 + x\right)\ y^2 + \left(-x^3 + x\right)\ y + x^4 - x^2$$

et :

$$p_2 = -y^4 + (2\ x - 1)\ y^3 + \left(-3\ x^2 + 2\ x + 1\right)\ y^2 + \left(-3\ x^2 + 1\right)\ y + 4\ x^3 - 2\ x$$

Avec l'ensemble triangulaire $T = \emptyset$, l'opération subResNormalizedGcd (p_1, p_2, y, T) retourne :

```
                18       17        16        15        14        13         12        11         10
  [[  val = 4x    + 8x    - 24x    - 40x    + 73x    + 74x    - 133x    - 48x    + 133x
 +
       9       8       7      6      5    4
  - 14x   - 59x   + 24x   + 7x   - 6x   + x, tower= {}]]
```

qui est en fait le résultant en y de ces polynômes. Sa partie sans carrés, soit f, définit l'extension algébrique séparable associée à l'ensemble de Lazard :

$$T\ =\ \left\{2\ x^6 - 5\ x^4 + 2\ x^3 + 2\ x^2 - x\right\}$$

Maintenant subResNormalizedGcd (p_1, p_2, y, T) retourne :

```
             3
   [[val= y   - y,tower= {x}], [val= y - 1,tower= {x - 1}],

              2         3      2             3      2
     [val= y   + (- 4x   - 8x   + 4)y + 4x   + 7x   - 2x - 3,
                  4      3      2
      tower= {2x   + 2x   - 3x   - x + 1}]]
```

Détaillons les calculs qui ont conduit à ce résultat. Avant cela, notons qu'il est clair que :

$$S\ =\ \left\{\left\{2\ x^4 + 2\ x^3 - 3\ x^2 - x + 1\right\},\ \{x - 1\},\ \{x\}\right\}$$

soit un scindage de Lazard de T. Désignons par S_1, S_2, S_3 les trois composantes de ce scindage dans l'ordre dans lequel elles sont données ci dessus.

Vérifions maintenant le résultat produit par subResNormalizedGcd (p_1, p_2, y, T). Remarquons tout d'abord que p_1 et p_2 sont réduits et normalisés par rapport à T. Le pseudo-reste de p_1 par p_2 est :

$$p_3 \;=\; (-2\;x+1)\;y^3+\left(-4\;x^3+3\;x^2\right)\;y^2+\left(3\;x^2-1\right)\;y+5\;x^4-4\;x^3-3\;x^2+2\;x$$

Sa normalisation donne :

```
[[  val =
          3        5     4      3     2       2
         y  + (- 8x  - 4x  + 18x  + 3x  - 8x)y
       +
            5     4      3    2               5    4     3    2
         (8x  + 4x  - 18x  - x  + 6x - 1)y - 2x  - x  + 2x  + x
     ,
               6     4     3     2
     tower= {2x  - 5x  + 2x  + 2x  - x} ]]
```

Pour obtenir ce polynôme normalisé par rapport à T, on a calculé l'inverse de l'initial $-2\;x+1$ de p_3 dans $\mathbf{A}_T$. Ceci se fait en calculant un pgcd étendu (c'est-à-dire avec coefficients de Bezout) des polynômes $-2\;x+1$ et $f = -8\;x^6+20\;x^4-8\;x^3-8\;x^2+4\;x$. En fait, seul le coefficient de Bezout de $-2\;x+1$ est utile. On trouve une relation de la forme :

$$(-256\;x^5-128\;x^4+576\;x^3+32\;x^2-240\;x+8)\;\mathsf{init}(p_3)+\cdots f \quad = \quad 8$$

Désignons par q le coefficient de Bezout de $\mathsf{init}(p_3)$ et par r le pgcd (égal à 8) ci-dessus. En réduisant par rapport à T, le polynôme $r\;\mathsf{mmon}(p_3)\;+\;q\;\mathsf{tail}(p_3)$ (voir l'algorithme de normalizeElseSplit dans la proposition V.5) on obtient le normalisé de p_3, que l'on dénote encore par p_3. Puis, le pseudo-reste de p_2 par p_3 est p_4 égal à :

```
          10       9        8        7        6        5        4       3      2  2
   (- 64x   - 64x  + 272x  + 192x  - 412x  - 164x  + 243x  + 42x  - 49x )y
 +
        10       9        8        7        6        5        4       3      2
   (64x   + 64x  - 272x  - 176x  + 404x  + 118x  - 214x  - 22x  + 36x )y
 +
         10       9       8       7       6       5      4       3    2
   - 16x   - 16x  + 48x  + 40x  - 41x  - 32x  + 8x  + 12x  + x  - 2x
```

Pour normaliser ce polynôme par rapport à T, on procède comme précédemment. Cependant, on trouve cette fois-ci une relation de la forme :

$$(10048\;x^8+\cdots)\;\mathsf{init}(p_4)+\cdots f \quad = \quad 16\;x$$

Par conséquent, $\mathsf{init}(p_4)$ n'est pas inversible dans $\mathbf{A}_T$ sur sa composante où est x est nulle. Il faut donc scinder T comme suit :

$$T \;\longmapsto\; \left\{\left\{2\;x^5-5\;x^3+2\;x^2+2\;x-1\right\},\;\{x\}\right\}$$

Bref, normalize(p_4, T) retourne :

```
            2      4      3      2                    4      3      2
[ [val= 2y   + (30x   + 22x   - 61x   - 15x + 23)y - 26x   - 18x   + 53x   + 9x - 19
    ,
                5     3     2
   tower= {2x   - 5x   + 2x   + 2x - 1}],  [val= 0,tower= {x}]]
```

Laissons provisoirement de côté la tour définie par $\{x\}$ et désignons encore par p_4 le polynôme en y^2 normalisé par rapport à $\{2\ x^5 - 5\ x^3 + 2\ x^2 + 2\ x - 1\}$. Le pseudo-reste de p_3 par p_4 est p_5 égal à :

```
            9        8       7        6        5        4        3        2
      480x  + 1492x  - 560x  - 4876x  - 760x  + 5951x  + 1092x  - 3069x
    +
      - 316x + 563
  *
    y
 +
          9        8       7        6       5        4       3        2
    - 416x  - 1276x  + 528x  + 4152x  + 468x  - 5019x  - 690x  + 2505x  + 188x
 +
    - 437
```

L'opération normalize$(p_5, \{2\ x^5 - 5\ x^3 + 2\ x^2 + 2\ x - 1\})$ donne :

```
                                                   4     3     2
[[val= y - 1,tower= {x - 1}],[val= 0,tower= {2x  + 2x  - 3x  - x + 1}]]
```

Laissons provisoirement de côté la tour définie par $\{2\ x^4 + 2\ x^3 - 3\ x^2 - x + 1\}$ et désignons encore par p_5 le polynôme $y - 1$ normalisé par rapport à $\{x - 1\}$. Le pseudo-reste de p_4 par p_5 est $4\ x^4 + 4\ x^3 - 8\ x^2 - 6\ x + 6$ qui se réduit à 0 par $x - 1$.

Bref, si $x = 1$, les polynômes p_1 et p_2 de départ ont $y - 1$ pour pgcd. En reprenant les calculs modulo S_1 et S_3, on trouve finalement le résultat annoncé plus haut.

Notons que nous n'avons pas suivi exactement l'algorithme donné dans la proposition V.8 pour calculer subResNormalizedGcd (p_1, p_2, y, T) avec $T = \{2\ x^6 - 5\ x^4 + 2\ x^3 + 2\ x^2 - x\}$. En fait, nous avons prise égale au contenu de p_3 (en tant que polynôme de $\mathbf{Q}[x, y]$) la quantité β par laquelle on divise p_3 dans l'algorithme. Avec ce choix, les spécifications de sortie de l'algorithme sont encore valables si celui ne retourne pas un résultant. Cependant, ce choix n'a pour intérêt que d'allèger l'exposé des calculs dans notre exemple. En règle général, c'est une mauvaise stratégie (voir [DST93]).

Nous reviendrons sur cet exemple plus loin. En effet, il nous reste à retrouver le scindage de Lazard de la variété associée à $\{p_1, p_2\}$ vue dans la remarque V.10.

Remarque V.15 Nous reprenons maintenant l'exemple de la remarque V.9. Nous nous intéressons aux polynômes :

$$p_1 = \left(t\ z - t^2 - t\right)\ y - z + a\ t + a \text{ et } p_2 = \left(z^2 - t\ z + 1\right)\ y + (-t + a)\ z - a$$

Leur résultant en y, soit r, définit un ensemble de Lazard :

$$T = \left\{z^3 + \left(-t^2 - t - a\right)\ z^2 + \left(t^3 + t^2 - a\ t + 1\right)\ z + a\ t^2 - a\right\}$$

Les polynômes $\overline{p_1}^{\widehat{T}}$ et $\overline{p_2}^{\widehat{T}}$ ont pour pgcd normalisé :

```
  [ val =
                    4        2       3        2      2
          ((a + 1)t  + (- a  + 2)t  - 4a t  + 4a t)y
        +
                    2        2                2  2
          ((a + 1)t  + (- a  - 2a)t + 2a )z
        +
                         4      2       3         2       2     3    2            3
              (- a - 1)t  + (a  + 2a)t  + (- 3a  + 1)t  + (a  + a  - 2a)t - 2a
            +
                2
              2a
         *
            z
        +
                2       3       3      2       2        3      2       3
          (- 2a  - 2a)t  + (2a  + 3a  - a)t  + (- 3a  + 2a )t - 2a
      ,
               3       2          2     3    2                  2
     tower= {z  + (- t  - t - a)z  + (t  + t  - a t + 1)z + a t  - a}]]
```

Pour obtenir ce pgcd, on normalise les polynômes p_1 et p_2 par rapport à T et l'on vérifie que le normalisé de p_2 réduit à 0 par pseudo-division le normalisé de p_1. Le pgcd ci-dessus est donc en fait le normalisé de p_2 par rapport à T. Comme on le verra plus loin, pour normaliser p_i par rapport à T, on calcule un pgcd étendu (i.e. avec coefficients de Bezout) de $\mathsf{init}(p_i)$ et du résultant r précédent vus comme polynômes de $\mathbb{Q}(a,t)[z]$. En fait, seul le coefficient de Bezout de $\mathsf{init}(p_i)$ est nécessaire. Ainsi, pour p_1, on a une relation de la forme :

$$(t^2\ z^2 + \cdots)\ \mathsf{init}(p_1) + \cdots\ r \quad = \quad (a-1)\ t^5 + (3\ a - 3)\ t^4 + (2\ a - 2)\ t^3$$

Puis, pour p_2, on a une relation de la forme :

$$((t - 2\ a)\ z^2 + \cdots)\ \mathsf{init}(p_2) + \cdots\ r \quad = \quad (a+1)\ t^4 + \left(-a^2 + 2\right)\ t^3 - 4\ a\ t^2 + 4\ a^2\ t$$

Pour retrouver le scindage de Lazard de $\mathbf{V}(p_1, p_2)$ donné dans la remarque V.9 on doit encore *faire des calculs*. C'est ce que nous allons voir avec les propositions et algorithmes suivants.

Proposition V.9 *Soit $T \subseteq \mathbf{R}_n$ un ensemble de Lazard. Soit p un polynôme non constant de $\mathbf{R}_n$ normalisé par rapport à T, et de variable principale v telle que : $v >$ $\mathsf{mvar}(T)$. Alors l'opération* squareFreePart(p, T) *retourne un ensemble $\{[f_1, S_1], \ldots, [f_l, S_l]\}$ où les f_i sont des polynômes de $\mathbf{R}_n$ et les S_i sont des ensembles Lazard tel que si $\mathbf{A}_i$ désigne la tour associée à S_i on ait pour chaque i :*

(i) $\overline{f_i}^{\widehat{S_i}}$ *est unitaire et a pour variable principale v*

(ii) f_i *est normalisé par rapport à S_i*

(iii) *le polynôme* $\overline{f_i}^{\widehat{S_i}}$ *de $\mathbf{A}_i[v]$ est une partie sans carrés de* $\overline{p}^{\widehat{S_i}}$

(iv) $\{S_1, \ldots, S_l\}$ *est un scindage de Lazard de T avec $T \prec_{alg} S_i$ pour $1 \leq i \leq l$ si $l \geq 2$.*

squareFreePart(p, T) ==
  mdeg$(p) = 1$ => **return** $\{[p, T]\}$
  $v :=$ mvar(p)
  $q := \frac{\partial p}{\partial v}$
  $\gamma :=$ subResNormalizedGcd (p, q, v, T)
  $S := \{\}$
  for $[g_i, S_i] \in \gamma$ **repeat**
    $p_i :=$ reduce(p, S_i)
    if deg$(g_i, v) > 0$
      then
        $h_i :=$ lazyPquo$(p_i,$ reduce$(g_i, S_i))$
        $f_i :=$ reduce(h_i, T)
    $S := S \cup \{[f_i, S_i]\}$
  return S

Preuve ▷ Nous utiliserons les notations de l'algorithme. En particulier, nous désignerons par $\{S_1, \ldots, S_s\}$ le scindage de T calculé par subResNormalizedGcd (p, q, v, T), par g_i le pgcd normalisé de p et q dans $\mathbf{A}_i[v]$ et par p_i le polynôme reduce(p, S_i). Vérifions le point (i). Puisque p est normalisé par rapport à T, son initial est inversible dans la tour d'extensions simples séparables $\mathbf{A}_T$ (proposition V.6). Par conséquent, l'initial de p n'appartient à aucun des idéaux premiers associés à sat(T) (proposition III.18 p. 103). Or d'après la proposition III.22 p. 111 on a :

$$\mathsf{sat}(T) \ = \ \cap_1^l \ \mathsf{sat}(S_i)$$

Donc init(p) n'appartient à aucun des idéaux premiers associés à l'un des sat(S_i). Par conséquent (proposition III.18 p. 103) init(p) est inversible dans chaque $\mathbf{A}_i$. Le même raisonnement est valable pour chacun des initiaux itérés de p. Par suite, la réduction de p par chacun des S_i n'annule aucun des initiaux itérés de p, seules leurs queues sont éventuellement réduites. On distingue alors deux cas.

- Soit deg$(g_i, v) = 0$. Alors, $\overline{f_i}^{\widehat{S_i}}$ est unitaire et mvar$(\overline{f_i}^{\widehat{S_i}}) = v$ d'après ce qui précède.
- Soit deg$(g_i, v) > 0$. Alors, $\overline{h_i}^{\widehat{S_i}}$ est unitaire puisque $\overline{p_i}^{\widehat{S_i}}$ et $\overline{g}^{\widehat{S_i}}$ sont unitaires et que l'on a :

$$\overline{p_i}^{\widehat{S_i}} = \overline{h_i \, g}^{\widehat{S_i}}$$

D'autre part comme :

$$\mathsf{deg}(g_i, v) \leq \mathsf{deg}(q, v) \text{ et } \mathsf{deg}(q, v) < \mathsf{deg}(p, v)$$

le polynôme $\overline{h_i}^{\widehat{S_i}}$ est de degré non nul en v. Bref, le polynôme $\overline{\mathsf{reduce}(h_i)}^{\widehat{S_i}} = f_i$ est unitaire et a pour variable principale v

On en déduit (i). Le point (iv) résulte du théorème V.1. Le point (iii) résulte de ce même théorème et de la proposition III.5. Enfin, nous admettrons pour le moment le point (ii). De toute manière, on peut remplacer $\{[f_i, S_i]\}$ par $\mathsf{normalize}(f_i, S_i)$ dans la dernière ligne de l'algorithme, puisque $\overline{f_i}^{\widehat{S_i}}$ est unitaire. $\triangleleft$

Proposition V.10 *Soit $T \subseteq \mathbf{R}_n$ un ensemble de Lazard. Soit p un polynôme non constant de $\mathbf{R}_n$ normalisé par rapport à T, et de variable principale $v = X_i$ telle que : $v >$ $\mathsf{mvar}(T)$. On suppose que $\mathbf{R}$ est un anneau factoriel effectif (i.e. avec un algorithme de pgcd). Alors l'opération* $\mathsf{primitivePart}(p, T)$ *retourne un ensemble $[[p_1, S_1], \ldots, [p_l, S_l]]$ où les p_i sont des polynômes de $\mathbf{R}_n$ et les S_i sont des ensembles de Lazard tel que si $\mathbf{A}_i$ désigne la tour associée à S_i on ait pour chaque i :*

(i) *p_i est normalisé et a pour variable principale v*

(ii) *p_i est primitif en tant que polynôme $\mathbf{A}_{\,{S_i}_{\bar{X}_j}}[X_j, \ldots, X_i]$ pour tout $1 \leq j \leq i$.*

(iii) $\langle \overline{p}^{\widehat{S_i}} \rangle_{\mathbf{A}_i[v]} = \langle \overline{p_i}^{\widehat{S_i}} \rangle_{\mathbf{A}_i[v]}$

(iv) *$\{S_1, \ldots, S_l\}$ est un scindage de Lazard de T avec $T \prec_{alg} S_i$ pour $1 \leq i \leq l$ si $l \geq 2$.*

$\mathsf{primitivePart}(p, T) ==$
  $p = 0 =>$ **return** $\{[p, T]\}$
  $p \in \mathbf{R} =>$ **return** $\{[1, T]\}$
  $\mathsf{var}(p) \cap \mathsf{var}(T) = \emptyset =>$ **return** $\{[\mathsf{primitivePart}_{\mathbf{R}_{i-1}[X_i]}(p), T]\}$
  $S := \{\}$
  $v := \mathsf{mvar}(p)$
  $\gamma := \mathsf{normalizedGcd}\ (\mathsf{coefficients}(p, v), T)$
  for $[c, T] \in \gamma$ **repeat**
    $f := \mathsf{reduce}(\mathsf{lazyPquo}(\mathsf{reduce}(p, T), \mathsf{reduce}(c, T)), T)$
    $S := S \cup \mathsf{primitivePart}(f, \emptyset)$
  return S

où :

- $\mathsf{primitivePart}_{\mathbf{R}_{i-1}[X_i]}$ désigne la partie primitive *usuelle* des polynômes univariés en X_i à coefficients dans l'anneau factoriel effectif $\mathbf{R}_{i-1}$
- $\mathsf{normalizedGcd}$ est l'opération à deux aguments définie dans la preuve du théorème V.1

3 Décomposition des variétés

Remarque V.16 Rappelons tout d'abord que l'on désigne par $\mathbf{K}$ une clôture algébrique de $\mathbf{k}$. Dans cette section, nous présentons un algorithme de décomposition d'une variété affine au sens du théorème III.11 p. 111. Ainsi, étant donné un ensemble (fini) de polynômes $F \subseteq \mathbf{R}_n$ on va décrire une opération solve qui à F fait correspondre une décomposition de Lazard sans redondances de $\mathbf{V}_{\mathbf{K}}(F)$. Elle repose sur une opération $\mathsf{intersect}$ qui pour un polynôme $p \in \mathbf{R}_n$ et un ensemble de Lazard $T \subseteq \mathbf{R}_n$ retourne une décomposition de Lazard de :

$$\mathbf{V}(p) \cap \mathbf{W}(T)$$

Afin de décrire notre algorithme, nous aurons besoin de la *convention* suivante.

Définition V.3 *Soit* $\mathsf{decompose}$ *une opération qui pour un polynôme* $p \in \mathbf{R}_n$ *et un ensemble de Lazard* $T \subseteq \mathbf{R}_n$ *retourne une décomposition de Lazard de :*

$$\mathbf{V}(p) \cap \mathbf{W}(T)$$

On associe à cette opération, une opération de même nom et, qui pour un ensemble fini F de polynômes de $\mathbf{R}_n$ et un ensemble fini S d'ensembles de Lazard de $\mathbf{R}_n$, retourne un ensemble S' d'ensembles de Lazard de $\mathbf{R}_n$ à l'aide de l'algorithme suivant :

```
decompose(F, S) ==
    F = ∅ => return S
    R := {[F, T] for T ∈ S}
    S' := ∅
    for [F = {p_1, ..., p_s}, T] ∈ R repeat
            if s = 1
                then
                    S' := S' ∪ decompose(p_1, T)
                else
                    F := {p_2, ..., p_s}
                    D := decompose(p_1, T)
                    R := R ∪ {[F, T] for T ∈ D}
    return S'
```

Proposition V.11 *Soit $F \subseteq \mathbf{R}_n$ un ensemble fini de polynômes et $S = \{S_1, \ldots, S_l\}$ un ensemble d'ensembles de Lazard de $\mathbf{R}_n$. Alors avec les hypothèses et notations de la définition précédente,* $\mathsf{decompose}(F, S)$ *retourne une décomposition de Lazard de :*

$$\mathbf{V}(F) \cap (\cup_1^l \mathbf{W}(S_j)$$

Preuve $\triangleright$ Si F est vide, l'algorithme est correct grâce à la convention : $\mathbf{V}_{\mathbf{K}}(\emptyset) = \mathbf{K}$. Si F est un singleton $\{p_1\}$, l'algorithme retourne $\cup_1^l \mathsf{decompose}(p_1, S_i)$. Posons :

$$\mathsf{decompose}(p_1, S_i) = \{S_{i,1}, \ldots, S_{i,l_i}\}$$

On a donc pour chaque i :

$$\mathbf{V}(p_1) \cap \mathbf{W}(S_i) \ \subseteq \ \cup_1^{l_i} \mathbf{W}(S_{i,j}) \ \subseteq \ \overline{\mathbf{V}(p_1) \cap \mathbf{W}(S_i)}$$

La fermeture commutant avec la réunion finie, il vient :

$$\mathbf{V}(p_1) \cap (\cup_1^{l} \mathbf{W}(S_i)) \ \subseteq \ \cup_{i,j} \mathbf{W}(S_{i,j}) \ \subseteq \ \overline{\mathbf{V}(p_1) \cap (\cup_1^{l} \mathbf{W}(S_i))}$$

On pose maintenant $F = \{p_1, \ldots, p_s\}$ avec $s > 1$. On peut supposer que l'algorithme calcule une décomposition de Lazard $\{T_1, \ldots, T_k\}$ de :

$$\mathbf{V}(p_2, \ldots, p_s) \cap (\cup_1^{l} \mathbf{W}(S_j))$$

On a donc :

$$\mathbf{V}(p_2, \ldots, p_s) \cap (\cup_1^{l} \mathbf{W}(S_j)) \ \subseteq \ \cup_i \mathbf{W}(T_i) \ \subseteq \ \overline{\mathbf{V}(p_2, \ldots, p_s) \cap (\cup_1^{l} \mathbf{W}(S_j))}$$

Or :

$$\mathbf{V}(p_1, \ldots, p_s) \cap (\cup_1^{l} \mathbf{W}(S_i)) \ = \ \mathbf{V}(p_1) \cap (\mathbf{V}(p_2, \ldots, p_s) \cap (\cup_1^{l} \mathbf{W}(S_i)))$$

D'où :

$$\mathbf{V}(p_1, \ldots, p_s) \cap (\cup_1^{l} \mathbf{W}(S_j)) \ \subseteq \ \mathbf{V}(p_1) \cap (\cup_i \mathbf{W}(T_i)) \ \subseteq \ \overline{\mathbf{V}(p_1, \ldots, p_s) \cap (\cup_1^{l} \mathbf{W}(S_j))}$$

D'autre part, on peut vérifier :

$$\mathsf{decompose}(\{p_1, \ldots, p_s\}, S) \ = \ \mathsf{decompose}(p_1, \mathsf{decompose}(\{p_2, \ldots, p_s\}, S))$$

Posons : $\mathsf{decompose}(p_1, T_i) = \{S_{i,1}, \ldots, S_{i,l_i}\}$ D'après le cas $s = 1$, il vient :

$$\mathbf{V}(p_1) \cap (\cup_1^{l} \mathbf{W}(T_i)) \ \subseteq \ \cup_{i,j} \mathbf{W}(S_{i,j}) \ \subseteq \ \overline{\mathbf{V}(p_1) \cap (\cup_1^{l} \mathbf{W}(T_i))}$$

Par suite, en utilisant les propriétés de la fermeture, on obtient :

$$\mathbf{V}(p_1, \ldots, p_s) \cap (\cup_1^{l} \mathbf{W}(S_j)) \ \subseteq \ \cup_{i,j} \mathbf{W}(S_{i,j}) \ \subseteq \ \overline{\mathbf{V}(p_1, \ldots, p_s) \cap (\cup_1^{l} \mathbf{W}(S_j))}$$

Ainsi, l'algorithme $\mathsf{decompose}(F, S)$ satisfait la spécification annoncée. $\triangleleft$

Remarque V.17 Le théorème V.2 suivant donne un l'algorithme pour l'opération intersect présentée dans la remarque précédente. Cet algorithme fait appel à deux sous-procédures algebraicIntersect et transcendentalIntersect. dont les spécifications sont les suivantes. Soit $p \in \mathbf{R}_n$ et $T \subseteq \mathbf{R}_n$ un ensemble de Lazard. On pose $q = \mathsf{iRed}(p, T)$ et on suppose $q \notin \mathbf{R}$ (le cas $q \in \mathbf{R}$ étant traité par intersect).

- si $\mathsf{mvar}(q)$ est algébrique par rapport à T, alors $\mathsf{algebraicIntersect}(p, T)$ retourne une décomposition de Lazard de $\mathbf{V}(p) \cap \mathbf{W}(T)$,
- si $\mathsf{mvar}(q)$ n'est pas algébrique par rapport à T, alors $\mathsf{transcendentalIntersect}(p, T)$ retourne une décomposition de Lazard de $\mathbf{V}(p) \cap \mathbf{W}(T)$.

Dans l'algorithme original de Daniel Lazard ([Laz91a]) la procédure intersect se réduit à notre sous-procédure transcendentalIntersect. De plus, les instructions encadrées n'y figurent pas.

Remarque V.18 L'algorithme pour l'opération intersect présenté dans le théorème V.2 est récursif. En effet, ses sous procédures algebraicIntersect et transcendentalIntersect font appel à deux autres opérations *semblables* à intersect et qui appellent intersect.

La première de ces deux opérations, augment, a la même spécification qu'intersect à ceci-près que le polynôme passé comme premier argument doit êtue normalisé par rapport à l'ensemble de Lazard passé comme second argument. Cette opération est en fait une sous-procédure de intersect. C'est elle qui gère la discussion $\mathsf{init}(p) = 0$, $\mathsf{init}(p) \neq 0$ (dans $\mathbf{W}(T)$). De plus, dans le second cas, elle calcule une partie sans carrés puis une partie primitive p par rapport à T au sens des spécifications des opérations $\mathsf{squareFreePart}(p, T)$ (proposition V.9) et $\mathsf{primitivePart}(p, T)$ (proposition V.10). Ceci permet de *rajouter* la relation $p = 0$ dans la tour $\mathbf{A}_T$ associée à T.

La seconde opération, extend, pour un polynôme $p \in \mathbf{R}_n$ et un ensemble de Lazard $T \subseteq \mathbf{R}_n$, tels que : $p \notin \mathbf{R}$ et : $\mathsf{mvar}(p) > \mathsf{mvar}(T)$ associe au couple (p, T) un scindage de Lazard de :

$$(\mathbf{V}(p) \setminus \mathbf{V}(\mathsf{init}(p))) \cap \mathbf{W}(T)$$

En particulier, si l'initial de p est inversible dans la tour $\mathbf{A}_T$ associée à T, l'opération $\mathsf{extend}(p, T)$ décrit l'extension de $\mathbf{A}_T$ par $\overline{p}^{\widehat{T}}$. Avec les mêmes hypothèses sur p et T, il n'en va pas de même pour $\mathsf{augment}(p, T)$ comme le montre l'exemple suivant. Nous revenons à l'exemple de l'introduction de ce chapitre. Considérons le polynôme :

$$p = \left(t^{93} - t^{90}\right)\, x - t^{90}\, y - t^{89}\, z^2$$

et l'ensemble de Lazard :

$$T = \left\{z^5 - t^4,\ t^3\, y^2 + 2\, t^2\, z^2\, y + \left(-t^6 + 2\, t^3 + t - 1\right)\, z^4\right\}$$

On pose :

$$S_3 = \left\{z^5 - t^4,\ t^3\, y^2 + 2\, t^2\, z^2\, y + \left(-t^6 + 2\, t^3 + t - 1\right)\, z^4,\ \left(t^4 - t\right)\, x - t\, y - z^2\right\}$$

et :

$$S_2 = \left\{t^3 - 1,\ z^5 - t,\ y + t^2\, z^2\right\}$$

On a :

$$\mathsf{augment}(p, T) = \{S_2, S_3\}$$

tandis que :

$$\mathsf{extend}(p, T) = \{S_3\}$$

Notons que :

$$\mathsf{normalize}(\mathsf{init}(p), S_2) = \{[0, S_2]\}$$

tandis que :

$$\mathsf{normalize}(\mathsf{init}(p), S_3) = \{[\mathsf{init}(p), S_3]\}$$

Remarque V.19 On effectue pour l'opération extend la même convention que celle faite pour l'opération intersect plus haut dans la remarque V.16. Plus précisément, étant donnés un ensemble de polynômes $F = \{p_1, \ldots, p_s\} \subseteq \mathbf{R}_n$ et une famille $S = (S_1, \ldots, S_m)$ d'ensembles de Lazard tel que : $\mathsf{mvar}(p_s) > \cdots > \mathsf{mvar}(p_1) > \mathsf{mvar}(S_j)$, on pose :

$\mathsf{extend}(F, S) ==$
    $F = \emptyset$ => **return** S
    $R := \{[F, T]$ **for** $T \in S\}$
    $S := \emptyset$
    for $[F = \{p_1, \ldots, p_s\}, T] \in R$ **repeat**
        if $s = 1$
            then
                $S := S \cup \mathsf{extend}(p_1, T)$
            else
                $F := \{p_2, \ldots, p_s\}$
                $D := \mathsf{extend}(p_1, T)$
                $R := R \cup \{[F, T]$ **for** $T \in D\}$
    return S

Ainsi, $\mathsf{extend}(F, S)$ retourne un scindage de Lazard de

$$(\mathbf{V}(F) \setminus \mathbf{V}(\mathsf{init}(p_1) \cdots \mathsf{init}(p_s))) \cap (\cup_1^l \mathbf{W}(S_j)$$

La preuve est analogue à celle de la proposition V.11.

Théorème V.2 *Soit $T \subseteq \mathbf{R}_n$ un ensemble de Lazard et $p \in \mathbf{R}_n$ un polynôme. Alors on a :*

(*i*) *l'opération* $\mathsf{initIntersect}(p, T)$ *retourne un scindage de Lazard de :*

$$(\mathbf{V}(p) \cap \mathbf{V}(\mathsf{init}(p))) \cap \mathbf{W}(T)$$

(*ii*) *Si p est normalisé par rapport à T, alors l'opération* $\mathsf{augment}(p, T)$ *retourne un scindage de Lazard de :*

$$\mathbf{V}(p) \cap \mathbf{W}(T)$$

(*iii*) *l'opération* $\mathsf{intersect}(p, T)$ *retourne un scindage de Lazard de :*

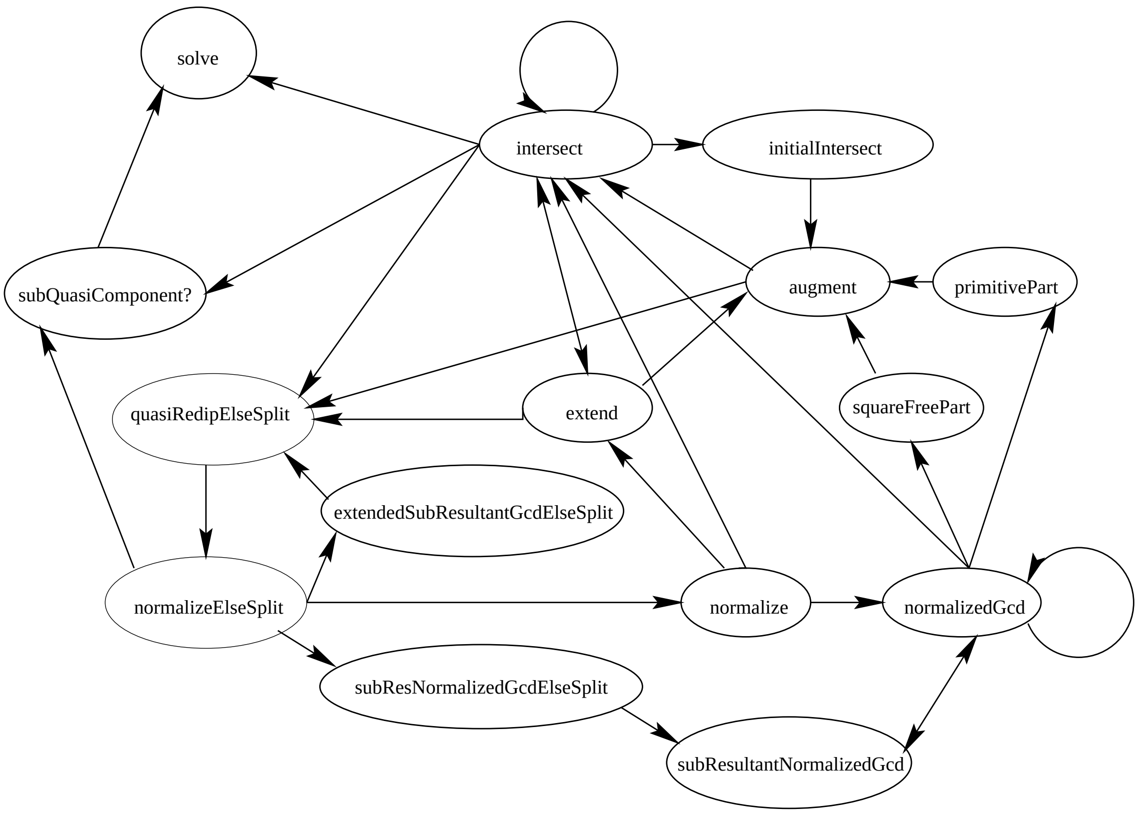
$$\mathbf{V}(p) \cap \mathbf{W}(T)$$

(*iv*) *Si $p \notin \mathbf{R}$ et $\mathsf{mvar}(p) > \mathsf{mvar}(T)$, alors l'opération* $\mathsf{extend}(p, T)$ *retourne un scindage de Lazard de :*

$$(\mathbf{V}(p) \setminus \mathbf{V}(\mathsf{init}(p))) \cap \mathbf{W}(T)$$

Remarque V.20 Nous donnons sur deux des pages suivantes des algorithmes pour :

- les opérations spécifiées dans le théorème V.2 ci-dessus et
- l'opération quasiRecipElseSplit de la proposition V.4 p. 139.



Le diagramme ci-dessus montre les *liens* entre les différents algorithmes de ce chapitre. Les algorithmes solve et subQuasiComponent ? seront abordés dans la remarque V.29 p. 179. Plus précisément, il s'agit d'un graphe orienté dont les sommets sont les algorithmes et tel qu'un arc va d'un algorithme a_1 vers un algorithme a_2 si dans a_2 un appel est fait à a_1. Ce graphe comporte de nombreux cycles. Nous vérifierons cependant que toute exécution de l'un de ces algorithmes se termine. Nous ne donnerons pas une preuve complète des algorithmes du théorème V.2 et de la proposition V.4, mais simplement quelques remarques pour aider à leur lecture. Ces remarques constituent une preuve de la validité et de la terminaison de L'algorithme original de Lazard ([Laz91a]). En effet, nous passerons en revue les opérations :

$$\begin{array}{llll} \bullet(p,T) \longmapsto \mathsf{initIntersect}(p,T) & & \bullet(p,T) \longmapsto \mathsf{augment}(p,T) \\ \bullet(p,T) \longmapsto \mathsf{intersect}(p,T) & & \bullet(p,T) \longmapsto \mathsf{extend}(p,T) \\ \bullet(p,T) \longmapsto \mathsf{quasiRecipElseSplit}(p,T) & & \end{array}$$

et nous établirons que :

- ces 5 algorithmes satisfont leurs spécifications dans les cas $n = 1$ (remarque V.24)
- chaque exécution de chacun de ces 5 algorithmes se termine (proposition V.12)
- ces 5 algorithmes satisfont leurs spécifications dans le cas général (remarques V.22, V.23 et V.26).

```
initIntersect(p, T) ==
    init(p) ∈ R => return ∅
    tail(p) ∈ reg(R) => return ∅
    tail(p) = 0 => return intersect(init(p), T)
    return intersect(tail(p), intersect(init(p), T))

augment(p, T) ==
    p = 0 => return {T}
    p ∈ R => ∅
    R := {[p, T]} ; S := {}
    v := mvar(p)
    U := T_v^+
    φ := squareFreePart(p, T_v^-)
    R := R ∪ φ
    π := {}
    for [f, T] ∈ φ repeat
        π := π ∪ primitivePart(f, T)
    R := R ∪ π
    for [f, T] ∈ π repeat
        T := T ∪ {f}
        T := extend(U, T)
        S := S ∪ {T}
    for [p, T] ∈ R repeat
        S := S ∪ initIntersect(p, T)
    return S

extend(p, T) ==
    p = 0 => {T}
    p ∈ R => ∅
    init(p) ∈ R => intersect(p, T)
    iRed(init(p), T) = 0 => ∅
    S := ∅
    for U_i ∈ intersect(p, T) repeat
        for [n_i, W_i] ∈ normalize(init(p), U_i) repeat
            n_i ≠ 0 => S := S ∪ {W_i}
    return S

intersect(p, T) ==
    q := iRed(p, T)
    q = 0 => {T}
    q ∈ R => ∅
    v := mvar(q)
    algebraic?(v, T) => algebraicIntersect(p, T)
    transcendentalIntersect(p, T)
```

```
algebraicIntersect(p, T) ==
    q := iRed(p, T)
    v := mvar(q)
    S := ∅
    for [g_i, T_i] ∈ normalizedGcd(q, T_v, T_v^-) repeat
        g_i ∈ R => iterate
        for U_i ∈ augment(g_i, T_i) repeat
            mvar(U_i) < v =>
                    L := extend(T_v, U_i)
                    L := intersect(p, L)
                    S := S ∪ extend(T_v^+, L)
            S := S ∪ extend(T_v^+, U_i)
    return S

transcendentalIntersect(p, T) ==
    q := iRed(p, T)
    v := mvar(q)
    S := ∅; R := [[q, T]]
    for [q, T] ∈ R repeat
        q := iRed(q, T)
        for [n_i, T_i] ∈ normalize(q, T) repeat
            n_i = 0 =>
                    S := S ∪ {T_i}
            n_i ∉ R =>
                    for U_i ∈ augment(n_i, T_i) repeat
                        [mvar(U_i) = v =>]
                                [S := S ∪ {U_i}]
                        R := cons([p, U_i], R)
    return S

quasiRecipElseSplit(p, T) ==
    v := mvar(p)
    ε := extendedSubResGcdElseSplit (p, T_v, T_v^-)
    split?(ε) => return extend(T_v^+ ∪ {T_v}, ε)
    [f, h, g] := ε
    deg(g, v) > 0 =>
            q := lazyPquo(T_v, g)
            S := ∅
            for U_i ∈ augment(g, T_v^-) ∪ intersect(q, T_v^-) repeat
                    mvar(U_i) = v =>
                            S := S ∪ extend(T_v^+, U_i)
                    S := S ∪ extend(T_v^+ ∪ {T_v}, U_i)
            return S
    return (f, g)
```

Remarque V.21 Afin de présenter le principe général de la méthode de Lazard, nous commençons l'étude de ces algorithmes en donnant une version ramassée de l'algorithme $\mathsf{intersect}$ du théorème V.2. Cet $\mathsf{intersect}$ compacté, mais nécessairement moins précis, est équivalent à l'$\mathsf{intersect}$ de [Laz91a]. Il est par conséquent moins optimisé que l' $\mathsf{intersect}$ dont l'algorithme est donné précédemment. D'autre part, il se raconte sans $\mathsf{augment}$ qui est en fait caché dedans.

Soit $T \subseteq \mathbf{R}_n$ un ensemble de Lazard et $p \in \mathbf{R}_n$ un polynôme. Les spécifications de cet $\mathsf{intersect}(p, T)$ compacté sont exactement celles décrites dans le théorème V.2. Certaines étapes du pseudo-algorithme suivant sont étiquetées $\boxed{\triangle \textbf{ Scindages possibles.}}$ Cela signifie que, pour des polynômes $p_1, \ldots, p_n$ de $\mathbf{R}_n$, une opération $\mathsf{computeElseSplit}(p_1, \ldots, p_s, T)$ devant a prioiri retourner des polynômes $q_1, \ldots, q_t$ de $\mathbf{R}_n$ retourne en fait un scindage de Lazard $\{S_1, \ldots, S_l\}$ de T. On doit alors dupliquer l'exécution du pseudo-algorithme suivant en remplaçant $\mathsf{computeElseSplit}(p_1, \ldots, p_s, T)$ par $\mathsf{computeElseSplit}(p_1, \ldots, p_s, S_i)$ dans la $i-$*ième* copie, pour chaque $i \in \{1, \ldots, l\}$.

(l_1) Si p est normalisé par rapport à T, alors allez en (l_2) avec $r = p$, sinon passez à ($l_1{}'$).

($l_1{}'$) Si p n'est pas normalisé par rapport à T, calculez deux polynômes $q, r \in \mathbf{R}_n$ tels que r soit normalisé par rapport à T avec

$$\mathsf{modulo}(pq - r, T) = 0 \ \text{ et } \ \mathsf{modulo}(p, T) = 0 \iff \mathsf{modulo}(r, T) = 0$$

$\boxed{\triangle \textbf{ Scindages possibles.}}$ Puis, passez à l'étape suivante.

(l_2) Si $r = 0$ alors retournez $\{T\}$, sinon et si $r \in \mathbf{k}$ alors retournez $\{\,\}$. Passez à l'étape suivante.

(l_3) Retournez $\mathsf{intersect}(\mathsf{tail}(r), \mathsf{intersect}(\mathsf{init}(r), T))$ puis passez à l'étape suivante.

(l_4) Remplacez r par sa partie primitive, ce polynômes r étant vu comme univarié en $\mathsf{mvar}(r)$. Puis, passez à l'étape suivante.

(l_5) Si l'ensemble régulier $T^-_{\mathsf{mvar}(r)} \cup \{r\}$ est sans carrés alors allez en (l_7).

(l_6) Posons $v = \mathsf{mvar}(r)$. Calculez un pgcd normalisé par rapport à T_v^- de r et de sa dérivée par rapport à v en interprétant leurs coefficients dans la tour d'extensions simples associés à T_v^-. $\boxed{\triangle \textbf{ Scindages possibles.}}$ Soit g un tel pgcd. Remplacez r par $\mathsf{pquo}(r, g)$. Ainsi, l'ensemble régulier $T_v^- \cup \{r\}$ est maintenant sans carrés. Passez à l'étape (l_3).

(l_7) Posons $v = \mathsf{mvar}(r)$ et $T_v^+ = \{t_k, \ldots, t_l\}$ avec $\mathsf{mvar}(t_k) < \cdots < \mathsf{mvar}(t_l)$. Calculez $\mathcal{D} = \mathsf{intersect}(t_l, \mathsf{intersect}(\ldots, \mathsf{intersect}(t_k, \ T_v^- \cup \{r\})))$. Effacez dans $\mathcal{D}$ tout ensemble triangulaire U tel que $\mathsf{normalize}(\mathsf{init}(t_i), \ U^-_{\mathsf{mvar}(t_i)}) = 0$ pour un $i \in \{k, \ldots, l\}$.

$\boxed{\triangle \textbf{ Scindages possibles.}}$ Passez enfin à la dernière étape.

(l_8) Retournez $\mathsf{intersect}(p, \mathcal{D})$ où p est le polynôme de départ. $\boxed{\triangle \textbf{ Scindages possibles.}}$

Remarque V.22 Observons tout d'abord que si les algorithmes des opérations :

- $(p, T) \longmapsto \mathsf{algebraicIntersect}(p, T)$
- $(p, T) \longmapsto \mathsf{transcendentalIntersect}(p, T)$

satisfont leurs spécifications (remarque V.17) alors il va de même de l'algorithme de l'opération : $(p, T) \longmapsto \mathsf{intersect}(p, T)$. $\triangleright$ En effet, si $\mathsf{iRed}(p, T)$ est nul cela entraîne : $p \in \mathsf{sat}(T)$ et donc : $\mathbf{W}(T) \subseteq \mathbf{V}(p)$. Par suite, dans ce cas, $\{T\}$ est bien un scindage de Lazard de $\mathbf{V}(p) \cap \mathbf{W}(T)$. Le cas où $\mathsf{iRed}(p, T)$ n'est pas nul est traîté soit par $\mathsf{algebraicIntersect}(p, T)$, soit par $\mathsf{transcendentalIntersect}(p, T)$. $\triangleleft$

Remarque V.23 Observons que si l'algorithme de l'opération : $(p, T) \longmapsto \mathsf{intersect}(p, T)$ satisfait ces spécifications (théorème V.2, point (*iii*)) alors il en va de même de l'algorithme de l'opération : $(p, T) \longmapsto \mathsf{extend}(p, T)$ avec les hypothèses :

$$p \notin \mathbf{R} \text{ et } \mathsf{mvar}(p) > \mathsf{mvar}(T)$$

Notons que ces hypothèses sont toujours vérifiées lors d'un appel à extend dans nos algorithmes. $\triangleright$ Si $\mathsf{init}(p) \in \mathbf{R}$ alors $\mathsf{extend}(p, T)$ retourne $\mathsf{intersect}(p, T)$ ce qui est correct. Si $\mathsf{iRed}(\mathsf{init}(p), T) = 0$ alors : $\mathbf{W}(T) \subseteq \mathbf{V}(\mathsf{init}(p))$. Donc $(\mathbf{V}(p) \setminus \mathbf{V}(\mathsf{init}(p))) \cap \mathbf{W}(T)$ est vide, Or $\mathsf{extend}(p, T)$ retourne $\emptyset$ dans ce cas. Supposons maintenant $\mathsf{iRed}(\mathsf{init}(p), T)$ nul. Par suite, la variable v égale à $\mathsf{mvar}(\mathsf{iRed}(p, T))$ n'est pas algébrique dans T et $\mathsf{intersect}(p, T)$ appelle $\mathsf{transcendentalIntersect}(p, T)$ qui retourne $\{S_1, \ldots, S_s\}$ un scindage de Lazard de $\mathbf{V}(p) \cap \mathbf{W}(T)$. On distingue alors deux cas pour chacun des S_i.

- Soit S_i a été retourné car : $v = \mathsf{mvar}(S_i)$. Posons $t = S_{iv}$. Si $\mathsf{mdeg}(t) = \mathsf{mdeg}(p)$, par construction (voir $\mathsf{augment}$) $\overline{t}^{\widehat{S_i}}$ divise $\overline{p}^{\widehat{S_i}}$ dans $\mathbf{A}_{S_i}[v]$ et $\overline{\mathsf{init}(p)}^{\widehat{S_i}}$ est à une unité près de $\mathbf{A}_{S_i}$ égal à $\overline{\mathsf{init}(t)}^{\widehat{S_i}}$. Comme $\mathsf{init}(t)$ est non nul et normalisé par rapport à S_i, on en déduit que $\overline{\mathsf{init}(p)}^{\widehat{S_i}}$ est une unité de $\mathbf{A}_{S_i}$ (proposition V.6 point (*iii*)). Or, si $\mathsf{init}(p)$ s'annulait sur tout ou partie de $\mathbf{W}(S_i)$, ce polynôme appartiendrait à l'un des idéaux premiers associés à $\mathsf{sat}(S_i)$. Ainsi, l'ensemble S_i est retourné par $\mathsf{extend}(p, T)$ ce qui est correct. Si $\mathsf{mdeg}(t) < \mathsf{mdeg}(p)$, il y a deux sous-cas :
  - soit $\overline{t}^{\widehat{S_i}}$ est une partie sans carrés de $\overline{p}^{\widehat{S_i}}$ et l'on conclut comme précédemment,
  - soit $\overline{\mathsf{init}(p)}^{\widehat{S_i}}$ est nul et $\mathsf{extend}(p, T)$ ne retournera pas S_i ce qui est correct.
- Soit S_i a été retourné car p se normalise à zéro par S_i. Si cela est du fait des coefficients de p et en particulier de son initial, $\mathsf{extend}(p, T)$ ne retournera pas S_i ce qui est correct. Si par contre, cela est du fait d'un polynôme en v, sans que $\mathsf{init}(p)$ se normalise à 0, cela signifie que ce polynôme en v est une partie sans carrés de p dans $\mathbf{A}_{S_i}[v]$, cas déjà traité $\triangleleft$.

Remarque V.24 Vérifions que les algorithmes des opérations spécifiées dans le théorème V.2 satisfont leurs spécifications dans le cas $n = 1$. Notons d'une part que, d'après la remarque V.22, étudier $\mathsf{intersect}$ se ramène à étudier $\mathsf{algebraicIntersect}$ et $\mathsf{transcendentalIntersect}$. D'autre part, d'après la remarque V.23, pour tout couple (p, T) pour lequel l'algorithme de $\mathsf{intersect}$ est correct l'algorithme de extend l'est aussi. Par suite, nous n'examinons ci-après que les algorithmes des opérations $\mathsf{initIntersect}$, $\mathsf{augment}$, $\mathsf{algebraicIntersect}$, $\mathsf{transcendentalIntersect}$ et $\mathsf{quasiRecipElseSplit}$.

- Avec $n = 1$, on a : $\mathsf{init}(p) \in \mathbf{R}$ et $\mathsf{initIntersect}(p, T)$ est trivialement correcte
- Sauf cas triviaux, T est nécessairement vide dans $\mathsf{augment}(p, T)$. Par suite, U est vide. D'autre part, π et R sont égaux à $\mathsf{squareFreePart}(p, \emptyset)$. Comme $\mathsf{initIntersect}(p, T)$ est vide, $\mathsf{augment}(p, T)$ retourne $\mathsf{squareFreePart}(p, \emptyset)$ ce qui est correct d'après la proposition III.5 p. 81.
- Passons à $\mathsf{algebraicIntersect}(p, T)$. On suppose donc $T = \{t\}$ où t a X_1 comme variable principale, comme p. On doit distinguer les cas suivants.
  - Si p et t sont premiers entre eux alors $\mathsf{normalizedGcd}\ (p, t, \emptyset)$ retourne une constante (non nulle) et $\mathsf{algebraicIntersect}(p, T)$ retourne $\emptyset$ ce qui est correct.
  - Si p et t ont un pgcd non trivial, celui-ci est rajouté à l'ensemble triangulaire vide par $\mathsf{augment}$. Soit U l'ensemble ainsi obtenu. On vérifie que $\mathsf{algebraicIntersect}(p, T)$ retourne $\{U\}$ ce qui est correct.
- Passons à $\mathsf{transcendentalIntersect}(p, T)$. On suppose donc T vide et on vérifie que $\mathsf{transcendentalIntersect}(p, T)$ retourne $\mathsf{augment}(p, T)$ ce qui est correct.
- Passons pour finir à $\mathsf{quasiRecipElseSplit}(p, T)$. Rappelons (proposition V.4) qu'un appel à cette opération suppose : $p \notin \mathbf{R}$ et $\mathsf{mvar}(p) \in \mathsf{algVar}(T)$. On suuppose donc $T = \{t\}$ où t a X_1 comme variable principale, comme p. Notons qu'alors T_v^- et T_v^+ sont vides. Par suite, $\mathsf{extendedSubResGcdElseSplit}\ (p,\ T_v,\ T_v^-)$ retourne des polynômes f, h, g de $\mathbf{R}_1$ tel que :
$$f\,p + h\,t \;=\; g$$
  On distingue alors deux cas.
  - Soit $g \notin \mathbf{R}$ et :
$$\mathsf{quasiRecipElseSplit}(p, T) = \mathsf{augment}(g, \emptyset) \cup \mathsf{augment}(t\ \mathsf{exquo}\ g, \emptyset)$$
  - Soit $g \in \mathbf{R}$ et :
$$\mathsf{quasiRecipElseSplit}(p, T) = (f, g)$$

  Ces deux cas satisfont les points (ii) et (i) de la proposition V.4.

Remarque V.25 Pour continuer l'examen de nos algorithmes nous allons vérifier que chaque exécution de l'un d'eux se termine. La difficulté provient essentiellement des algorithmes intersect, augment et extend qui s'appellent mutuellement. On munit pour cela les couples (p, T) où, comme d'habitude, p est un polynôme de $\mathbf{R}_n$ et T un ensemble de Lazard de $\mathbf{R}_n$, de l'ordre $\prec_{Laz}$ défini comme suit :

$$(p_1, T_1) \prec_{Laz} (p_2, T_2) \iff \begin{cases} T_2 \prec_{alg} T_1 & \text{ou} \\ T_2 \sim_{alg} T_1 & \text{et} \quad p_1 <_{rwl} p_2 \end{cases}$$

On vérifie sans peine qu'il n'y a pas de suites infinies strictement décroissantes pour cet ordre. Le fait que chaque exécution de l'un des algorithmes intersect, augment et extend se termine fait l'objet de la proposition suivante.

Proposition V.12 *Pour tout couple (p, T) où p est un polynôme de $\mathbf{R}_n$ et T un ensemble de Lazard de $\mathbf{R}_n$, les algorithmes des opérations* augment(p, T), intersect(p, T) *et* extend(p, T) *se terminent et retournent chacun un ensemble $\{S_1, \ldots, S_s\}$ d'ensembles de Lazard tels que :*

$$(s > 1 \text{ et } (\forall i \in \{1, \ldots, s\})\, T \prec_{alg} S_i) \quad \text{ou} \quad (s = 1 \text{ et } S_1 = T) \quad \text{ou} \quad (s = 0)$$

Preuve $\triangleright$ On raisonne par récurrence sur n. Notons que le cas $n = 1$ résulte de la remarque V.24. On suppose maintenant $n > 1$. On raisonne alors par induction sur l'ordre $\prec_{Laz}$. Autrement dit, nous allons passer en revue chacun des algorithmes initIntersect, augment, algebraicIntersect, transcendentalIntersect, intersect et extend pour un couple (p, T), en supposant que pour tout couple (q, L) tel que $(q, L) \prec_{Laz} (p, T)$ nos algorithmes se terminent en produisant chacun un ensemble $\{S_1, \ldots, S_s\}$ d'ensembles de Lazard tels que :

$$(s > 1 \text{ et } (\forall i \in \{1, \ldots, s\})\, T \prec_{alg} S_i) \quad \text{ou} \quad (s = 1 \text{ et } S_1 = T) \quad \text{ou} \quad (s = 0)$$

Dans la suite, un tel ensemble $\{S_1, \ldots, S_s\}$ d'ensembles de Lazard sera dit *scindage convergent de T*.

On en déduit immédiatement que quasiRecipElseSplit(p, T) se termine. De plus lorsqu'elle retourne un ensemble $\{T_1, \ldots, T_k\}$ d'ensembles de Lazard, on a :

$$k > 1 \quad \text{et} \quad (\forall i \in \{1, \ldots, k\})\, T \prec_{alg} T_i$$

Pour établir ce fait, on raisonne comme pour augment ci-après. Nous ne détaillerons donc pas cette vérification. Par suite, pour le couple (p, T) ou tout couple (q, L) tel que $(q, L) \prec_{Laz} (p, T)$, nous avons le droit d'utiliser la proposition V.4 (et les propositions qui en découlent) dans cette preuve. Rappelons que dans les premières sections quasiRecipElseSplit était une *boîte noire* et que nous ignorions ces liens (*indirects*) avec les opérations que nous en avons déduit.

– Nous commençons par initIntersect(p, T). Les cas où init$(p) \in \mathbf{R}$ et tail$(p) \in \mathbf{reg}(\mathbf{R})$ sont immédiats. Dans les autres cas, initIntersect fait tout d'abord appel à intersect sur le couple $(\text{init}(p), T)$ pour qui on a :

$$(\text{init}(p), T) \prec_{Laz} (p, T)$$

Par hypothèse d'induction, $\mathsf{intersect}(\mathsf{init}(p), T)$ se termine et retourne un ensemble $\{T_1, \ldots, T_k\}$ d'ensembles de Lazard qui est un scindage convergent de T. Par suite, si $\mathsf{tail}(p) = 0$, l'opération $\mathsf{initIntersect}(p, T)$ se termine en retournant $\{T_1, \ldots, T_k\}$. Sinon, que k vale 1 ou pas, $\mathsf{intersect}$ est rappelé sur les couples $(\mathsf{tail}(p), T_i)$ pour qui on a encore :

$$(\mathsf{tail}(p), T_i) \prec_{Laz} (p, T)$$

Bref, la propriété désirée est vérifiée pour $\mathsf{initIntersect}(p, T)$ dans tous les cas.

- Nous passons à $\mathsf{augment}(p, T)$. Si $p \in \mathbf{R}$ c'est immédiat. On suppose maintenant que : $p \notin \mathbf{R}$. Nous nous plaçons tout d'abord juste après la première boucle **for**. D'après les propositions V.9 et V.10, la réunion $\{T_1, \ldots, T_k\}$ des ensembles de Lazard de la liste π constitue un scindage de Lazard de T_v^- qui vérifie :

$$k > 1 \implies (\forall i \in \{1, \ldots, k\}) \; T_v^- \prec_{alg} T_i \quad \text{ou} \quad T_1 = T$$

Nous nous plaçons maintenant au début de la deuxième boucle **for**. Notons que chacun des ensembles T'_i produit par la première instruction de cette boucle vérifie (même si $k = 1$) :

$$T_v^- \prec_{alg} T'_i$$

On distingue maintenant deux cas.

  - Soit T_v^+ est vide et l'on quitte cette seconde boucle **for**,
  - Soit T_v^+ n'est pas vide et l'on remarque que extend est appelé sur des couples (q, L) pour qui on a :

$$(q, L) \prec_{Laz} (p, T)$$

  Par hypothèse d'induction, on quitte donc cette seconde boucle **for** avec un ensemble $S = \{S_1, \ldots, S_l\}$ d'ensembles de Lazard vérifiant :

$$l > 1 \implies (\forall i \in \{1, \ldots, l\}) \, T \prec_{alg} S_i \quad \text{ou} \quad S_1 = T$$

Enfin, d'après ce qui précède, on quitte la troisième et dernière boucle **for** en ayant obtenue la propriété désirée pour $\mathsf{augment}(p, T)$.

- Nous passons à $\mathsf{algebraicIntersect}(p, T)$. Nous donnons rapidement les remarques essentielles pour traiter cet algorithme. On reprend les notations de l'algorithme. L'appel à $\mathsf{augment}$ ne pose pas de problème puisque l'on a :

$$(T_i = T_v^- \text{ ou } T_v^- \prec_{alg} T_i) \quad \text{et} \quad g_i <_{rwl} p$$

Puis, si $\mathsf{mvar}(U_i) < v$, on a nécessairement : $T_i \prec_{alg} U_i$ (car alors $\mathsf{augment}(g_i, T_i)$ a retourné au moins deux ensembles de Lazard). Par suite, chaque ensemble de Lazard des deux ensembles L est supérieure à T_v^- (et donc à T) pour $\prec_{alg}$. Enfin, si $\mathsf{mvar}(U_i) = v$, on a :

$$\mathsf{mdeg}(U_{iv}) \leq \mathsf{mdeg}(g_i)$$

Or :

$$\mathsf{mdeg}(g_i) < \mathsf{mdeg}(T_v)$$

sans quoi q serait nul. Par suite :

$$T_v^- \cup \{ T_v \} \prec_{alg} U_i$$

Bref, les différents appels à augment, intersect et extend dans algebraicIntersect(p, T) se font sur des couples inférieures à (p, T) pour l'ordre $\prec_{Laz}$. Ainsi, algebraicIntersect(p, T) possède bien la propriété désirée.

- Nous passons à transcendentalIntersect(p, T). Là encore, nous sommes un peu *expéditifs* pour ne pas rendre *kilométrique* la preuve de cette proposition. On reprend les notations de l'algorithme. Soit $[q, T] \in R$. Pour chaque couple $[n_i, T_i]$ de normalize(q, T) il y a deux cas.
  - Soit n_i est nul. Par suite, le polynôme *p de départ* est normalisé à zéro par T_i. Si T_i n'est pas le seul ensemble retourné, c'est qu'il a été construit par normalize ou l'algorithme augment (que nous avons déjà examiné) en *scindant* l'ensemble *T de départ*. On a donc bien : $T \prec_{alg} T_i$.
  - Soit n_i est non nul. Si $T_i = T$ alors soit $n_i = p$, soit $n_i <_{rwl} p$. Si $T_i \neq T$, alors nécessairement, T_i a été produit par scindage de T grâce à augment et l'on a : $T \prec_{alg} T_i$ Par suite, l'appel augment(n_i, T_i) se termine et produit un ensemble $\{T_{i,1}, \ldots, T_{i,s_i}\}$ d'ensembles de Lazard tel que :

    $$s_i > 1 \implies (\forall j \in \{1, \ldots, s_i\})\, T_i \prec_{alg} T_{i,j}$$

    Si $s_i = 1$, alors nécessairement iRed$(n_i, T_{i,1})$ est nul et *on sort* $T_{i,1}$ au plus tard à la prochaine itération. Si $s_i > 1$, on remplace dans R le couple $[n_i, T_i]$ par des couples $[n_i, T_{i,j}]$ tels que :

    $$(n_i, T_{i,j}) \prec_{Laz} (n_i, T_i)$$

    Ce processus devant se terminer tôt ou tard du fait de l'ordre $\prec_{Laz}$.
- Le cas de intersect(p, T) résulte des deux cas précédents.
- Le cas de extend(p, T) résulte de celui de intersect(p, T)

Ceci termine la preuve de cette proposition. $\triangleleft$

Remarque V.26 Il résulte de la preuve de la proposition précédente que vérifier la validité de l'algorithme transcendentalIntersect se ramène à vérifier la validité de l'algorithme de l'algorithme augment. En gros, transcendentalIntersect fait *ce qu'il faut* pour que intersect(p, T) se ramène à augment(p, T). L'algorithme algebraicIntersect doit être compris comme une optimisation de intersect. En effet, algebraicIntersect permet de se contenter d'un calcul de pgcd normalisé là où transcendentalIntersect calculerait un pgcd normalisé avec coefficients de Bezout (pour normaliser).

Par conséquent, pour terminer l'exploration de nos algorithmes nous nous contenterons d'étudier augment(p, T). Nous supposerons pour simplifier : mvar$(p) >$ mvar(T) et l'on pose : $v =$ mvar(p). Par conséquent U est vide. Nous supposerons également que T n'est pas scindé par les calculs de partie sans carrés et de partie primitive. Par conséquent, il existe des polynômes f et t de $\mathbf{R}_n$ tels que :

- squareFreePart$(p, T) = \{[f, T]\}$

– $\mathsf{primitivePart}(f, T) \;=\; \{[t, T]\}$

On distingue alors les cas suivants :

– Si $p = f = t$, alors :

$$\mathsf{augment}(p, T) \;=\; \{T \cup \{p\}\} \;\cup\; \mathsf{initIntersect}(p, T)$$

ce qui est correct puisque l'on a :

$$\mathbf{V}(p) \cap \mathbf{W}(T) \;=\; \mathbf{W}(T \cup \{p\}) \;\cup\; \mathbf{V}(p, \mathsf{init}(p)) \cap \mathbf{W}(T)$$

et que dans ce cas $T \cup \{p\}$ est bien un ensemble de Lazard

– Si $p \neq f$ et $f = t$, alors :

$$\mathsf{augment}(p, T) \;=\; \{T \cup \{f\}\} \;\cup\; \mathsf{initIntersect}(f, T) \;\cup\; \mathsf{initIntersect}(p, T)$$

Or d'après la proposition V.9 on a :

$$\sqrt{\langle \overline{p}^{\widehat{T}} \rangle_{\mathbf{A}_T[v]}} \;=\; \langle \overline{f}^{\widehat{T}} \rangle_{\mathbf{A}_T[v]}$$

Par conséquent, si les polynômes $\overline{p}^{\widehat{T}}$ et $\overline{f}^{\widehat{T}}$ sont unitaires, ils ont les mêmes zéros en tant que polynômes à coefficients dans chacun des facteurs du produit de corps $\mathbf{A}_T$. Puisque les polynômes p et f sont normalisés par rapport à T, dire que $\overline{\mathsf{init}(p)}^{\widehat{T}}$ et $\overline{\mathsf{init}(f)}^{\widehat{T}}$ sont des unités revient à dire que ces quantités sont non nulles.

– Si $p \neq f$ et $f \neq t$, alors :

$$\mathsf{augment}(p, T) \;=\; \{T \cup \{t\}\} \cup \mathsf{initIntersect}(t, T) \cup \mathsf{initIntersect}(f, T) \cup \mathsf{initIntersect}(p, T)$$

On remarque alors que l'on a :

$$\langle \overline{t}^{\widehat{T}} \rangle_{\mathbf{A}_T[v]} \;=\; \langle \overline{f}^{\widehat{T}} \rangle_{\mathbf{A}_T[v]}$$

Lorsque les calculs de partie sans carrés et de partie primitive *remplace* T par un scindage de Lazard $\{S_1, \ldots, S_s\}$ de T on reprend le cas *non scindé* sur chacun des S_i et l'on conclut en raisonnant comme dans la preuve de la proposition V.11.

Enfin, pour traiter le cas où U n'est pas vide, on *incorpore* l'algorithme de extend dans augment. Puis, on vérifie en raisonnant sur les couples (p, T) à l'aide de l'ordre $\prec_{Laz}$ (comme dans la proposition V.12) que les algorithmes de augment et intersect satisfont les spécifications du théorème V.2.

Remarque V.27 Reprenons l'exemple des points singuliers de la courbe traité dans la remarque V.10. Nous avons posé :

$$p_1 = y^5 - x\ y^4 + \left(x^2 - x - 1\right)\ y^3 + \left(-x^3 + x^2 + x\right)\ y^2 + \left(-x^3 + x\right)\ y + x^4 - x^2$$

et :

$$p_2 = -y^4 + (2\ x - 1)\ y^3 + \left(-3\ x^2 + 2\ x + 1\right)\ y^2 + \left(-3\ x^2 + 1\right)\ y + 4\ x^3 - 2\ x$$

On cherche une décomposition de Lazard de $\mathbf{V}(p_1, p_2)$. On vérifie sans peine que le polynôme p_1 en tant que polynôme de $\mathbb{Q}(x)[y]$ est sans carrés. Son initial étant constant, une décomposition de Lazard de $\mathbf{V}(p_1)$ est :$\{\ \{p_1\}\ \}$. Pour déterminer une décomposition de Lazard de $\mathbf{V}(p_1, p_2)$, il suffit par conséquent d'exécuter $\mathsf{intersect}(p_2, \{p_1\})$. On vérifie immédiatement que $\mathsf{intersect}(p_2, \{p_1\})$ vaut dans ce cas $\mathsf{algebraicIntersect}(p_2, \{p_1\})$. On calcule alors $\mathsf{normalizedGcd}\ (p_2, p_1, \emptyset)$ ce qui donne le résultant en y de ces deux polynômes, soit r. On calcule ensuite $\mathsf{augment}(r, \emptyset)$ ce qui donne : $\{U\}$ avec :

$$U\ =\ \left\{2\ x^6 - 5\ x^4 + 2\ x^3 + 2\ x^2 - x\right\}$$

Comme $\mathsf{mvar}(U) < y$, on calcule ensuite $\mathsf{extend}(p_1, U)$. Ceci conduit à calculer $\mathsf{intersect}(p_1, U)$ qui appelle $\mathsf{transcendentalIntersect}(p_1, U)$ qui appelle $\mathsf{augment}(p_1, U)$ qui appelle $\mathsf{squareFreePart}(p_1, U)$ ce qui donne :

```
           4              2                 2
 [[val= 2y  + (- 2x - 1)y  + x,tower= {2x  - 1}],

           3     3    2        2       3               3
   [val= y  + (x  + x  - 2x)y  + (- x  + x - 1)y - x  + x,
              4     2
    tower= {x  - 2x  + x}]
```

Les initiaux de ces parties sans carrés étant constants, $\mathsf{extend}(p_1, U)$ retourne tout simplement L donné par :

```
     2      4              2
 [{2x  - 1,2y  + (- 2x - 1)y  + x},
    4     2     3     3    2        2       3               3
  {x  - 2x  + x,y  + (x  + x  - 2x)y  + (- x  + x - 1)y - x  + x}]
```

Nous revenons dans $\mathsf{algebraicIntersect}(p_2, \{p_1\})$. Cette opération se réduit maintenant à $\mathsf{intersect}(p_2, L)$. Posons $L = \{T_1, T_2\}$ où T_i est le *i-ième* ensemble de Lazard de la liste ci-dessus.

Commençons par $\mathsf{intersect}(p_2, T_1)$, c'est-à-dire $\mathsf{algebraicIntersect}(p_2, T_1)$. On doit calculer :

$$\mathsf{normalizedGcd}(p_2, 2\ y^4 + (-2\ x - 1)\ y^2 + x, \{2\ x^2 - 1\})$$

ce qui donne :

```
                                 2
     [[val= y - x,tower= {2x  - 1}]]
```

Puis, on doit calculer $\mathsf{augment}(y - x, \{2\ x^2 - 1\})$ ce qui donne clairement :

$$\left\{\left\{2\ x^2 - 1,\ y - x\right\}\right\}$$

résultat que retourne finalement $\mathsf{intersect}(p_2, T_1)$.

Passons à $\mathsf{intersect}(p_2, T_2)$, c'est-à-dire $\mathsf{algebraicIntersect}(p_2, T_2)$. On doit calculer :

$$\mathsf{normalizedGcd}(p_2, y^3 + (x^3 + x^2 - 2\ x)\ y^2 + (-x^3 + x - 1)\ y - x^3 + x, \{x^4 - 2\ x^2 + x\})$$

ce qui donne :

```
          2               2
[[val= y   - x,tower= {x   + x - 1}], [val= y - 1,tower= {x - 1}],
          3
 [val= y   - y,tower= {x}]]
```

On en tire $\mathsf{intersect}(p_2, T_2)$:

```
          3       2           2
[{x,y   - y},{x   + x - 1,y   - x},{x - 1,y - 1}]
```

Remarque V.28 Nous reprenons maintenant l'exemple de la remarque V.9. Nous nous intéressons aux polynômes :

$$p_1 \ = \ \left(t\ z - t^2 - t\right)\ y - z + a\ t + a \ \text{ et } \ p_2 \ = \ \left(z^2 - t\ z + 1\right)\ y + (-t + a)\ z - a$$

Déterminons un scindage de Lazard de $\mathbf{V}(p_1, p_2)$. Une décomposition de Lazard de $\mathbf{V}(p_1)$ est donné par $\mathsf{intersect}(p_1, \emptyset)$ qui est donné par $\mathsf{augment}(p_1, \emptyset)$:

```
             2
[{(t z - t   - t)y - z + a t + a},{a - 1,z - t - 1},{t + 1,z},{t,z - a}]
```

Notons respectivement S_1, S_2, S_3, S_4 les ensembles ainsi obtenus.

Passons à $\mathsf{intersect}(p_2, \{S_1, S_2, S_3, S_4\})$. Tout d'abord $\mathsf{intersect}(p_2, \{S_2, S_3, S_4\})$ retourne (par simple substitution ou presque) :

```
                                 2              2           2
[{a - 1,z - t - 1,(t + 2)y - t }, {t,z - a,(a   + 1)y + a   - a},
 {t + 1,z,y - a}]
```

Reste à traiter $\mathsf{intersect}(p_2, S_1)$, c'est-à-dire $\mathsf{algebraicIntersect}(p_2, S_1)$. On doit calculer :

$$\mathsf{normalizedGcd}(p_2, (t\ z - t^2 - t)\ y - z + a\ t + a, \emptyset)$$

On trouve leur résultant, soit r. Puis, on calcule $\mathsf{augment}(r, \emptyset)$ ce qui donne : $\{U\}$ avec :

$$U \ = \ \left\{z^3 + \left(-t^2 - t - a\right)\ z^2 + \left(t^3 + t^2 - a\ t + 1\right)\ z + a\ t^2 - a\right\}$$

On passe ensuite à $\mathsf{extend}((t\ z - t^2 - t)\ y - z + a\ t + a, U)$ ce qui donne :

```
   3       2           2     3    2                  2
 {z  + (- t  - t - a)z  + (t  + t  - a t + 1)z + a t  - a,
    2           2      2
   (t  + 2t)y + z  + (- t  - a + 1)z - 2a t - a}
 ,
       2                                          2
{t + 2,z  + (- a - 3)z + 3a,(8a + 8)y + (- a + 1)z + a  + 7a},
       2   2                             2
{a - 1,z  - t z - t + 1,t y - 1}, {t + 1,z  - a z + a + 1,y + 1},
       3            2            2
{a - 1,t  - t - 2,z - t  + t + 1,2y - t  + 1}, {a + 1,t + 1,z + 1,y + 1},
{a + 1,t + 2,z - 3,4y + 1}]
```

Soit L le scindage obtenu. Il reste à calculer $\mathsf{intersect}(p_2, L)$ ce qui donne :

```
   3       2           2     3    2                  2
 {z  + (- t  - t - a)z  + (t  + t  - a t + 1)z + a t  - a,
    2           2      2
   (t  + 2t)y + z  + (- t  - a + 1)z - 2a t - a},
       2                                          2
{t + 2,z  + (- a - 3)z + 3a,(8a + 8)y + (- a + 1)z + a  + 7a},
{a + 1,t + 2,z - 3,4y + 1}]
```

Revenons au problème (remarque V.9) du calcul d'une décomposition de $\mathbf{V}(p_1, p_2, p)$ où

$$p \;=\; (z-t)\; y^2 + \left(-z^2 + t\; z + a\right)\; y - a\; z - t + a$$

On reprend les notations de la remarque V.9 La seule intersection non triviale à calculer est celle de $\mathbf{V}(p)$ avec $\mathbf{W}(T_1)$. On procède comme dans $\mathsf{transcendentalIntersect}$. On commence par normaliser p par rapport à T_1 ce que nous avons déjà fait dans la remarque V.9. Les facteurs sans carrés (sur les entiers) du normalisé de p par rapport à T_1 sont :

```
[t, t + 2,
     11                10       2             9                8
   4t   + (- 8a + 12)t   + (4a  - 18a + 18)t  + (- 28a + 24)t
 +
      3      2                7        3      2              6
   (6a  + 16a  - 42a + 20)t  + (- 4a  + 22a  - 42a + 12)t
 +
        4    3      2                5        4     3      2           4
   (- 2a  + a  + 26a  - 39a + 10)t  + (- 4a  - 2a  + 30a  - 32a + 4)t
 +
       5     4     3      2                3      4     2             2
   (- a  - 2a  - 6a  + 20a  - 13a + 6)t  + (2a  + 8a  - 12a + 2)t
 +
     5     4    3     2                  4    3
   (a  + 2a  - a  + 7a  - 2a + 1)t + 2a  - a  - a]
```

Notons f le dernier facteur de la liste ci-dessus. On peut alors calculer $\mathsf{augment}(f, T_1)$ en 10 heures et l'on vérifie (quelques secondes) que p se réduit à 0 sur chacune des quasi-composantes de $\mathsf{augment}(f, T_1)$. La solution (trop longue pour être reproduite ici) est ainsi obtenue sans recours à la base lexicographique, ou à la factorisation en facteurs irréductibles.

Remarque V.29 Soient T et U deux ensembles de Lazard de $\mathbf{R}_n$. Nous terminons ce chapitre par un algorithme pour une opération $\mathsf{subQuasiComponent?}(T, U)$ permettant de détecter si $\mathbf{W}(T)$ est contenu dans $\mathbf{W}(U)$. On laisse le soin au lecteur de combiner cette opération avec intersect afin d'obtenir, pour un ensemble de polynômes F de $\mathbf{R}_n$, une opération $\mathsf{solve}(F)$ produisant une décomposition de Lazard sans redondances de $\mathbf{V}(F)$. Notons que l'on a intérêt pour éliminer les quasi-composantes redondantes à maintenir les scindages de Lazard triés par dimension décroissante et à travailler en priorité avec les quasi-composantes de dimension maximum. Pour cela, une façon commode d'implanter intersect est d'utiliser un *arbre de scindage* comme dans la méthode de Wu (voir le chapitre suivant). Nous reviendrons ultérieurement sur les optimisations à apporter à l'opération solve. Il fallait bien conclure un jour ce chapitre. Remarquons quand-même qu'avant d'exécuter $\mathsf{intersect}(F, \emptyset)$ on a souvent intérêt à :

- *inter-réduire un peu* les polynômes de F (au sens base de Gröbner) pour éliminer une variable par exemple (comme nous l'avons vu pour le système de Liu-Lorenz)
- *contraindre* intersect à utiliser en priorité les polynômes impliquant qu'un produit de variables est non nul (comme $abcde - 1$ dans `Cyclic-5`) afin d'éviter certains scindages.

```
subQuasiComponent?(T, U) ==
    |T| < |U| => false
    algVar(U) ⊈ algVar(T) => false
    R := {[U, T]}
    b := true
    for [U, T] in R while b repeat
        U = ∅ => iterate
        p := first(U)
        q := iRed(p, T)
        ν := normalizeElseSplit(q, T)
        if split?(ν)
            then
                R := R ∪ {[U, S] for S ∈ ν}
            else
                if b := (ν = 0)
                    then
                        i_q := iRed(init(p), T)
                        ν := normalizeElseSplit(i_q, T)
                        if b := (ν ≠ 0)
                            then
                        R := R ∪ {[rest(U), T]}
    return b
```

Chapitre VI

Comparaison expérimentale de quelques méthodes

Introduction

Le but de ce chapitre est de donner une comparaison expérimentale de différentes méthodes décomposant une variété affine en zéros réguliers (ou fermetures de zéros réguliers) d'ensembles triangulaires. Nous ne nous intéresserons **pas uniquement** à la rapidité d'obtention d'un résultat mais aussi à diverses questions comme par exemple :

1. les résultats produits par une implantation de la méthode sont-ils corrects ?
2. la méthode produit-elle souvent (resp. jamais) des ensembles triangulaires inconsistants ? des décompositions avec redondances ?

Les méthodes les plus connues (au moment où nous avons commencé cette thèse) sont dues à Daniel Lazard ([Laz91a]), Michael Kalkbrener ([Kal91, Kal93]), Dongming Wang ([Wan93b]) et Wu Wen-Tsün ([Wu87]). Ce sont celles que nous allons étudier. De l'opinion même de ces auteurs, il semblait nécessaire de confronter ces méthodes, qui, lorsqu'elles ont été implantées, ne l'ont pas été dans des conditions comparables (logicielles, matérielles, humaines). De plus, il ne paraît pas impossible qu'une méthode efficace dans un maximum de situations soit un *compromis* de diverses méthodes.

Il est apparu nécessaire de définir un cadre commun à toutes ces méthodes. Une terminologie et des concepts tentant d'unifier tous ces points de vue ont été proposés par Daniel Lazard ([Laz92b]). Nous les avons rappelé dans le premier chapitre. A partir de là, nous avons implanté en AXIOM les outils communs à ces méthodes. Nous décrirons ces outils dans la seconde section de ce chapitre. Nous avons aussi implanté diverses opérations relatives aux idéaux et aux variétés affines, notions d'algèbre et de géométrie liées à la résolution des systèmes d'équations algébriques (voir le premier chpitre). Ceci afin de pouvoir étudier les solutions calculées à l'aide de chacune des méthodes. En particulier, les calculs effectués par chaque méthode peuvent être vérifiés à l'aide des techniques fondées sur les bases de Gröbner et du logiciel GB de J.-C. Faugère ([Fau94]). Ainsi, a-t-on pu mettre au point les implantations de ces différentes méthodes. Ce fut en particulier le cas pour celle de Daniel Lazard qui n'avait pas encore été mise en œuvre. Mais ce fut également le cas pour les trois autres méthodes dont les résultats ne sont pas canoniques. Plus précisément, pour un système de départ donné, la solution calculée par une de ces méthodes peut non seulement dépendre de l'implantation mais aussi de l'ordre des polynômes dans ce système de départ. De plus la seule vérification *à la main* de quelques calculs (nécessairement peu *compliqués*) produits par l'implantation d'une méthode ne peut évidemment permettre d'affirmer que cette implantation est correcte. J'avoue avoir eu une implantation erronée pendant plus de trois ans de la méthode de Wu Wen-Tsün, du fait d'une erreur de programmation dans une procédure d'élimination des cas redondants. Il m'a fallu mettre au point le processus de vérification décrit plus loin pour être aujourd'hui un confiant dans mon implantation de cette méthode. Ce processus de vérification :

- intensivement testé depuis bientôt un an,
- fondé sur des algorithmes simples, bien connus et implantés de manière directe,
- comme *sur-couche* AXIOM du logiciel GB

sera considéré comme *très probablement fiable* dans nos expériences.

L'implantation des quatre méthodes évoqués plus haut est un travail commun avec Philippe Aubry (LITP, Paris 6). Philippe Aubry a réalisé une implantation performante des méthodes de Michael Kalkbrener et Dongming Wang. Ces méthodes sont présentées respectivement dans les sections 3 et 4 de ce chapitre par Philippe Aubry.

Pour ma part j'ai programmé la méthode de Ritt et Wu Wen-Tsün rappelée dans la cinquième section et réalisé une première implantation de la méthode de Daniel Lazard en y incorporant certains outils (voir le chpitre 3).

La sixième section présente les solutions éventuellement calculées par nos quatre méthodes sur une trentaine d'exemples. Pour certains exemples, on donne aussi la base de Gröbner pour l'ordre lexicographique (calculée par AXIOM ou GB) , la transformation par l'algorithme *lexTriangular* (voir chapitre 2) de cette dernière lorsqu'il s'agit d'un système de dimension zéro, ainsi que la décomposition primaire (lorsque nous avons pu la calculer).

Dans la septième et dernière section, nous essayons de dégager quelques conclusions de ce travail de longue haleine largement commun avec Philippe Aubry. Cependant nous donnons dès la première section des tableaux récapitulatifs sur les calculs présentés dans la sixième section. Ces tableaux pourront servir de *guide* au lecteur dans ce long chapitre.

1 Tableaux récapitulatifs

Remarque VI.1 Nous allons maintenant donner deux tableaux récapitulatifs sur les exemples précédents : le premier portera sur les exemples de dimension zéro, et le second sur ceux de dimension strictement positive. Soit F l'ensemble de polynômes de départ de l'un des exemples et V l'ensemble de ses variables ordonnées. On examinera la décomposition de $\mathbf{V}(F)$ calculée par chacune des méthodes de Wang, Wu, Kalkbrener et Lazard si une telle décomposition a pu être obtenue. De plus, pour les exemples de dimension zéro, on étudiera la décomposition calculée par l'algorithme lexTriangular à partir de la base lexicographique de F pour l'odre sur les monômes induit par V. Soit $\Sigma = \{\sigma_1, \ldots, \sigma_s\}$ la décomposition en systèmes triangulaires calculée par l'une de ces méthodes. Rappelons que pour chaque méthode, sauf celle de Wang, chaque σ_i est réduit à un ensemble triangulaire T_i. Pour les méthodes de Wu, Lazard et pour lexTriangular, on a :

$$\mathbf{V}(F) = \cup_i \mathbf{ZR}(T_i)$$

Pour la méthode de Kalkbrener, on a :

$$\mathbf{V}(F) = \cup_i \overline{\mathbf{ZR}(T_i)}$$

Enfin, pour celle de Wang, on a :

$$\mathbf{V}(F) = \cup_i \mathbf{ZR}(\sigma_i)$$

Définition VI.1 *Nous dirons que deux ensembles de systèmes triangulaires Σ et Λ de $\mathbf{R}_n$ sont* semblables *si l'on a :*

$$\{\mathsf{sat}(\sigma) \mid \sigma \in \Sigma\} \quad = \quad \{\mathsf{sat}(\lambda) \mid \lambda \in \Lambda\}$$

Remarque VI.2 Dans chacun des tableaux suivantes, pour chaque exemple F et chaque décomposition σ on donnera les informations suivantes (si elles ont pu être obtenues) :

- le temps de calcul,
- une séquence d'entiers naturels $e_1, \ldots, e_s$ où :
  - si F est de dimension zéro, alors e_i est le nombre de zéros réguliers de σ_i,
  - si F est de dimension strictement positive e_i est la dimension de $\mathsf{sat}(\sigma_i)$
- une *étiquette* égale à l'une des minuscules (a), (b), (c), (d) ou (e) telle que deux décompositions semblables de $\mathbf{V}(F)$ aient la même étiquette,
- un ensemble d'étiquettes donnant des propriétés parmi les suivantes :
  - (I) si chaque T_i est initialement réduit,
  - (R) si chaque T_i est réduit,
  - (N) si chaque T_i est normalisé,

Les temps de calcul sont donnés en secondes, sauf mention explicite du contraire. Ils comprennent le temps d'*évaluation* (EV) et le temps de *gestion de la mémoire* (GC). À l'exception des calculs de bases lexicographiques, tous les calculs sont réalisés par les programmes AXIOM que Philippe Aubry et moi-même avons écrit. Les bases lexicographiques sont calculées soit par les librairies AXIOM, soit par le logiciel GB de J.-C. Faugère lorsqu'elles ne peuvent l'être par AXIOM. Lorsqu'un temps est présenté comme une somme, son premier terme est le temps de calcul de la base lexicographique et son second terme est le temps de calcul de la décomposition de $\mathbf{V}(F)$ à partir de ladite base. Lorsque le premier terme est (GB) cela signifie que la base a été calculée par le logiciel GB et le temps est alors de l'ordre de la dizaine de secondes. Cependant pour lexTriangular cette somme a généralement été effectuée. Autrement dit, chaque temps donné dans la colonne lexTriangular du premier tableau, sauf ceux de `Cyclic 6` et `Cyclic 7`, comprend le temps de calcul de la base lexicographique en AXIOM et le temps de calcul de la décomposition par lexTriangular à partir de cette base. Enfin, pour la méthode de Lazard, il y a généralement deux temps de calcul pour chaque exemple. Le premier est alors le temps de calcul avec la version de notre implantation à l'époque où cette comparaison a été faite (été 1996). Le second est celui obtenu par notre implantation actuelle. Lorsque les deux temps sont très proches, on ne donne que le premier. L'optimisation entre ces deux versions est la suivante. l'opération squareFreePart de la proposition V.10 p. 161 a été transformée en une opération squareFree (pour reprendre les noms AXIOM) calculant une *factorisation sans facteurs carrés* au lieu d'une *partie sans carrés*. Pour cela nous avons adapté l'algorithme exposé dans l'appendice de [DST93]. Nous avions tout d'abord essayé d'implanter celui de Yun ([Yun76]). Cependant, notre notion de *polynôme primitif* (définition III.1 p. 79) semble trop *faible*. En effet, deux polynômes associés et primitifs (dans notre sens) peuvent différer d'un facteur lui-même polynômial.

Notations VI.1 Pour alléger la présentation des résultats, voici quelques notations.

- Pour un exemple F de dimension zéro à la place d'une séquence d'entiers naturels $e = e_1, \ldots, e_s$ telle que :

$$e_{i_1} = \cdots = e_{i_1+s_1-1} < e_{i_2} = \cdots = e_{i_2+s_2-1} < \cdots < e_{i_k} = \cdots = e_{i_k+s_k-1}$$

$$\text{avec : } i_1 = 1, \ \ i_j + s_j = i_{j+1} \text{ et } i_k + s_k - 1 = s$$

  on écrira plutôt : $e_{i_1}^{s_1}, e_{i_2}^{s_2}, \ldots, e_{i_k}^{s_k}$.
- Pour un exemple F de dimension strictement positive, dans la séquence d'entiers naturels $e = e_1, \ldots, e_s$ donnant les dimensions des saturés on indicera les dimensions de façon à se référer à la composante correspondante. De plus, si $\mathsf{sat}(\sigma_i)$ contient $\mathsf{sat}(\sigma_j)$, alors on fera suivre la dimension indicée de $\mathsf{sat}(\sigma_i)$ de $\subseteq$ puis de la dimension indicée de $\mathsf{sat}(\sigma_j)$. Donnons un exemple. La séquence :

$$1_a, 1_b \subseteq 2_a, 1_c \subseteq 2_a, 2_a, 2_b$$

  se lit de la manière suivante. La décomposition calculée comporte trois idéaux saturés de dimension 1 et deux idéaux saturés de dimension 2. De plus, deux des idéaux saturés de dimension 1 contiennent le même idéal saturé de dimension 2. Ceci signifie que la fermeture des zéros de deux des systèmes triangulaires est contenue dans la fermeture des zéros de l'un des autres systèmes triangulaires.
- Nous combinerons parfois les deux systèmes de notations précédents. Plus précisément, pour un exemple F de dimension strictement positive, si tous les idéaux saturés de dimension d contiennent l'un des idéaux saturés de dimension δ on écrira :

$$\ldots, d^k \subseteq \delta_a, \ldots, \delta_b, \ldots$$

  au lieu de :

$$\ldots, d_1 \subseteq \delta_a, d_2 \subseteq \delta_a, \ldots, d_k \subseteq \delta_a, \ldots, \delta_a, \delta_b, \ldots$$

- Enfin, lorsqu'une information n'a pu être obtenue dans un temps *raisonnable* nous remplacerons cette information par un point d'interrogation.

Exemple	lexTriangular	Wang	Wu	Kalkbrener	Lazard
easyExample 4	2.56	1.07	0.57	0.15	0.17
p. 235	$2^2, 3^2$	$2^2, 3^2$	$0^4, 2^2, 3^2$	$1^3, 2^2, 3$	$2^2, 3^2$
	(a, N, R)	(a)	(b, R)	(c, N, R)	(a, N, R)
easyExample 5	0.66	0.25	0.13	0.27	0.13
p. 238	4^2	$2, 3$	$2, 4$	$1^3, 2$	$2, 3$
	(a, N, R)	(b, I, N)	(c, R)	(d)	(b, N, R)
Solotare	0.53	1.32	0.35	0.25	0.50 | 0.23
p. 253	4	4	4	$2, 2$	4
	(a, N, R)	(a, I)	(a, R)	(b, N, R)	(a, N, R)
Moeller 4	0.57	2.72	0.30	1.22	1.80
p. 248	8	8	8	8	8
	(a, N, R)	(a, I)	(a, I)	(a, I)	(a, N, R)
Trinks 2	3.23	0.57	0.47	0.82	1.42
p. 267	0	0	0	0	0
	(a, N, R)	(a, N, R)	(a, N, R)	(a, N, R)	(a, N, R)
Trinks diff.	4.96	3.63	126	1.57	317 | 50
p. 282	10	10	10	10	10
	(a, N, R)	(a, I)	(a, I)	(a, I)	(a, N, R)
Katsura 3	0.65	3.35	7.50	3.02	170 | 131
p. 276	8	10	10	1,7	8
	(a, N, R)	(b, I)	(b, I)	(c, I)	(a, N, R)
easyExample 3	0.88	0.63	?	21.95	0.38
p. 289	$1, 120$	$1, 120$		$1, 120$	$1, 120$
	(a, N, R)	(a, I)		(a, N, R)	(a, N, R)
Caprasse	24	88	?	4.02	15.41 | 3.38
p. 293	$6, 16^3, 18$	$2^2, 4, 16^2$		$2, 6, 8, 16$	$2^2, 4^3, 8^2$
	(a, N, R)	(b, I)		(c, I)	(d, N, R)
Points	1.05	2.50	?	2.50	250 | 0.52
Singuliers	4^2	4^2		$(1, 3, 4)$	$(1, 3, 4)$
p. 301	(a, N, R)	(b, I)		(c, I)	(d, N, R)
Cyclic 5	22	?	?	1808	66 | 13.62
p. 297	$10^2, 20, 30$			$10^5, 20$	$10^5, 20$
	(a, N, R)			(b, I)	(c, N, R)
Cyclic 6	GB + 0.80	?	?	?	GB + (35 | 7)
Cyclic 7	GB + 1h	?	?	?	?

Exemple	Base Lex	Wang	Wu	Kalkbrener	Lazard
Discrimant	0.82	6.91	6.43	1.30	2.33
en $d^0 = 4$		$-1, 1 \subset 2_a, 2_b$	$1 \subset 2_a, 2_b$	$1 \subset 2_a, 2_b$	2^2
p. 240		(a)	(b, R)	(b, R)	(c, N, R)
Cyclic-4	0.35	0.72	0.35	0.13	0.27
p. 246		$1^2 \subset 2$	$1^2 \subset 2$	1	1
		(a, R)	(a, R)	(b, N, R)	(b, N, R)
Buchberger	0.33	1.57	0.47	1.33	0.90
p. 255		$2^3 \subset 4$	$2^3 \subset 4$	$4, 4$	$2^3 \subset 4$
		(a, R)	(a, R)	(b, N, R)	(a, N, R)
Donati	9.55	0.75	0.90	0.67	0.77
Traverso		$0^2 \subset 1$	$0^2 \subset 1$	1	$0^2 \subset 1$
p. 259		(a, R)	(a, R)	(b, R)	(a, N, R)
Alonso	0.35	1.90	0.35	0.25	0.45
p. 263		$2^2 \subset 3$	$2^2 \subset 3$	3	$2^2 \subset 3$
		(a, I, N)	(a, N, R)	(b, I, N)	(a, N, R)
Rob. Plano	0.37	3.97	9.10	1.52	0.94
fácil		$0^2 \subset 2_a, 1^2 \subset 2_a$	$0^2 \subset 2_a, 1^2 \subset 2_a$	2	$0 \subset 2_a, 1 \subset 2_a$
p. 265		(a, N, R)	(a, N, R)	(b)	(c, N, R)
Teorema	0.50	3.30	1.15	2.33	1.63
de Euler		$2^3, 3$	$2^3, 3$	$2^3, 3$	$2^3, 3$
p. 268		(a)	(a, R)	(a)	(a, N, R)
Wang	1.90	8.03	13.62	5.40	13.00 | 7.33
p. 271		1^2	1^2	1^2	1^4
		(a)	(a, R)	(a)	(b, N, R)
Wu	0.50	0.40	0.13	0.59	111.92 | 1.85
p. 279		$0 \subset 1$	$0 \subset 1$	1	$0^6 \subset 1$
		(a, R)	(a, R)	(b, I)	(c, N, R)
Rob. Plano	1.23	8.27	293	18.08	42 | 10.83
difícil		$2^4 \subset 4_a, 3^2 \subset 4_a$	$2^4 \subset 4_a, 3^2 \subset 4_a$	4	$2^3 \subset 4_a, 3^3 \subset 4_a$
p. 286		(a)	(a, R)	(b)	(c, N, R)
f633	45	32.60	?	?	109 | 180
p. 303		$(\ldots, 2)$			$0^2, 1^2, 2$
		(a)			(b, N, R)
Robot	3.45	27	22	80	3.45 + (125 | 26)
Romin		?	?	5	$2^2, 3^{11}, 4^6, 5$
p. 308		?	?	(a, I)	(b, N, R)
Neural	GB	?	?	?	GB + 136
Network					$0^2, 1^2$
p. 312					(a, N, R)
Liu	GB	?	?	?	(GB + 2h) ou 10h
Lorenz					(a, N, R)
p. 143					$(\ldots, 1)$

2 Implantation des méthodes

Nous allons présenter dans cette section les librairies AXIOM constituant le cadre commun dans lequel nous avons implanté les méthodes (de résolution d'un système d'équations algébriques sous forme d'ensembles triangulaires) dues à Daniel Lazard, Michael Kalkbrener, Dongming Wang et Wu Wen-Tsün. Cette section s'adresse plus particulièrement aux lecteurs familiers avec le logiciel AXIOM ([JS92]). De plus cette partie est probablement un peu technique, longue et moins rédigée que les sections suivantes. Elle permet cependant de définir les objets informatiques (i.e. les constructeurs AXIOM de catégories et domaines) mis en œuvre tout au long de ce chapitre. Elle montre peut-être aussi l'une de nos motivations essentielles dans cette comparaison expérimentale des quatre méthodes : faire que leurs implantations ne diffèrent que par les principes de leurs algorithmes.

Nous allons tout d'abord préciser ce que sera l'implantation d'une des méthodes de résolution d'un système d'équations algébriques sous forme d'ensembles triangulaires. Comme nous le reverrons plus loin, à chacune de ces méthodes est associée une certaine *notion* plus ou moins *riche* d'ensemble triangulaire. tant donnés une méthode et un domaine de polynômes `P`, soit `T` le domaine des ensembles triangulaires de polynômes de `P` au sens de ladite méthode. L'implantation de cette méthode sera donnée par la définition d'une opération exportée par `T` et définie comme suit :

```
zeroSetSplit : List P -> List $
    ++ zeroSetSplit(lp) returns a list \spad{lts} of triangular sets
    ++ such that the zero set of \spad{lp} is the union (of the
    ++ closures) of the regular zero sets of the members of \spad{lts}.
```

Pour cela, il est bien sr commode de se donner un constructeur de catégories d'ensembles triangulaires. Ce constructeur, que nous avons appelé `TSETCAT` (en abrégé), exporte également l'opération de *vérification d'un calcul*, opération qui est déclarée comme suit :

```
zeroSetSplit? : (List $, List P) -> Boolean
    ++ zeroSetSplit?(lts,lp) returns true iff the zero set of
    ++ \spad{lp} is the union of the closures of the regular
    ++ zero sets of the members of \spad{lts}.
```

Cette opération est définie dans notre constructeur `TSETCAT` par :

```
zeroSetSplit? (lts:(List $),lq:(List P)) ==
  llp : List List P := [saturate(ts)$$ for ts in lts]
  lps : List GPS := [construct(lp)$GPS for lp in llp]
  lps := eliminateRedundantVarieties(lps)$GPS
  varietySplit?(lps,lq)$GPS
```

Dans cette définition `GPS` est un domaine d'ensembles (non nécessairement triangulaires) de polynômes de `P`. L'introduction d'un constructeur de domaines d'ensembles quelconques de polynômes est en effet nécessaire pour implanter les opérations sur les idéaux et les variétés. Nous n'avons pas réutilisé le constructeur `PolynomialIdeals` d'AXIOM pour des raisons qui apparaîtront plus loin. Donnons maintenant le plan de la suite de cette section.

Nous décrivons tout d'abord rapidement quelques constructeurs d'anneaux de polynômes ou de fractions. Puis nous présentons un paquetage AXIOM permettant d'effectuer diverses opérations relatives aux idéaux et aux variétés affines à l'aide des techniques fondées sur les bases de Gröbner et du logiciel GB de J.-C. Faugère. Nous passons ensuite à un constructeur d'arbres orientés fournissant une structure de données dans deux des quatre méthodes. Nous terminons avec les constructeurs de catégories ou domaines d'ensembles triangulaires ou non communs aux quatre méthodes.

2.1 Polynômes

Le logiciel AXIOM possède de nombreux constructeurs de catégories ou domaines d'anneaux de polynômes. Nous avons cependant été conduits à définir une catégorie `RPOLCAT` de polynômes afin de :

- disposer des opérations élémentaires sur les polynômes *vus récursivement* (voir la sous-section 6.2. du permier chapitre) comme init, mvar, lazyprem ...
- afficher ces polynômes de façon récursive tout en se gardant la possibilité de les représenter (de façon interne) sous forme récursive ou distribuée (en utilisant l'ordre de son choix sur les monômes).

Ainsi, chacune des quatre méthodes de résolution d'un système d'équations algébriques sous forme d'ensembles triangulaires utilise les mêmes polynômes et en particulier les mêmes algorithmes pour les opérations de réduction d'un polynôme par rapport à un autre.

Voici un extrait de la définition du constructeur `RPOLCAT` :

```
)abbrev category RPOLCAT RecursivePolynomialCategory

RecursivePolynomialCategory(R:Ring,
 E:OrderedAbelianMonoidSup, V:OrderedSet): Category ==
  PolynomialCategory(R, E, V) with
     mvar : $ -> V
         ++  mvar(p) returns an error if p belongs to R, else returns its main
         ++  variable w. r. t. to the total ordering on the elements in V.
     ..............................................................
     initiallyReduce: ($, $) ->  $
         ++ initiallyReduce(a,b) returns a polynomial r such that
         ++ initiallyReduced?(r,b) holds and there exists an integer e
         ++ such that \spad{init(b)^e a - r} is zero modulo b
     ..............................................................
     if R has IntegralDomain
     then
       subResultantPRS : ($, $) -> List $
           ++ subResultantPRS(a,b) where a and b are not contant polynomials
           ++ and have the same main variable returns the complete PRS
           ++ computed to get \spad{resultant(a,b)}. The last term of this
           ++ sequence is \spad{resultant(a,b)} while both first are a and b.
     ..............................................................
     if (V has ConvertibleTo(Symbol))
     then
       CoercibleTo(Polynomial R)
       ConvertibleTo(Polynomial R)
     ..............................................................
```

À l'aide de ce constructeur de catégories, nous avons défini deux constructeurs de domaines :

- l'un, `NSMP`, utilisant une représentation récursive :

```
)abbrev domain NSMP NewSparseMultivariatePolynomial

NewSparseMultivariatePolynomial(R,V) : Exports == Implementation where
  R:Ring
  V:OrderedSet
```

```
  SMPR ==> SparseMultivariatePolynomial(R, V)
  Exports == Join(RecursivePolynomialCategory(R,IndexedExponents V, V),
                  CoercibleTo(SMPR),RetractableTo(SMPR))

  Implementation ==  SparseMultivariatePolynomial(R, V) add

     D := SparseUnivariatePolynomial($)
     VPoly:=  Record(v:V,ts:D)
     Rep:= Union(R,VPoly)

     mvar p ==
       p case R => error" Error in mvar from NSMP : #1 has no variables."
       p.v
     ..........................................................................
```

- l'autre NDMP une représentation distribuée suivant l'ordre lexicographique pure :

```
)abbrev domain NDMP NewDistributedMultivariatePolynomial

NewDistributedMultivariatePolynomial(R,vl) : Exports == Implementation where

  R  : Ring
  vl : List Symbol
  N ==> NonNegativeInteger
  E ==> DirectProduct(#vl,N)
  V ==> OrderedVariableList(vl)
  DMPR ==> DistributedMultivariatePolynomial(vl,R)
  NSMPR ==> NewSparseMultivariatePolynomial(R,V)
  Vec ==> Vector(N)

  Exports == Join(RecursivePolynomialCategory(R,E,V),
                  CoercibleTo(DMPR),RetractableTo(DMPR),
                  CoercibleTo(NSMPR),RetractableTo(NSMPR))

  Implementation ==  DistributedMultivariatePolynomial(vl,R) add
     Term := Record(k:E,c:R)
     Rep := List Term

     localMvar (v : Vec) : Z ==
       i := 1$Z
       while (zero?(v.i)$N) and (i < n) repeat i := i + 1
       zero?(v.i)$N => error"in localMvar from NDMP : ground #1"
       i
     mvar p ==
       zero?(p) => error"in mvar from NDMP : #1 is zero."
       t : Term := first(p pretend Rep)
       v : Vec := t.k pretend Vec
       i := localMvar(v)
       variable(vl.i)::V
     ..........................................................................
```

2.2 Dénominateurs

Dans l'une de nos méthodes (celle décrite dans le chapitre 3) ainsi que dans l'algorithme *lexTriangular* (voir chapitre 2) il est nécessaire d'inverser (si possible) des quantités algébriques modulo une tour d'extensions simples. Ceci conduit à manipuler des fractions. D'autre part, dans les deux cas, on utilise une *version étendue d'un algorithme de pgcd* où les *coefficients de Bezout* sont également des fractions. Pour que la gestion de ces fractions ne soit pas trop coûteuse, nous avons défini (entre autre) divers constructeurs de catégories et domaines d'ensembles de dénominateurs et de fractions. Pour les dénominateurs, l'idée essentielle et naturelle est d'essayer de rendre peu coûteuses les opérations de multiplication, division exacte et pgcd en représentant les objets sous une forme *paresseusement factorisée* (voir plus loin le constructeur `LAZFAC`). Nous donnons dans cette sous-section les principaux constructeurs relatifs aux ensembles de dénominateurs :

- un constructeur de catégories pour les parties multiplicativement stables d'un anneau :

```
)abbrev category MUSECAT MultiplicativeSubsetCategory
MultiplicativeSubsetCategory (R : CommutativeRing) :
          Category == Join (Monoid,CoercibleTo(R)) with
    commutative("*")
    retractIfCan : R -> Union($,"failed")
      ++  retractIfCan(r) returns r as an element of $ if it can
      ++  "easily" be retracted to $, else returns "failed".
    retract : R -> $
      ++  retract(r) returns r as an element of $ if it can
      ++  "easily" be retracted to $, else an error is produced.
    "*" : ($,R) -> R
    commonMultiple : ($,$) -> $
    equivalent : (R,$) -> Record(num : R, den : $)
      ++  equivalent(a,s) returns \spad{[b,t]} such that
      ++  there exist u in $ such that \spad{u * (a*t - b*s) = 0}
      ++  More equivalent is "idempotent".
    equivalent : ($,$) -> Record(num : $, den : $)
      ++  equivalent(s,t) returns \spad{[u,v]} such that
      ++  there exist w in $ such that \spad{w * (s*v - t*u) = 0}
      ++  More equivalent is "idempotent".
    if R has IntegralDomain
    then
      noZeroDivisors
      "exquo" : ($,$) -> Union($,"failed")
      commonDivisor : (R,$) -> $
      commonDivisor : ($,$) -> $
      "exquo" : (R,$) -> Union(R,"failed")
     ........................................................................
```

- un constructeur de domaines pour la partie multiplicativement stable engendrée par les éléments non nuls de l'anneau des coefficients d'un anneau de polynômes (à coefficients dans un anneau intègre) :

```
)abbrev domain NZBRING  NonZeroBaseRingMultiplicativeSubSet
NonZeroBaseRingMultiplicativeSubSet(R,E,V,P)  : Exports == Implementation where

  R : IntegralDomain
  E : OrderedAbelianMonoidSup
  V : OrderedSet
```

```
  P : RecursivePolynomialCategory(R,E,V)
  Exports == Join(MultiplicativeSubsetCategory(P),
                  MultiplicativeSubsetCategory(R)) with
    coerce : R -> $
  Implementation == add
    Rep ==> R
     ....................................................................
```

- un constructeur de domaines pour la partie multiplicativement stable engendrée par les éléments non nuls d'un anneau intègre. Un élément d'un tel domaine est représenté par une liste de facteurs élevés à une puissance entière (pouvant être négative dans des calculs intermédiaires) et dont le produit est égal à cet élément ; ce produit n'est évalué que lorsque cela est nécessaire. Nous n'avons réutilisé le constructeur `Factored` d'AXIOM que pour l'affichage. En effet, ce constructeur comporte diverses erreurs, en particulier pour le test d'égalité.

```
)abbrev domain LAZFAC LazyFactored
LazyFactored (R) : Exports == Implementation where

  R : IntegralDomain
  RRZ ==> Record(factor: R,exponent: Integer)
  FR ==> Factored(R)
  LRRZ ==> List Record(factor: R,exponent: Integer)
  Exports == MultiplicativeSubsetCategory (R) with
    coerce : R -> $
      ++  coerce(r) returns r as an element of $, if it is not zero.
    coerce : $ -> FR
      ++  coerce(s) returns s as an element of Factored(R).
    coerce : FR -> $
      ++  coerce(fr) returns fr as an element of $, if it is not zero.
    power : (R, $) -> Z
      ++  power(r,s) returns the power of \spad{r} as a factor
      ++  appearing in the representation of \spad{s}.
    remove : (R, $) -> $
      ++  remove(r,s) divides \spad{s} by \spad{r ^ power(r,s)}
    commonFactors : ($, $) -> $
      ++  commonFactors(s,t) returns the common factors by comparing
      ++  the lists \spad{factors(s)} and \spad{factors(t)}.
      ++  The only operation which is called from \spad{R} is \spad{=}.
      ++  This operation calls only the \spad{=} operation from \spad{R} .
     ....................................................................
  Implementation == add
    Rep ==> LRRZ
    rep(s:$):Rep == s pretend Rep
    per(l:Rep):$ == l pretend $
    reciprocal(s:$):$ ==
      one? s => s
      per [[x.factor, -x.exponent]$RRZ for x in rep(s)]
     ....................................................................
    expand(lx:Rep):R ==
      r : R := 1
      while not empty? lx repeat
        x := first lx
        x.exponent < 0 =>
           error "in expand : $ -> R from LazyFactored  : bad #1"
```

```
    lx := rest lx
    r := r *$R (x.factor **$R (x.exponent)::N)
  r
 ..........................................................................
(s:$) exquo (t:$) ==
  one? t => s::Union($,"failed")
  cf := commonFactors(s,t)
  lx := rep(s * reciprocal(cf))
  ly := rep(t * reciprocal(cf))
  empty? ly => per(lx)::Union($,"failed")
  r : R := expand(lx)
  u : Union(R,"failed")
  while not empty? ly repeat
    y := first ly
    ly := rest ly
    p := y.exponent
    while p > 0 repeat
      u := r exquo$R y.factor
      not (u case R) => return("failed"::Union($,"failed"))
      r := u::R
      p := p - 1
  (r::$)::Union($,"failed")
 ..........................................................................
(s:$) * (t:$) ==
  one? s => t
  one? t => s
  lx := rep(s)
  ly := rep(t)
  lrrz :  Rep := []
  while (not empty? lx) and (not empty? ly) repeat
    x := first lx
    lx := rest lx
    ye := power(x.factor, per(ly))
    p := x.exponent + ye
    if p ~= 0 then lrrz := cons([x.factor,p],lrrz)
    if ye ~= 0 then ly := rep remove(x.factor, per(ly))
    (not empty? ly) =>
      y := first ly
      ly := rest ly
      xe := power(y.factor,per(lx))
      p := y.exponent + xe
      if p ~= 0 then lrrz := cons([y.factor,p],lrrz)
      if xe ~= 0 then lx := rep remove(y.factor,per(lx))
  (empty? lx) and (empty? ly) => per(reverse lrrz)
  empty? lx => per(concat!(reverse lrrz, ly))
  per(concat!(reverse lrrz, lx))
 ..........................................................................
(s:$) = (t:$) ==
  cf := commonFactors(s,t)
  lx := rep(s * reciprocal(cf))
  ly := rep(t * reciprocal(cf))
  x : R := 1
  y : R := 1
```

```
      while (not empty? lx) and (not empty? ly) repeat
        x := x * expand([first lx])
        y := y * expand([first ly])
        (lx,ly) := (rest lx, rest ly)
        if x = y then (x,y) := (1$R,1$R)
      (x * expand(lx)) =$R (y * expand(ly))
```

- un constructeur de domaines pour la partie multiplicativement stable engendrée par les éléments non nuls d'un anneau intègre de polynômes :

```
)abbrev domain LZPRING LazyFactoredPolynomialRing
LazyFactoredPolynomialRing (R,E,V,P) : Exports == Implementation where

  R : IntegralDomain
  E : OrderedAbelianMonoidSup
  V : OrderedSet
  P : RecursivePolynomialCategory(R,E,V)
  RRLFP ==> Record(base:R,pols:LazyFactored(P))

  Exports == MultiplicativeSubsetCategory (P) with
     ..................................................................
    if R has GcdDomain
    then
      gcd : ($, $) -> $
        ++ gcd(s,t) returns a gcd of s and t
      gcd : (R, $) -> R
        ++ gcd(r,t) returns a gcd of r and t
      gcd : (P,$) -> $
        ++ gcd(p,s) returns a gcd of p and s
     ..................................................................
```

Dans ce constructeur, nous avons réimplanté l'algorithme de calcul du pgcd de deux polynômes multivariés fondé sur celui des sous-résultants en exploitant la *forme paresseusement factorisée* des objets. Nous ne reproduisons pas ici le code correspondant qui est long et technique, mais nous donnons ci-après une session AXIOM illustrant ce travail. Dans cette session, comme dans les suivantes, le type du résultat, ainsi que le temps de calcul, de certaines commandes ne sont pas reproduits ici afin d'aérer ce texte. De plus, rappelons que si une commande se termine par un point-virgule, son résultat n'est pas affiché. Ainsi, les commandes `(7)` à `(15)` ci-après définissent des polynômes `f1` à `f9` de l'anneau `P` sans les *retourner*.

```
(1) ->R := Integer

   (1)  Integer

(2) ->ls : List Symbol := [X,Y,Z]

   (2)  [X,Y,Z]

(3) ->V:= OVAR(ls)

   (3)  OrderedVariableList [X,Y,Z]

(4) ->E := IndexedExponents V
```

```
   (4)  IndexedExponents OrderedVariableList [X,Y,Z]

(5) ->P := NSMP(R, V)

   (5)  NewSparseMultivariatePolynomial(Integer,OrderedVariableList [X,Y,Z])

(6) ->S := LZPRING(R,E,V,P)

   (6)
   LazyFactoredPolynomialRing(Integer,IndexedExponents
   OrderedVariableList [X,Y,Z],OrderedVariableList [X,Y,Z],
   NewSparseMultivariatePolynomial(Integer,OrderedVariableList [X,Y,Z]))

(7) ->f1 := (X-1)::P;
(8) ->f2 := (X+1)::P;
(9) ->f3 := (X-2)::P;
(10) ->f4 := (X+2)::P;
(11) ->f5 := (Y-3)::P;
(12) ->f6 := (Y+3)::P;
(13) ->f7 := (Z + X + Y)::P;
(14) ->f8 := (Z - X + Y)::P;
(15) ->f9 := (Z + X - Y)::P;

(16) ->a :=  1::S

   (16)  1

(17) ->a := a * (f1 * f2)::S

          2
   (17)  X  - 1

(18) ->a :=  a * (2 * f1 * f3)::S * f4::S * f5::S

                              2            2
   (18)  2(Y - 3)(X + 2)(X  - 3X + 2)(X  - 1)

(19) ->b := f2::S * (2 * f1 * f5)::S * (-f4)::S

   (19)  - 2(X + 1)(X + 2)((Y - 3)X - Y + 3)

(20) ->b := f2::S * b

                    2
   (20)  - 2(X + 1) (X + 2)((Y - 3)X - Y + 3)

(21) ->pa := a::P

   (21)
              5              4                3              2
     (2Y - 6)X  + (- 2Y + 6)X  + (- 10Y + 30)X  + (10Y - 30)X  + (8Y - 24)X - 8Y
   +
```

```
     24
  Type: NewSparseMultivariatePolynomial(Integer,OrderedVariableList [X,Y,Z])

(22) ->pb := b::P

                 4                3              2
  (22)  (- 2Y + 6)X  + (- 6Y + 18)X  + (- 2Y + 6)X  + (6Y - 18)X + 4Y - 12

(23) ->c := commonFactors(a,b)

  (23)  X + 2

                         Time: 0.02 (IN) + 0.02 (EV) + 0.02 (OT) = 0.05 sec
(24) ->exquo(a,c)::S

                2            2
  (24)  2(Y - 3)(X  - 3X + 2)(X  - 1)

  Type: LazyFactoredPolynomialRing(Integer,IndexedExponents
  OrderedVariableList [X,Y,Z],OrderedVariableList [X,Y,Z],
  NewSparseMultivariatePolynomial(Integer,OrderedVariableList [X,Y,Z]))
                                     Time: 0.03 (IN) + 0.02 (OT) = 0.05 sec
(25) ->gcd(pa,pb)

                3             2
  (25)  (2Y - 6)X  + (4Y - 12)X  + (- 2Y + 6)X - 4Y + 12

  Type: NewSparseMultivariatePolynomial(Integer,OrderedVariableList [X,Y,Z])
                                     Time: 0.05 (EV) + 0.03 (OT) = 0.08 sec
(26) ->gcd(a,b)

                           2
  (27)  2(Y - 3)(X + 2)(X  - 1)

  Type: LazyFactoredPolynomialRing(...
                                     Time: 0.02 (EV) + 0.03 (OT) = 0.05 sec

(27) ->gcd(pa::S,pb::S)

                3     2
  (27)  2(Y - 3)(X  + 2X  - X - 2)

                                     Time: 0.02 (EV) + 0.05 (OT) = 0.07 sec
(28) ->gcd(pa**2,pb**2)

  (28)
      2             6       2              5      2             4
    (4Y  - 24Y + 36)X  + (16Y  - 96Y + 144)X  + (8Y  - 48Y + 72)X
  +
       2                3         2                2       2
    (- 32Y  + 192Y - 288)X  + (- 28Y  + 168Y - 252)X  + (16Y  - 96Y + 144)X
  +
      2
```

```
    16Y  - 96Y + 144

   Type: NewSparseMultivariatePolynomial(Integer,OrderedVariableList [X,Y,Z])
                             Time: 0.02 (IN) + 0.15 (EV) + 0.10 (OT) = 0.27 sec
(29) ->gcd((pa**2)::S,(pb**2)::S)

             2             6     5     4     3     2
   (29)  4(Y  - 6Y + 9)(X  + 4X  + 2X  - 8X  - 7X  + 4X + 4)

   Type: LazyFactoredPolynomialRing(...
                             Time: 0.07 (IN) + 0.08 (EV) + 0.07 (OT) = 0.22 sec
(30) ->gcd((pa::S)**2,(pb::S)**2)

                    2  6     5     4     3     2
   (30)  4(Y - 3) (X  + 4X  + 2X  - 8X  - 7X  + 4X + 4)

                             Time: 0.07 (IN) + 0.03 (EV) + 0.07 (OT) = 0.17 sec

(31) ->p1  : P := f1 * f2 *f4 * f5 * f7 * f8;

(32) ->q1 : P := f2 * f3 * f5 * f6 * f8 * f9;

(33) ->gcd(p1::S,q1::S)

                    2
   (33)  (Y - 3)(X  + (- Y - Z + 1)X - Y - Z)

   Type: LazyFactoredPolynomialRing(...
                             Time: 0.03 (IN) + 0.12 (EV) + 0.02 (OT) = 0.17 sec
(34) ->gcd(p1,q1)$P

                    2       2                          2
   (34)  (Y - 3)X  + (- Y  + (- Z + 4)Y + 3Z - 3)X - Y  + (- Z + 3)Y + 3Z

   Type: NewSparseMultivariatePolynomial(Integer,OrderedVariableList [X,Y,Z])
                             Time: 0.02 (IN) + 0.67 (EV) + 0.03 (OT) = 0.72 sec
```

2.3 Fractions

Notre principal constructeur de fractions LLCPLR gère les fractions de la façon paresseuse suivante : seules les simplifications *peu coûteuses* peuvent être effectuées systématiquement, les autres étant laissé à l'initiative du programmeur. Plus précisément, si l'on localise un anneau de polynômes à coefficients dans un anneau R à pgcd, sont réputées *peu coûteuses* les simplifications des fractions dont le numérateur et le dénominateur sont dans R ou dans la partie multiplicativement stable par laquelle on localise. Ces fractions **paresseuses** jouent un rôle **crucial** dans l'implantation de la version étendue du calcul du pgcd (fondé sur les sous-résultants) de deux polynômes à coefficients dans une tour d'extensions simples. En effet, les coefficients des polynômes intermédiaires appartiennent à un anneau localisé. Cependant, ces coefficients (qui sont donc des fractions) ne doivent pas être simplifiés pour ne pas perdre l'avantage de l'algorithme des sous-résultants (les divisions exactes des numérateurs de ces fractions par les β_i).

Nous donnons dans cette sous-section les principaux constructeurs relatifs aux ensembles de fractions et illustrons le cas du constructeur LLCPLR :

- un constructeur de catégories pour les anneaux localisés :

```
)abbrev category LOCRNGM LocalizedRingModel
++ Description: A model for a localized commutative ring R,
++ by a multiplicatively closed subset S of R.
++ This should not be viewed as a "mathematical category"
++ but as the common specifications of distinct implementations
++ of \spad{R / S}. So the elements of a domain of this "axiom
++ category" are fractions with numerators from R and
++ denominators from S.
LocalizedRingModel(R:CommutativeRing, S : MultiplicativeSubsetCategory(R)) :
          Category == Join(Monoid,Module(R)) with

    if R has OrderedRing then OrderedRing
    _/ : (R,S) -> $
    _/ :(%,S) -> $
    numer : $ -> R
    denom: $ -> S
    simplify : $ -> $
       ++  \spad{simplify(p/s)} returns \spad{q/t} such that there exists
       ++  an u in S such that \spad{u * (p*t - q*s) = 0}. In other words,
       ++  \spad{p/s} and \spad{q/t} are equal as elements of the domain.
       ++  More, simplify is "idempotent". This operation is of course
       ++  intended to remove common factors from denominators and the
       ++  numerators, when R is an integral domain.
    simplify! : $ -> $
       ++  \spad{simplify!(p/s)} destructevly replaces \spad{p/s}
       ++  by \spad{simplify(p/s)}.
    localize : R -> $
       ++ localize(p) returns p / 1.
    numerator: $ -> $
    denominator: $ -> $
    "*" : (S,$) -> $
    "*" : ($,S) -> $
    retractIfCan : $ -> Union(R,"failed")
    retract : $ -> R
    if R has GcdDomain
    then
      coerce : $ -> Fraction R
      retractIfCan : Fraction R -> Union($,"failed")
      retract :  Fraction R -> $
```

- un constructeur de catégories pour les anneaux de polynômes localisés :

```
)abbrev category LCPLRM LocalizedPolynomialRingModel

++ Description: A model for a \spad{S^(-1) P} where \spad{P} is a polynomial
++ ring over an integral domain \spad{R} and \spad{S} is a multiplicatively
++ closed subset of \spad{P}.

R : IntegralDomain
E : OrderedAbelianMonoidSup
```

```
V : OrderedSet
P : RecursivePolynomialCategory(R,E,V)
S : MultiplicativeSubsetCategory(P)

LocalizedPolynomialRingModel(R,E,V,P,S) :
          Category == LocalizedRingModel(P,S) with

    if P has RetractableTo(R) then RetractableTo(R)
    construct : (R,P,S) -> $
       ++ construct(r,p,s) returns the same as \spad{(r * p)/s}. Thus,
       ++ for fractions managed in a lazy way, this operation does not
       ++ involve any simplification. Note that for such fractions the
       ++ same would not occur with \spad{(r * (p/s)} because this
       ++ expression is understood as \spad{((r/1) * (p/s)} (and then as
       ++ \spad{(p * (r/s)}) and fractions with numerator from \spad{R}
       ++ are always managed in a busy way. If you want to create \spad{r/s}
       ++ (in a lazy fraction domain) without any simplifications simply use
       ++ \spad{construct(r,1,s)}. Note that \spad{p@P * x@$} (in a lazy
       ++ fraction domain) does not involve any simplifications even if
       ++ \spad{p} lies in \spad{R} whereas \spad{ground(p)@R * x@$} does.
    "*" : (R,$) -> $
       ++ r * l returns the product of r and l
    "*" : ($,R) -> $
       ++ l * r returns the product of r and l
    _/ : (R,S) -> $
       ++ r / s divides the element r of R  by the element s of S.
    polPart : $ -> P
      ++ polPart(x) returns the polymomial part of the numerator, which may
      ++ be its primitive part, if R is a gcd domain and if the domain
      ++ constructor keeps numerators in a primitively factored form.
    numPart : $ -> R
      ++ numPart(x) returns the base ring part of the numerator, which may
      ++ ibe ts content w.r.t. R, if R is a gcd domain and if the domain
      ++ constructor keeps numerators in a primitively factored form
    modulo :  (P,P) -> $
      ++ modulo(a,b) returns the image \spad{c} of \spad{a} in the residue class
      ++ ring of \spad{R^(-1) P} by spad{<b>} s.t. \spad{reduced?(polPart(c),b)},
      ++ if \spad{init(b)} can be retracted to \spad{S}, else produces an error.
    "exquo" : ($,P) -> Union($,"failed")
      ++ exquo(l,p) returns the same as \spad{(exquo(numer(l),p)::P /denom(w)}
      ++ if \spad{exquo(numer(l),p)} does not return \spad{"failed"}
      ++ else \spad{"failed"} is returned.
     ..........................................................................
```

- un constructeur de domaines de la catégorie précédente gérant les fractions de façon paresseuse :

```
)abbrev domain LLCPLR LazyLocalizedPolRing
LazyLocalizedPolRing(R,E,V,P,S) : Exports == Implementation where

  R : IntegralDomain
  E : OrderedAbelianMonoidSup
  V : OrderedSet
  P : RecursivePolynomialCategory(R,E,V)
  S : MultiplicativeSubsetCategory(P)
```

```
Z ==> Integer
N ==> NonNegativeInteger
RPS ==> Record(base:R, pol:P, den:S)

Exports == LocalizedPolynomialRingModel(R,E,V,P,S)

Implementation == add

  Rep ==> RPS

  rep(x:$):Rep == x pretend Rep
  per(y:Rep):$ == y pretend $

  construct(r:R,p:P,s:S) ==
    (zero? r) or (zero? p) => 0
    per([r,p,s]$Rep)

  (p:P) / (s:S) ==
    construct(1$R,p,s)

  (s:S) * (x:$) ==
    one? s => x
    zero? x => 0$$
    ss : Record(num:S, den:S) := equivalent(s,rep(x).den)$S
    p : P :=  (ss.num)::P
    ground? p =>  per([rep(x).base * ground(p), rep(x).pol, ss.den])
    per([rep(x).base, rep(x).pol * p, ss.den])

  (p:P) * (x:$) ==
    ground? p =>  per([rep(x).base * ground(p), rep(x).pol , rep(x).den])
    per([rep(x).base, rep(x).pol * p, rep(x).den])
  ......................................................................
```

Nous illustrons ce dernier constructeur à l'aide d'une petite session AXIOM :

```
(1) ->R := Integer;
(2) ->ls : List Symbol := [X,Y,Z];
(3) ->V:= OVAR(ls);
(4) ->E := IndexedExponents V;
(5) ->P := NSMP(R, V);
(6) ->K := Fraction P;
(7) ->S := LZPRING(R,E,V,P);
(8) ->L := LLCPLR(R,E,V,P,S)

   (8)
  LazyLocalizedPolRing(....

(9) ->f10 := 165 * (Z^2 - Y^2)::P;
(10) ->f11 := 225 * (Y^2 - X^2)::P;
(11) ->f12 := 150 * (X^2 - Z^2)::P;
(12) ->f13 := (-11 * (Z - Y) * (Y - X))::P;
(13) ->f14 := (-15 * (Y - X) * (X - Z))::P;
(14) ->f15 := (-10 * (X - Z) * (Z - Y))::P;
(15) ->q1 : L := f10 / f13::S
```

```
                     1                    2       2
   (15)  ------------------------- (- 165Y  + 165Z )
                          2
         - 11((Y - Z)X - Y  + Z Y)

                                      Time: 0.45 (IN) + 0.28 (OT) = 0.73 sec
(16) ->q1::K

         15Y + 15Z
   (16)  ---------
           X - Y
Type: Fraction NewSparseMultivariatePolynomial(Integer,OrderedVariableList [X,Y,Z])
                                      Time: 0.25 (IN) + 0.02 (OT) = 0.27 sec
(19) ->q2 : L := f11 / f14::S

                     1                    2       2
   (19)  ------------------------- (- 225X  + 225Y )
             2
         15(X  + (- Y - Z)X + Z Y)

                                      Time: 0.02 (EV) + 0.07 (OT) = 0.08 sec
(20) ->q1 + q2

   (20)
                 1
     --------------------------
     - 165(Y - Z)(X - Y)(X - Z)
  *
                      2           2        2          3          2        3
     ((2475Y - 2475Z)X  + (- 2475Y  + 2475Z )X - 2475Y  + 4950Z Y  - 2475Z )

                          Time: 0.02 (IN) + 0.03 (EV) + 0.05 (OT) = 0.10 sec

(22) ->simplify(q1 + q2)

                 15          2                  2          2
   (22)  - -------------- (X  + (- Y - Z)X - Y  + Z Y + Z )
           (X - Y)(X - Z)

                          Time: 0.02 (IN) + 0.02 (EV) + 0.05 (OT) = 0.08 sec
(23) ->q1::K + q2::K

              2                     2             2
         - 15X  + (15Y + 15Z)X + 15Y  - 15Z Y - 15Z
   (23)  -------------------------------------------
                     2
                    X  + (- Y - Z)X + Z Y

                          Time: 0.92 (IN) + 0.02 (EV) + 0.03 (OT) = 0.97 sec
```

2.4 Arbres de scindages

Nous décrivons maintenant une structure de données que nous utilisons dans la situation typique suivante. Soit q une certaine quantité à évaluer par une opération eval sous une certaine hypothèse h. Par exemple, soit à calculer la racine carrée de $x + 1$ sachant que la valeur absolue de x est 1. Si l'évaluation de q par eval sachant h requiert une distinction de cas (l'examen de chaque cas pouvant lui-même amener une distinction de cas) il est naturellement commode de *représenter* l'ensemble du calcul par un arbre. Nous ne chercherons pas ici à formaliser ce *type d'évaluation* et nous nous contenterons de présenter deux constructeurs AXIOM. Nous renvoyons à [Gom92] pour un exposé complet sur ce sujet.

Nous allons tout d'abord nous intéresser aux sommets de nos *arbres de scindages*. Chaque sommet contient 3 informations :

1. une *quantité algébrique* q,
2. un ensemble d'hypothèses h,
3. un booléen qui vaut :
   (a) vrai si la quantité algébrique q est le résultat d'une évaluation sous l'ensemble d'hypothèses h,
   (b) faux si la quantité algébrique q est à évaluer sous l'ensemble d'hypothèses h.

Notons qu'en pratique q et h pourront très bien être du même *type informatique*, par exemple : liste de polynômes. Dans ce cas, q pourra être un ensemble de générateurs d'un idéal dont on cherche à extraire une certaine information, tandis que h définira une tour d'extensions simples fournissant les coefficients de nos polynômes. Voici la définition de notre constructeur AXIOM de domaine de sommets. Les trois champs précédents y sont respectivement appelés `value, condition, status`. On notera que le test d'égalité entre sommets ne prend en compte que les deux premiers champs. On notera aussi qu'il existe un sommet particulier que l'on notera $[\emptyset, \emptyset, \mathrm{true}]$ représentant l'abscence de résultat pour un calcul.

```
)abbrev domain SPLNODE SplittingNode
SplittingNode(V,T) : Exports == Implementation where

  V:Join(SetCategory,Aggregate)
  T:Join(SetCategory,Aggregate)
  B ==> Boolean
  VT ==> Record(val:V, tower:T)
  VTB ==> Record(val:V, tower:T, flag:B)

  Exports ==  SetCategory with
     value : % -> V
     condition : % -> T
     status : % -> B
     construct : (V,T,B) -> %
       ++ construct(v,t,b) returns the non-empty node with
       ++ value v, condition t and flag b
     construct : (V,T) -> %
       ++ construct(v,t) returns the same as
       ++ \spad{construct(v,t,false)}
     setValue! : (%,V) -> %
     setCondition! : (%,T) -> %
     setStatus!: (%,B) -> %
```

```
    subNode? : (%,%,(T,T) -> B) -> B

 Implementation == add

    Rep ==> VTB

    n1:% = n2:% ==
       (rep n1).val ~= (rep n2).val => false
       (rep n1).tower = (rep n2).tower

    subNode?(n1,n2,o2) ==
       (rep n1).val = (rep n2).val =>
          o2((rep n1).tower,(rep n2).tower)
       false
    ..................................................................
```

Nous passons maintenant aux *arbres de scindages* : Il s'agit d'arbres orientés avec une et une seule racine et dont les sommets sont du type précédent. On dira que les sommets d'un tel arbre sont *à jour* lorsque la condition suivante est remplie : si tous les successeurs d'un sommet ont `true` pour `status` alors ce sommet a aussi `true` pour `status`. L'opération de signature : updateStatus ! : $\% \longrightarrow \%$ du constructeur `SplittingTree` ci-après réalise cette mise à jour.

Dans nos algorithmes, nous utiliserons les arbres de scindages de la manière suivante. Soit encore q une certaine quantité à évaluer par une opération eval sous une certaine hypothèse h. On effectuera successivement les opérations suivantes :

(1) initialisation d'un arbre $\mathcal{A}$ réduit au seul sommet $[q, h, \text{false}]$. Cette opération s'écrira : $\mathcal{A} := \bullet\ [q, h, \text{false}]$ dans nos algorithmes et `a := [[q,h]]` dans nos programmes.

(2) recherche dans l'arbre $\mathcal{A}$ de la feuille (i.e. sommet pendant) la plus à gauche dont le status est `false`. Si cette recherche est vaine, l'évaluation de q par eval sous `h` est réputée terminée et on passe au point (4). Sinon, soit $\mathcal{L}$ l'arbre réduit à cette feuille. Dans nos algorithmes comme dans nos programmes, cette procédure sera effectuée par l'opération de signature : extractSplittingLeaf : $\% \longrightarrow$ Union(%,"failed")

(3) Soit $\mathcal{L}$ un arbre réduit à un sommet [q,h,false] sous-arbre de l'arbre $\mathcal{A}$. On suppose que l'appel de la fonction eval sur la quantité q sous l'hypothèse `h` retourne une liste ls de sommets 2 à 2 distincts :

$$[[q_1, h_1, b_1], \ldots, [q_n, h_n, b_n]]$$

où les b_i sont des booléens. Tout sommet de la liste ls déjà présent dans $\mathcal{A}$ est effacé de ls. Puis, si la liste ls est vide, le *sommet-racine* de l'arbre $\mathcal{L}$ est *froidement* remplacé par $[\emptyset, \emptyset, \text{true}]$. Si la liste ls n'est pas vide, ses éléments deviennent alors les successeurs immédiats de [q,h,false] dans $\mathcal{L}$ et donc dans $\mathcal{A}$. Enfin, la mise à jour de l'arbre $\mathcal{A}$ est effectuée au moyen de l'opération updateStatus ! présentée plus haut Cette opération s'écrira : $\mathcal{L} := \mathcal{L} \longrightarrow ls$ dans nos algorithmes et `splitNodeOf !(l,a,ls)` dans nos programmes. On revient alors au point (2).

(4) sortie du résultat, à savoir la liste des feuilles de l'arbre $\mathcal{A}$ distinctes de $[\emptyset, \emptyset, \text{true}]$.

Voici les points principaux de la définition de notre constructeur d'*arbres de scindages*. Notez la variante avec paramètre de l'opération `splitNodeOf !` décrite ci-dessus.

```
)abbrev domain SPLTREE SplittingTree
SplittingTree(V,T) : Exports == Implementation where

  V:Join(SetCategory,Aggregate)
  T:Join(SetCategory,Aggregate)
  B ==> Boolean
  VT ==> Record(val:V, tower:T)
  VTB ==> Record(val:V, tower:T, flag:B)
  S ==> SplittingNode(V,T)
  A ==> Record(root:S,subTrees:List(%))

  Exports ==  RecursiveAggregate(S) with
     shallowlyMutable
     finiteAggregate
     extractSplittingLeaf : % -> Union(%,"failed")
     updateStatus! : % -> %
     construct : S -> %
     result : % -> List VT
     nodeOf? : (S,%) -> B
       ++ nodeOf?(s,a) returns true iff some node of \spad{a}
       ++ is equal to \spad{s}
     subNodeOf? : (S,%,(T,T) -> B) -> B
       ++ subNodeOf?(s,a,sub?) returns true iff for some node
       ++ \spad{n} in \spad{a} we have
       ++ \spad{status(n)} and \spad{subNode?(s,n,sub?)}.
     splitNodeOf! : (%,%,List(S)) -> %
     splitNodeOf! : (%,%,List(S),(T,T) -> B) -> %
  Implementation == add

     Rep ==> A

     rep(n:%):Rep == n pretend Rep
     per(r:Rep):% == r pretend %

     construct(s:S) ==
        per [s,[]]$A
     setchildren!(a:%,la:List(%)) ==
         -- operation inherited from RecursiveAggregate(S)
        (rep a).subTrees := la
        a
     setvalue!(a:%,s:S) ==
         -- operation inherited from RecursiveAggregate(S)
        (rep a).root := s
        s
     subNodeOf?(s:S,a:%,sub?:((T,T) -> B)) ==
       empty? a => false
       status(value(a)$%)$S and subNode?(s,value(a),sub?)$S => true
       la := children(a)
       while (not empty? la) and (not subNodeOf?(s,first la,sub?)) repeat
          la := rest la
       not empty? la

     splitNodeOf!(l:%,a:%,ls:List(S)) ==
```

```
      ln := removeDuplicates ls
      la : List % := []
      while not empty? ln repeat
         if not nodeOf?(first ln,a)
           then
             la := cons([first ln]$%, la)
         ln := rest ln
      la := reverse la
      setchildren!(l,la)$%
      if empty? la then (rep l).root := [empty()$V,empty()$T,true]$S
      updateStatus!(a)

    splitNodeOf!(l:%,a:%,ls:List(S),sub?:((T,T) -> B)) ==
      ln := removeDuplicates ls
      la : List % := []
      while not empty? ln repeat
         if not subNodeOf?(first ln,a,sub?)
           then
             la := cons([first ln]$%, la)
         ln := rest ln
      la := reverse la
      setchildren!(l,la)$%
      if empty? la then (rep l).root := [empty()$V,empty()$T,true]$S
      updateStatus!(a)
    ....................................................................
```

2.5 Le paquetage GBLUP

Le nom complet de ce paquetage est `GBLazyUserPackage`. En effet, son but initial est d'utiliser GB depuis AXIOM d'une manière aussi souple que MAPLE . Plus précisément, on souhaite manipuler des systèmes quasi-algébriques dans le mode interprété d' AXIOM avec les facilités suivantes :

- ne pas avoir à typer (explicitement) les polynômes,
- pouvoir rajouter ou enlever une équation ou une inéquation dans un système,
- pouvoir préciser l'ordre des variables pour chaque système ou laisser faire la règle *naturelle* suivante : la dernière indéterminée introduite dans un système est plus grande que toutes les autres,
- remplacer l'ensemble des polynômes définissant les équations par une base de Gröbner (pour l'ordre degré-lexico-inverse ou l'ordre lexicographique pure), puis simplifier l'ensemble des polynômes définissant les inéquations par rapport à cette base,
- tester si un système est contradictoire, inconsistant, ...

Voici une première petite session illustrant ce paquetage. Son seul paramètre est l'anneau des coefficients des polynômes. Celui-ci doit être un anneau factoriel effectif `R` (i.e. un `GcdDomain` dans la terminologie AXIOM). Les idéaux de polynômes sont toujours vus dans l'anneau des polynômes à coefficients dans le corps des fractions de `R`.

```
(14) ->)ca
   All user variables and function definitions have been cleared.
```

```
(1) ->R := Integer;
(2) ->pa := GBLUP(R)

   (2)  GBLazyUserPackage Integer

(3) ->f1 := x^2 + y + z - 1 ;

                 2
   (3)  z + y + x  - 1
                                                         Type: Polynomial Integer
(4) ->f2 := x + y^2 + z  -1 ;
(5) ->f3 := x + y + z^2 -1 ;
(6) ->f4 := x * (x -1) ;
(7) ->f5 := x * (x -1) * (x^2 + 2*x - 1) ;
(8) ->leq := [f1,f2,f3]

                  2          2          2
   (8)  [z + y + x  - 1,z + y  + x - 1,z  + y + x - 1]
                                                    Type: List Polynomial Integer
                                           Time: 0.02 (IN) + 0.02 (OT) = 0.03 sec
(9) ->syst := system(leq,[])$pa

   (9)
                   2          2          2
   [leq= [z + y + x  - 1,z + y  + x - 1,z  + y + x - 1], bs= false, lnoneq= [],
    lvar= [z,y,x], si= false]
Type: Record(leq: List Polynomial Integer,bs: Boolean,
              lnoneq: List Polynomial Integer,lvar: List Symbol,si: Boolean)
                                           Time: 0.02 (IN) + 0.07 (OT) = 0.08 sec
(10) ->unknowns(syst)$pa

   (10)  [z,y,x]
                                                                Type: List Symbol
(11) ->equalities(syst)$pa

                   2          2          2
   (11)  [z + y + x  - 1,z + y  + x - 1,z  + y + x - 1]
                                                    Type: List Polynomial Integer
(12) ->inequalities(syst)$pa

   (12)  []
                                                    Type: List Polynomial Integer
```

Notez le type de la variable `syst`. Le paquetage `GBLUP` code les systèmes quasi-algébriques comme des enregistrements à cinq champs :

- le premier `leq` donne la liste *actuelle* des équations,
- le second `bs` est un booléen qui vaut `vrai` si le premier champ est la base pour l'ordre du degré-lexicographique-inverse (exemple I.1 et exemple I.1 p. 21) de l'idéal qu'il engendre,
- le troisième `lnoneq` donne la liste *actuelle* des inéquations,
- le quatrième donne le liste des variables en ordre décroissant

– le dernier `si` est un booléen qui vaut `vrai` si les polynômes définissant les inéquations sont simplifiés (i.e. en forme normale, voir théorème I.1) par rapport aux polynômes définissant les équations.

Dans la commande suivante, on demande de remplacer la liste des équations par la base degré-lexicographique-inverse, puis, de remplacer les polynômes définissant les inéquations par leurs formes normales par rapport à cette dernière. Sur cet exemple, il n'y a rien à faire, à part changer la valeur du champs `bs`. En effet, les polynômes définissant les équations forment déjà la base degré-lexicographique-inverse de l'idéal qu'ils engendrent. D'autre part, il n'y a pas d'inéquations.

```
(13) ->simplify(syst)$pa
    GB time: 0.10000000000000001
   (13)
          2                    2                    2
   [leq= [z  + y + x - 1,z + y  + x - 1,z + y + x  - 1], bs= true, lnoneq= [],
    lvar= [z,y,x], si= true]

(14) ->contradictory?(syst)$pa
    GB time: 0.11666700000000001
   (14)  false
```

On demande maintenant la base lexicographique. Puis on change l'ordre des variables et on demande la nouvelle base lexicographique.

```
(15) ->lexNormalForm syst
    GB time: 0.10000000000000001
   (15)
                      2      2        2       2     4    2  6     4     3    2
   [leq= [z + y + x  - 1,y  - y - x  + x,2x y + x  - x ,x  - 4x  + 4x  - x ],
    bs= false, lnoneq= [], lvar= [z,y,x], si= true]

(16) ->setUnknownListInSyst!(syst,[x,y,z])

   (16)
                      2          2          2
   [leq= [z + y + x  - 1,z + y  + x - 1,z  + y + x - 1], bs= false, lnoneq= [],
    lvar= [x,y,z], si= false]

(17) ->lexNormalForm syst
    GB time: 0.11666700000000001
   (17)
          2                2        2      4             2  6     4     3    2
   [leq= [z  + y + x - 1,- z  + z + y  - y,z  + (2y - 1)z ,z  - 4z  + 4z  - z ],
    bs= false, lnoneq= [], lvar= [x,y,z], si= true]
```

Dans ce qui suit, on ajoute une inéquation. Puis on demande l'idéal saturé correspondant (définition III.25). Ceci la donne la fermeture des zéros du système (théorème III.12) Comme nous sommes ici en dimension zéro, cela donne en fait, exactement les zéros du système. Le procédé de calcul de l'idéal saturé repose directement sur la formule de la proposition I.27.

```
(19) ->newNonEqInSyst!(syst,[f4])$pa
   (19)
                      2          2          2
   [leq= [z + y + x  - 1,z + y  + x - 1,z  + y + x - 1], bs= false,
              2
    lnoneq= [x  - x], lvar= [x,y,z], si= false]
```

```
(20) ->saturate(syst)$pa
    GB time: 0.10000000000000001    GB time: 0.0    GB time: 0.16666700000000001
           2
   (20)  [z  + 2z - 1,- z + x,- z + y]
                                                      Type: List Polynomial Integer
                                   Time: 0.02 (IN) + 1.97 (EV) + 0.25 (OT) = 2.23 sec
```

On demande maintenant une base de l'idéal d'élimination en `x`. On rajoute alors une nouvelle inéquation. On simplifie le nouveau système. Enfin on constate qu'il est inconsistent sans pour autant être contradictoire (définition III.18) En effet, aucun des polynômes définissant les inéquations ne se trouve dans l'idéal des équations, mais leur produit y appartient.

```
(21) ->elimIdeal(leq,[x::Symbol])$pa
    GB time: 0.10000000000000001
           6     4     3    2
   (21)  [x  - 4x  + 4x  - x ]

(22) ->newNonEqInSyst!(syst,[x^2 + 2*x - 1])$pa
   (22)
                   2          2          2
   [leq= [z + y + x  - 1,z + y  + x - 1,z  + y + x - 1], bs= false,
              2      2
    lnoneq= [x  - x,x  + 2x - 1], lvar= [x,y,z], si= false]

(23) ->simplify(syst)$pa
    GB time: 0.083333000000000004
   (23)
                   2          2          2
   [leq= [z + y + x  - 1,z + y  + x - 1,z  + y + x - 1], bs= true,
    lnoneq= [(2y - x)z - x y,1], lvar= [x,y,z], si= true]

(24) ->contradictory?(syst)$pa
    GB time: 0.10000000000000001
   (24)  false

(25) ->consistent?(syst)$pa
    GB time: 0.066667000000000004
   (25)  false

(26) ->saturate(syst)$pa
    GB time: 0.066667000000000004
   (26)  [1]
```

Une autre opération importante sur les systèmes quasi-algébriques réalisée par ce paquetage `GBLUP` est la *réunion de deux systèmes* au sens suivant. Pour deux systèmes quasi-algébriques σ_1 et σ_2 trouver un système quasi-algébrique σ_3 tel que :

$$\mathbf{Z}(\sigma_3) = \overline{\mathbf{Z}(\sigma_1)} \cup \overline{\mathbf{Z}(\sigma_2)}$$

Cette opération intervient dans la vérification d'une décomposition en systèmes quasi-algébriques d'une variété affine. Le s.q.a. σ_3 est obtenu comme l'intersection des idéaux saturés de σ_1 et σ_2. L'intersection de deux idéaux est calculée à l'aide de la formule de la proposition I.12 Continuons notre session AXIOM pour illustrer cette opération.

```
(30) ->syst1 := system([x^2 - 3*x +2, (x -1)*y +1],[x -1])$pa
   (30)
```

```
          2
   [leq= [x  - 3x + 2,(x - 1)y + 1], bs= false, lnoneq= [x - 1], lvar= [y,x],
    si= false]

(31) ->saturate(syst1)$pa
    GB time: 0.10000000000000001
   (31)  [y + 1,x - 2]

(32) ->syst2 := system([x-1],[])$pa

   (32)  [leq= [x - 1],bs= false,lnoneq= [],lvar= [x],si= false]

(33) ->zeroSetUnionClosure(syst1,syst2)$pa
    GB time: 0.083333000000000004     GB time: 0.10000000000000001
   (33)
                             2
   [leq= [(x - 1)y + x - 1,x  - 3x + 2], bs= true, lnoneq= [], lvar= [y,x],
    si= true]
```

2.6 Catégories et domaines d'ensembles de polynômes

On dispose tout d'abord d'un constructeur de catégories `PolynomialSetCategory` de signature :

```
PolynomialSetCategory(R:Ring,E:OrderedAbelianMonoidSup,V:OrderedSet,
                      P:RecursivePolynomialCategory(R,E,V)): Category
```

Les éléments d'un domaine `D` de la catégorie `PolynomialSetCategory(R,E,V,P)` sont des sous-ensembles finis de polynômes de `P`. L'égalité dans `D` est l'égalité entre parties de `P`. Autrement dit, chaque élément de `D` n'est pas identifié à l'idéal qu'il engendre dans `P`. Cependant lorsque `R` est un anneau intègre, chaque élément `ps` de `D` peut être vu, selon les besoins d'un utilisateur, soit comme l'idéal qu'il engendre dans $R^{-1}P$, soit comme la variété affine associée, soit comme la tour d'extensions simples associée (si `ps` est un ensemble triangulaire régulier). Pour se faire `PolynomialSetCategory(R,E,V,P)` exporte différentes opérations sur les idéaux et variétés affines. Ces opérations y sont implantées grâce au paquetage `GBLUP`. On dispose d'un constructeur de domaines associé au constructeur de catégorie précédent et défini comme suit :

```
)abbrev domain GPOLSET GeneralPolynomialSet

GeneralPolynomialSet(R,E,V,P) : Exports == Implementation where

  R:Ring
  V:OrderedSet
  E:OrderedAbelianMonoidSup
  P:RecursivePolynomialCategory(R,E,V)
  LP ==> List P

  Exports ==  PolynomialSetCategory(R,E,V,P)

  Implementation == add

     Rep := LP

     construct lp ==
       (removeDuplicates(lp)$LP)::$
     ..................................................................
```

Nous illustrons ce constructeur à l'aide d'une session AXIOM :

```
(1) ->ls : List Symbol := [x,y,z];
(2) ->V := OVAR(ls);
(3) ->R := Integer;
(4) ->E := IndexedExponents V ;
(5) ->P := NSMP(R, V);
(6) ->LP := List P;
(7) ->PS := GPOLSET(R,E,V,P);
(8) ->polset(lp :LP):PS == construct(lp)$PS;
(9) ->d1 : P := x*(x+1);
(10) ->d2 : P := (y+x)*(y-1);
(11) ->ld : LP := [d1,d2]

           2                  2
   (11)  [x  + x,(y - 1)x + y  - y]

(12) ->lps := [polset([x,y]),polset([x,y - 1]),polset([x + 1,y^2  - 2*y + 1])]
```

```
                            2
   (12)  [{y,x},{y - 1,x},{y  - 2y + 1,x + 1}]
```

Dans les commandes suivantes, on demande tout d'abord si les idéaux engendrés par les trois ensembles ci-dessus forment un scindage (définition II.4) de l'idéal engendré par `ld`. La réponse est : *oui*. Puis, on rajoute deux ensembles à la liste `lps`. On repose la question précédente. La réponse est alors : *non*. En effet, on a augmenté la multiplicité de certaines composantes. On demande maintenant si les variétés affines des ensembles de `lps` ont pour réunion celle de `ld`. Pour finir, on élimine les variétés redondantes dans `lps` c'est-à-dire celles qui sont contenues dans une autre variété.

```
(13) ->idealSplit?(lps,ld)
   (13)  true
                         Time: 0.03 (IN) + 2.50 (EV) + 0.05 (OT) = 2.58 sec
(14) ->lps := concat([polset([x^2,y^2]),polset([x,y*(y-1)])], lps)

            2  2    2                           2
   (14)  [{y ,x },{y  - y,x},{y,x},{y - 1,x},{y  - 2y + 1,x + 1}]
(15) ->idealSplit?(lps,ld)
   (15)  false
                         Time: 0.02 (IN) + 0.12 (EV) + 0.02 (OT) = 0.15 sec
(16) ->varietySplit?(lps,ld)
   (16)  true
                         Time: 0.03 (IN) + 4.97 (EV) + 0.07 (OT) = 5.07 sec
(17) ->lps := eliminateRedundantVarieties(lps)
            2         2
   (17)  [{y  - y,x},{y  - 2y + 1,x + 1}]
                         Time: 0.02 (IN) + 2.82 (EV) + 0.15 (OT) = 2.98 sec
(18) ->varietySplit?(lps,ld)
   (18)  true
                         Time: 0.02 (IN) + 1.38 (EV) + 0.02 (OT) = 1.42 sec
```

Notons que :

- pour tester l'inclusion entre deux idéaux, l'opération `idealSplit ?` utilise naturellement des calculs de forme normale.
- pour tester l'inclusion entre deux variétés affines, sans pour autant calculer (des bases) d'idéaux radicaux, l'opération `eliminateRedundanVarieties` se ramène au *test d'appartenance au radical* (th. I.6 p. 32)
- pour tester qu'une réunion $V_1, \ldots, V_s$ de variétés affines est égale à une variété affine V, l'opération `varietySplit ?` procède comme suit :
  - elle teste si chaque V_i est contenue dans V
  - elle calcule l'intersection I des idéaux associés aux V_i
  - enfin, elle teste si I est contenu dans le radical de l'idéal associé à V

  de sorte qu'aucun radical n'est jamais calculé.

2.7 Catégories et domaines d'ensembles triangulaires

On dispose tout d'abord d'une catégorie générale d'ensembles triangulaires. il s'agit donc d'ensembles de polynômes au sens de la définition III.6. Cette catégorie exporte les différentes opérations *de base* (notation III.3, remarque III.14) en particulier les opérations de réduction. Cette catégorie exporte aussi les opérations liées à la décomposition des variétés affines en (fermeture de) zeros réguliers d'ensembles triangulaires. Elle n'exporte cepedant aucun algorithme de décomposition. Noter le rôle de l'opération `extendIfCan` : elle encode les éventuelles propriété des ensembles triangulaires d'une catégorie ou d'un domaine héritant de `TSETCAT`. Cette catégorie est définie comme suit :

```
)abbrev category TSETCAT TriangularSetCategory

R:IntegralDomain
E:OrderedAbelianMonoidSup
V:OrderedSet
P:RecursivePolynomialCategory(R,E,V)

TriangularSetCategory(R,E,V,P):Category == PolynomialSetCategory(R,E,V,P) with
     saturate : $ -> List P
         ++ saturate(ts) returns the saturated ideal of the ideal generated
         ++ by ts w.r.t. its initials in the polynomial ring (R)^(-1) P.
         ++ It is given by its Groebner base w.r.t. lexicographical
         ++ ordering induced by V
     quasiComponent : $ -> Record(close:List P,open:List P)
         ++ quasiComponent(ts) returns [lp,lq] where lp is the list of the
         ++ members of ts and lq is the list of their initials
     initiallyReduce : (P,$) -> P
     ......................................................................
     zeroSetSplit : List P -> List $
     ......................................................................
     extendIfCan : ($,P) -> Union($,"failed")
         ++ extendIfCan(ts,p) returns a triangular set which encodes the simple
         ++ extension by p of the extension of the base field defined by ts,
         ++ according to the properties of triangular sets of the current category.
         ++ If the required properties do not hold then "failed" is returned.
         ++ This operation encodes in some sense the properties of the
         ++ triangular sets of the current category. Is is used to implement
         ++ the \spad{construct} operation to guarantee that every triangular
         ++ set build from a list of polynomials has the required properties.
     ......................................................................
  add
     construct(lp:List(P)) ==
       rif := retractIfCan(lp)@Union($,"failed")
       not (rif case $) => error"in TSETCAT construct: la vache qui rit est-elle folle?"
       rif::$

     retractIfCan(lp:List(P)) ==
       empty? lp => (empty()$$)::Union($,"failed")
       lp := sort(supRittWu?,lp)
       rif := retractIfCan(rest(lp))@Union($,"failed")
       not (rif case $) => error"in retractIfCan : LP -> ... from TSETCAT : bad arg"
       extendIfCan(rif::$,first(lp))@Union($,"failed")
     ......................................................................
```

On dispose aussi d'un constructeur de domaines de la catégorie TSETCAT n'encodant aucune méthode de résolution. Ce constructeur sert de *point de départ* pour implanter une méthode de résolution. Il est défini comme suit :

```
)abbrev domain GTSET GeneralTriangularSet

GeneralTriangularSet(R,E,V,P) : Exports == Implementation where

  R : IntegralDomain
  E : OrderedAbelianMonoidSup
  V : OrderedSet
  P : RecursivePolynomialCategory(R,E,V)

  Exports ==  TriangularSetCategory(R,E,V,P)

  Implementation == add

     Rep ==> List P
     rep(s:$):Rep == s pretend Rep
     per(l:Rep):$ == l pretend $
     ........................................................................
     mvar ts ==
       empty? ts => error"Fucking Mad Cow in mvar : $ -> V from GTSET"
       mvar(first(rep(ts)))$P
     -- for another domain of TSETCAT build on this domain GTSET
     -- the following operations must be redefined
     extendIfCan(ts:$,p:P) ==
       ground? p => "failed"::Union($,"failed")
       empty? ts => (per([unitCanonical(p)]$LP))::Union($,"failed")
       not (mvar(ts) < mvar(p)) => "failed"::Union($,"failed")
       (per(cons(p,rep(ts))))::Union($,"failed")
     ........................................................................
```

Les méthodes de Wang et Wu sont chacunes implantées dans un constructeur de domaines de la catégorie TSETCAT. Ainsi pour Wu, on a le constructeur suivant ne contenant que les algorithmes tels qu'ils sont exposés dans la quatrième section de ce chapitre.

```
)abbrev domain WUTSET WuWenTsunTriangularSet

WuWenTsunTriangularSet(R,E,V,P) : Exports == Implementation where

  R : IntegralDomain
  E : OrderedAbelianMonoidSup
  V : OrderedSet
  P : RecursivePolynomialCategory(R,E,V)
     ........................................................................

  Exports ==  TriangularSetCategory(R,E,V,P) with
     medialSet : (LP,((P,P)->B),((P,P)->P)) -> Union($,"failed")
     characteristicSet : (LP,((P,P)->B),((P,P)->P)) -> Union($,"failed")
     characteristicSerie  : (LP,((P,P)->B),((P,P)->P)) -> List $

  Implementation == GeneralTriangularSet(R,E,V,P) add
     ........................................................................
```

Pour les méthodes de Kalkbrener et Lazard, il y a quelques catégories supplémentaires. Tout d'abord on se donne une catégorie correspondant aux tours d'extensions simples (sans plus d'hypothèses). Les initiaux des polynômes définissant une telle tour étant inversibles dans cette tour, on peut alors *localiser* par la partie multiplicativement stable qu'ils engendrent et définir une opération modulo (remarque III.10) :

```
)abbrev category TOSE TowerOfSimpleExtensions
R:IntegralDomain
E:OrderedAbelianMonoidSup
V:OrderedSet
P:RecursivePolynomialCategory(R,E,V)
S:MultiplicativeSubsetCategory(P)
L:LocalizedPolynomialRingModel(R,E,V,P,S)

TowerOfSimpleExtensions(R,E,V,P,S,L):
         Category ==
   TriangularSetCategory(R,E,V,P) with
     modulo : (P,$) -> L
         ++ modulo(p,ts) returns \spad{l / s} the image
         ++ of \spad{p} in the residue class ring of
         ++ \spad{P} defined by \spad{ts} such that \spad{numer(l)}
         ++ is reduced w.r.t. \spad{ts} in Wu Wen Tsun sense.
     ............................................................
```

On dispose aussi d'une notion de pgcd comme celle définie dans la méthode de Kalkbrener, mais pas comme celle exposée dans le troisième chapitre de cette thèse. Pour cette dernière, il faut rajouter l'hypothèse de *séparabilité*. D'où la catégorie suivante :

```
)abbrev category STOSE SeparableTowerOfSimpleExtensions

R:IntegralDomain
E:OrderedAbelianMonoidSup
V:OrderedSet
P:RecursivePolynomialCategory(R,E,V)
S:MultiplicativeSubsetCategory(P)
L:LocalizedPolynomialRingModel(R,E,V,P,S)

SeparableTowerOfSimpleExtensions(R,E,V,P,S,L):
         Category ==
   TowerOfSimpleExtensions(R,E,V,P,S,L)
```

De plus, on se donne une catégorie pour les ensembles triangulaires réguliers normalisés. En effet, nos algorithmes de pgcd reposent également sur la notion de *normalisation* :

```
)abbrev category NTOSE NormalizedTowerOfSimpleExtensions

R:IntegralDomain
E:OrderedAbelianMonoidSup
V:OrderedSet
P:RecursivePolynomialCategory(R,E,V)
S:MultiplicativeSubsetCategory(P)
L:LocalizedPolynomialRingModel(R,E,V,P,S)

NormalizedTowerOfSimpleExtensions(R,E,V,P,S,L):
         Category ==
   TowerOfSimpleExtensions(R,E,V,P,S,L)
```

Par suite, la catégorie d'un domaine implantant la méthode de Lazard (le chapitre 3 de cette thèse) est :

```
)abbrev category NSTOSE NormalizedSeparableTowerOfSimpleExtensions
```

```
R:IntegralDomain
E:OrderedAbelianMonoidSup
V:OrderedSet
P:RecursivePolynomialCategory(R,E,V)
S:MultiplicativeSubsetCategory(P)
L:LocalizedPolynomialRingModel(R,E,V,P,S)

NormalizedSeparableTowerOfSimpleExtensions(R,E,V,P,S,L):
         Category ==
   Join(NormalizedTowerOfSimpleExtensions(R,E,V,P,S,L),
        SeparableTowerOfSimpleExtensions(R,E,V,P,S,L))
```

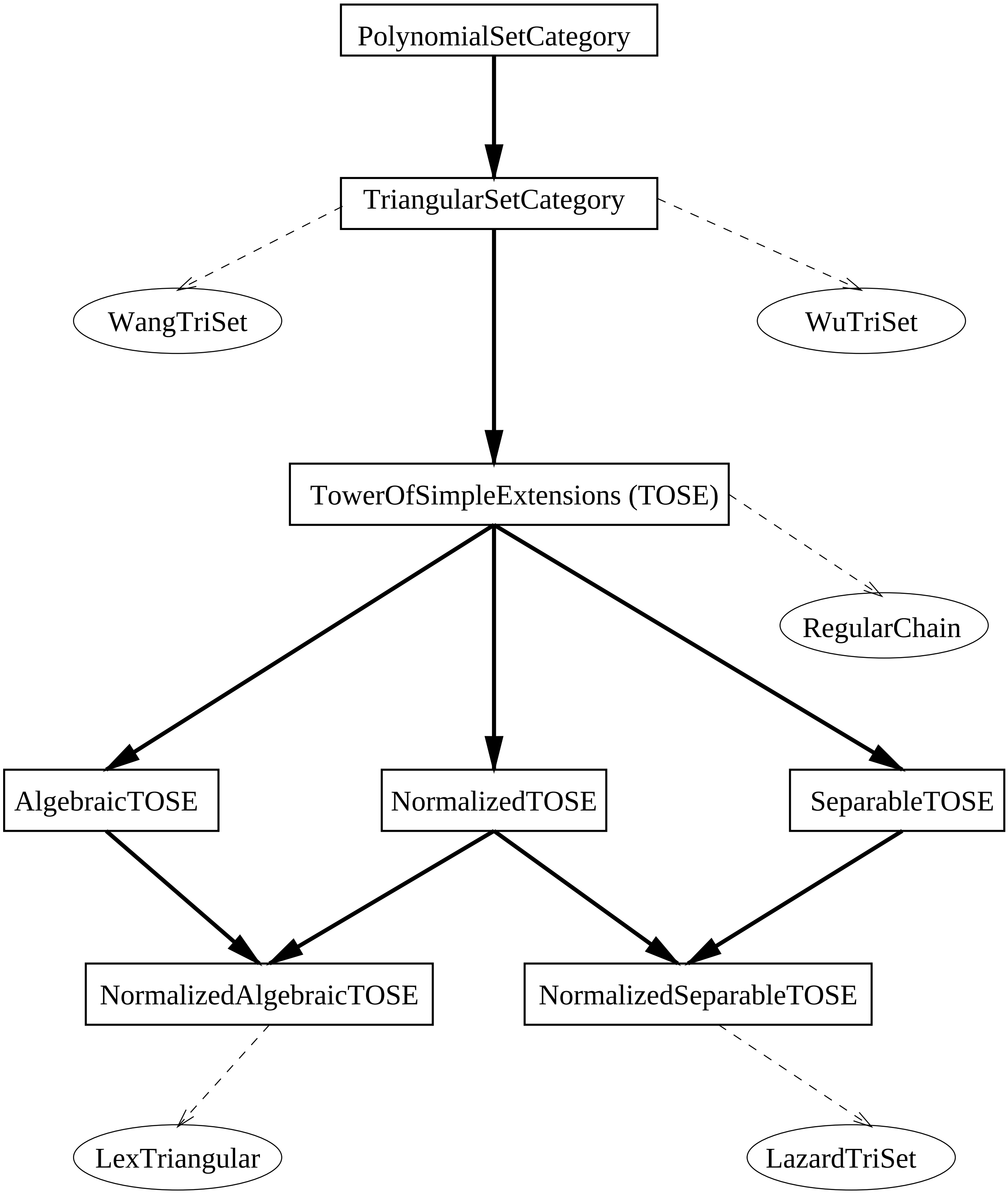
On a également une notion de pgcd *à la Kalkbrener* avec des ensembles triangulaires normalisés si l'on rajoute l'hypothèse d' *algébricité* (i.e. toutes les variables sont algébriques). C'est l'objet du second chapitre de cette thèse. Noter ci-après la catégorie de `S` dans `NATOSE`.

```
)abbrev category ATOSE AlgebraicTowerOfSimpleExtensions
R:IntegralDomain
E:OrderedAbelianMonoidSup
V:OrderedSet
P:RecursivePolynomialCategory(R,E,V)
S:MultiplicativeSubsetCategory(P)
L:LocalizedPolynomialRingModel(R,E,V,P,S)
AlgebraicTowerOfSimpleExtensions(R,E,V,P,S,L):
         Category ==
   TowerOfSimpleExtensions(R,E,V,P,S,L)

)abbrev category NATOSE NormalizedAlgebraicTowerOfSimpleExtensions
R:IntegralDomain
E:OrderedAbelianMonoidSup
V:OrderedSet
P:RecursivePolynomialCategory(R,E,V)
S:Join(MultiplicativeSubsetCategory(P),MultiplicativeSubsetCategory(R))
L:LocalizedPolynomialRingModel(R,E,V,P,S)
NormalizedAlgebraicTowerOfSimpleExtensions(R,E,V,P,S,L):
         Category ==
   Join(NormalizedTowerOfSimpleExtensions(R,E,V,P,S,L),
        AlgebraicTowerOfSimpleExtensions(R,E,V,P,S,L))
```

Par suite, nous avons repoussé dans la catégorie `TOSE` les algorithmes de normalisation (première section du troisième chapitre) et de pgcd (seconde section du troisième chapitre) Ces algorithmes sont en effet les mêmes dans le cas *purement algébrique* (second chapitre). La différence entre le cas purement algébrique et le cas séparable se fait au niveau de l'opération quasiRecipElseSplit. Elle est implantée dans la catégorie `NATOSE` pour le cas purement algébrique et dans la catégorie `NSTOSE` pour le cas séparable.

Quant à l'opération zeroSetSplit, elle s'implante avec l'algorithme lexTriangular dans `NATOSE` et l'algorithme intersect dans `NSTOSE`.


PolynomialSetCategory
TriangularSetCategory
WangTriSet
WuTriSet
TowerOfSimpleExtensions (TOSE)
RegularChain
AlgebraicTOSE
NormalizedTOSE
SeparableTOSE
NormalizedAlgebraicTOSE
NormalizedSeparableTOSE
LexTriangular
LazardTriSet

3 Méthode de D.M. Wang

On rappelle que $\mathbf{R}$ désigne un anneau nœthérien intègre, $\mathbf{k}$ son corps des fractions et $\mathbf{K}$ une clôture algébrique de $\mathbf{k}$. On pose $\mathbf{R}_n = \mathbf{R}[X_1, \ldots, X_n]$ et $\mathbf{P}_n = \mathbf{P}[X_1, \ldots, X_n]$ et l'on convient que $\mathbf{R}_0 = \mathbf{R}$.

Définition VI.2 *Soit (T, Q) un s.q.a. triangulaire de $\mathbf{R}_n$. On note $h = \prod_{t \in T} \mathsf{init}(t)$ et $\mathsf{prem}(Q, T) = \{\mathsf{prem}(q, T)\ q \in Q\}$. Le système (T, Q) sera dit* standard *s'il vérifie les deux conditions suivantes :*

- $\mathbf{V}_{\mathbf{K}}(h) \cap \mathbf{ZR}_{\mathbf{K}}(T, Q)) = \emptyset$
- $0 \notin \mathsf{prem}(Q, T)$.

Remarque VI.3 Avec la définition ci-dessus certains types de s.q.a. triangulaires inconsistants sont éliminés, mais on peut tout de mme trouver des s.q.a. triangulaires standards inconsistants. Par exemple, considérer (T, Q) avec :

$$T = \{x^2 - 1, xy^2 - x + 1, y(x+1)z + 1\} \text{ et } Q = \{x, y\}$$

La méthode de Wang est une méthode de résolution des s.q.a., où la famille $\mathcal{R}$ de systèmes résolus (rem. III.18 p. 100) est formée par les systèmes triangulaires standards de $\mathbf{R}_n$.

Par souci de clarté, nous présentons sous forme récursive les algorithmes proposés par D.M. Wang ([Wan93b]).

Proposition VI.1 *Soit f un polynôme non constant dans $\mathbf{R}_n$ et (F, Q) un système quasi-algébrique dans $\mathbf{R}_n$. On pose $G = \mathsf{prem}(F, f) \setminus \{0\}$ et $h = \mathsf{init}(f)$. Alors*

$$\mathbf{ZR}_{\mathbf{K}}(F \cup \{f\}, Q) = \mathbf{ZR}_{\mathbf{K}}(G \cup \{f\}, Q \cup \{h\}) \cup \mathbf{ZR}_{\mathbf{K}}(F \cup \{h, \mathsf{tail}(f)\}, Q).$$

Preuve. $\triangleright$ Soit $p \in F$ et $\mathrm{r} = \mathsf{prem}(p, f)$. Il existe $e \in \mathbb{N}$ et $q \in \mathbf{R}_n$ tels que

$$h^e\, p \;=\; q\, f \;+ r$$

Cette relation entraîne

$$\mathbf{ZR}_{\mathbf{K}}(\{p, f\}, \{h\}) = \mathbf{ZR}_{\mathbf{K}}(\{r, f\}, \{h\}).$$

On a ainsi

$$\mathbf{ZR}_{\mathbf{K}}(F \cup \{f\}, \{h\}) = \mathbf{ZR}_{\mathbf{K}}(G \cup \{f\}, \{h\}). \tag{VI.1}$$

Puisque de façon évidente

$$\mathbf{V}_K(F \cup \{f\}) = \mathbf{ZR}_{\mathbf{K}}(F \cup \{f\}, \{h\}) \cup \mathbf{V}_K(F \cup \{h\}, \mathsf{tail}(f)\}),$$

on obtient alors avec VI.1

$$\mathbf{V}_K(F \cup \{f\}) = \mathbf{ZR}_{\mathbf{K}}(G \cup \{f\}, \{h\}) \cup \mathbf{V}_K(F \cup \{\{h\}, \mathsf{tail}(t)\}).$$

L'égalité de la proposition s'ensuit immédiatement. $\triangleleft$

Soit $\Sigma = (F, Q)$ un s.q.a. dans $\mathbf{R}_n$, tel que $\mathsf{mvar}(F) = X_i$. l'algorithme $\mathsf{elimination}$ présenté plus bas permet d'éliminer la variable X_i dans Σ, c-à-d de scinder le s.q.a. Σ en plusieurs s.q.a. Σ_k qui contiennent chacun au plus une équation de variable principale X_i. Plus précisément, nous dirons

Définition VI.3 *Soit $F \subseteq \mathbf{R}_i$ et $Q \subseteq \mathbf{R}_n$. Une* élimination *de la variable X_i dans le s.q.a. $\Sigma = (F, Q)$ est un ensemble Λ de triplets (F_k, Q_k, τ_k) vérifiant :*

(i) pour tout k, F_k, Q_k et τ_k sont des parties de $\mathbf{R}_n$ telles que

- $F_k \neq \emptyset \Rightarrow \mathsf{mvar}(F_k) < X_i$
- $\tau_k \neq \emptyset \;\Rightarrow\; \exists t \in \mathbf{R}_i \setminus \mathbf{R}_{i-1} \;\mid\; \tau_k = \{t\}$

(ii)
$$\mathbf{ZR}_{\mathbf{K}}(\Sigma) = \bigcup_{(F_j, Q_j, \tau_j) \in \Lambda} \mathbf{ZR}_{\mathbf{K}}(F_j \cup \{\tau_j\}, Q_j).$$

Nous présentons maintenant l'algorithme $\mathsf{elimination}$:

Entrée : v, une variable de $\{X_1, \ldots, X_n\}$
F une partie de $\mathbf{R}_n$ telle que $\mathsf{mvar}(F) \leq v$
$Q \subseteq \mathbf{R}_n$.

Sortie : $\Lambda = \{(F_k, Q_k, \tau_k)\}$ une élimination de la variable v dans le s.q.a. (F, Q).
En particulier, si Λ est vide, alors $\mathbf{ZR}_{\mathbf{K}}(F, Q) = \emptyset$.

$\mathsf{elimination}(v, F, Q) ==$
$F := F \setminus \{0\}$
$(0 \in Q)$ **or** $(F \cap \mathbf{R} \neq \emptyset)$ $=>$ $\{\,\}$
$F_v := \{p \in F \mid \mathsf{mvar}(p) = v\}$
$F_v^- := \{p \in F \mid \mathsf{mvar}(p) < v\}$
$F_v = \emptyset$ $=>$ $\{\,(F, Q, \emptyset)\,\}$
$f :=$ un polynôme de F_v de degr minimal en v
$F_v := F_v \setminus \{f\}$
$F_1 := F_v \cup \{\mathsf{init}(f), \mathsf{tail}(f)\} \cup F_v^-$
$Q_2 := Q \cup \{\mathsf{init}(f)\}$
if empty ? F_v
  then
    $\Lambda := \{\,(\,F_v^-, \mathsf{prem}(Q_2, f), \{f\})\,\} \cup \mathsf{elimination}(v, F_1, Q)$
  else
    $F_2 := \mathsf{prem}(\,F_v, f) \cup \{f\} \cup F_v^-$
    $\Lambda := \mathsf{elimination}(v, F_2, Q_2) \cup \mathsf{elimination}(v, F_1, Q)$
return Λ

On utilise le lemme suivant dans la preuve de la correction de cet algorithme.

Proposition VI.2 *Soit f un polynôme non constant et (F, Q) un s.q.a. dans $\mathbf{R}_n$. Si* $\mathsf{init}(f) \in Q$, *alors*
$$\mathbf{ZR}_{\mathbf{K}}(F \cup \{f\}, Q) = \mathbf{ZR}_{\mathbf{K}}(F \cup \{f\}, \mathsf{prem}(Q, f))$$

Preuve. $\triangleright$ Notons $h = \mathsf{init}(f)$. On remarque tout d'abord que si $h \in Q$, alors $h \in \mathsf{prem}(Q, f)$. Soit $q \in Q$, et $r = \mathsf{prem}(q, f)$. Il existe un entier positif e et un polynôme q' tel que $h^e\, q = f\, q' + r$.

Supposons que x soit un zéro de f tel que $h(x) \neq 0$. On a alors $q(x) \neq 0 \Leftrightarrow r(x) \neq 0$, d'où le résultat. $\triangleleft$

Preuve de l'algorithme. $\triangleright$ On peut supposer sans restriction que $0 \notin F$. On raisonne par induction sur $s(F) = \sum_{p \in F \setminus \{0\}} \mathsf{deg}(p, v)$.

Éliminons tout de suite le cas trivial où il y a une constante dans F ou si $0 \in Q$.

Dans le cas contraire, et si $s(F) = 0$, alors $F_v = \emptyset$; La terminaison et la correction sont encore immédiates.

Supposons maintenant $s(F) > 0$. Cela signifie que F_v n'est pas vide. Remarquons d'abord que $s(F_1) < s(F)$ puisque $\mathsf{deg}(\mathsf{init}(f), v) = 0$ et $\mathsf{deg}(\mathsf{tail}(f), v) < \mathsf{deg}(f, v)$. On distingue alors deux cas :

1. $F_v = \{f\}$. Par récurrence, $\mathsf{elimination}(v, F_1, Q)$ termine et est correct. $\mathsf{elimination}(v, F, Q)$ se termine donc, et la correction s'obtient par une application directe de la proposition VI.1, puis du lemme VI.2.
2. $F_v \setminus \{f\} \neq \emptyset$. Pour tout $p \in F_v \setminus \{f\}$, $\mathsf{deg}(\mathsf{prem}(p, f), v) < \mathsf{deg}(f, v) \leq \mathsf{deg}(p, v)$. $F_v \setminus \{f\}$ n'étant pas vide, $s(\mathsf{prem}(F_v \setminus \{f\}, f)) < s(F_v \setminus \{f\})$, d'où $s(F_2) < s(F)$. Ainsi, par application de la proposition VI.1 et de l'hypothèse de récurrence, on obtient la terminaison et la correction du calcul. $\triangleleft$

Remarque VI.4 Si (F_j, Q_j, τ_j) est un triplet retourné par l'algorithme elimination tel que $\tau_j = \{f\}$, il est aisé de voir qu'alors $\mathsf{init}(f)$ appartient à Q_j.

Un s.q.a quelconque étant donné, l'algorithme elimination, appliqué successivement aux variables dans l'ordre décroissant, permet d'obtenir des systèmes triangulaires :

Proposition VI.3 *Soit (F, Q) un s.q.a. de $\mathbf{R}_n$ tel que $F \subseteq \mathbf{R}_i$, où $i \in \{0, 1, \ldots, n\}$. Soit T un ensemble triangulaire de $\mathbf{R}_n$ tel que $T \cap \mathbf{R}_i = \emptyset$. L'algorithme* triangulation *ci-dessous calcule une famille finie Θ de s.q.a. triangulaires (T_k, Q_k) vérifiant*

$$\mathbf{ZR_K}(F \cup T, Q) = \bigcup_{(T_k, Q_k) \in \Theta} \mathbf{ZR_K}(T_k, Q_k).$$

En particulier, si Θ est vide, alors $\mathbf{ZR_K}(F, Q) = \emptyset$.

triangulation(F, Q, T) ==
 $F := F \setminus \{0\}$
 $(0 \in Q)$ **or** $(F \cap \mathbf{R} \neq \emptyset)$ => { }
 empty ? F => { (T, Q) }
 $v :=$ mvar(F)
 $\Lambda :=$ elimination(v, F, Q)
 $\Theta := \bigcup_{(F_j, Q_j, \tau_j) \in \Lambda}$ triangulation$(F_j, Q_j, \tau_j \cup T)$

Preuve. $\triangleright$ On raisonne par induction sur $i(F)$, plus petit entier tel que $F \subseteq \mathbf{R}_i$.
Pour $i(F) = 0$, c-a-d $F \subseteq R$, le résultat est évident. Supposons $i(F) > 0$. Passons sur les cas $0 \in Q$, $F \cap \mathbf{R} \neq \emptyset$, ou $\Lambda = \emptyset$, puisqu'on sait que $\mathbf{ZR_K}(F, Q) = \emptyset$. Sinon, d'après les spécifications de l'algorithme elimination,

$$\mathbf{ZR_K}(F, Q) \cap \mathbf{V}_K(T) = \bigcup_{(F_j, Q_j, \tau_j) \in \Lambda} \mathbf{ZR_K}(F_j \cup \{\tau_j\}, Q_j) \cap \mathbf{V}_K(T)$$

c'est-à-dire

$$\mathbf{ZR}_{\mathbf{K}}(F \cup T, Q) = \bigcup_{(F_j, Q_j, \tau_j) \in \Lambda} \mathbf{ZR}_{\mathbf{K}}(F_j \cup (\{\tau_j\} \cup T), Q_j)$$

Notons $T_j = \{\tau_j\} \cup T$. Les triplets (F_j, Q_j, T_j) vérifient les conditions d'entrée de triangulation. Comme $i(F_j) < i(F)$, le résultat est obtenu par récurrence. $\lhd$

Remarque VI.5 Plaçons nous dans les hypothèses de la proposition VI.3. Supposons de plus que prem(Q, T) ne contient pas 0. Alors pour tout $(T_k, Q_k) \in \Theta$ prem(Q_k, T_k) ne contient pas 0. On vérifie facilement que ceci est d au remplacement dans l'algorithme elimination, de Q_2 par prem(Q_2, f).

Théorème VI.1 *Soit $\Sigma = (F, Q)$ un s.q.a. de $\mathbf{R}_n$. Il existe une famille Θ de s.q.a. triangulaires standards (T_k, Q_k) telle que*

$$\mathbf{ZR}_{\mathbf{K}}(\Sigma) = \bigcup_{(T_k, Q_k) \in \Theta} \mathbf{ZR}_{\mathbf{K}}(T_k, Q_k).$$

L'algorithme triangulation$(F, Q, \emptyset)$ *calcule une telle famille.*

Preuve. $\rhd$ La famille de s.q.a. (T_k, Q_k) calculée par l'algorithme triangulation$(F, Q, \emptyset)$ vérifie l'égalité ci-dessus d'après la proposition VI.3. Il nous reste à vérifier que les s.q.a. triangulaires obtenus sont standards. En choisissant $T = \emptyset$, la remarque VI.5 s'applique. D'autre part, la remarque VI.4 entraîne que les pseudo-restes non nuls des initiaux de T_k par T_k se trouvent dans l'ensemble Q_k. On déduit alors facilement avec la formule du reste (définition III.7) que $\mathbf{V}_{\mathbf{K}}(h) \cap \mathbf{ZR}_{\mathbf{K}}(T, Q) = \emptyset$.

Les systèmes (T_k, Q_k) de Θ sont donc standards, $\lhd$

Remarque VI.6 Pour une implantation efficace de son algorithme, Wang propose plusieurs pistes([Wan93b]). Les optimisations peuvent être nombreuses, mais nécessitent une certaine prudence du fait du manque de propriétés algébriques et de *stabilité* des s.q.a. triangulaires standards.

• L'élimination descendante des variables opérée dans triangulation fournit des ensembles triangulaires qui ne sont mme pas réduits au sens de la réduction des initiaux itérés. Si l'algorithme est appliqué *brutalement*, cela génère éventuellement de grosses composantes en sortie, qui peuvent se révéler inconsistantes. La performance comme la lisibilité des sorties est alors médiocre. Malheureusement les tentatives d'auto-réduction d'un ensemble triangulaire obtenu par l'algorithme triangulation peuvent faire apparaître deux polynômes de mme variable principale comme dans l'exemple suivant :

$$T = \{x^2 - x, x^2y + 1, (xy + 1)z - y^2\}$$

avec l'ordre $x < y < z$ sur les variables. Si on réduit T au sens de la réduction des initiaux itérés, on obtient l'ensemble

$$\{x^2 - x, xy + 1, -x\}$$

qui n'est plus triangulaire.

Autre exemple de l'instabilité de ces s.q.a. triangulaires : si (T, Q) est un s.q.a. triangulaire, $t \in T$, et $h = \prod_{q \in Q} q$, on peut légitimement supprimer dans t tout facteur qui est

facteur de h. Une fonction removeFactors(t,Q), effectue ce travail dans notre implantation. Mais t peut voir sa variable principale baisser à la suite de cette opération. Par conséquent, si $T' = \{\mathsf{removeFactors(t, Q)},\ t \in T\}$, alors $\mathbf{ZR_K}(T', Q) = \mathbf{ZR_K}(T, Q)$, mais le s.q.a. (T', Q) n'est plus forçément triangulaire.
Dans notre implantation de triangulation, lorsque nous obtenons un s.q.a. triangulaire (T, Q), nous essayons de réduire T comme suit :

$F :=$ autoRemainder(T)
$(F, Q) :=$ removeFactors(F,Q)

où autoRemainder(T) consiste en une auto-réduction au sens des bases de Gröbner (qui conserve $\mathbf{Id}(T)$) et removeFactors(F,Q) $= \{\mathsf{removeFactors(f, Q)},\ f \in F\}$ Si dans cette phase de *nettoyage des sorties*, l'une de ces opérations conduit à un ensemble F qui n'est plus triangulaire, alors on relance triangulation$(F, Q, \emptyset)$ dont le résultat est une décomposition de $\mathbf{ZR}(T, Q)$.

- La *triangulation par le haut* nécessite aussi une optimisation de l'algorithme elimination :

1. Avec les notations de l'algorithme, les polynômes de F_v^- ne sont pas pris en compte dans elimination(v, F, Q). On peut illustrer ce phénomène par une élimination calculée au début du problème de Donati-Traverso. L'ordre des variables étant $t < z < y < x$, on considère $F = \{t, z, z^3x^7 - x^6 - x - y, x^8 - z\}$, et $Q = \emptyset$. Pour calculer elimination(x, F, Q), si on ne tient pas compte des polynômes t et z, cela génère de nombreuses pseudo-divisions et de grosses données, alors on a clairement $\mathbf{V_K}(F) = \mathbf{V_K}(\{t, z, y, x\})$.

   Dans notre implantation de elimination(v, F, Q), pour utiliser ces polynômes, ainsi que ceux de Q, nous commençons par quelques opérations destinés à faire un peu de *ménage* dans F.

   Les deux premières opérations conservent $\sqrt{\mathbf{Id}(F)}$ et la troisième $\mathbf{ZR}(F, Q)$:

   - F est réduit par ceux de ses polynômes qui sont à initiaux dans $\mathbf{R}$.
   - Tout sous-ensemble de F composé de polynômes univariés de mme variable est remplacé par le gcd de ces polynômes.
   - F est remplacé par removeFactors(F,Q), fonction vue plus haut.

2. Dans l'algorithme elimination, il est clair que dans les lignes

   $F_1 := F \cup \{\mathsf{init}(f), \mathsf{tail}(f)\}$
   $Q_2 := Q \cup \{\mathsf{init}(f)\}$

   init(f) peut tre remplacé par squareFreePart(init(f)).

   Il est possible d'tre plus efficace : Soit p est un polynôme de $\mathbf{R}_n$, L'opération square-FreeFactors(p) retourne un ensemble de polynômes $\{p_1^{e_1} \dots p_k^{e_k}\}$ sans facteurs carrés et premiers entre eux, tels qu'il existe $e_1, \dots, e_k \in \mathbb{N}$ et $c \in \mathbf{R}$ vérifiant $p = cp_1^{e_1} \dots p_k^{e_k}$.

   On peut alors transformer les lignes ci-dessus de la manière suivante

   $\{p_1, \dots, p_s\} :=$ squareFreeFactors(init(f))
   $F_1 := F \cup \{\prod_{i=1}^{i=s} p_i\ , \mathsf{tail}(f)\}$
   $Q_2 := Q \cup \{p_1, \dots, p_s\}$

Précisons tout de mme que les performances de l'algorithme ne sont pas affectées par ce changement. En effet, le temps d'exécution d'une opération squareFreeFactors(p) en AXIOM est le mme que celui de la fonction squareFreePart(p) d'AXIOM(qui commence par calculer la décomposition sans facteur carré de p).

3. La dernière optimisation sert à obtenir des ensembles triangulaires plus *lisibles* en sortie. Elle s'effectue lorsque elimination(v, F, Q) isole un polynôme f de variable principale v.

   On remplace la ligne

   $\Lambda := \{\ (\ F_v^-, \mathsf{prem}(Q_2, f), \{f\})\ \}\ \cup\ \mathsf{elimination}(v, F_1, Q)$

   par

   $Q_2 := \mathsf{prem}(Q_2, f)$
   $f\ := \mathsf{squareFreePart}(f)$
   $f\ := \mathsf{removeFactors}(f, Q_2)$
   if $f \notin \mathbf{R}$ **and** mvar$(f)= v$
   then
   $\Lambda := \{[\ F_v^-, Q_2, \{f\}]\} \cup\ \mathsf{elimination}(v, F_1, Q)$
   else
   $\Lambda := \{[\ F_v^- \cup \{f\}, Q_2, \{\}]\} \cup\ \mathsf{elimination}(v, F_1, Q)$

   On notera ici encore que l'opération removeFactors peut abaisser la variable principale du polynôme f et qu'il faut adapter la sortie de elimination en conséquence.

4 Méthode de Kalkbrener

4.1 Présentation

Dans son habilitation ([Kal95]), M. Kalkbrener étudie le transfert d'algorithmes d'un anneau commutatif nœtherien $\mathbf{A}$ vers $\mathbf{A}[X]$. Dans ce cadre général, il construit un algorithme de décomposition d'un idéal radical dans $\mathbf{A}[X]$ à partir d'un tel algorithme dans $\mathbf{A}$. Lorsque $\mathbf{A}$ est un anneau intègre, l'application récursive de ce transfert permet de donner une méthode de décomposition d'un idéal radical de $\mathbf{A}[X_1, \dots, X_n]$ en une intersection de représentations de chaînes régulières (définition III.11) $T_1, \dots, T_k$. Soit F est un ensemble fini de polynômes de $\mathbf{P}_n$. On aura alors

$$\mathbf{V}_K(F) = \bigcup_{i=1}^{k} \mathbf{V}_K(\mathsf{Rep}_n(T_i))$$

c-à-d

$$\mathbf{V}_K(F) = \bigcup_{i=1}^{k} \overline{\mathbf{ZR}_{\mathbf{K}}(T_i)}.$$

Remarquons que cette décomposition ne comporte aucune composante vide d'après la proposition suivante.

Proposition VI.4 *Soit $n \in \mathbb{N}$ et T une chaîne régulière de $\mathbf{P}_n$. On a*

$$\mathsf{Rep}_n(T) \neq \mathbf{P}_n.$$

Preuve. $\triangleright$ Montrons par récurrence sur l'entier n que $1 \notin \mathsf{Rep}_n(T)$. Lorsque $n = 0$, on a par définition $\mathsf{Rep}_0(T) = \{0\}$. Supposons maintenant $n > 0$.

Dans le cas où $X_n \notin \mathsf{algVar}(T)$, on a avec la proposition III.10

$$\mathsf{Rep}_n(T) = \sqrt{\mathsf{Rep}_{n-1}(T)}[X_n]$$

Par récurrence $1 \notin \mathsf{Rep}_{n-1}(T)$, donc $1 \notin \mathsf{Rep}_n(T)$.

Si $X_n \in \mathsf{algVar}(T)$, alors $1 \in \mathsf{Rep}_n(T)$ si et seulement $1 \in \mathsf{Rep}_{n-1}(T^-_{X_n})$ d'après la proposition III.17 Le résultat découle donc immédiatement de l'hypothèse de récurrence. $\triangleleft$

L'algorithme de décomposition $\mathsf{decompose}_n(F)$ ci-après fait appel à un calcul de gcd modulo une chaîne par un algorithme $\mathsf{gcd}_n(T, F)$, et à un algorithme $\mathsf{split}_n(T, f)$, qui *scinde* T en deux groupes de chaînes régulières : les premières déterminent les idéaux premiers associés à $\mathsf{Rep}_n(T)$ auxquels f appartient, et les secondes les idéaux premiers associés à $\mathsf{Rep}_n(T)$ auxquels f n'appartient pas.

A partir des algorithmes decompose et split dans $\mathbf{P}_i$, on construit dans $\mathbf{P}_{i+1}$ les algorithmes gcd, split et decompose. On dispose de façon triviale de ces algorithmes dans $\mathbf{P}_0 = \mathbf{k}$, on peut donc par induction les obtenir dans $\mathbf{P}_n$ pour tout entier n.

Nous présentons ci-dessous les différents algorithmes. Nous laissons le soin au lecteur intéressé par leur preuve de se référer au travail de Kalkbrener([Kal95],[Kal93]). On utilisera les notations introduites page 88.

4.2 Algorithmes

- algorithme $\mathsf{gcd}_n(T, F)$

Entrée : T une chaîne régulière de $\mathbf{P}_{n-1}$
F un ensemble fini de polynômes de $\mathbf{R}_n$

Sortie : $\Gamma = \{(T_1, g_1), \ldots, (T_k, g_k)\}$, où les T_i sont des chaînes régulières de $\mathbf{P}_{n-1}$ et les g_i des polynômes de $\mathbf{R}_n$ vérifiant

- $\bigcup_1^k \mathsf{ap}(\mathsf{Rep}_{n-1}(T_i)) = \mathsf{ap}(\mathsf{Rep}_{n-1}(T))$
- pour tout $\mathcal{P} \in \mathsf{ap}(\mathsf{Rep}_{n-1}(T_i))$,
  1. $F = \emptyset \Rightarrow k = 1$ et $g_k = 0$,
     $F \neq \emptyset \Rightarrow \overline{\overline{g_i}}^{\mathcal{P}}$ est le pgcd de $\overline{\overline{F}}^{\mathcal{P}}$ dans $\mathbf{k}(\mathcal{P})[X_n]$
  2. $g_i \neq 0 \Rightarrow \mathsf{lc}(g_i, X_n) \notin \mathcal{P}$
  3. $g_i \in \mathbf{Id}_{\mathbf{P}_n}(\mathsf{Rep}_n(T_i) \cup F)$.

$\mathsf{gcd}_n(T, F) ==$
  $F := F \setminus \{0\}$
  empty ? $F \Rightarrow \{(T, 0)\}$
  $f := \mathbf{min}(<_{rwl}, F)$
  $F := F \setminus \{f\}$
  c := $\mathsf{leadCoeff}(f, X_n)$
  $(M_1, M_2) := \mathsf{split}_{n-1}(T, c)$
  $F_1 := F \cup \{\mathsf{tail}(f)\}$
  if $f \in \mathbf{R}[X_{n-1}]$
    then
      $F_2 := \emptyset$ else
      $F_2 := \{\mathsf{prem}(p, f) \mid p \in F\}$
  $F_2 \subseteq \{0\} \Rightarrow$
      return $\Gamma := \bigcup_{T_1 \in M_1} \mathsf{gcd}_n(T_1, F_1) \cup \bigcup_{T_2 \in M_2} \{(T_2, f)\}$
  return $\Gamma := \bigcup_{T_1 \in M_1} \mathsf{gcd}_n(T_1, F) \cup \bigcup_{T_2 \in M_2} \mathsf{gcd}_n(T_2, F_2)$

- algorithme $\mathsf{split}_n(T, p)$

Entrée : T une chaîne régulière de $\mathbf{P}_n$
p un polynôme de $\mathbf{R}_n$.

Sortie : $(\{C_1, \ldots, C_k\}, \{S_1, \ldots, S_l\})$, un couple d'ensembles de chaînes régulières de $\mathbf{P}_n$ vérifiant

$$\bigcup_{i=1}^{k} \mathsf{ap}(\mathsf{Rep}_n(C_i)) = \{\mathcal{P} \in \mathsf{ap}(\mathsf{Rep}_n(T)) \mid p \notin \mathcal{P}\}$$

$$\bigcup_{i=1}^{l} \mathsf{ap}(\mathsf{Rep}_n(S_i)) = \{\mathcal{P} \in \mathsf{ap}(\mathsf{Rep}_n(T)) \mid p \in \mathcal{P}\}.$$

```
split_n(T, p) ==
    p=0 => ({T}, ∅)
    p ∈ R => (∅, {T})
    T = ∅ or mvar(T) < X_n =>
        {(T_1, g_1), ..., (T_k, g_k)} := gcd_n(T, p)
        return ({T_i | g_i = 0}, {T_i | g_i ≠ 0})
    t := T_{X_n}
    d := deg(t, X_n)
    Γ := gcd_n( T^-_{X_n}, {p, t})
    (Common, Separate) := (∅, ∅)
    for (T_i, g_i) ∈ Γ repeat
        if g_i ∈ R or mvar(g_i) < X_n
            then
                Separate := Separate ∪ {T_i ∪ {t}}
            else
                Common := Common ∪ {T_i ∪ {g_i}}
                deg(g_i, X_n) < d =>
                        q := pquo(t, g_i)
                        T' := T_i ∪ {q}
                        (C', S') := split_n(T', p)
                        Separate := Separate ∪ S'
        return (Common, Separate)
```

- algorithme $\mathsf{decompose}_n(F)$

Entrée : F un ensemble fini de polynômes de $\mathbf{R}_n$

Sortie : Θ un ensemble fini de chaînes régulières de $\mathbf{P}_n$ tel que $\sqrt{\mathbf{Id}_{\mathbf{P}_n}(F)} = \bigcap_{T \in \Theta} \sqrt{\mathsf{sat}(T_i)}$

```
decompose_n(F) ==
    F := F \ {0}
    empty? F  => {∅}
    F ∩ R ≠ ∅ => { }
    Θ := ∅
    F' := F ∩ R_{n-1}
    Δ := decompose_{n-1}(F')
    for T ∈ Δ repeat
        Γ := gcd_n(T, F \ F')
        for (T_i, g_i) ∈ Γ repeat
            g_i = 0  => Θ := Θ ∪ {T_i}
            Θ := Θ ∪ decompose_n(F ∪ {leadCoeff(g_i, X_n)})
            mvar(g_i)=X_n and |T_i| < |F| => Θ := Θ ∪ {T_i ∪ {g_i}}
    return Θ
```

4.3 Implantation et remarques

L'algorithme decompose peut rendre en sortie des chaînes dont les représentations contiennent d'autres chaînes de l'ensemble Θ. Pour éviter le calculs de certaines branches inutiles, Kalkbrener conseille d'utiliser le *primeidealkettensatz* de Krull ([Wa91], page 148), qui permet d'éliminer les chaînes qui ont un nombre d'éléments plus grand que celui des générateurs de départ. Ce critère a été inséré dans l'algorithme présenté ci-dessus ($|T_i| < |F|$).

Du fait de la non-canonicité des chaînes régulières, lorsque dans $\mathsf{decompose}_n(F)$ on ajoute à Θ une chaîne $T_i \cup \{g_i\}$, on a intérêt à chercher un polynôme g *plus simple* que g_i, tel que $T_i \cup \{g\}$ soit une chaîne régulière de $\mathbf{P}_n$ et $\mathsf{Rep}_n(T_i \cup \{g_i\}) = \mathsf{Rep}_n(T_i \cup \{g\})$. On peut même chercher plusieurs polynômes $g_{i,j}$ tels que les $T_i \cup \{g_{i,j}\}$ soient des chaînes régulières de $\mathbf{P}_n$ et $\mathsf{Rep}_n(T_i \cup \{g_i\})= \bigcap \mathsf{Rep}_n(T_i \cup \{g_{i,j}\})$.

- On peut choisir pour g la partie primitive de g_i dans $\mathbf{P}_{n-1}[X_n]$. En effet, soit $g_i = cg$ où c est le contenu de g_i dans $\mathbf{P}_{n-1}[X_n]$. Soit $\mathcal{P} \in \mathsf{ap}(\mathsf{Rep}_{n-1}(T_i))$. On sait que $\mathsf{leadCoeff}(g_i, X_n) \notin P$ et c divise $\mathsf{leadCoeff}(g_i, X_n)$. Comme $\mathcal{P}$ est premier, on a $c \notin P$ et l'égalité des chaînes s'obtient facilement à partir de leur définition.
- On peut aussi vérifier que $g = \mathsf{prem}(g_i, T_i)$ convient.
- Soit $f_1^{e_1} \dots f_s^{e_s}$ une décomposition sans facteur carré de g_i, alors $\overline{g_i}^{\mathcal{P}} = (\overline{f_1}^{\mathcal{P}})^{e_1} \dots (\overline{f_s}^{\mathcal{P}})^{e_s}$. On prend alors $g_{i,j} = f_j$.

En fait,dans l'implantation de decompose, la ligne $\Theta := \Theta \cup \{T_i \cup \{g_i\}\}$ est remplacée par

$g := \mathsf{prem}(g_i, \{t \in T_i \mid \ \mathsf{init}(t) \in \mathbf{R}\})$
$\{f_1, \dots, f_s\} := \mathsf{squareFreeFactors(g)}$
$G := \{f_j, 1 \leq j \leq s \mid \mathsf{mvar}(f_j) = X_n\}$
$\Theta := \Theta \cup \{T_i \cup \{g\}\}, g \in G\}$

Pourquoi ne réduisons-nous pas fortement le polynôme g_i par rapport à T_i ? En pratique, nous avons constaté que si la réduction forte est quelquefois avantageuse, elle peut se révéler très pénalisant. Dans l'exemple de Trinks1 le temps d'exécution est multiplié par 40 par rapport à la réduction par les seuls polynômes quasi-unitaires. Les ensembles obtenus en sortie ne sont donc pas toujours complètement réduits. Mais il suffit pour cela d'appliquer la fonction de réduction forte uniquement après la sortie ; Les calculs intermédiaires ne seront pas pénalisés.
La deuxième ligne de l'optimisation fait appel à la fonction squareFreeFactors décrite dans la section précédente (remarque VI.6).

Indiquons de plus l'utilisation du domaine Table d'AXIOM pour garder en mémoire les calculs de pgcd effectués. La structure de $\mathsf{decompose}_n(F)$ entraîne que ces calculs peuvent être repris plusieurs fois au cours d'une décomposition. Un bon exemple de l'intérêt de cette stratégie est l'exemple `Katsura4`, avec l'ordre $t < y < u < z < x$. La décomposition avec le domaine de chaînes régulières de nos tests qui implante cette mémorisation se fait en 85 secondes environ, contre plus de 300 secondes sans.

Intéressons-nous maintenant à l'algorithme gcd. Kalkbrener précise lui-même([Kal95]) que son algorithme de calcul de pgcd présenté ci-dessus doit être implanté à l'aide d'adaptations techniques telles que les sous-résultants, ou des essais de divisions(voir [Hea79]) pour

éviter la croissance exponentielle des coefficients.
Nous avons donc implanté une adaptation de la technique des sous-résultants qui tient compte des scindages de chaînes au fur et à mesure du calcul, de manière analogue à celle des algorithmes de pgcd de polynômes univariés à coefficients dans une tour d'extension séparables (chapitre 3). L'algorithme gcd tel qu'il est présenté plus haut, est effectivement trop handicapant sur des problèmes difficiles. Avec le domaine utilisant cet algorithme, nous avons arrêté la décomposition de l'exemple de Katsura4, déjà cité, au bout de 30 minutes sans avoir obtenu de réponse.

L'implantation de gcd et de split utilise aussi le fait de connaître dans certains cas la non-appartenance d'un coefficient de tête aux idéaux premiers associés à la chaîne sur laquelle on travaille. Mais avec la mémorisation des calculs de pgcd dans des tables de hachage, cette optimisation perd de son intérêt.

5 Méthode de Ritt et Wu Wen-Tsün

Définition VI.4 *Soient $T = \{t_1, \ldots, t_l\}$ et $S = \{s_1, \ldots, s_k\}$ deux ensembles triangulaires de $\mathbf{R}_n$. On dira que T est* inférieur à S pour l'ordre de Ritt et Wu avec le raffinement de Lazard *et l'on écrira $T \prec_{rwl} S$ si l'une des conditions suivantes est réalisée*

(i) *il existe un entier $1 \leq i \leq \min(k, l)$ tel que :*

$$(\forall j \in \{1, \ldots, i-1\})\ t_j \sim_{rwl} s_j \quad \text{et} \quad t_i <_{rwl} s_i$$

(ii) $l > k$ et $(\forall j \in \{1, \ldots, k\})\ t_j \sim_{rwl} s_j$

On dira que T est supérieur à S *(pour ce même ordre) et l'on écrira $T \succ_{rwl} S$ si on a :$S \prec_{rwl} T$. Enfin, on dira que S et T sont* incomparables *(pour ce même ordre) dans tous les autres cas c'est-à-dire si l'on a :*

$$l = k \quad \text{et} \quad (\forall j \in \{1, \ldots, k\})\ t_j \sim_{rwl} s_j$$

Remarque VI.7 Il n'existe pas de suites infinies d'ensembles triangulaires strictement décroissantes pour l'ordre $\prec_{rwl}$.

Proposition VI.5 *Soit F une famille finie non vide de polynômes non constants de $\mathbf{R}_n$ et* redOp ? *un prédicat de réduction dans $\mathbf{R}_n$. Alors il existe un sous-ensemble B de F appelé* ensemble basique de F pour le prédicat redOp ? *et tel que :*

(i) *B est triangulaire et auto-réduit au sens de* redOp ?

(ii) *B est minimal pour l'ordre $\prec_{rwl}$ parmi tous les ensembles triangulaires contenus dans F et auto-réduits au sens de* redOp ?

(iii) *de plus l'algorithme suivant calcule un tel ensemble B*

Remarque VI.8 Plus précisément, l'opération basicSet(I, redOp?) suivante retourne $[B, R]$ tel que B soit ensemble basique de F pour redOp ? et $R = F \setminus B$.

basicSet(F, redOp?) ==
    $B := \{\ \}$
    $F :=$ **sort**$(<_{rwl}, I)$
    $R := \{\ \}$
    while not empty ? F **repeat**
        $b :=$ **first** F
        $F :=$ **rest** F
        $B := B \cup \{b\}$
        while (**not empty ?** F) **and** (**not** redOp?($p :=$ **first** F, B)) **repeat**
            $R := R \cup \{p\}$
            $F :=$ **rest** F
    return $[B, R]$

Proposition VI.6 *Soit F une famille non vide de polynômes non constants de $\mathbf{R}_n$ et* redOp ? *un prédicat de réduction dans $\mathbf{R}_n$ (définition III.16 p. 96). Soit B une famille basique de F pour* redOp ?*. Soit de plus un polynôme $p \in \mathbf{R}_n$ non constant réduit par rapport à B pour* redOp ?*. Alors, $I \cup \{p\}$ admet une famille basique B' tel que $B' \prec_{rwl} B$.*

Preuve. $\triangleright$ En effet, on a : $B^{-}_{\mathsf{mvar}(p)} \cup \{p\} \prec_{rwl} B$. $\triangleleft$

Proposition VI.7 *Soit F une famille non vide de polynômes non constants de $\mathbf{R}_n$ et $\mathbf{K}$ une clôture algébrique de $\mathbf{k}$. Soit* redOp ? *un prédicat de réduction dans $\mathbf{R}_n$ et* redOp *une opération de réduction associée (définition III.16 p. 96).*

Il existe un ensemble triangulaire C auto-réduit pour redOp ? *appelé* ensemble caractéristique au sens de Wu de F pour redOp ? *et tel que si l'on pose :*

$$h = \prod_1^l \mathsf{init}(t_i) \text{ avec } C = \{t_1, \ldots, t_l\}$$

alors on a :

(i) $\mathbf{Id}(C) \subseteq \mathbf{Id}(F) \subseteq \mathbf{Id}(C) : h^{\infty}$

(ii) $\mathbf{ZR}_{\mathbf{K}}(C) \subseteq \mathbf{V}_{\mathbf{K}}(F) \subseteq \mathbf{V}_{\mathbf{K}}(C)$

(iii) $\mathbf{V}_{\mathbf{K}}(F) = \mathbf{ZR}_{\mathbf{K}}(C) \cup_1^l \mathbf{V}_{\mathbf{K}}(F \cup \{\mathsf{init}(t_i)\}\ )$

(iv) $\sqrt{F} \longmapsto (\sqrt{C} : h, \sqrt{F + \langle \mathsf{init}(t_1) \rangle}, \ldots, \sqrt{F + \langle \mathsf{init}(t_l) \rangle})$

(v) (F premier et $F \cap \langle \mathsf{init}(t_1), \ldots, \mathsf{init}(t_l) \rangle = \emptyset$) $\Longrightarrow$ $\overline{\mathbf{ZR}_{\mathbf{K}}(C)} = \mathbf{V}_{\mathbf{K}}(F)$

(vi) (F premier et $\overline{\mathbf{ZR}_{\mathbf{K}}(C)} \neq \emptyset$) $\Longrightarrow$ $\overline{\mathbf{ZR}_{\mathbf{K}}(C)} = \mathbf{V}_{\mathbf{K}}(F)$

(vi) (F premier et C normalisé) $\Longrightarrow$ $\overline{\mathbf{ZR}_{\mathbf{K}}(C)} = \mathbf{V}_{\mathbf{K}}(F)$

($viii$) *l'algorithme suivant calcule un tel ensemble C*

Remarque VI.9 Plus précisément, l'opération charSet(F, redOp ?,redOp) suivante retourne soit $\{1\}$ si un polynôme constant non nul appartenant à $\mathbf{Id}(F)$ est découvert au cours des calculs, soit C un ensemble caractéristique au sens de Wu de F pour redOp ?. Notons que si cette opération retourne un ensemble caractéristique C, rien ne prouve en général que $\mathbf{Id}(F)$ ne soit pas l'idéal-unité.

charSet(F, redOp ?,redOp) ==
    $R := F$
    $Q := \emptyset$
    $C := \emptyset$
    while ($R \neq \emptyset$) **and** ($R \cap \mathbf{reg}(\mathbf{R}) = \emptyset$) **repeat**
        $Q := Q \cup R \cup C$
        $[C, Q] :=$ basicSet(Q, redOp ?)
        $R :=$ rewriteSet(Q, C, redOp)
    $R \cap \mathbf{reg}(\mathbf{R}) \neq \emptyset$ => **return**$\{1\}$
    return C

Nous donnons maintenant la preuve de la proposition précédente.

Preuve. $\triangleright$ Nous laissons de côté le cas trivial où $\{1\}$ est retourné par l'algorithme précédent. On remarque tout d'abord que l'algorithme se termine par application de la remarque VI.7 de la proposition VI.6 et des axiomes (ii) et (iii) d'une opération de réduction (définition III.16). Observons maintenant que chaque ensemble C calculé est contenu dans l'ensemble Q *courant*. Du fait de l'axiome (i) d'une opération de réduction, chaque ensemble R calculé est donc contenu dans l'idéal engendré par l'ensemble Q calculé au début de chaque itération de la boucle `while`. Par suite, l'idéal engendré par Q au début d'une itération de la boucle `while` est le même idéal que celui engendré par Q au début de l'itération précédente. Ainsi, à chaque étape on a :

$$\mathbf{Id}(F) = \mathbf{Id}(Q)$$

Et il vient :

$$\mathbf{Id}(C) \subseteq \mathbf{Id}(F)$$

D'autre part, à la fin de la dernière itération de la boucle `while`, on a :

$$(\forall q \in Q)\ (\exists e \in S_E) \mid eq \in \mathbf{Id}(C)$$

où S_E désigne la partie multiplicativement stable de $\mathbf{R}_n$ engendrée par les initiaux des polynômes de C. Par conséquent :

$$(\forall q \in Q)\ (\exists n \in \mathbb{N}) \mid h^n q \in \mathbf{Id}(C)$$

Comme Q engendre $\mathbf{Id}(F)$, on donc bien :

$$\mathbf{Id}(F) \subseteq \mathbf{Id}(C) : h^\infty$$

Ceci termine la preuve de (i) L'assertion (ii) résulte de la précédente, du théorème I.8 et de la proposition I.1. Vérifions (iii). On a clairement :

$$\mathbf{V}_{\mathbf{K}}(F) \supseteq \mathbf{ZR}_{\mathbf{K}}(C) \cup \cup_1^l \mathbf{V}_{\mathbf{K}}(F \cup \{\mathsf{init}(t_i)\}\ )$$

Inversement, si $\zeta \in \mathbf{V}_{\mathbf{K}}(F)$ n'appartient pas à $\mathbf{ZR}_{\mathbf{K}}(C)$ c'est parce qu'il appartient à l'un des $\mathbf{V}_{\mathbf{K}}(F \cup \{\mathsf{init}(t_i)\}\ )$. Ceci prouve (iii). Vérifions (iv). D'après le théorème III.3 p. 101 on sait que :

$$\overline{\mathbf{ZR}_{\mathbf{K}}(C)} = \mathbf{V}_{\mathbf{K}}(C : h^\infty)$$

Par conséquent, par passage aux radicaux (théorème des zéros fort, théorème I.9), il vient :

$$\sqrt{F} = \sqrt{C : h^\infty} \cap \cap_1^l \sqrt{F + \langle \mathsf{init}(t_i)\rangle}$$

Il ne reste plus qu'à remarquer que, avec la proposition I.17, les égalités :

$$\sqrt{C : h^\infty} = \sqrt{\sqrt{C} : h} = \sqrt{C} : h$$

Vérifions (v). Si F engendre un idéal premier, alors cet idéal doit être égal à l'un des idéaux engendrés par l'une des composantes du scindage précédent. S'il s'agit de l'une des composantes de la forme $\sqrt{F + \mathsf{init}(t_i)}$ alors l'un des initiaux des polynômes de C appartient à F. Ceci est impossible par hypothèse, d'où la conclusion. La proposition (vi) résulte de (ii) et du fait que la $\mathbf{V}_{\mathbf{K}}(F)$ ne peut contenir strictement aucune variété affine non vide, puisque F est premier. La proposition (vii) résulte de la précédente et du fait qu'un ensemble triangulaire normalisé est nécessairement consistent (proposition III.21). $\lhd$

Théorème VI.2 (Wu Wen Tsün) *Soit F une famille non vide de polynômes non constants de $\mathbf{R}_n$ et $\mathbf{K}$ une clôture algébrique de $\mathbf{k}$. Soit* redOp ? *un prédicat de réduction dans $\mathbf{R}_n$ et* redOp *une opération de réduction associée. Alors, il existe une famille d'ensembles triangulaires $(C_1, \ldots, C_s)$ auto-réduits pour* redOp ? *appelée* série caractéristique *pour F au sens de* redOp *et telle que :*

$$\mathbf{V}_{\mathbf{K}}(F) \;=\; \cup_1^s \, \mathbf{ZR}_{\mathbf{K}}(C_i)$$

De plus l'algorithme présenté dans la remarque suivante calcule une telle famille.

Remarque VI.10 Nous donnons une version *un peu optimisée* de l'algorithme naïf que l'on déduit sans peine de la proposition VI.7. L'optimisation essentielle, déjà évoquée par Wang ([Wan91]), consiste à ne pas *décomposer* (au sens du théorème VI.2) un ensemble de polynômes qui l'aurait déjà été précédemment. Pour cela l'algorithme manipule un arbre de scindage.

```
charSerie(F, redOp ?,redOp) ==
    A := •[F, ∅, false]
    while not (L := extractSplittingLeaf A) case failed repeat
        •[F, ∅, false] := L
        C := charSet(F, redOp ?,redOp)
        if C = {1}
            then
                L := •[∅, ∅, true]
            else
                I := ∅
                for t ∈ C repeat
                    I := I ∪ {f ∉ R | f ∈ factors(init(t)) }
                if empty ? I
                    then
                        L := L ⟶ •[C, ∅, true]
                    else
                        L := L ⟶ [•[F ∪ C ∪ {f}, ∅, false] for f ∈ I]
        A := updateStatus ! A
    return leaves A
```

Preuve. ▷ L'algorithme se termine du fait de la proposition VI.6. La formule du théorème VI.2 est une conséquence immédiate de la proposition VI.7. ◁

Remarque VI.11 Nous donnons maintenant quelques stratégies ou optimisations proposées dans la littérature, outre celles données dans l'algorithme précédent.

1. l'utilisation de la réduction au sens des initiaux itérés plutôt que la réduction totale permet d'accroître la rapidité d'obtention d'un résultat.
2. l'utilisation de l'odre de Ritt et Wu avec le raffinement de Lazard contraint l'algorithme basisSet à choisir les polynômes les plus *réduits*.
3. le remplacement (légitime) de la ligne $Q := Q \cup R \cup C$ par $Q := F \cup R \cup C$ dans l'algorithme charSet permet en général de travailler avec des polynômes *moins gros*. En effet, les pseudo-divisions successives font *enfler* les polynômes intermédiaires

4. la ligne $Q := Q \cup R \cup C$ peut en fait être remplacé par toute ligne du type $Q :=$ $\mathsf{foo}(F, Q, C, R)$ du moment que $\mathsf{foo}(F, Q, C, R)$ engendre le même idéal que F et admette une famille basique strictement inférieure à C. Wang a beaucoup travaillé sur ce sujet ([Wan91, Wan92a, Wan92b]). Une idée, déjà présente dans le papier original de Wu ([Wu87]) est la suivante. Remplacer la ligne $Q := Q \cup R \cup C$ de l'algorithme charSet par $Q := R \cup C$ et bâptiser le nouvel algorithme ainsi obtenu medialSet(F, redOp ?,redOp) (pour reprendre la terminologie de Wang, [Wan91, Wan92a, Wan92b]). Notons que cet algorithme ne produit pas nécessairement (mais souvent) un ensemble caractéristique de F au sens de Wu. Puis remplacer l'algorithme charSet par le suivant :

charSet(F, redOp ?,redOp) $==$
    $R := F$
    $Q := \emptyset$
    $C := \emptyset$
    while $(R \neq \emptyset)$ **and** $(R \cap \mathbf{reg(R)} = \emptyset)$ **repeat**
        $Q := F \cup R \cup C$
        $C :=$ medialSet(Q, redOp ?,redOp)
        $R :=$ rewriteSet(F, C, redOp)
    $R \cap \mathbf{reg(R)} \neq \emptyset$ => $\{1\}$
    return C

6 Expérimentation

Cette longue section rassemble les résultats des calculs effectués par nos implantations des méthodes de résolution présentées précédemment (et dues à DongMing Wang, Wu Wen-Tsün, Michael Kalkbrener et Daniel Lazard) sur une trentaine d'exemples. Nous n'avons pas voulu nous contenter d'un tableau de temps récapitulatif. Les spécifications des différentes méthodes n'étant pas exactement les mêmes, un tel résumé serait trop réducteur. Nous voulons contraindre le lecteur à ne pas dissocier le temps nécessaire à une méthode pour traiter un problème de *l'utilisabilité* du résultat produit.

Rappelons que ces méthodes ont été implantées en AXIOM dans la version 2.0 de ce logiciel. Les calculs ont été effectués sur une Sparc 10 disposant de 96 méga-octets de mémoire vive. Tous ces calculs se sont déroulés les uns à la suite des autres (sans interruption ou intervention de l'utilisateur) par lecture d'un fichier `.input`. La très grande majorité des exemples traités se trouvent dans la base d'exemples du projet européen PoSSo ([Eur92]). Quelques uns, également, nous ont été communiqués ou ont été créés pour cette expérience.

L'ensemble de ce qui suit se présente comme une session AXIOM eentrecoupée de remarques. Cette session a été produite de la manière suivante. Chaque exemple à traiter se trouve dans un fichier `.input` contenant successivement.

1. Le nom *usuel* de l'exemple (avec sa source) précédé et suivi d'une ligne de tirets.
2. Les initialisations communes à tous nos constructeurs de domaines d'ensembles triangulaires. Essentiellement il s'agit de la liste des variables (rangées par ordre décroissant), de l'anneau des coefficients, et de l'anneau de polynômes. Ces variables sont respectivement notées `ls`, `R` et `P`.
3. les polynômes définissant le système d'équations algébriques à résoudre. Ces polynômes sont généralement notés `p1`, `p2`, ... Ils sont rassemblés dans une liste appelée `lp`.
4. Le calcul de la base lexicographique effectué :
   – par AXIOM si cela peut être fait (en un temps raisonnable)
   – sinon par GB

   Cette base n'apparaît pas systématiquement pour chaque exemple dans la session reproduite ci-après ; en particulier si elle est trop longue pour être imprimée ici.
5. Le calcul de la décomposition en ensembles triangulaires de la variété associée à `lp` par l'algorithme *lexTriangular* (s'il s'agit d'une variété de dimension zéro.).
6. Pour un domaine donné d'ensembles triangulaires `T`
   – le calcul de la décomposition de la variété associée à `lp` en éléments de `T` si ce calcule ne prend pas plus d'une heure. Dans ce cas, ce calcul est effectué 2 fois, le second calcul étant précédé de la commande de *nettoyage de la mémoire* :

   ```
   )lisp (gbc t)
   ```

   De plus les éventuelles variables globales de chacun de nos constructeurs sont réinitialisées au début ou en fin de calcul. Cette commande ainsi que le premier calcul n'apparaissent pas dans ce qui suit. Comme pour la base lexicographique, si le résultat est trop encombrant, il n'est reproduit ici. Notons que, suivant la valeur de `T`, deux cas se présentent. S'il s'agit de la méthode de DongMing Wang, la décomposition est obtenue par :

```
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T
```

Pour les trois autres méthodes, elle est obtenue par :

```
lts := zeroSetSplit(lp)$T
```

En effet, le type du résultat dans la méthode de DongMing Wang n'est pas le même que pour les autres méthodes. Pour la méthode de DongMing Wang, il s'agit d'une liste d'enregistrements dont le premier champ, appelé `close`, est un ensemble triangulaire et le second, appelé `open`, est une liste de polynômes, chacun des termes de cette liste représentant bien sur un système quasi-algébrique triangulaire. Pour les autres méthodes, il s'agit simplement d'une liste d'ensembles triangulaires.

– Le calcul de l'idéal saturé associé à chacun des systèmes ou ensembles triangulaires de la décomposition précédente. S'il s'agit de la méthode de DongMing Wang, ce calcul est obtenu par :

```
lps : List GPS := [[lexSaturate(ts.close,ts.open)$T]$GPS for ts in ltqas]
```

Pour les trois autres méthodes, il est obtenu par :

```
lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]
```

Rappelons que ces calculs d'idéaux saturés se font grâce à GB (qui calcule les bases de Gröbner nécessaires) et à notre paquetage `GBLUP` qui implante l'algorithme. Ces idéaux saturés sont donnés sous forme de base lexicographique et les temps des différentes opérations effectuées par GB ne sont en général pas reproduits ici. Ces calculs d'idéaux saturés permettent d'une part de détecter les systèmes ou ensembles triangulaires inconsistants produits dans une décomposition de Wang ou Wu. Ils permettent aussi une meilleure comparaison des décompositions. Lorsque dans une décomposition, chaque ensemble triangulaire est clairement un système de générateurs de son idéal saturé, la liste des idéaux saturés correspondante n'est pas reproduite ici. Enfin, pour certains gros exemples, cette liste n'a pu être calculée.

– L'élimination dans la liste `lps` précédente de chaque idéal saturé dont le radical est contenu dans celui d'un autre idéal saturé. Dans tous les cas, ce calcul est obtenu par :

```
lps := eliminateRedundantVarieties(lps)$GPS
```

Les buts de ce calcul sont nombreux. Tout d'abord rendre plus concises les différentes listes `lps` sur un exemple donné. Puis détecter qu'une méthode a su mieux qu'une autre décomposer les composantes de dimension maximum de l'idéal associé à `lp`. Ceci permet aussi de comparer nos décompositions de Wang, Wu, Kalkbrener et Lazard avec la décomposition primaire minimale de l'idéal associé à `lp` (lorsque cette dernière peut être calculée). Enfin, cet étape de nettoyage apparaît expérimentalement comme une optimisation pour l'étape suivante.

– La vérification (au sens de la fermeture) de la décomposition `ltqas` ou `lts` . Plus précisément, on vérifie que l'intersection des radicaux des idéaux saturés de la liste `lps` est égale au radical de l'idéal associée à `lp`. Dans tous les cas, ce calcul est obtenu par :

```
varietySplit?(lps,lp)$GPS
```

Ce calcul permet de vérifier une décomposition de Kalkbrener dans tous les cas. Pour une décomposition de Wu ou Lazard, cela constitue aussi une vérification si chaque ensemble triangulaire est un système de générateurs du radical de son idéal saturé. Ceci est toujours le cas pour une décomposition de Lazard en dimension zéro.

```
--------------------------------------------------------------------------------
--% Easy example4
--------------------------------------------------------------------------------
```

Ce premier exemple, que nous avons fabriqué, illustre en particulier le fait qu'une quasi-composante produite par la méthode de Wu (telle que nous l'avons implantée) peut être inconsistante.

```
ls : List Symbol := [d,c,b,a];
V := OVAR ls;
R := Integer;
E := IndexedExponents V;
P := NSMP(R, V);
LP := List(P);
GPS := GPOLSET(R,E,V,P);

p1 := (a*(a + 1))::P;
p2 := (b*(b + 2)  )::P;
p3 := (a*b*c**3 + (a + 1)*(b + 2)*c**2 + c + 1)::P;
p4 := (a*(b + 2)*d**3 + b*(a + 1)*d**2 + d + c + 1)::P;
lp := [p1,p2,p3,p4]

     2       2            3                        2
   [a  + a, b  + 2b, a b c  + ((a + 1)b + 2a + 2)c  + c + 1,
                3             2
    (a b + 2a)d  + (a + 1)b d  + d + c + 1]


gb := reverse groebner lp

     2       2
   [a  + a, b  + 2b, (b - 2a)c + b - 2a,
             2
    (2a + 2)c  + (a + 1)c + (a + 1)b + a + 1,
      3
    4c  + (- a + 1)c + (- 5a - 3)b - 7a - 1,
                                                   2
    (b + 2a + 2)d + (4a + 2)c + 4a + 2, (c + 1)d + c  + 2c + 1,
             2
    (2a + 2)d  + (- a - 1)d + (- 4a - 4)c + (a + 1)b - 2a - 2,
      3                 2
    4d  + (a - 1)d + 12c  + (2a + 10)c + (5a + 3)b + 8a + 4]

               Time: 0.03 (IN) + 0.88 (EV) + 0.22 (OT) + 1.12 (GC) = 2.25 sec

T := LXATSET(R,E,V,P);
lts := lexTriangular(gb)$T

                   3                             2
   [{a + 1,b + 2,2c  + c + 1,d + c + 1}, {a,b,2c  + c + 1,d + c + 1},
                     3                         2
    {a + 1,b,c + 1,2d  - d}, {a,b + 2,c + 1,2d  - d}]

                           Time: 0.02 (IN) + 0.53 (EV) + 0.62 (OT) = 1.17 sec
```

```
T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T

                                     2
   [[close= {a,b,((a + 1)b + 2a + 2)c  + c + 1,d + c + 1},open= []],

                               3                       2
     [close= {a + 1,b + 2,a b c  + ((a + 1)b + 2a + 2)c  + c + 1,d + c + 1},
      open= []]
     ,
                                     2
    [close= {a,b + 2,c + 1,(a + 1)b d  + d + c + 1},open= []],
                                      3             2
    [close= {a + 1,b,c + 1,(a b + 2a)d  + (a + 1)b d  + d + c + 1},open= []]]

                          Time: 0.02 (IN) + 1.07 (EV) + 0.17 (OT) = 1.25 sec

lps : List GPS := [[lexSaturate(ts.close,ts.open)$T]$GPS for ts in ltqas]

          2                                   3
   [{a,b,2c  + c + 1,d + c + 1}, {a + 1,b + 2,2c  + c + 1,d + c + 1},
                    2                       3
    {a,b + 2,c + 1,2d  - d}, {a + 1,b,c + 1,2d  - d}]

lps := eliminateRedundantVarieties(lps)$GPS

          2                                     3
   [{a,b,2c  + c + 1,d + c + 1}, {a + 1,b,c + 1,2d  - d},
                  3                                     2
    {a + 1,b + 2,2c  + c + 1,d + c + 1}, {a,b + 2,c + 1,2d  - d}]

varietySplit?(lps,lp)$GPS

   (22)  true
                          Time: 6.68 (EV) + 0.07 (OT) + 2.43 (GC) = 9.18 sec

T := WuWenTsunTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

          2                                     2
   [{a,b,2c  + c + 1,d + c + 1}, {a,b + 2,c + 1,2d  - d},
       2             2           2                                3
    {a,b  + 2b,(b + 2)c  + c + 1,b d  + d + c + 1}, {a + 1,b,c + 1,2d  - d},
                  3
    {a + 1,b + 2,2c  + c + 1,d + c + 1},
      2             2           3
    {a  + a,b,(2a + 2)c  + c + 1,2a d  + d + c + 1},
      2               3            2
    {a  + a,b + 2,2a c  - c - 1,(2a + 2)d  - d - c - 1},
       2       2          3                       2
     {a  + a, b  + 2b, a b c  + ((a + 1)b + 2a + 2)c  + c + 1,
                3              2
      (a b + 2a)d  + (a + 1)b d  + d + c + 1}
```

```
   ]
                              Time: 0.02 (IN) + 0.57 (EV) + 0.12 (OT) = 0.70 sec

lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]

           2                                       2
   [{a,b,2c  + c + 1,d + c + 1}, {a,b + 2,c + 1,2d  - d}, {1},
                     3                       3
    {a + 1,b,c + 1,2d  - d}, {a + 1,b + 2,2c  + c + 1,d + c + 1}, {1}, {1},
    {1}]
```

Notons que 4 des quasi-composantes de Wu sont inconsistentes, puisque leurs idéaux saturés sont égaux à l'idéal-unité. On remarque aussi que la liste des saturés des quasi-composantes consistantes produites par notre implantation de la méthode de Wu est la même, à l'ordre près, que la liste des saturés des systèmes quasi-algébriques produits dans notre décomposition de Wang. Par suite, l'élimination des variétés redondantes et le calcul de leur réunion n'est pas nécessaire. Nous passons donc à la méthode de Kalkbrener.

```
T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T

                   3                                       2
   [{a + 1,b + 2,2c  + c + 1,d + c + 1}, {a + 1,b,c + 1,2d  - 1},
    {a + 1,b,c + 1,d}, {a,b + 2,c + 1,2d - 1}, {a,b + 2,c + 1,d},
           2
    {a,b,2c  + c + 1,d + c + 1}]

                              Time: 0.02 (IN) + 0.15 (EV) + 0.07 (OT) = 0.23 sec
```

Notons l'efficacité de la méthode de Kalkbrener (temps de calcul et lisibilité du résultat) sur cet exemple. D'autre part, chaque chaîne régulière ne comptant que des initiaux constants (i.e. dans l'anneau de base) est égale à la base lexicographique de son idéal saturé. Le calcul de ces saturés n'est donc pas nécessaire. De plus les variétés associées sont clairement 2 à 2 incomparables pour l'inclusion : il s'agit du reste de variétés irréductibles comme on le retrouve à l'aide de la décomposition primaire minimale de `lp` ci-après. Il ne reste donc plus qu'à prendre la réunion de ces variétés.

Le fait que chaque ensemble triangulaire d'une décomposition soit la base lexicographique de son idéal saturé et que ces idéaux saturés soient incomparables pour l'inclusion est un phénomène fréquent en dimension 0 pour la méthode de Kalkbrener et général (toujours en dimension 0) pour celle de Lazard.

```
lps : List GPS := [[ts::LP]$GPS for ts in lts];
varietySplit?(lps,lp)$GPS

   true
                             Time: 8.25 (EV) + 0.03 (OT) + 3.75 (GC) = 12.03 sec

primaryDecomp([lp]$GPS)

    GB time: 0.066667000000000004

                 3                           2
   [[d + c + 1,2c  + c + 1,b + 2,a + 1], [2d  - 1,c + 1,b,a + 1],
```

```
                 2
    [d + c + 1,2c  + c + 1,b,a], [d,c + 1,b,a + 1], [2d - 1,c + 1,b + 2,a],
    [d,c + 1,b + 2,a]]

            Time: 0.03 (IN) + 14.07 (EV) + 0.45 (OT) + 10.50 (GC) = 25.05 sec

T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

                   3                            2
   [{a + 1,b + 2,2c  + c + 1,d + c + 1}, {a,b,2c  + c + 1,d + c + 1},
                     2                        3
    {a,b + 2,c + 1,2d  - d}, {a + 1,b,c + 1,2d  - d}]

                                       Time: 0.17 (EV) + 0.07 (OT) = 0.23 sec

lps : List GPS := [[ts::LP]$GPS for ts in lts];
varietySplit?(lps,lp)$GPS

   true
                          Time: 6.45 (EV) + 0.02 (OT) + 3.80 (GC) = 10.27 sec

-----------------------------------------------------------------------------
--% Easy Example 5
-----------------------------------------------------------------------------
```

Nous retrouvons ici un exemple familier (voir l'introduction du premier chapitre).

```
ls : List Symbol := [x,y,z];
V := OVAR ls;
R := Integer;
E := IndexedExponents V;
P := NSMP(R, V);
LP := List(P);
GPS := GPOLSET(R,E,V,P);

p1 : P := (x^2 + y + z - 1)::P;
p2 : P := (x + y^2 + z  -1)::P;
p3 : P := (x + y + z^2 -1)::P;
lp := [p1,p2,p3]

          2                  2                  2
   (10)  [x  + y + z - 1,x + y  + z - 1,x + y + z  - 1]

T := LXATSET(R,E,V,P);
gb := reverse groebner lp

          6     4     3    2   2     4    2  2        2              2
   (14)  [z  - 4z  + 4z  - z ,2z y + z  - z ,y  - y - z  + z,x + y + z  - 1]

                           Time: 0.03 (IN) + 0.23 (EV) + 0.03 (OT) = 0.30 sec
lts := lexTriangular(gb)

      2  2                     4     2               2           2
   [{z ,y  - y + z,x + y - 1},{z  - 4z  + 4z - 1,2y + z  - 1,2x + z  - 1}]
```

```
                           Time: 0.02 (IN) + 0.43 (EV) + 0.05 (OT) = 0.50 sec

T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T

                2              2
   [[close= {z,y  - y,x + y + z  - 1},open= []],
              3    2               2              2
    [close= {z  + z  - 3z + 1,2y + z  - 1,x + y + z  - 1},open= [z]]]

                                       Time: 0.25 (EV) + 0.07 (OT) = 0.32 sec

lps : List GPS := [[lexSaturate(ts.close,ts.open)$T]$GPS for ts in ltqas]

              2                  3    2                2           2
   (19)  [{z,y  - y,x + y - 1},{z  + z  - 3z + 1,2y + z  - 1,2x + z  - 1}]

                           Time: 0.10 (IN) + 1.10 (EV) + 0.12 (OT) = 1.32 sec

varietySplit?(lps,lp)$GPS

   (20)  true
               Time: 0.02 (IN) + 1.37 (EV) + 0.02 (OT) + 1.32 (GC) = 2.72 sec

T := WuWenTsunTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

        2
   [{z,y  - y,x + y - 1},
      4    3     2       2     3     2       2     3     2
    {z  + z  - 3z  + z,2z y - z  + 2z  - z,2z x - z  + 2z  - z}]

                           Time: 0.02 (IN) + 0.13 (EV) + 0.05 (OT) = 0.20 sec

lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]

              2                  3    2                2           2
   (25)  [{z,y  - y,x + y - 1},{z  + z  - 3z + 1,2y + z  - 1,2x + z  - 1}]

                           Time: 0.02 (IN) + 0.72 (EV) + 0.02 (OT) = 0.75 sec

T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T

   [{z - 1,y,x}, {z,y - 1,x}, {z,y,x - 1},
      2
    {z  + 2z - 1,(2z - 1)y + 5z - 2,x + y - 2z}]

                           Time: 0.02 (IN) + 0.27 (EV) + 0.07 (OT) = 0.35 sec
```

Contrairement, à l'exemple précédent, les chaînes régulières produites ci-dessus par notre implantation de la méthode de Kalkbrener ne sont pas égales aux bases lexicographiques de leurs idéaux saturés. Cependant, elles décrivent encore les idéaux premiers associés à `lp`.

Dans le décomposition de Lazard ci-après, on retrouve les saturés associés aux décompositions de Wu et Wang. Aucun travail supplémentaire n'est donc nécessaire pour vérifier cette dernière décomposition.

```
lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]

                                                     2
   (31)  [{z - 1,y,x},{z,y - 1,x},{z,y,x - 1},{z  + 2z - 1,y - z,x - z}]

                           Time: 0.02 (IN) + 0.57 (EV) + 0.05 (OT) = 0.63 sec

varietySplit?(lps,lp)$GPS

   (32)  true
                           Time: 0.02 (IN) + 2.08 (EV) + 0.02 (OT) = 2.12 sec

primaryDecomp([lp]$GPS)

    GB time: 0.016667000000000001
   (39)
                      2                               2
   [[- x + z,- y + z,z  + 2z - 1], [- x + z,y + z - 1,z ],
                        2                          2
    [x + z - 1,- y + z,z ], [x + z - 1,y + z - 1,z  - 2z + 1]]

                           Time: 0.02 (IN) + 5.73 (EV) + 0.08 (OT) = 5.83 sec

T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

            3    2                 2            2          2
   (35)  [{z  + z  - 3z + 1,2y + z  - 1,2x + z  - 1},{z,y  - y,x + y - 1}]

                           Time: 0.02 (IN) + 0.13 (EV) + 0.05 (OT) = 0.20 sec


-------------------------------------------------------------------------------
--% Discrimant problem for degree 4
-------------------------------------------------------------------------------
```

Voici un premier exemple de dimension (strictement) positive. On considère un polynôme `f` univarié unitaire de degré 4 et dont les quatre autres coefficients sont des indéterminées. Le système d'équations `lp` à résoudre est contitué du discrimant `d` de `f` et des dérivées partielles de `d` par rapport à chacun des coefficients (indéterminés) de `f`.

On vérifie que la variété associée à `lp` est réunion de deux variétés irréductibles de dimension 2.

```
ls : List Symbol := [x,a4,a3,a2,a1];
V := OVAR ls;
R := Integer;
E := IndexedExponents V;
P := NSMP(R, V);
LP := List(P);
```

```
GPS := GPOLSET(R,E,V,P)

f := (x^4 + a1 * x^3 + a2 * x^2 + a3 * x + a4)::P;
d := discriminant(f,x)$P;
p1 := differentiate(d,a1)$P;
p2 := differentiate(d,a2)$P;
p3 := differentiate(d,a3)$P;
p4 := differentiate(d,a4)$P;

lp := [d,p1,p2,p3,p4];

groebner lp

   [
            2                        2          2           4                  2   2
      384a4  + (- 192a1 a3 - 128a2  + 144a1 a2 - 27a1 )a4 + (72a2 - 3a1 )a3
    +
                 2      3           4      2  3
      (- 40a1 a2  + 9a1 a2)a3 + 8a2  - 2a1 a2
    ,
               2         3           3         2  2         4          6
      (3456a3  - 432a1 a3 - 512a2  - 288a1 a2  + 432a1 a2 - 81a1 )a4
    +
                  3          2          2           4   2
      - 2160a1 a3  + (864a2  + 1512a1 a2 - 297a1 )a3
    +
                  3       3  2        5             5      2  4      4  3
      (- 832a1 a2  + 72a1 a2  + 27a1 a2)a3 + 128a2  - 8a1 a2  - 6a1 a2
    ,
                       2               2        3         5           3
      ((576a2 - 216a1 )a3 - 288a1 a2  + 180a1 a2 - 27a1 )a4 - 216a3
    +
                     3   2           3       2  2      4              4     3  3
    (180a1 a2 - 27a1 )a3  + (- 16a2  - 36a1 a2  + 9a1 a2)a3 + 8a1 a2  - 2a1 a2
    ,
              4          2  3         4  2         6         8            4
      (4096a2  - 6144a1 a2  + 3456a1 a2  - 864a1 a2 + 81a1 )a4 - 10368a3
    +
                             3   3
      (20736a1 a2 - 5184a1 )a3
    +
                3         2  2          4          6   2
      (- 7680a2  - 6912a1 a2  + 4536a1 a2 - 567a1 )a3
    +
                 4         3  3        5  2       7              6        2  5
      (6656a1 a2  - 3840a1 a2  + 648a1 a2  - 27a1 a2)a3 - 1024a2  + 640a1 a2
    +
              4  4      6  3
      - 120a1 a2  + 6a1 a2
    ,
              5                          3   4
      93312a3  + (- 233280a1 a2 + 58320a1 )a3
    +
               3           2  2          4            6   3
      (55296a2  + 171072a1 a2  - 93312a1 a2 + 11664a1 )a3
    +
                    4          3  3          5  2          7          9   2
      (- 82944a1 a2  - 2592a1 a2  + 29160a1 a2  - 8748a1 a2 + 729a1 )a3
    +
              6          2  5          4  4         6  3        8  2
      (8192a2  + 23040a1 a2  - 20952a1 a2  + 5616a1 a2  - 486a1 a2 )a3
```

```
    +
                7         3  6        5  5       7  4
      - 4096a1 a2  + 3328a1 a2  - 900a1 a2  + 81a1 a2
    ]
```

```
                             Time: 0.03 (IN) + 0.82 (EV) + 0.43 (OT) = 1.28 sec
```

```
T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T
```

```
   newTriangulation

   [
     [
       close =
                  2           3
         {8a2 - 3a1 , 16a3 - a1 ,

                    2                             2           2            4
             442368a4  + (- 221184a1 a3 - 147456a2  + 165888a1 a2 - 31104a1 )a4
           +
                       2           3           5            4          2  3
             (119808a1 a2  - 48384a1 a2 + 2160a1 )a3 + 21504a2  - 78848a1 a2
           +
                     4  2         6          8
             44640a1 a2  - 7560a1 a2 + 243a1
           }
         ,
      open= []]
     ,

     [
       close =
                  2           3
         {8a2 - 3a1 , 16a3 - a1 ,

                        2                          3           3         2  2
                 3456a3  + (- 6912a1 a2 + 2160a1 )a3 - 512a2  + 3168a1 a2
               +
                          4          6
                 - 1728a1 a2 + 243a1
            *
               a4
           +
                     2         2          4   2
             (864a2  - 288a1 a2 - 27a1 )a3
           +
                       3         3  2        5              5        2  4       4  3
           (- 672a1 a2  + 432a1 a2  - 63a1 a2)a3 + 128a2  - 88a1 a2  + 14a1 a2
           }
         ,
                        3
      open= [16a3 - a1 ]]
     ,

     [
       close =
                              3
         {8a3 - 4a1 a2 + a1 ,

                        2                          3           3         2  2
                 3456a3  + (- 6912a1 a2 + 2160a1 )a3 - 512a2  + 3168a1 a2
               +
                          4          6
                 - 1728a1 a2 + 243a1
            *
               a4
```

```
          +
                 2        2         4   2
            (864a2  - 288a1 a2 - 27a1 )a3
          +
                    3        3  2       5             5       2  4       4  3
          (- 672a1 a2  + 432a1 a2  - 63a1 a2)a3 + 128a2  - 88a1 a2  + 14a1 a2
          }
       ,
                    2
      open= [8a2 - 3a1 ]]
     ,

     [
       close =
             2                  3         3      2  2
         {108a3  + (- 108a1 a2 + 27a1 )a3 + 32a2  - 9a1 a2 ,
                    2         2             3
          (72a2 - 27a1 )a4 - 27a3  + 9a1 a2 a3 - 2a2 }
       ,
                    2                   3
      open= [8a2 - 3a1 ,8a3 - 4a1 a2 + a1 ]]
     ]
           Time: 0.02 (IN) + 5.35 (EV) + 0.57 (OT) + 1.55 (GC) = 7.48 sec
```

```
lps : List GPS := [[lexSaturate(ts.close,ts.open)$T]$GPS for ts in ltqas]
```

```
              2          3        2        4       8
   [{8a2 - 3a1 ,16a3 - a1 ,65536a4  - 512a1 a4 + a1 }, {1},
                        3            2      2       4
    {8a3 - 4a1 a2 + a1 ,64a4 - 16a2  + 8a1 a2 - a1 },
          2                     3          3      2  2                 2
    {108a3  + (- 108a1 a2 + 27a1 )a3 + 32a2  - 9a1 a2 ,12a4 - 3a1 a3 + a2 }]

                      Time: 0.08 (IN) + 3.80 (EV) + 0.17 (OT) = 4.05 sec
```

```
lps := eliminateRedundantVarieties(lps)$GPS
```

```
                        3            2      2       4
   [{8a3 - 4a1 a2 + a1 ,64a4 - 16a2  + 8a1 a2 - a1 },
          2                     3          3      2  2                 2
    {108a3  + (- 108a1 a2 + 27a1 )a3 + 32a2  - 9a1 a2 ,12a4 - 3a1 a3 + a2 }]

           Time: 0.02 (IN) + 4.00 (EV) + 0.13 (OT) + 1.55 (GC) = 5.70 sec
```

```
varietySplit?(lps,lp)$GPS
```

```
   (20)  true
                                Time: 0.02 (IN) + 3.45 (EV) = 3.47 sec
```

Dans le calcul de notre décomposition de Wang, on notera tout d'abord que l'optimisation `newTriangulation` est appelée. Cette phase du calcul permet de *nettoyer* les systèmes quasi-algébriques produits par l'*algorithme brut* de Wang (voir la section décrivant la méthode de Wang). Ceci permet de découvrir facilement que le second système est inconsistent (ce que nous aurions pu automatiser un peu plus). D'autre part, notons qu'encore une fois la décomposition de Wang est *assez voisine* de celle de Wu (ci-après). Plus précisément, elles ont les mêmes familles de radicaux d'idéaux saturés. Par suite, la vérification de la décomposition de Wu n'est pas nécessaire. Voici cette décomposition.

```
T := WuWenTsunTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

             2         3           4
   [{8a2 - 3a1 ,16a3 - a1 ,256a4 - a1 },

                 2             2       3        5
     {(64a2 - 24a1 )a3 - 32a1 a2  + 20a1 a2 - 3a1 ,
                   2           3        2  2       4        6
      (512a2 - 192a1 )a4 - 128a2  + 112a1 a2  - 32a1 a2 + 3a1 }
     ,

     {
             3                        3   2
         864a3  + (- 1296a1 a2 + 324a1 )a3
       +
              3        2  2        4         6              4       3  3
         (256a2  + 360a1 a2  - 216a1 a2 + 27a1 )a3 - 128a1 a2  + 68a1 a2
       +
             5  2
         - 9a1 a2
       ,

                       2              2        3         5
         ((768a2 - 288a1 )a3 - 384a1 a2  + 240a1 a2 - 36a1 )a4
       +
                          3   2        3       2  2       4        6
         (- 192a1 a2 + 72a1 )a3  + (64a2  + 72a1 a2  - 60a1 a2 + 9a1 )a3
       +
                4       3  3      5  2
         - 32a1 a2  + 20a1 a2  - 3a1 a2
       }]
              Time: 0.02 (IN) + 3.27 (EV) + 0.22 (OT) + 1.55 (GC) = 5.05 sec

lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]

             2         3           4
   [{8a2 - 3a1 ,16a3 - a1 ,256a4 - a1 },
                      3            2      2       4
    {8a3 - 4a1 a2 + a1 ,64a4 - 16a2  + 8a1 a2 - a1 },
          2                    3          3      2  2                  2
    {108a3  + (- 108a1 a2 + 27a1 )a3 + 32a2  - 9a1 a2 ,12a4 - 3a1 a3 + a2 }]

                            Time: 0.03 (IN) + 3.13 (EV) + 0.08 (OT) = 3.25 sec

T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T
   [
          2                    3          3      2  2
     {108a3  + (- 108a1 a2 + 27a1 )a3 + 32a2  - 9a1 a2 ,

                                3           2        2         4             3
         (864a3 - 432a1 a2 + 108a1 )a4 + (72a2  - 108a1 a2 + 27a1 )a3 + 28a1 a2
```

```
        +
               3  2
          - 9a1 a2
        }
      ,
                         3            2      2       4
    {8a3 - 4a1 a2 + a1 ,64a4 - 16a2  + 8a1 a2 - a1 },
           2          3          4
    {8a2 - 3a1 ,16a3 - a1 ,256a4 - a1 }]

                              Time: 0.02 (IN) + 2.75 (EV) + 0.13 (OT) = 2.90 sec
```

```
lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]

          2                     3          3      2  2                  2
   [{108a3  + (- 108a1 a2 + 27a1 )a3 + 32a2  - 9a1 a2 ,12a4 - 3a1 a3 + a2 },
                         3            2      2       4
    {8a3 - 4a1 a2 + a1 ,64a4 - 16a2  + 8a1 a2 - a1 },
           2          3          4
    {8a2 - 3a1 ,16a3 - a1 ,256a4 - a1 }]

                              Time: 0.02 (IN) + 1.30 (EV) + 0.10 (OT) = 1.42 sec
```

Notons, qu'à la normalisation près de l'initial du second polynôme dans la première chaîne régulière, la décomposition de Kalkbrener est identique à la famille des saturés de la décomposition de Wu. Bref, sur cet exemple, seule la décomposition de Lazard (ci-après) se distingue en donnant directement les deux variétés irréductibles obtenues à partir de chacune des 3 autres décompositions *après nettoyage*.

```
T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

          2                     3          3      2  2                  2
   [{108a3  + (- 108a1 a2 + 27a1 )a3 + 32a2  - 9a1 a2 ,12a4 - 3a1 a3 + a2 },
                         3            2      2       4
    {8a3 - 4a1 a2 + a1 ,64a4 - 16a2  + 8a1 a2 - a1 }]

                                            Time: 2.23 (EV) + 0.07 (OT) = 2.30 sec
```

```
llq := primaryDecomp([lp]$GPS);

    GB time: 0.066667000000000004

          Time: 0.02 (IN) + 111.20 (EV) + 0.48 (OT) + 63.13 (GC) = 174.83 sec
```

```
[lexRadical([lq]$GPS) for lq in llq]

    GB time: 0.066667000000000004    GB time: 0.016667000000000001

                          2      2                     3          3      2  2
   [[12a4 - 3a1 a3 + a2 ,108a3  + (- 108a1 a2 + 27a1 )a3 + 32a2  - 9a1 a2 ],
                 2      2       4                  3
    [64a4 - 16a2  + 8a1 a2 - a1 ,8a3 - 4a1 a2 + a1 ]]
```

```
              Time: 0.03 (IN) + 8.83 (EV) + 0.05 (OT) + 3.90 (GC) = 12.82 sec


-----------------------------------------------------------------------------
--% Cyclic 4
-----------------------------------------------------------------------------
```

A partir de cet exemple, nous ne reproduisons plus dans notre session AXIOM que les initialisations nécessaires à son intelligibilité. Sur l'exemple qui suit on notera que les 4 méthodes décomposent le radical de l'idéal engendré par `lp` et ne *s'égarent* pas dans les différentes composantes de la décomposition primaire minimale ci-dessous. Avec les deux premières méthodes, où l'usage de la factorisation sans facteurs carrés est plus une optimisation (utilisée pour *nettoyer*) qu'un ingrédient de la méthode, les décompositions sont moins concises que celles produites par les méthodes de Kalkbrener et Lazard.

```
ls : List Symbol := [a,b,c,d];
R := Integer;
lp := [p1,p2,p3,p4]

   [a + b + c + d,(b + d)a + c b + d c,((c + d)b + d c)a + d c b,d c b a - 1]

groebner lp
                    2           2              4 2           2
   [a + b + c + d, b  + 2d b + d , (c - d)b + d c  + d c - 2d ,
      4          5       2 3    3 2            6    2  2    4
    (d  - 1)b + d  - d, d c  + d c  - c - d, (d  - d )c  - d  + 1]

                           Time: 0.03 (IN) + 0.35 (EV) + 0.08 (OT) = 0.47 sec

primaryDecomp([lp]$GPS)

                2                                      2
   [[a + b + 2,b  + 2b + 1,(d - 1)b + d - 1,c + d - 2,d  - 2d + 1],
                          2                            2
    [a + d - 2,b + c + 2,c  + 2c + 1,(d - 1)c + d - 1,d  - 2d + 1],
                          2                            2
    [a + d + 2,b + c - 2,c  - 2c + 1,(d + 1)c - d - 1,d  + 2d + 1],
                2                                      2
    [a + b - 2,b  - 2b + 1,(d + 1)b - d - 1,c + d + 2,d  + 2d + 1],

             3                3        2           2    2          3
     [- a + d  + 2d, b + c + d  + 3d, c  + 2d c + d , (d  + 1)c + d  + d,
       4     2
      d  + 2d  + 1]
     ,

               3        2           2    2          3             3
     [a + b + d  + 3d, b  + 2d b + d , (d  + 1)b + d  + d, - c + d  + 2d,
       4     2
      d  + 2d  + 1]
     ,
    [a + c,b + d,d c + 1], [a + c,b + d,d c - 1]]

            Time: 0.03 (IN) + 35.90 (EV) + 0.23 (OT) + 10.73 (GC) = 46.90 sec
```

```
T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T

              4
   [[close= {d  - 1,c - d,b + d,a + b + c + d},open= [d]],
              4             2     3      3
    [close= {d  - 1,c + d,(d c - d )b + d c - 1,a + b + c + d},open= [d]],
              2 2                                   4
    [close= {d c  - 1,b + d,a + b + c + d},open= [d,d  - 1]]]

                           Time: 0.02 (IN) + 0.72 (EV) + 0.12 (OT) = 0.85 sec

lps : List GPS := [[lexSaturate(ts.close,ts.open)$T]$GPS for ts in ltqas]

      4                          4
   [{d  - 1,c - d,b + d,a + d}, {d  - 1,c + d,b + d,a - d},
      2 2
    {d c  - 1,b + d,a + c}]

                           Time: 0.08 (IN) + 2.00 (EV) + 0.10 (OT) = 2.18 sec

lps := eliminateRedundantVarieties(lps)$GPS

            2 2
   (19)  [{d c  - 1,b + d,a + c}]

                           Time: 0.02 (IN) + 1.13 (EV) + 0.02 (OT) = 1.17 sec

T := WuWenTsunTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

      4            2              2
   [{d  - 1,d c - d ,b + d,d a + d },
      4      2    2   2     3      3        2     3      3
    {d  - 1,c  - d ,(d c - d )b + d c - 1,(d c - d )a - d c + 1},
      2 4       4      2    2   2    2        2    3   2    2      3    2
    {d c  + (- d  - 1)c  + d ,(c  - d )b + d c  - d ,(c  - d )a + c  - d c}]

                           Time: 0.02 (IN) + 0.35 (EV) + 0.12 (OT) = 0.48 sec

lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]

      4                          4
   [{d  - 1,c - d,b + d,a + d}, {d  - 1,c + d,b + d,a - d},
      2 2
    {d c  - 1,b + d,a + c}]

T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T

            2 2
   (29)  [{d c  - 1,b + d,a + c}]
                                       Time: 0.13 (EV) + 0.05 (OT) = 0.18 sec
```

```
T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

            2 2
   (35)  [{d c  - 1,b + d,a + c}]
                                            Time: 0.27 (EV) + 0.03 (OT) = 0.30 sec

-----------------------------------------------------------------------------
--% Moeller example 4 (from the PoSSo data base)
-----------------------------------------------------------------------------
```

Voici maintenant le prototype de la situation où les méthodes de triangulation ne sont généralement pas compétitives vis-à-vis des méthodes de calcul de bases de Gröbner lexicographiques : la base lexicographique de notre ensemble de polynômes `lp` est du type *shape lemma* (i.e. l'idéal associé est de dimension zéro, ladite base est un ensemble triangulaire et seul son polynôme univarié n'est pas de degré 1). En effet, réduire par rapport à un tel ensemble au sens des bases de Gröbner ou au sens des ensembles triangulaires (à l'aide de la pseudo-division *standard*) revient au même théoriquement. Mais la seconde réduction apparaît expérimentalement plus coteuse (voir dans le premier chapitre la section sur les réductions et images modulaires).

De plus, dans notre implantation de la méthode de Wu, si pour les calculs intermédiaires, nous utilisons la réduction au sens des initiaux itérés, nous *nettoyons* cependant les ensembles triangulaires obtenus à l'aide de la réduction forte. Ceci est intéressant sur certains exemples pour *y voir plus clair*, mais se paye ici. Nous donnons donc également pour la méthode Wu le résultat non fortement réduit. Du reste, dans les décompositions de Wang et Kalkbrener, où ce nettoyage (par la réduction forte) n'est fait pas, nous retrouvons le résultat *non nettoyé* de Wu.

Le cas de la décomposition de Lazard est particulier. En effet, chaque ensemble triangulaire produit devant être normalisé, on doit *aller plus loin dans les calculs* (sur cet exemple) que dans les trois autres méthodes. Ceci conduit en fait à retrouver la base lexicographique.

Notons qu'aucune des 4 méthodes n'a découvert que le polynôme univarié (i.e. le polynôme en `v`) était divisible par `2 * v -3`. En particulier, la méthode de Kalkbrener ne *s'est pas rapprochée* de la décomposition en variétés irréductibles comme dans les précédents exemples de dimension 0.

```
ls : List Symbol := [x,y,z,t,u,v];
R := Integer;

lp := [p1,p2,p3,p4,p5,p6]

   [y + u + v - 1, z + t + 2u - 3, y + t + 2v - 1, x - y - z - t - u - v,
              2        3
    31250u t x  - 1569z y, - 587t y + 15625v z]

gb := reverse groebner lp;

                              Time: 0.02 (IN) + 0.57 (EV) + 0.02 (OT) = 0.60 sec
[head(g) for g in gb]

                             8
  [8724935291855297898986v ,
```

```
      53668068257504085191741240048305018795214839641836688939783792219572488 35
      96739896
    *
       u
     ,

      536680682575040851917412400483050187952148396418366889397837922195724883 5
      96739896
    *
       t
     ,

      178893560858346950639137466827683395984049465472788963132612640731908294 5
      32246632
    *
       z
     ,

      536680682575040851917412400483050187952148396418366889397837922195724883 5
      96739896
    *
       y
     ,

      268340341287520425958706200241525093976074198209183444698918961097862441 7
      98369948
    *
       x
     ]
                                        Time: 0.02 (IN) + 0.08 (OT) = 0.10 sec
T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T
   [
     [
       close =
         {
                                   8                            7
             8724935291855297898986v  - 39262208813348840545437v
           +
                                      6                            5
             - 46237578773854460684161v  + 389268076523678581626231v
           +
                                       4                           3
             - 404694959447676777910385v  + 19876132921799844380393v
           +
                                     2
           - 186538649326279838211v  - 163172805926330371 5v + 5029665611823243
           ,

                                    3                      2
                 8282838608877853969v  - 315490964385538173v  - 25594942638053v
               +
                 33851175608089
            *
               u
           +
                                     4                      3
             - 2667985333760708531v  - 8319462093247392142v
           +
                                  2
             318993777538462620v  + 34163367871142v - 8568425233089
           ,
          t - u + v, z + 3u - v - 3, y + u + v - 1, x + t + u + v - 4}
       ,
```

```
      open =
        [
                                   3                      2
            8282838608877853969v  - 315490964385538173v  - 25594942638053v
          +
            33851175608089
          ]
      ]
    ]
                                         Time: 2.72 (EV) + 0.20 (OT) = 2.92 sec
```

```
T := WuWenTsunTriangularSet(R,E,V,P);
characteristicSerie(lp,initiallyReduced?,initiallyReduce)
```

```
   [
     {
                                 8                            7
        872493529185529789898986v  - 392622088133488405454437v
      +
                                   6                             5
        - 4623757877385446068 4161v  + 3892680765236785816 26231v
      +
                                    4                             3
        - 4046949594476767779 10385v  + 1987613292179984438 0393v
      +
                                 2
        - 186538649326279838211v  - 163172805926330 3715v + 5029665611823243
      ,
                                   3                      2
            8282838608877853969v  - 315490964385538173v  - 25594942638053v
          +
            33851175608089
        *
          u
      +
                                4                         3                         2
        - 266798533376070853 1v  - 831946209324739214 2v  + 318993777538462620v
      +
        34163367871142v - 8568425233089
      ,
     t - u + v, z + t + 2u - 3, y + u + v - 1, x - y - z - t - u - v}

                          Time: 0.05 (IN) + 0.25 (EV) + 0.18 (OT) = 0.48 sec
```

```
lts := zeroSetSplit(lp)$T;
```

```
                                         Time: 2.22 (EV) + 0.02 (OT) = 2.23 sec
```

```
[head(p) for p  in  (first lts)::LP]
```

```
   [
         971108706805456999744854033718642771331540875732221724062366548360016
         655832757800461217941975704748474894886975509381124119375038112913858
         855836657581779133016774445922573030950271514897614106699209334154720
         873
       *
          7
         v
      +
        -
           1247232126681925078208817765200977913957461613366272991004688 06175
           912929615751688917233172353946043765894664561985947393866697947444
```

```
          76563510593227032396098021597355318088641919282651433841255360 1262
          5458187428667
        *
           6
          v
    +
      -
          80871186949693910523013874037334432128123516656325114744722672 2121
          00424087485024494136640425116532856868299866190558073553426714 8559
          92697272710948587955678049572843847409488245452242170061432950 9074
          5526190237995
        *
           5
          v
    +
       120577715526926304254079964593747761812772687353249935767475638673415
       124170569075316032103535356365006609421339094703229320632366807870360
       219884793782318196842889820070783173368041494613904501729440333876383
       52345
      *
         4
        v
    +
      -
          59946399388342462548322119637507545841363114981701144670375490 4892
          85955904739442775739345516600689644205315230850000990542310384 4787
          76554936708328927649653010266958540442930908475506657415838904 4991
          081393345253
        *
           3
          v
    +
       565723676149401527371365604976462591432674273853518577929824651573040
       916765736143194449998591996072888569390020795590862230565092397767232
       653142650442658806593675207023075849219162204243512024085550248710271
       9
      *
         2
        v
    +
       493081103574833084750705179431314905451991478702286127273175746668902
       767873788969990398172364044318719541312339299234354991067031301947696
       15950202843835105230536864182047881500973586145523227827112634692039
      *
        v
    +
      -
        152131182701206885065724867525899386816095805079596381578777224521 17
        039704649628370010534446303048229874565133045954457906365982490588 43
        614558394292914699812804495365376643200912095598028252225701598704 733
*
    x
  ,

                           3                          2
    8282838608877853969v  - 315490964385538173v  - 25594942638053v
    +
    33851175608089
*
    y
  ,

                                             6
    68605415420717623158009097894889052961v
    +
                                              5
    - 52263214811292871429033103073811727 4v
```

```
      +
                                                4
        99110551050767957105114494527213215v
      +
                                              3
        576917594837711413277591342104820v
      +
                                               2
        - 2135882497527176518781863053l985v  - 1732837795839393672212021434v
      +
        1145902090049679678922231921
    *
      z
    ,

                              3                      2
        8282838608877853969v  - 315490964385538173v  - 25594942638053v
      +
        33851175608089
    *
      t
    ,

                              3                      2
        8282838608877853969v  - 315490964385538173v  - 25594942638053v
      +
        33851175608089
    *
      u
    ,
                              8
    8724935291855297898986v ]
```

```
T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T
```

```
   [
    {
                                 8                            7
        8724935291855297898986v  - 3926220881334884054 5437v
      +
                                   6                              5
        - 4623757877385446068416lv  + 3892680765236785816262 31v
      +
                                   4                              3
        - 4046949594476767779l0385v  + 198761329217998443803 93v
      +
                                 2
        - 186538649326279838211v  - 163172805926330371 5v + 5029665611823243
      ,

                                 3                      2
            8282838608877853969v  - 315490964385538173v  - 25594942638053v
          +
            33851175608089
         *
           u
      +
                              4                      3                      2
        - 2667985333760708531v  - 8319462093247392142v  + 318993777538462620v
      +
        34163367871142v - 8568425233089
      ,
     t - u + v, z + 3u - v - 3, y + u + v - 1, x + 2u - 4}
    ]
                                         Time: 1.22 (EV) + 0.18 (OT) = 1.40 sec
```

```
T := LazardTriangularSet(R,E,V,P);
```

```
lts := zeroSetSplit(lp)$T;
                              Time: 0.02 (IN) + 1.78 (EV) + 0.02 (OT) = 1.82 sec

--------------------------------------------------------------------------------
--% solotare (example from the PoSSo data base)
--------------------------------------------------------------------------------
```

Voici un autre exemple où la base lexicographique est du type *shape lemma* et qui est *un peu plus favorable* à nos quatre *méthodes de triangulation*. Elles vont en effet découvrir des facteurs (non triviaux) dans le polynôme univarié de la base lexicographique. Pour chaque décomposition la vérification `varietySplit?(lps,lp)` prend entre 2 et 3 secondes.

```
ls : List Symbol := [x,y,a,b]; R := Integer;
lp := [p1,p2,p3,p4]
      2           3    2                            2           3    2
   [3x  - 2x - a,x  - x  - a x + a - 2b - 2,3y  - 2y - a,y  - y  - a y - a + 2]

groebner lp
   [
                                 5             4               3              2
      27734835200x + 1227687759b  - 897078969b  - 17174846826b  - 2571316626b
    +
      82286604715b + 42073006043
    ,
                                 5              4              3
      13867417600y + 2835001539b  - 7132522149b  - 25287557346b
    +
                    2
      38243071254b  + 77175816815b + 9745294303
    ,
                               5              4              3             2
      1733427200a - 537286851b  + 1274021541b  + 5113254114b  - 7388075286b
    +
      - 15455988335b - 6499394527
    ,
         6         5          4        3          2
   19683b  - 21870b  - 236115b  - 6724b  + 830093b  + 860946b + 218691]
                              Time: 0.02 (IN) + 0.53 (EV) + 0.12 (OT) = 0.67 sec

T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T
   [
     [
       close =
              4       3        2
          {729b  + 108b  - 4802b  - 5732b - 1551,
                2                        3       2
           (567b  + 1710b + 1543)a - 486b  - 3249b  - 4720b - 2357,
           (3a + 1)y + 5a - 9, (3a + 1)x - 4a + 9b + 9}
       ,
                  2                      3       2
      open= [567b  + 1710b + 1543,729b  + 5157b  + 7935b + 4307]]
     ]
                              Time: 0.02 (IN) + 1.32 (EV) + 0.12 (OT) = 1.45 sec

lps : List GPS := [[lexSaturate(ts.close,ts.open)$T]$GPS for ts in ltqas]
```

```
   [
         4      3       2
    {729b  + 108b  - 4802b  - 5732b - 1551,
                3       2
     920a + 729b  - 1431b  - 3161b - 1921,
                 3       2
     1840y - 729b  + 1431b  + 3161b - 839,
                   3     2
     29440x - 9477b  - 27b  + 81113b + 42223}
    ]
                         Time: 0.07 (IN) + 1.70 (EV) + 0.13 (OT) = 1.90 sec
```

```
T := WuWenTsunTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T
```

```
   [
         4      3       2                                   2
    {729b  + 108b  - 4802b  - 5732b - 1551, (18b + 38)a - 27b  - 56b - 49,
          3       2                                3       2
     (729b  + 3213b  + 5199b + 3515)y + 1215b  + 3627b  + 1369b - 1843,
          3       2                               3       2
     (729b  + 3213b  + 5199b + 3515)x + 486b  + 3546b  + 6634b + 2774}
    ]
                                   Time: 0.35 (EV) + 0.10 (OT) = 0.45 sec
```

```
lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]
   [
         4      3       2
    {729b  + 108b  - 4802b  - 5732b - 1551,
                3       2
     920a + 729b  - 1431b  - 3161b - 1921,
                 3       2
     1840y - 729b  + 1431b  + 3161b - 839,
                   3     2
     29440x - 9477b  - 27b  + 81113b + 42223}
    ]
```

```
T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T
        2
   [{27b  + 38b + 11,a - 1,y - 1,4x + 9b + 5},
        2
    {27b  - 34b - 141,a - 5,y + 1,16x + 9b - 11}]

                         Time: 0.02 (IN) + 0.25 (EV) + 0.05 (OT) = 0.32 sec
```

```
lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]
        2
   [{27b  + 38b + 11,a - 1,y - 1,4x + 9b + 5},
        2
    {27b  - 34b - 141,a - 5,y + 1,16x + 9b - 11}]
```

```
T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T
   [
         4      3       2
    {729b  + 108b  - 4802b  - 5732b - 1551,
                3       2
     920a + 729b  - 1431b  - 3161b - 1921,
```

```
                 3        2
      1840y - 729b  + 1431b  + 3161b - 839,
                   3      2
      29440x - 9477b  - 27b  + 81113b + 42223}
     ]

-----------------------------------------------------------------------------
--% Buchberger (from the PoSSo data base)
-----------------------------------------------------------------------------
```

Définition VI.5 *Soit I un idéal de $\mathbf{P}_n$ et $\sigma = \{\sigma_1, \ldots, \sigma_k\}$ un ensemble de systèmes quasi-algébriques dans $\mathbf{P}_n$ tel que :*

$$\mathbf{V}_{\mathbf{K}}(I) = \bigcup_i \mathbf{ZR}_{\mathbf{K}}(\sigma_i)$$

On dira que σ est une décomposition paresseuse de $\mathbf{V}_{\mathbf{K}}(I)$ *si les 2 conditions suivantes sont remplies :*

1. *I n'est pas un idéal premier*
2. *l'un des systèmes quasi-algébriques σ_i vérifie :*

$$\sqrt{\mathsf{sat}(\sigma_i)} = \sqrt{I} \text{ et } \sqrt{I} \neq \bigcap_{i \neq j} \sqrt{\mathsf{sat}(\sigma_j)}$$

Dans cet exemple, les methodes de Wang, Wu et Lazard donnent des decompositions paresseuses de la variété affine associée à `lp`. Cependant, elles sont sans redondances au sens des décompositions en quasi-composantes.

La méthode de Kalkbrener, quant à elle, factorise le second polynôme de la base lexicographique et découvre la décomposition primaire minimale de I.

```
ls : List Symbol := [x,y,z,t,a,b,c,d];
R := Integer;

lp := [p1,p2,p3,p4]

   [t - b - d, x + y + z + t - a - c - d,
    (z + t)x + z y + t z + (- c - d)a - d c, t z x - d c a]
                                       Time: 0.02 (IN) + 0.08 (OT) = 0.10 sec

groebner lp

   [x + y + z - a + b - c,
               2                                 2
    (b + d)y + z  + (- a + b - c)z + (- b + c)a + b  + (- c + d)b,
     3              2
    z  + (- a - c - d)z  + ((c + d)a + d c)z - d c a, t - b - d]

                           Time: 0.02 (IN) + 0.33 (EV) + 0.08 (OT) = 0.43 sec

T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T
   [
     [
       close =
```

```
                       2
      {d,b + d,t,z  + (- a - c - d)z + (c + d)a + d c,x + y + z - a + b - c}
     ,
    open= []]
   ,
   [
     close =
                       2
      {c,b + d,t,z  + (- a - c - d)z + (c + d)a + d c,x + y + z - a + b - c}
     ,
    open= [d]]
   ,
   [
     close =
                       2
      {b + d,a,t,z  + (- a - c - d)z + (c + d)a + d c,x + y + z - a + b - c}
     ,
    open= [d,c]]
   ,
   [
     close =
                        3                  2
      {t - b - d, z  + (- a - c - d)z  + ((c + d)a + d c)z - d c a,
                        2                                   2
       (b + d)y + z  + (- a + b - c)z + (- b + c)a + b  + (- c + d)b,
       x + y + z - a + b - c}
     ,
    open= [b + d]]
   ]
                          Time: 0.02 (IN) + 1.57 (EV) + 0.15 (OT) = 1.73 sec
lps : List GPS := [[lexSaturate(ts.close,ts.open)$T]$GPS for ts in ltqas]

           2
   [{d,b,t,z  + (- a - c)z + c a,x + y + z - a - c},
               2
    {c,b + d,t,z  + (- a - d)z + d a,x + y + z - a - d},
               2
    {b + d,a,t,z  + (- c - d)z + d c,x + y + z - c - d},

                  3             2
     {t - b - d, z  + (- a - c - d)z  + ((c + d)a + d c)z - d c a,
                  2                               2
      (b + d)y + z  + (- a + b - c)z + (- b + c)a + b  + (- c + d)b,
      x + y + z - a + b - c}
     ]

lps := eliminateRedundantVarieties(lps)$GPS
   [
                  3             2
     {t - b - d, z  + (- a - c - d)z  + ((c + d)a + d c)z - d c a,
                  2                               2
      (b + d)y + z  + (- a + b - c)z + (- b + c)a + b  + (- c + d)b,
      x + y + z - a + b - c}
     ]

varietySplit?(lps,lp)$GPS

   (19)  true

T := WuWenTsunTriangularSet(R,E,V,P);
```

```
lts := zeroSetSplit(lp)$T

            2
   [{d,b,t,z  + (- a - c)z + c a,x + y + z - a - c},
                    2            2      2                          2
    {d c,b + d,t,d z  + (- d a - d )z + d a,d x + d y + d z - d a - d },

                            2          2    2       2 2
     {b + d, d c a, t, d c z  + (- d c  - d c)z + d c ,
                                   2    2
      d c x + d c y + d c z - d c  - d c}
     ,
                  3                 2
     {t - b - d, z  + (- a - c - d)z  + ((c + d)a + d c)z - d c a,
                  2                                      2
      (b + d)y + z  + (- a + b - c)z + (- b + c)a + b  + (- c + d)b,
                  2
      (b + d)x - z  + (a + c + d)z + (- c - d)a - d c}
     ]
                                       Time: 0.47 (EV) + 0.18 (OT) = 0.65 sec
```

```
lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]

            2
   [{d,b,t,z  + (- a - c)z + c a,x + y + z - a - c},
                 2
    {c,b + d,t,z  + (- a - d)z + d a,x + y + z - a - d},
                 2
    {b + d,a,t,z  + (- c - d)z + d c,x + y + z - c - d},

                  3                 2
     {t - b - d, z  + (- a - c - d)z  + ((c + d)a + d c)z - d c a,
                  2                                      2
      (b + d)y + z  + (- a + b - c)z + (- b + c)a + b  + (- c + d)b,
      x + y + z - a + b - c}
     ]
```

```
T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T
   [
     {t - b - d, z - d,
                                    2                        2
      (b + d)y + (- b + c - d)a + b  + (- c + 2d)b - d c + d ,
      x + y - a + b - c + d}
     ,
                                       2
    {t - b - d,z - c,(b + d)y - b a + b  + d b,x + y - a + b},
                                 2
    {t - b - d,z - a,(b + d)y + b  + (- c + d)b,x + y + b - c}]

                                       Time: 1.33 (EV) + 0.07 (OT) = 1.40 sec
```

```
primaryDecomp([lp]$GPS)

    GB time: 0.033333000000000002
   [
     [x + y - a + b - c + d,
                                   2                       2
      (b + d)y + (- b + c - d)a + b  + (- c + 2d)b - d c + d , z - d,
      t - b - d]
```

```
     ,
                                           2
    [x + y - a + b,(b + d)y - b a + b  + d b,z - c,t - b - d],
                                  2
    [x + y + b - c,(b + d)y + b  + (- c + d)b,z - a,t - b - d]]
            Time: 0.02 (IN) + 19.33 (EV) + 0.12 (OT) + 4.23 (GC) = 23.70 sec
```

```
T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T
   [
                3           2
     {t - b - d, z  + (- a - c - d)z  + ((c + d)a + d c)z - d c a,
                2                             2
      (b + d)y + z  + (- a + b - c)z + (- b + c)a + b  + (- c + d)b,
                2
      (b + d)x - z  + (a + c + d)z + (- c - d)a - d c}
     ,
              2
    {b + d,a,t,z  + (- c - d)z + d c,x + y + z - c - d},
          2
    {d,b,t,z  + (- a - c)z + c a,x + y + z - a - c},
              2
    {c,b + d,t,z  + (- a - d)z + d a,x + y + z - a - d}]
                          Time: 0.02 (IN) + 0.90 (EV) + 0.15 (OT) = 1.07 sec
```

```
lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]
   [
                3           2
     {t - b - d, z  + (- a - c - d)z  + ((c + d)a + d c)z - d c a,
                2                             2
      (b + d)y + z  + (- a + b - c)z + (- b + c)a + b  + (- c + d)b,
      x + y + z - a + b - c}
     ,
              2
    {b + d,a,t,z  + (- c - d)z + d c,x + y + z - c - d},
          2
    {d,b,t,z  + (- a - c)z + c a,x + y + z - a - c},
              2
    {c,b + d,t,z  + (- a - d)z + d a,x + y + z - a - d}]
                          Time: 0.02 (IN) + 4.12 (EV) + 0.15 (OT) = 4.28 sec
```

```
---------------------------------------------------------------------------
--% Cercle d'Apollonius
---------------------------------------------------------------------------
```

Voici un exemple classique et facile. Nous nous contentons de donner le résultat produit par chaque méthode.

```
ls : List Symbol := [c,d,a,b]; R := Integer;

lp := [f1,f2,f3]
                                      2           2
   (10)  [b c + a d - 4b a,a c - b d,c  - 2a c + d  - 2b d]

groebner lp
           2    2                                  2    2         2
   (12)  [c  + d  - 4b d,a c - b d,b c + a d - 4b a,(a  + b )d - 4b a ]

T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T
```

```
                 2    2
   [[close= {b,a,c  + d },open= []], [close= {b,d,c},open= [a]],
              2    2         2                            2    2
    [close= {(a  + b )d - 4b a ,b c + a d - 4b a},open= [b,a  + b ]]]
                           Time: 0.02 (IN) + 0.30 (EV) + 0.12 (OT) = 0.43 sec


T := WuWenTsunTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

               2    2                2    2          2     2    3       3
   (22)  [{b,a,c  + d },{b,a d,a c},{(a  + b )d - 4b a ,(b a  + b )c - 4b a}]
                                       Time: 0.27 (EV) + 0.07 (OT) = 0.33 sec


T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T

            2    2         2                        2    2
   (28)  [{(a  + b )d - 4b a ,b c + a d - 4b a},{b,a,c  + d },{b,d,c}]
                           Time: 0.02 (IN) + 0.17 (EV) + 0.07 (OT) = 0.25 sec

T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

            2    2         2   2    2       2         2    2
   (34)  [{(a  + b )d - 4b a ,(a  + b )c - 4b a},{b,a,c  + d }]
                                       Time: 0.32 (EV) + 0.05 (OT) = 0.37 sec

primaryDecomp([lp]$GPS)

    GB time: 0.0
           2    2                                   2    2         2
   (38)  [[c  + d  - 4b d,- a c + b d,b c + a d - 4b a,(a  + b )d - 4b a ]]
                           Time: 0.03 (IN) + 2.27 (EV) + 0.10 (OT) = 2.40 sec

-----------------------------------------------------------------------------
--% exemple de [Traverso  & Donati - 1989]
-----------------------------------------------------------------------------
```

Voici un exemple tout aussi classique où les méthodes de triangulation donnent un résultat plus rapidement et plus exploitable que la base lexicographique. De plus, cet exemple illustre clairement la différence entre résoudre au sens de Lazard, Wang et Wu d'une part et résoudre au sens de Kalkbrener. Les 3 premieres méthodes donnent une décomposition des zéros de `lp` en zéros réguliers de systèmes quasi-algébriques. Sur cet exemple, l'un des saturés de ces systèmes (qui est de dimension 1) est contenu les saturés des 2 ou 3 autres ssytèmes (qui eux sont de dimension zéro). Cependant en tant que décomposition en zéros réguliers de s.q.a., chacune de ces 3 décompositions est sans redondance. La méthode de Kalkbrener, quant à elle, donne une unique quasi-composante dont le saturé est l'idéal engendré par le système `lp` de départ.

```
ls : List Symbol := [x,y,z,t]; R := Integer;
lp :LP := [p1,p2,p3]
```

```
           31    6           8       10
   (12)  [x   - x  - x - y,x  - z,x   - t]
```

groebner lp

```
   [
      2     7          2            2
     x  - 2y  + (- 41t  - 13t - 1)y
    +
          17      14      12      11       9      8      7       6      5
        2t   - 12t   + 42t   + 30t   - 168t  - 40t  + 70t  + 252t  + 30t
       +
              4       3     2
        - 140t  - 168t  + 2t  - 12t + 16
     *
        2
       z y
    +
             16      13      11       10       8       7      6       5
        - 12t   + 72t   - 28t   - 180t   + 112t  + 240t  + 28t  - 127t
       +
              4      3      2
        - 167t  - 55t  + 30t  + 58t - 15
     *
        4
       z
    ,
           2 2      7       2           2
     (y + t z )x + y  + (20t  + 6t + 1)y
    +
           17     14      12      11      9      8      7       6      5
        - t   + 6t   - 21t   - 15t   + 84t  + 20t  - 35t  - 126t  - 15t
       +
            4      3    2
        70t  + 84t  - t  + 5t - 9
     *
        2
       z y
    +
          16      13      11      10      8       7      6      5      4
        6t   - 36t   + 14t   + 90t   - 56t  - 120t  - 14t  + 64t  + 84t
       +
            3      2
        27t  - 16t  - 30t + 7
     *
        4
       z
    ,

       3          6
     (t  - 1)x - y
    +
          13      10      8      7      5      4     3      2
     (- 6t   + 24t   - 20t  - 36t  + 40t  + 24t  - 6t  - 20t  - 6t - 1)y
    +
          17     14     12      11      9      8     7      6      5      4
        t   - 6t   + 9t   + 15t   - 36t  - 20t  - 5t  + 54t  + 15t  + 10t
       +
             3      2
        - 36t  - 11t  - 5t + 9
     *
        2
       z
    ,
      8      3      2 2
     y  + 8t y  - 16z y
    +
```

```
         16     13     11      10      8      7     6      5      4
       8t   - 48t   + 56t   + 120t   - 224t  - 160t  + 56t  + 336t  + 112t
     +
             3      2
       - 112t  - 224t  - 24t + 56
   *
      4
     z y
  +
          24     21     19     18      16     15     14      13
       - t   + 8t   - 20t   - 28t   + 120t   + 56t   - 14t   - 300t
     +
             12     11      10     9      8      7     6      5     4
       - 70t   + 56t   + 400t   + 84t  - 84t  - 268t  - 84t  + 56t  + 63t
     +
           3     2
       36t  - 46t  + 12t - 1
   *
     z
  ,
       5     2       11     8      6      5      3     2           3
    2t y  + z y  + (- 2t   + 8t  - 20t  - 12t  + 40t  + 8t  - 10t - 20)z y
  +
      14      11      8    7      5     4     2
    8t   - 32t   + 48t  - t  - 32t  - 6t  + 9t  - t
  ,
    3      7     4     2           6     3          2
  z y  + (- t  + 2t  - 3t  - t)y + (2t  - 4t  - 2t + 2)z ,

      2 3    2 2       5     2     4
    2t y  + z y  + (- 2t  + 4t  - 6)z y
  +
       8    7     5     4     3     2
    (4t  - t  - 8t  + 2t  - 4t  + 5t  - t)z
  ,
   3 2     3        7     4     2     2        2     3     8     5    3    2
  z y  + 2t y + (- t  + 2t  + t  - t)z , t z y  + 2z y - t  + 2t  + t  - t ,
   3 2    2 2       6     3         4   5    4
  t y  + 2t z y + (- t  + 2t  + t - 1)z , z  - t ]
            Time: 0.02 (IN) + 3.98 (EV) + 0.77 (OT) + 5.55 (GC) = 10.32 sec
```

```
T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T
```

```
   [[close= {t,z,y,x},open= []],
              3     5     3         2
    [close= {t  - 1,z  - t,z y + 1,z x  - t},open= [z]],

               5    4       2     3     8     5    3    2   3       3     3     3
     [close= {z  - t ,t z y  + 2z y - t  + 2t  + t  - t ,(t  - 1)z x - z y - t }
      ,
               3
      open= [t,t  - 1,z]]]
                         Time: 0.02 (IN) + 0.73 (EV) + 0.13 (OT) = 0.88 sec
```

```
T := WuWenTsunTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T
```

```
                 3      5      3         2
   [{t,z,y,x}, {t  - 1,z  - t,z y + 1,z x  - t},
      5    4  2 2 2       4        9     6    4    3     3      3     3     3
    {z  - t ,t z y  + 2t z y + (- t  + 2t  + t  - t )z,(t  - 1)z x - z y - t }]
                                       Time: 0.90 (EV) + 0.08 (OT) = 0.98 sec
```

```
T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T
```

```
            5    4      2     3     8     5    3    2   4                2
   (29)  [{z  - t ,t z y  + 2z y - t  + 2t  + t  - t ,(t  - t)x - t y - z }]
                           Time: 0.02 (IN) + 0.65 (EV) + 0.07 (OT) = 0.73 sec
```

```
T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T
```

```
      5    4  3 2     2 2        6     3          4   4                2
   [{z  - t ,t y  + 2t z y + (- t  + 2t  + t - 1)z ,(t  - t)x - t y - z },
      2          5                   2  2    4           5          2  2    4
    {t  + t + 1,z  - t,y + (- t - 1)z ,x  - z }, {t - 1,z  - 1,y + z ,x  - z },
    {t,z,y,x}]
                                       Time: 0.77 (EV) + 0.10 (OT) = 0.87 sec
```

```
----------------------------------------------------------------------------
--% Exemple de Alonso (from the PoSSo data base)
----------------------------------------------------------------------------
```

Cet exemple est analogue au précédent, à ceci près que la base lexicographique est plus ramassée mais tout aussi inexploitable numériquement. Il s'agit encore d'un idéal premier (de dimension 3). La méthode de Kalkbrener le décrit encore à l'aide d'une unique chaîne régulière, les autres méthodes en extrait quelques points qui sont dans la fermeture d'une quasi-composante redonnant la variété de départ. On se contente, pour cet exemple, de donner la vérification de la sortie de notre implantation de la méthode de Wang. Les autres vérifications ne sont pas reproduites ici.

```
ls : List Symbol := [x,y,z,v,u,r,t];
R := Integer;

lp := [f1,f2,f3,f4]

                              4             2
   (11)  [(u - r - 2)x - u - 2t  + 1,r y - t u - 1,t u z - 1,v - t r]
                                                   Time: 0.10 (OT) = 0.10 sec
groebner lp

                         2         5
   [((2t z - 1)y + t z + t )x + ((2t  - t)z + 1)y,
                           5                                  4
    ((t r + 2t)z - 1)x + (2t  - t)z + 1, (u - r - 2)x - u - 2t  + 1,
           2
    r y - t u - 1, t u z - 1, v - t r]
                          Time: 0.02 (IN) + 0.35 (EV) + 0.05 (OT) = 0.42 sec

T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T
   [
     [
       close =
                4            4             5         5
         {r + 2t  + 1, u + 2t  - 1, v + 2t  + t, (2t  - t)z + 1,
             4           6    2
          (2t  + 1)y - 2t  + t  + 1}
        ,
                 4       4
      open= [t,2t  - 1,2t  + 1]]
     ,

                 2                                      4
     [close= {r,t u + 1,v,t u z - 1,(u - r - 2)x - u - 2t  + 1},
                 2
      open= [t,2t  + 1]]
     ,

                                      2                           4
     [close= {v - t r,t u z - 1,r y - t u - 1,(u - r - 2)x - u - 2t  + 1},
      open= [t,r,u,u - r - 2]]
     ]
                          Time: 0.03 (IN) + 1.90 (EV) + 0.17 (OT) = 2.10 sec
```

```
lps : List GPS := [[lexSaturate(ts.close,ts.open)$T]$GPS for ts in ltqas]
   [
           4           4           5        5
     {r + 2t  + 1, u + 2t  - 1, v + 2t  + t, (2t  - t)z + 1,
        4          6    2          3        4         5
      (2t  + 1)y - 2t  + t  + 1, (z + t )y + (- t  + 1)z - t  + t}
     ,
      2                2          6    2                    4
    {r,t u + 1,v,z + t,(2t  + 1)x + 2t  - t  - 1,(u - 2)x - u - 2t  + 1},

                                2                            4
     {v - t r, t u z - 1, r y - t u - 1, (u - r - 2)x - u - 2t  + 1,
                               5
      ((t r + 2t)z - 1)x + (2t  - t)z + 1,
                             2        5
      ((2t z - 1)y + t z + t )x + ((2t  - t)z + 1)y}
     ]
            Time: 0.10 (IN) + 9.87 (EV) + 0.22 (OT) + 3.83 (GC) = 14.02 sec

lps := eliminateRedundantVarieties(lps)$GPS
   [
                                2                            4
     {v - t r, t u z - 1, r y - t u - 1, (u - r - 2)x - u - 2t  + 1,
                               5
      ((t r + 2t)z - 1)x + (2t  - t)z + 1,
                             2        5
      ((2t z - 1)y + t z + t )x + ((2t  - t)z + 1)y}
     ]
             Time: 0.02 (IN) + 4.52 (EV) + 0.08 (OT) + 4.17 (GC) = 8.78 sec
```

```
T := WuWenTsunTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

        2                2    3           7    3
   [{r,t u + 1,v,t z + t ,(2t  + t)x + 2t  - t  - t},

            4           4           5         5
     {r + 2t  + 1, u + 2t  - 1, v + 2t  + t, (2t  - t)z + 1,
         4          6    2
       (2t  + 1)y - 2t  + t  + 1}
      ,
                              2                          4
    {v - t r,t u z - 1,r y - t u - 1,(u - r - 2)x - u - 2t  + 1}]
                          Time: 0.02 (IN) + 0.35 (EV) + 0.12 (OT) = 0.48 sec

T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T
                                     2                          4
   (28)  [{v - t r,t u z - 1,r y - t u - 1,(u - r - 2)x - u - 2t  + 1}]
                                    Time: 0.25 (EV) + 0.05 (OT) = 0.30 sec

T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

                              2                          4
   [{v - t r,t u z - 1,r y - t u - 1,(u - r - 2)x - u - 2t  + 1},

            4           4           5         5
     {r + 2t  + 1, u + 2t  - 1, v + 2t  + t, (2t  - t)z + 1,
```

```
           4            6    2
      (2t   + 1)y - 2t   + t   + 1}
     ,
      2                 2           6    2
    {r,t u + 1,v,z + t,(2t   + 1)x + 2t   - t   - 1}]
                                          Time: 0.45 (EV) + 0.10 (OT) = 0.55 sec

-----------------------------------------------------------------------------
--% ROBOT plano facil (Santander)
-----------------------------------------------------------------------------
```

Voici un exemple un peu différent des deux précédents, Il s'agit d'un idéal premier de dimension 2. La base lexicographique, dont le calcul est aisée, n'est toujours pas exploitable numériquement. Les sorties de nos implantations des méthodes de Wang et Wu sont un peu longues ; celle de Wu n'est pas reproduite ici. Elles ont pu cependant passer dans notre moulinette de vérification. Sur cet exemple, la sortie la plus rapide et la plus exploitable est clairement celle de notre implantation de la méthode de Lazard.

```
ls : List Symbol := [c2,s2,c1,s1,b,a]; R := Integer;
lp := [f1,f2,f3,f4]
                                                             2     2       2     2
   [c1 c2 - s1 s2 + c1 - a,s1 c2 + c1 s2 + s1 - b,c1  + s1  - 1,c2  + s2  - 1]

groebner lp
           2    2                         2     2
   [2c2 - b  - a  + 2, s2 - b c1 + a s1, c1  + s1  - 1,
               2           2               2    3
    (4b s1 - 2b )c1 - 4a s1  + 2a b s1 - a b  - a  + 4a,
                     2    2
    2a c1 + 2b s1 - b  - a ,
       2     2  2        3     2        4     2 2    4     2
    (4b  + 4a )s1  + (- 4b  - 4a b)s1 + b  + 2a b  + a  - 4a ]
                              Time: 0.03 (IN) + 0.37 (EV) + 0.10 (OT) = 0.50 sec

T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T

                      2
   [[close= {a,b,s1,c1  - 1,c1 s2 - b,c1 c2 + c1 - a},open= [c1]],

               4     2 2    4     2             2    2
     [close= {b  + 2a b  + a  - 4a ,s1,2a c1 - b  - a ,c1 s2 - b,c1 c2 + c1 - a}
       ,
                2    2
      open= [a,b  + a ]]
     ,

                    2     2
     [close= {a,b,c1  + s1  - 1,s2 - b c1 + a s1,s1 c2 + c1 s2 + s1 - b},
      open= [s1]]
     ,

                          2     2
     [close= {a,2s1 - b,c1  + s1  - 1,s2 - b c1 + a s1,s1 c2 + c1 s2 + s1 - b},
      open= [b]]
```

```
     ,

     [
       close =
             2     2   2         3     2         4     2 2    4     2
         {(4b  + 4a )s1  + (- 4b  - 4a b)s1 + b  + 2a b  + a  - 4a ,
                                2    2
          2a c1 + 2b s1 - b  - a , s2 - b c1 + a s1, s1 c2 + c1 s2 + s1 - b}
       ,
                2    2
      open= [a,b  + a ,s1]]
     ]
              Time: 0.03 (IN) + 1.97 (EV) + 0.27 (OT) + 1.97 (GC) = 4.23 sec

T := WuWenTsunTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T;

                         Time: 4.50 (EV) + 0.52 (OT) + 6.00 (GC) = 11.02 sec

T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T
   [
         2     2   2         3     2         4     2 2    4     2
     {(4b  + 4a )s1  + (- 4b  - 4a b)s1 + b  + 2a b  + a  - 4a ,
                        2    2
      2a c1 + 2b s1 - b  - a , s2 - b c1 + a s1,
                   2
      s1 c2 + b c1  - a s1 c1 + s1 - b}
     ]
                                      Time: 1.52 (EV) + 0.10 (OT) = 1.62 sec

T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T
   [
         2     2   2         3     2         4     2 2    4     2
     {(4b  + 4a )s1  + (- 4b  - 4a b)s1 + b  + 2a b  + a  - 4a ,
                        2    2             2     2      3    2
      2a c1 + 2b s1 - b  - a , 2a s2 + (2b  + 2a )s1 - b  - a b,
             2    2
      2c2 - b  - a  + 2}
     ,
                   2    2                      2
    {a,2s1 - b,4c1  + b  - 4,s2 - b c1,2c2 - b  + 2},
           2     2
    {a,b,c1  + s1  - 1,s2,c2 + 1}]
                          Time: 0.02 (IN) + 0.92 (EV) + 0.13 (OT) = 1.07 sec

primaryDecomp([lp]$GPS)
   [
            2    2                         2     2
     [2c2 - b  - a  + 2, s2 - b c1 + a s1, c1  + s1  - 1,
                    2           2                 2    3
      (- 4b s1 + 2b )c1 + 4a s1  - 2a b s1 + a b  + a  - 4a,
```

```
                       2    2
      2a c1 + 2b s1 - b  - a ,
         2     2   2       3     2        4     2 2    4     2
      (4b  + 4a )s1  + (- 4b  - 4a b)s1 + b  + 2a b  + a  - 4a ]
     ]
              Time: 0.03 (IN) + 7.75 (EV) + 0.13 (OT) + 2.17 (GC) = 10.08 sec

---------------------------------------------------------------------------
--% Exemple de Trinks 2 (from the PoSSo data base)
---------------------------------------------------------------------------
```

Voici un exemple qui n'est intéressant que pour les temps d'exécution ...

```
ls : List Symbol :=  [x,y,t,z,u,v] ; R := Integer;
lp := [f1,f2,f3,f4,f5,f6,f7]

   [45y + 35u - 165v - 36, 35y + 40t + 25z - 27u,
                                   2                                  3
    15x + 25u y + 30t - 18z - 165v , - 9x + 15z y + 20u t, y x + 2z t - 11v ,
                    2        3
    99x - 11v u + 3v , 10000v  + 6600v + 2673]

groebner lp

   (16)  [1]
              Time: 0.02 (IN) + 0.67 (EV) + 0.03 (OT) + 2.60 (GC) = 3.32 sec

T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T

   (19)  []
                          Time: 0.02 (IN) + 0.55 (EV) + 0.03 (OT) = 0.60 sec

T := WuWenTsunTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

   (22)  []
                          Time: 0.02 (IN) + 0.45 (EV) + 0.05 (OT) = 0.52 sec

T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T

   (25)  []
                                      Time: 0.82 (EV) + 0.05 (OT) = 0.87 sec

T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

   (28)  []
                          Time: 0.02 (IN) + 1.40 (EV) + 0.03 (OT) = 1.45 sec
```

```
----------------------------------------------------------------------------
--% Teorema de Euler (Teresa Gomez Diaz, tesis)
----------------------------------------------------------------------------

ls : List Symbol := reverse [a1,a2,a3,a4,a5,a6,a7,a8,a9,a10];
R := Integer;

lp := [f1,f2,f3,f4,f5,f6,f7,f8,f9]

   [2a3 - a1, 2a4 - a1, 2a5 - a2, 2a6 - a2,
                                2     2     2                        2     2
    2a5 a8 + (2a4 - 2a3)a7 - a5  - a4  + a3 , 2a6 a8 - 2a3 a7 - a6  + a3 ,
    a1 a10 + a2 a9 - a1 a2, a2 a10 - a1 a9,
         2             2                       2
    - a10  + 2a8 a10 - a9  + 2a7 a9 - 2a3 a7 + a3 ]
                                                   Time: 0.13 (OT) = 0.13 sec

groebner lp
       2              2
   [a10  - 2a8 a10 + a9  - 2a7 a9, a2 a10 - a1 a9, a1 a10 + a2 a9 - a1 a2,
                                                2     2           2
    (4a1 a8 - a1 a2)a9, (4a2 a7 - a1 a2)a9, (a2  + a1 )a9 - a1 a2 ,
               2            2
    4a2 a8 - a2 , 4a1 a7 - a1 , 2a6 - a2, 2a5 - a2, 2a4 - a1, 2a3 - a1]
                           Time: 0.02 (IN) + 0.48 (EV) + 0.10 (OT) = 0.60 sec

T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T

                                  2              2
   [[close= {a1,a2,a3,a4,a5,a6,a10  - 2a8 a10 + a9  - 2a7 a9},open= []],
    [close= {a1,a3,a4,2a5 - a2,2a6 - a2,4a8 - a2,a9,a10},open= [a2]],

     [
       close =
                                                      2     2           2
         {a2, 2a3 - a1, 2a4 - a1, a5, a6, 4a7 - a1, (a2  + a1 )a9 - a1 a2 ,
          a1 a10 + a2 a9 - a1 a2}
       ,
      open= [a1]]
     ,

     [
       close =
         {2a3 - a1, 2a4 - a1, 2a5 - a2, 2a6 - a2, 4a7 - a1, 4a8 - a2,
             2     2           2
          (a2  + a1 )a9 - a1 a2 , a1 a10 + a2 a9 - a1 a2}
       ,
                     2     2
      open= [a1,a2,a2  + a1 ]]
     ]
                           Time: 0.02 (IN) + 3.28 (EV) + 0.17 (OT) = 3.47 sec
lps : List GPS := [[lexSaturate(ts.close,ts.open)$T]$GPS for ts in ltqas]
```

```
                        2            2
   [{a1,a2,a3,a4,a5,a6,a10  - 2a8 a10 + a9  - 2a7 a9},
    {a1,a3,a4,2a5 - a2,2a6 - a2,4a8 - a2,a9,a10},
    {a2,2a3 - a1,2a4 - a1,a5,a6,4a7 - a1,a9,a10},

     {2a3 - a1, 2a4 - a1, 2a5 - a2, 2a6 - a2, 4a7 - a1, 4a8 - a2,
         2    2          2
      (a2  + a1 )a9 - a1 a2 , a1 a10 + a2 a9 - a1 a2, a2 a10 - a1 a9,
         2    2
      a10  + a9  - a1 a9}
     ]
                         Time: 0.10 (IN) + 5.93 (EV) + 0.27 (OT) = 6.30 sec

lps := eliminateRedundantVarieties(lps)$GPS

                        2            2
   [{a1,a2,a3,a4,a5,a6,a10  - 2a8 a10 + a9  - 2a7 a9},

     {2a3 - a1, 2a4 - a1, 2a5 - a2, 2a6 - a2, 4a7 - a1, 4a8 - a2,
         2    2          2
      (a2  + a1 )a9 - a1 a2 , a1 a10 + a2 a9 - a1 a2, a2 a10 - a1 a9,
         2    2
      a10  + a9  - a1 a9}
     ,
    {a1,a3,a4,2a5 - a2,2a6 - a2,4a8 - a2,a9,a10},
    {a2,2a3 - a1,2a4 - a1,a5,a6,4a7 - a1,a9,a10}]
                       Time: 26.45 (EV) + 0.37 (OT) + 7.60 (GC) = 34.42 sec

varietySplit?(lps,lp)$GPS

   (24)  true
             Time: 0.02 (IN) + 9.52 (EV) + 0.08 (OT) + 5.42 (GC) = 15.03 sec
```

```
T := WuWenTsunTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T


                        2            2
   [{a1,a2,a3,a4,a5,a6,a10  - 2a8 a10 + a9  - 2a7 a9},
                                           2
    {a1,a3,a4,2a5 - a2,2a6 - a2,4a2 a8 - a2 ,a2 a9,a2 a10},
                                           2
    {a2,2a3 - a1,2a4 - a1,a5,a6,4a1 a7 - a1 ,a1 a9,a1 a10},


                                                          2                  2
     {2a3 - a1, 2a4 - a1, 2a5 - a2, 2a6 - a2, 4a1 a7 - a1 , 4a1 a2 a8 - a1 a2 ,
         2    2          2        2    3           3
      (a2  + a1 )a9 - a1 a2 , (a1 a2  + a1 )a10 - a1 a2}
     ]
                                       Time: 1.15 (EV) + 0.13 (OT) = 1.28 sec
```

```
lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]
                        2            2
   [{a1,a2,a3,a4,a5,a6,a10  - 2a8 a10 + a9  - 2a7 a9},
    {a1,a3,a4,2a5 - a2,2a6 - a2,4a8 - a2,a9,a10},
    {a2,2a3 - a1,2a4 - a1,a5,a6,4a7 - a1,a9,a10},

     {2a3 - a1, 2a4 - a1, 2a5 - a2, 2a6 - a2, 4a7 - a1, 4a8 - a2,
         2    2          2
      (a2  + a1 )a9 - a1 a2 , a1 a10 + a2 a9 - a1 a2, a2 a10 - a1 a9,
         2    2
      a10  + a9  - a1 a9}]
                         Time: 0.02 (IN) + 4.68 (EV) + 0.12 (OT) = 4.82 sec
```

```
T := TableRegularChain(R,E,V,P);
```

```
lts := zeroSetSplit(lp)$T

                             2                 2
   [{a1,a2,a3,a4,a5,a6,a10  - 2a8 a10 + a9  - 2a7 a9},
                                2     2
    {a1,a2,a3,a4,a5,a6,a8,a10  + a9  - 2a7 a9},
    {a1,a3,a4,2a5 - a2,2a6 - a2,4a8 - a2,a9,a10},
                                                                 2     2
    {a1,a2,a3,a4,a5,a6,a8,a9,a10}, {a1,a2,a3,a4,a5,a6,a7,a8,a10  + a9 },

     {2a3 - a1, 2a4 - a1, 2a5 - a2, 2a6 - a2, 4a7 - a1, 4a8 - a2,
         2     2           2
      (a2  + a1 )a9 - a1 a2 , a1 a10 + a2 a9 - a1 a2}
     ,
    {a1,a3,a4,2a5 - a2,2a6 - a2,a7,4a8 - a2,a9,a10},
                                2                2
    {a1,a2,a3,a4,a5,a6,a7,a10  - 2a8 a10 + a9 },
    {a2,2a3 - a1,2a4 - a1,a5,a6,4a7 - a1,a9,a10},
    {a1,a2,a3,a4,a5,a6,a7,a9,a10}, {a1,a2,a3,a4,a5,a6,a7,a9,a10 - 2a8}]
                                           Time: 2.33 (EV) + 0.18 (OT) = 2.52 sec
lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]

                             2                 2
   [{a1,a2,a3,a4,a5,a6,a10  - 2a8 a10 + a9  - 2a7 a9},
                                2     2
    {a1,a2,a3,a4,a5,a6,a8,a10  + a9  - 2a7 a9},
    {a1,a3,a4,2a5 - a2,2a6 - a2,4a8 - a2,a9,a10},
                                                                 2     2
    {a1,a2,a3,a4,a5,a6,a8,a9,a10}, {a1,a2,a3,a4,a5,a6,a7,a8,a10  + a9 },

     {2a3 - a1, 2a4 - a1, 2a5 - a2, 2a6 - a2, 4a7 - a1, 4a8 - a2,
         2     2           2
      (a2  + a1 )a9 - a1 a2 , a1 a10 + a2 a9 - a1 a2, a2 a10 - a1 a9,
         2     2
      a10  + a9  - a1 a9}
     ,
    {a1,a3,a4,2a5 - a2,2a6 - a2,a7,4a8 - a2,a9,a10},
                                2                2
    {a1,a2,a3,a4,a5,a6,a7,a10  - 2a8 a10 + a9 },
    {a2,2a3 - a1,2a4 - a1,a5,a6,4a7 - a1,a9,a10},
    {a1,a2,a3,a4,a5,a6,a7,a9,a10}, {a1,a2,a3,a4,a5,a6,a7,a9,a10 - 2a8}]
                          Time: 0.03 (IN) + 1.95 (EV) + 0.15 (OT) = 2.13 sec

lps := eliminateRedundantVarieties(lps)$GPS

                             2                 2
   [{a1,a2,a3,a4,a5,a6,a10  - 2a8 a10 + a9  - 2a7 a9},
    {a2,2a3 - a1,2a4 - a1,a5,a6,4a7 - a1,a9,a10},
    {a1,a3,a4,2a5 - a2,2a6 - a2,4a8 - a2,a9,a10},

     {2a3 - a1, 2a4 - a1, 2a5 - a2, 2a6 - a2, 4a7 - a1, 4a8 - a2,
         2     2           2
      (a2  + a1 )a9 - a1 a2 , a1 a10 + a2 a9 - a1 a2, a2 a10 - a1 a9,
         2     2
      a10  + a9  - a1 a9}
     ]
                      Time: 37.50 (EV) + 0.35 (OT) + 10.42 (GC) = 48.27 sec
T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T
```

```
                         2               2
   [{a1,a2,a3,a4,a5,a6,a10  - 2a8 a10 + a9  - 2a7 a9},

     {2a3 - a1, 2a4 - a1, 2a5 - a2, 2a6 - a2, 4a7 - a1, 4a8 - a2,
         2     2           2     2     2         2
       (a2  + a1 )a9 - a1 a2 , (a2  + a1 )a10 - a1 a2}
     ,
    {a2,2a3 - a1,2a4 - a1,a5,a6,4a7 - a1,a9,a10},
    {a1,a3,a4,2a5 - a2,2a6 - a2,4a8 - a2,a9,a10}]
                          Time: 0.02 (IN) + 1.63 (EV) + 0.10 (OT) = 1.75 sec
```

```
lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]
                            2             2
   [{a1,a2,a3,a4,a5,a6,a10  - 2a8 a10 + a9  - 2a7 a9},

     {2a3 - a1, 2a4 - a1, 2a5 - a2, 2a6 - a2, 4a7 - a1, 4a8 - a2,
         2     2           2
       (a2  + a1 )a9 - a1 a2 , a1 a10 + a2 a9 - a1 a2, a2 a10 - a1 a9,
         2     2
       a10  + a9  - a1 a9}
     ,
    {a2,2a3 - a1,2a4 - a1,a5,a6,4a7 - a1,a9,a10},
    {a1,a3,a4,2a5 - a2,2a6 - a2,4a8 - a2,a9,a10}]
                            Time: 0.03 (IN) + 1.88 (EV) + 0.12 (OT) = 2.03 sec
```

```
------------------------------------------------------------------------------
--% Exemple de [Wang-1991c] et Pb3 {Wang "An implementation..."]
--% To illustrate the "rewriteSetWithRemainder" optimization in Wu's method !!!
------------------------------------------------------------------------------

ls : List Symbol := [y,a3,a2,a1,x6,x5,x4,x3,x2,x1];
R := Integer;
p1 :P := ((x4 + x5) * (x5 + x6) * ( x6 + x4) * (x1 * x2 * x3) ** 2)::P;
p2 :P := p1 + (a1 * (x2 ** 2 * x6 - 1) + a2 * ( x1 ** 2 * x4 - 1 ))::P;
p2 := p2 + (a3 * (x3 ** 2 * x5 - 1 ))::P;
f1 :P := (y ** 2)::P - p1;
f2 :P := differentiate (p2,'x1);
f3 :P := differentiate (p2,'x2);
f4 :P := differentiate (p2,'x3);
f5 :P := differentiate (p2,'x4);
f6 :P := differentiate (p2,'x5);
f7 :P := differentiate (p2,'x6);
f8 :P := differentiate (p2,'a1);
f9 :P := differentiate (p2,'a2);
f10 :P := differentiate (p2,'a3);

lp : List P := [f1,f2,f3,f4,f5,f6,f7,f8,f8,f9,f10];

groebner lp
     2        4      2  2     4   2         2      2
   [y  + (- x3  - 2x2 x3  - x2 )x4  + (- 2x3  - 2x2 )x4,
            4      2  2     4   2       2      2
    a3 + (x3  + 2x2 x3  + x2 )x4  + (2x3  + 2x2 )x4,
            4      2  2     4   2       2      2
    a2 + (x3  + 2x2 x3  + x2 )x4  + (2x3  + 2x2 )x4,
            4      2  2     4   2       2      2
    a1 + (x3  + 2x2 x3  + x2 )x4  + (2x3  + 2x2 )x4,
```

```
            4     4   3     2  2            4     4   3     2  2
    x6 + (x3  - x2 )x4  - x2 x4 , x5 + (- x3  + x2 )x4  - x3 x4 ,
      6       4     2  2     4    2          6     4  2     2  4     6
    x2 x4 + x3  - x1 x2  - x1 , x1 x4 - 1, x3  - x1 x3  + x1 x2  - x1 ,
      2  4     6     2  4     6    2  4     6     4  2     6
    x2 x3  - x2  - x1 x2  + x1 , x1 x3  + x2  - x1 x2  - x1 ,
       4     4   2     6     2  4     4  2     6    8     2  6     6  2     8
    (x2  - x1 )x3  + x2  + x1 x2  - x1 x2  - x1 , x2  + x1 x2  - x1 x2  - x1 ]
                           Time: 0.03 (IN) + 1.87 (EV) + 0.23 (OT) = 2.13 sec

T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T
   [
     [
       close =
            4     4    4     4    2          2          2
         {x2  - x1 , x3  - x1 , x1 x4 - 1, x3 x5 - 1, x2 x6 - 1,
                   2  2        2  2          2  2  2      2  2          2  2  2
          a1 + (2x1 x3 x5 + 2x1 x3 x4)x6 + x1 x3 x5  + 2x1 x3 x4 x5 + x1 x3 x4 ,
                 2  2  2       2  2        2  2          2  2  2      2  2
          a2 + x2 x3 x6  + (2x2 x3 x5 + 2x2 x3 x4)x6 + x2 x3 x5  + 2x2 x3 x4 x5,
                 2  2  2       2  2        2  2           2  2          2  2  2
          a3 + x1 x2 x6  + (2x1 x2 x5 + 2x1 x2 x4)x6 + 2x1 x2 x4 x5 + x1 x2 x4 ,

              2        2  2  2       2  2  2     2
             y  + (- x1 x2 x3 x5 - x1 x2 x3 x4)x6
           +
                  2  2  2  2      2  2  2          2  2  2  2
             (- x1 x2 x3 x5  - 2x1 x2 x3 x4 x5 - x1 x2 x3 x4 )x6
           +
                 2  2  2     2     2  2  2  2
             - x1 x2 x3 x4 x5  - x1 x2 x3 x4 x5
           }
       ,
      open= [x1,x2,x3]]
     ,

     [
       close =
            4     2  2     4     2     2   2     2  2    2          2
         {x2  + x1 x2  + x1 , (x2  + x1 )x3  + x1 x2 , x1 x4 - 1, x3 x5 - 1,
            2
          x2 x6 - 1,
                   2  2        2  2          2  2  2      2  2          2  2  2
          a1 + (2x1 x3 x5 + 2x1 x3 x4)x6 + x1 x3 x5  + 2x1 x3 x4 x5 + x1 x3 x4 ,
                 2  2  2       2  2        2  2          2  2  2      2  2
          a2 + x2 x3 x6  + (2x2 x3 x5 + 2x2 x3 x4)x6 + x2 x3 x5  + 2x2 x3 x4 x5,
                 2  2  2       2  2        2  2           2  2          2  2  2
          a3 + x1 x2 x6  + (2x1 x2 x5 + 2x1 x2 x4)x6 + 2x1 x2 x4 x5 + x1 x2 x4 ,

              2        2  2  2       2  2  2     2
             y  + (- x1 x2 x3 x5 - x1 x2 x3 x4)x6
           +
```

```
               2   2   2   2        2   2   2           2   2   2   2
          (- x1 x2 x3 x5  - 2x1 x2 x3 x4 x5 - x1 x2 x3 x4 )x6
        +
              2   2   2       2       2   2   2   2
          - x1 x2 x3 x4 x5  - x1 x2 x3 x4 x5
        }
      ,
                    2       2    2       2
     open= [x1,x2,x2  - x1 ,x2  + x1 ,x3]]
    ]
                         Time: 5.73 (EV) + 0.40 (OT) + 2.30 (GC) = 8.43 sec
```

```
lps : List GPS := [[lexSaturate(ts.close,ts.open)$T]$GPS for ts in ltqas]

   [
       4     4     4     4    2              2  2         2  2
     {x2  - x1 , x3  - x1 , x1 x4 - 1, x5 - x3 x4 , x6 - x2 x4 ,
             2  2  2       2       2
      a1 + 2x2 x3 x4  + (2x3  + 2x2 )x4 + 2,
             2  2  2       2       2
      a2 + 2x2 x3 x4  + (2x3  + 2x2 )x4 + 2,
             2  2  2       2       2
      a3 + 2x2 x3 x4  + (2x3  + 2x2 )x4 + 2,
       2      2  2  2        2       2
      y  - 2x2 x3 x4  + (- 2x3  - 2x2 )x4 - 2}
     ,

       4     2  2     4    2     2     2    2               2  2
     {x2  + x1 x2  + x1 , x3  + x2  + x1 , x1 x4 - 1, x5 - x2 x4 ,
            2  2                         2
      x6 + x2 x4  + x4, a1 - 1, a2 - 1, a3 - 1, y  + 1}
     ]
                         Time: 0.10 (IN) + 6.43 (EV) + 0.27 (OT) = 6.80 sec
-- no redundant varieties

varietySplit?(lps,lp)$GPS

   (28)  true
           Time: 0.02 (IN) + 200.78 (EV) + 0.38 (OT) + 75.08 (GC) = 276.27 sec
```

```
T := WuWenTsunTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T
   [
      4     4       5     5      2          2          2
    {x2  - x1 , x1 x3  - x1 x3, x1 x4 - 1, x3 x5 - 1, x2 x6 - 1,
       3  3            2      3   3       3  2      5
     x1 x3 a1 + (2x1 x2  + 2x1 )x3  + (2x1 x2  + 2x1 )x3,
       3  3            2      3   3       3  2      5
     x1 x3 a2 + (2x1 x2  + 2x1 )x3  + (2x1 x2  + 2x1 )x3,
       3  3            2      3   3       3  2      5
     x1 x3 a3 + (2x1 x2  + 2x1 )x3  + (2x1 x2  + 2x1 )x3,
       3  3 2            2      3   3         3  2      5
     x1 x3 y  + (- 2x1 x2  - 2x1 )x3  + (- 2x1 x2  - 2x1 )x3}
    ,

          9     3  7     7  3     9
    {x1 x2  + x1 x2  - x1 x2  - x1 x2,
          5     5    2        7     3  5     5  3     7      2
     (x1 x2  - x1 x2)x3  + x1 x2  + x1 x2  - x1 x2  - x1 x2, x1 x4 - 1,
```

```
            7     3  5     5  3     7             5     5      2
      (x1 x2  + x1 x2  - x1 x2  - x1 x2)x5 + x1 x2  - x1 x2, x2 x6 - 1,
            5     5             5     5          5     5             5     5
      (x1 x2  - x1 x2)a1 - x1 x2  + x1 x2, (x1 x2  - x1 x2)a2 - x1 x2  + x1 x2,
            5     5             5     5          5     5    2        5     5
      (x1 x2  - x1 x2)a3 - x1 x2  + x1 x2, (x1 x2  - x1 x2)y  + x1 x2  - x1 x2}
     ]
                           Time: 5.90 (EV) + 0.32 (OT) + 2.32 (GC) = 8.53 sec

lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts];

                          Time: 0.02 (IN) + 8.40 (EV) + 5.10 (GC) = 13.52 sec

-- the same saturated ideals as for Wang

T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T

   [
        4     4    4     4    2          2          2
     {x2  - x1 , x3  - x1 , x1 x4 - 1, x3 x5 - 1, x2 x6 - 1,
               2  2        2  2          2  2  2      2  2          2  2  2
      a1 + (2x1 x3 x5 + 2x1 x3 x4)x6 + x1 x3 x5  + 2x1 x3 x4 x5 + x1 x3 x4 ,
             2  2  2       2  2        2  2          2  2  2      2  2
      a2 + x2 x3 x6  + (2x2 x3 x5 + 2x2 x3 x4)x6 + x2 x3 x5  + 2x2 x3 x4 x5,
             2  2  2       2  2        2  2           2  2          2  2  2
      a3 + x1 x2 x6  + (2x1 x2 x5 + 2x1 x2 x4)x6 + 2x1 x2 x4 x5 + x1 x2 x4 ,

          2        2  2  2       2  2  2     2
         y  + (- x1 x2 x3 x5 - x1 x2 x3 x4)x6
       +
              2  2  2  2      2  2  2          2  2  2  2        2  2  2     2
         (- x1 x2 x3 x5  - 2x1 x2 x3 x4 x5 - x1 x2 x3 x4 )x6 - x1 x2 x3 x4 x5
       +
             2  2  2  2
         - x1 x2 x3 x4 x5
       }
     ,

        4     2  2     4     2      2   2      2  2     4    2
     {x2  + x1 x2  + x1 , (x2  + 2x1 )x3  + 2x1 x2  + x1 , x1 x4 - 1,
        2          2
      x3 x5 - 1, x2 x6 - 1,
               2  2        2  2          2  2  2      2  2          2  2  2
      a1 + (2x1 x3 x5 + 2x1 x3 x4)x6 + x1 x3 x5  + 2x1 x3 x4 x5 + x1 x3 x4 ,
             2  2  2       2  2        2  2          2  2  2      2  2
      a2 + x2 x3 x6  + (2x2 x3 x5 + 2x2 x3 x4)x6 + x2 x3 x5  + 2x2 x3 x4 x5,
             2  2  2       2  2        2  2           2  2          2  2  2
      a3 + x1 x2 x6  + (2x1 x2 x5 + 2x1 x2 x4)x6 + 2x1 x2 x4 x5 + x1 x2 x4 ,

          2        2  2  2       2  2  2     2
         y  + (- x1 x2 x3 x5 - x1 x2 x3 x4)x6
       +
```

```
              2  2  2  2      2  2  2          2  2  2  2        2  2  2     2
         (- x1 x2 x3 x5  - 2x1 x2 x3 x4 x5 - x1 x2 x3 x4 )x6 - x1 x2 x3 x4 x5
       +
             2  2  2  2
         - x1 x2 x3 x4 x5
       }]
                          Time: 0.03 (IN) + 5.37 (EV) + 0.45 (OT) = 5.85 sec

lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts];
-- the same saturated ideals as for Wu and Wang
               Time: 0.02 (IN) + 5.70 (EV) + 0.03 (OT) + 2.35 (GC) = 8.10 sec
T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T
   [
        4     2  2     4    2     2     2    2          4       2
     {x2  + x1 x2  + x1 , x3  + x2  + x1 , x1 x4 - 1, x1 x5 - x2 ,
        4       2     2                            2
      x1 x6 + x2  + x1 , a1 - 1, a2 - 1, a3 - 1, y  + 1}
     ,

        2     2    2     2    2          2          2
     {x2  - x1 , x3  - x1 , x1 x4 - 1, x1 x5 - 1, x1 x6 - 1, a1 + 8, a2 + 8,
               2
      a3 + 8, y  - 8}
     ,
       2     2   4     4   2         4       2   2
    {x2  + x1 ,x3  - x1 ,x1 x4 - 1,x1 x5 - x3 ,x1 x6 + 1,a1,a2,a3,y},
       2     2   2     2   2         2         2
    {x2  - x1 ,x3  + x1 ,x1 x4 - 1,x1 x5 + 1,x1 x6 - 1,a1,a2,a3,y}]
             Time: 0.03 (IN) + 10.43 (EV) + 0.17 (OT) + 2.37 (GC) = 13.00 sec
lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]
   [
       4     2 2     4    2     2    2              2 2
     {x2  + x1 x2  + x1 , x3  + x2  + x1 , x1 x4 - 1, x5 - x2 x4 ,
           2 2                     2
      x6 + x2 x4  + x4, a1 - 1, a2 - 1, a3 - 1, y  + 1}
     ,
      2     2   2     2   2                             2
    {x2  - x1 ,x3  - x1 ,x1 x4 - 1,x5 - x4,x6 - x4,a1 + 8,a2 + 8,a3 + 8,y  - 8},
      2     2   4     4   2            2  2
    {x2  + x1 ,x3  - x1 ,x1 x4 - 1,x5 - x3 x4 ,x6 + x4,a1,a2,a3,y},
      2     2   2     2   2
    {x2  - x1 ,x3  + x1 ,x1 x4 - 1,x5 + x4,x6 - x4,a1,a2,a3,y}]
              Time: 0.02 (IN) + 5.52 (EV) + 0.25 (OT) + 2.85 (GC) = 8.63 sec
-- no redundant varieties
varietySplit?(lps,lp)$GPS

   (41)  true
                     Time: 40.38 (EV) + 0.15 (OT) + 14.15 (GC) = 54.68 sec
```

```
---------------------------------------------------------------------------
--% exemple Katsura 3 (from the PoSSo data base)
---------------------------------------------------------------------------
```

Voici un exemple du type *shape lemma* qui n'est pas favorable aux implantations de nos *méthodes de triangulation*. En particulier, comme le montre les calculs ci-après, l'application de notre *implantation de la méthode brute* de Lazard (voir le chapitre 3) nécessite pratiquement une heure de calculs pour produire un résultat. Cependant, si l'on commence par éliminer `t` des polynômes `f1`, `f2`, `f3` à l'aide de `f4` on obtient un résultat au bout de 170 secondes.

Comme dans l'exemple `Moeller example 4` examiné précédemment, pour notre implantation de la méthode de Wu, nous ne donnons pas le résultat brut qui est trop long.

Notons que seule notre implantation de la méthode de Kalkbrener a découvert une factorisation du polynôme univarié de la base lexicographique.

```
ls : List Symbol :=  [t,z,y,x];
R := Integer;
lp:= [f1,f2,f3,f4]

     2         2     2     2                                   2
   [t  - t + 2z  + 2y  + 2x , 2z t + (2y - 1)z + 2x y, 2y t + z  + 2x z - y,
    t + 2z + 2y + 2x - 1]

gb := groebner lp
   [
                              7                6                5              4
      5913075t - 159690237696x  + 31246269696x  + 27439610544x  - 6475723368x
    +
                 3            2
      - 838935856x  + 275119624x  + 4884038x - 5913075
    ,
                             7                6                5              4
      1971025z - 97197721632x  + 73975630752x  - 12121915032x  - 2760941496x
    +
               3           2
      814792828x  - 1678512x  - 9158924x
    ,
                              7                6               5
      5913075y + 371438283744x  - 237550027104x  + 22645939824x
    +
                 4             3            2
      11520686172x  - 2024910556x  - 132524276x  + 30947828x
    ,
          8        7        6       5       4     3     2
   128304x  - 93312x  + 15552x  + 3144x  - 1120x  + 36x  + 15x  - x]
                          Time: 0.03 (IN) + 0.62 (EV) + 0.15 (OT) = 0.80 sec

T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T
   [
     [
       close =
         {
                      10            9           8          7          6
           171927360x   - 133762752x  + 20898000x  + 7727712x  - 2476640x
         +
                    5        4       3      2
           - 29656x  + 72532x  - 4124x  - 667x  + 49x
         ,
```

```
                   6             5            4            3           2
          47420640x  - 45632736x  + 4580216x  + 1753552x  - 343644x
        +
          - 14140x + 2401
       *
        y
      +
                 7             6            5            4          3
        - 57049920x  + 21090960x  + 2334960x  - 1416068x  + 64908x
      +
               2
        29454x  - 1946x
      ,
                           2                2
     (16y - 2x - 1)z + 15y  + (16x - 4)y + 3x  - x, t + 2z + 2y + 2x - 1}
   ,

   open =
          2
    [19764x  - 3424x + 287,

                  6             5            4            3           2
        47420640x  - 45632736x  + 4580216x  + 1753552x  - 343644x  - 14140x
      +
        2401
      ,

                   7             6           5             4           3
        817957440x  - 293610528x  - 887056x  + 14569768x  - 2104792x
      +
                 2
        - 99340x  + 40474x - 2401
      ]
   ]
  ]
                        Time: 0.02 (IN) + 3.33 (EV) + 0.27 (OT) = 3.62 sec
```

```
lps : List GPS := [[lexSaturate(ts.close,ts.open)$T]$GPS for ts in ltqas];
-- lps is exactly [gb]
                        Time: 0.10 (IN) + 3.13 (EV) + 0.02 (OT) = 3.25 sec

T := WuWenTsunTriangularSet(R,E,V,P);
zeroSetSplit(lp)$T;

                        Time: 0.02 (IN) + 5.80 (EV) + 2.45 (GC) = 8.27 sec

characteristicSerie(lp,initiallyReduced?,initiallyReduce)
```

```
  [
   {
                 10             9            8           7           6
     171927360x   - 133762752x  + 20898000x  + 7727712x  - 2476640x
    +
             5          4        3       2
     - 29656x  + 72532x  - 4124x  - 667x  + 49x
    ,

                  6             5            4            3           2
        47420640x  - 45632736x  + 4580216x  + 1753552x  - 343644x  - 14140x
      +
        2401
     *
       y
    +
```

```
                7              6            5             4           3          2
    - 57049920x  + 21090960x  + 2334960x  - 1416068x  + 64908x  + 29454x
  +
    - 1946x
  ,

                 7              6            5              4            3
      817957440x  - 293610528x  - 887056x  + 14569768x  - 2104792x
    +
              2
      - 99340x  + 40474x - 2401
   *
     z
  +
                 6              5             4             3            2
      711309600x  - 684491040x  + 68703240x  + 26303280x  - 5154660x
    +
      - 212100x + 36015
   *
      2
     y
  +
                 7              6              5            4             3
      758730240x  - 919806336x  + 255814400x  + 9735968x  - 12512512x
    +
              2
      1148336x  + 94976x - 9604
   *
     y
  +
               8              7             6           5            4
    142261920x  - 184318848x  + 59373384x  + 680440x  - 2784484x
  +
           3         2
    301224x  + 21343x  - 2401x
  ,
 t + 2z + 2y + 2x - 1}
]
                       Time: 5.05 (EV) + 0.28 (OT) + 2.45 (GC) = 7.78 sec

lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts];
-- lps is exactly [gb]
                       Time: 0.02 (IN) + 3.57 (EV) + 0.03 (OT) = 3.62 sec
```

```
T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T
   [
            7         6         5        4         3      2
     {128304x  - 93312x  + 15552x  + 3144x  - 1120x  + 36x  + 15x - 1,

                          6                     5                   4
         20475433563303312x  - 10133389047507264x  - 253504386749196x
       +
                         3                  2
         596008411599216x  - 50117980046508x  - 8578024644222x + 696659708514
      *
         y
     +
                          6                     5                   4
       - 2638183983365664x  + 1014333552623880x  + 112079255594916x
     +
                        3                  2
       - 69988938500464x  + 3630312399987x  + 980794710996x - 77115301123
```

```
     ,
                        2                    2
     (10y + 4x - 1)z + 12y  + (22x - 4)y + 12x  - 4x, t + 2z + 2y + 2x - 1}
    ,
   {x,y,z,t - 1}]
                                   Time: 3.02 (EV) + 0.17 (OT) = 3.18 sec

lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts];
-- each component is its own saturated ideal
                       Time: 0.02 (IN) + 1.85 (EV) + 0.03 (OT) = 1.90 sec


varietySplit?(lps,lp)$GPS

   (28)  true
                       Time: 4.85 (EV) + 0.03 (OT) + 2.38 (GC) = 7.27 sec

T := LazardTriangularSet(R,E,V,P);
ts : T := [[]];

intersect(lp,[ts])$T;

                 Time: 3065.53 (EV) + 0.20 (OT) + 16.50 (GC) = 3082.23 sec

lp := rewriteIdealWithQuasiMonicGenerators (lp,reduced?,lazyPrem)$pack

                        2                    2
   [t + 2z + 2y + 2x - 1, z  + (- 4y + 2x)z - 4y  + (- 4x + 1)y,
                      2                  2
    (16y - 2x - 1)z + 15y  + (16x - 4)y + 3x  - x,
                      2
    (18y - 4x - 1)z + 16y  + (14x - 4)y]
                       Time: 0.05 (IN) + 0.02 (EV) + 0.07 (OT) = 0.13 sec

lts := intersect(lp,[ts])$T;

          Time: 0.03 (IN) + 162.22 (EV) + 0.02 (OT) + 8.45 (GC) = 170.72 sec

-------------------------------------------------------------------------------
--% 2eme exemple de [Wu Wen-Tsun - 1987b] et Pb4 {Wang "An implementation..."]
--% To illustrate the "rewriteSetWithRemainder" optimization in Wu's method !!!
-------------------------------------------------------------------------------

ls : List Symbol := [x,y,z,r];
lp :LP := [p1,p2,p3]


          2    2    2    2        2        2            2
   (10)  [x  + y  + z  - r ,y x + z  - 1,- x  + z y x - y  - z + 1]

gb := groebner lp

    2    2    2    2         2            2                  3       2
   [x  + y  + z  - r , y x + z  - 1, (z + r  - 2)x + (z - 1)y  + (- r z + 1)y,
```

```
        4     2              2     2          2      3
      (r  - 4r  + 3)x + (- z  + (r  - 1)z - r  + 1)y
    +
        2      2      4     2           2
      ((r  - 1)z  + (- r  + 2r  - 1)z + 2r  - 2)y
    ,
    4     2    2  2    2       2          2       3    2    2
   y  + (z  - r )y  - z  + (- r  + 1)z - r  + 2, z  - z  + r  - 1]
                          Time: 0.03 (IN) + 0.47 (EV) + 0.10 (OT) = 0.60 sec

T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T

             4     2          2        2    2
   [[close= {r  - 4r  + 3,z + r  - 2,y,x  - r  + 1},open= []],

     [
       close =
           3    2    2       4     2    2  2    2       2          2
         {z  - z  + r  - 1, y  + (z  - r )y  - z  + (- r  + 1)z - r  + 2,
                  2
          y x + z  - 1}
        ,
      open= [y]]]

                          Time: 0.02 (IN) + 0.38 (EV) + 0.13 (OT) = 0.53 sec

lps : List GPS := [[lexSaturate(ts.close,ts.open)$T]$GPS for ts in ltqas]

       4     2          2        2    2
   [{r  - 4r  + 3,z + r  - 2,y,x  - r  + 1},

       3    2    2       4     2    2  2    2       2          2
     {z  - z  + r  - 1, y  + (z  - r )y  - z  + (- r  + 1)z - r  + 2,

          4     2              2     2          2      3
        (r  - 4r  + 3)x + (- z  + (r  - 1)z - r  + 1)y
      +
          2      2      4     2           2
        ((r  - 1)z  + (- r  + 2r  - 1)z + 2r  - 2)y
      ,
          2                3       2                 2
      (z + r  - 2)x + (z - 1)y  + (- r z + 1)y, y x + z  - 1,
       2    2    2    2
      x  + y  + z  - r }]

                          Time: 0.12 (IN) + 2.23 (EV) + 0.23 (OT) = 2.58 sec

lps := eliminateRedundantVarieties(lps)$GPS;

                                          Time: 0.03 (IN) + 0.07 (EV) = 0.10 sec
```

```
-- the first saturated ideal is contained in the second
-- which is [gb], so the verification is unnecessary.

T := WuWenTsunTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

      4     2          2        2    2
   [{r  - 4r  + 3,z + r  - 2,y,x  - r  + 1},
      3    2    2      4     2    2  2    2       2          2            2
    {z  - z  + r  - 1,y  + (z  - r )y  - z  + (- r  + 1)z - r  + 2,y x + z  - 1}]
                           Time: 0.02 (IN) + 0.13 (EV) + 0.08 (OT) = 0.23 sec

-- same verification as for Wang

T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T

      3    2    2      4     2    2  2    2       2          2            2
   [{z  - z  + r  - 1,y  + (z  - r )y  - z  + (- r  + 1)z - r  + 2,y x + z  - 1}]
                           Time: 0.02 (IN) + 0.57 (EV) + 0.05 (OT) = 0.63 sec

-- second component of Wu,
-- so the verification is unnecessary.

T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

   [
       3    2    2       4     2    2  2    2       2          2
     {z  - z  + r  - 1, y  + (z  - r )y  - z  + (- r  + 1)z - r  + 2,

           4     2             2     2          2      3
         (r  - 4r  + 3)x + (- z  + (r  - 1)z - r  + 1)y
       +
            2      2       4     2           2
         ((r  - 1)z  + (- r  + 2r  - 1)z + 2r  - 2)y
       }
     ,
      2      2           4            2                        3
    {r  - 3,z  - 2z + 2,y  + (2z - 5)y  - 4z + 1,5x + (2z - 1)y  + (- 4z - 3)y},
              4    2          3                 4    2          3
    {r + 1,z,y  - y  + 1,x + y  - y}, {r - 1,z,y  - y  + 1,x + y  - y},
      2            2           2              2         2
    {r  - 3,z + 1,y  - 2,x}, {r  - 3,z + 1,y,x  - 2}, {r  - 1,z - 1,y,x}]
           Time: 0.02 (IN) + 64.25 (EV) + 0.20 (OT) + 47.65 (GC) = 112.12 sec

-- each saturated ideal is contained in the first one,
-- which is [gb], so the verification is unnecessary.
```

```
------------------------------------------------------------------------------
--% Exemple 1 de Trinks, dit aussi "Trinks difficile" (PoSSo data base)
------------------------------------------------------------------------------
```

Voici un nouvel exemple où la base lexicographique est du type *shape lemma*. Cette base fait deux pages, nous n'en donnons que les monômes de tête.

On notera la *performance* de notre implantation de la méthode de Kalkbrener sur cet exemple. Non seulement, c'est la plus rapide de nos implantations sur cet exemple, mais elle produit ici un ensemble triangulaire initialement réduit contrairement à ce qu'elle avait produit sur des exemples comparables comme `Katsura 3`.

Encore une fois, sur un exemple de type *shape lemma*, pour notre implantation de la méthode de Wu, il est *préférable* de se contenter du *résultat brut* (i.e. non fortement réduit mais simplement initiallement réduit).

Enfin (une fois de plus sur un exemple de type *shape lemma*) pour notre implantation de la méthode de Lazard, il convient *d'inter-réduire* les polynômes du système de départ avant d'appliquer la méthode. Sans cette *phase de préparation* nous n'avons pas pu obtenir de résultat.

```
ls : List Symbol := [x,y,t,z,u,v]; R := Integer;
lp := [f1,f2,f3,f4,f5,f6]

   [45y + 35u - 165v - 36, 35y + 40t + 25z - 27u,
                                     2                                  3
    15x + 25u y + 30t - 18z - 165v , - 9x + 15z y + 20u t, y x + 2z t - 11v ,
                    2
    99x - 11v u + 3v ]

gb := groebner lp;

               Time: 0.02 (IN) + 2.07 (EV) + 0.72 (OT) + 2.88 (GC) = 5.68 sec

[head(p) for p in gb]

   [17766149161458472422166115589155691471353640232570952361584640x,
    10792935615586021996465915220412082568847336441286853559662668 80y,
    7994767122656312589974752015120061162109138104656928562713088 0t,
    539646780779301099823295761020604128442366822064342677983133 44z,
    239843013679689377699242560453601834863274143139707856881392 64u,
                          10
    4380800000000000000000v  ]

T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T

  [
    [
      close =
        {
                                10                        9
          43808000000000000000v    + 189995300000000000000v
        +
                                 8                        7
          343169730200000000000v  + 377900184178000000000v
        +
                                 6                        5
          277427432368460000000v  + 141636786601439800000v
```

```
      +
                           4                         3
        50921375336016834000v  + 12792266529459977340v
      +
                          2
        2215667232541084905v  + 23765355465806988 0v + 8984801833047216
      ,

                          5                     4                     3
            754773360000000v  - 1059998440600000v  - 3134170644330000v
          +
                              2
            - 2812280454319380v  - 1270378696625775v - 225507801878370
       *
          u
      +
                          6                     5                     4
        - 1714497120000000v  + 1715067895200000v  + 7542248713560000v
      +
                       3                    2
        8334341846526960v  + 4603451351394060v  + 1219770883257300v
      +
        117656384181336
      ,
                2                            2
     2079z + 1100u  + (- 5280v - 3432)u + 9360v  + 5445v + 1188,
                 2                            2
     20790t - 6875u  + (33000v - 6732)u - 58500v  + 32670v + 7128,
                         2                             2
     1485y + 2079z + 1100u  + (- 5280v - 2277)u + 9360v ,
     27x - 60u t + (35u - 165v - 36)z}
  ,

  open =
               2
    [444000000v  - 1116901775v - 885050034,

                     5                    4                    3
        13723152000000v  - 19272698920000v  - 56984920806000v
      +
                        2
        - 51132371896716v  - 23097794484105v - 4100141852334
      ]
  ]
]

                       Time: 0.03 (IN) + 3.60 (EV) + 0.27 (OT) = 3.90 sec

lps : List GPS := [[lexSaturate(ts.close,ts.open)$T]$GPS for ts in ltqas];
-- lps is [gb], so the verification is unnecessary
           Time: 0.10 (IN) + 4.97 (EV) + 0.05 (OT) + 2.47 (GC) = 7.58 sec


T := WuWenTsunTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T;

                 Time: 214.70 (EV) + 0.02 (OT) + 216.38 (GC) = 431.10 sec

lts := characteristicSerie(lp,initiallyReduced?,initiallyReduce);

          Time: 0.03 (IN) + 83.55 (EV) + 0.55 (OT) + 42.53 (GC) = 126.67 sec

ts := first lts;
```

```
[head(p) for p in ts::LP]

   [9x, 35y, 360t,

                                         8                                  7
        7247984841275127840000000000000v  + 2231064297463198728960000000000v
      +
                                          6                                  5
        3732719496999634163206800000000v  + 4031099780796384965747892000000v
      +
                                          4                                  3
        3050791030843700240387245680000v  + 1638172050150065990179890045600v
      +
                                         2
        6083968342614193173955429263 24v  + 1442302781563736185674169665 45v
      +
        162464355866582816520961074 06
    *
      z
     ,

                                      8                                  7
        7589125130000000000000000000v  + 2508565451552186400000000000v
      +
                                       6                                  5
        4409894479670886624000000000v  + 4833584477815938729060000000v
      +
                                       4                                  3
        3676227036982919083966400000v  + 1960279480427504660493036000v
      +
                                      2
        7283221234057300969962676 80v  + 1756537536157992231195395 25v
      +
        2005360441377984573732927 0
    *
      u
     ,
                                 12
    1112723200000000000000000000v   ]
                              Time: 0.03 (IN) + 0.02 (EV) + 0.15 (OT) = 0.20 sec
```

-- too much complicated to be checked

```
T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T
```

```
   [
     {
                             10                      9
        4380800000000000000000v   + 1899953000000000000000v
      +
                              8                      7
        34316973020000000000v  + 37790018417800000000v
      +
                              6                      5
        27742743236846000000v  + 14163678660143980000v
      +
                             4                         3                         2
        5092137533601683400 0v  + 1279226652945997734 0v  + 2215667232541084905v
      +
        237653554658069880v + 8984801833047216

      ,

                            5                      4                      3
           754773360000000v  - 1059998440600000v  - 3134170644330000v
         +
```

```
                                 2
           - 2812280454319380v  - 1270378696625775v - 225507801878370
        *
           u
       +
                              6                     5                     4
         - 1714497120000000v  + 1715067895200000v  + 7542248713560000v
       +
                            3                     2
         8334341846526960v  + 4603451351394060v  + 1219770883257300v
       +
         117656384181336
       ,
                   2                            2
      2079z + 1100u  + (- 5280v - 3432)u + 9360v  + 5445v + 1188,
                    2                           2
      20790t - 6875u  + (33000v - 6732)u - 58500v  + 32670v + 7128,
                                              2
      45y + 35u - 165v - 36, 99x - 11v u + 3v }
     ]
                                           Time: 1.57 (EV) + 0.23 (OT) = 1.80 sec
```

```
lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts];
-- lps is [gb], so the verification is unnecessary
```

```
                        Time: 0.02 (IN) + 4.07 (EV) + 0.02 (OT) = 4.10 sec
```

```
T := LazardTriangularSet(R,E,V,P);
lp := rewriteIdealWithQuasiMonicGenerators (lp,reduced?,lazyPrem)$pa
```

```
                  2
   [99x - 11v u + 3v , 45y + 35u - 165v - 36,
                  2                           2
    20790t - 6875u  + (33000v - 6732)u - 58500v  + 32670v + 7128,
                 2                            2
    2079z + 1100u  + (- 5280v - 3432)u + 9360v  + 5445v + 1188,

            3                       2           2
      39875u  + (- 282150v - 59664)u  + (774900v  + 372339v + 61182)u
     +
                 3         2
      - 772200v  - 616842v  - 196020v - 21384
     ,

                 2                                2
      (4884000000v  - 12285919525v - 9735550374)u
     +
                      3              2
      (- 23443200000v  + 55060212900v  + 60100838424v + 10277772912)u
     +
                 4              3              2
     41558400000v  - 31962179700v  - 80320947597v  - 26122801320v - 2588875344
     ]
                     Time: 0.03 (IN) + 0.08 (EV) + 0.12 (OT) = 0.23 sec
```

```
lts := zeroSetSplit(lp)$T;
-- lts is [gb], so the verification is unnecessary
                       Time: 0.03 (IN) + 308.52 (EV) + 8.82 (GC) = 317.37 sec
```

```
-------------------------------------------------------------------------------
--% ROBOT plano dificil (Tomas Recio, Santander)
-------------------------------------------------------------------------------

ls : List Symbol := [s1,c1,s2,c2,l2,l3,a,b]; R := Integer;
lp := [f1,f2,f3,f4]

   [- l3 s2 s1 + (l3 c2 + l2)c1 - a, (l3 c2 + l2)s1 + l3 s2 c1 - b,
      2     2        2     2
    s1  + c1  - 1, s2  + c2  - 1]

gb := groebner lp

      2     2
   [s1  + c1  - 1, l2 s2 s1 + (l2 c2 + l3)c1 - b s2 - a c2,
    l3 s2 s1 + (- l3 c2 - l2)c1 + a, (l2 c2 + l3)s1 - l2 s2 c1 + a s2 - b c2,
    (l3 c2 + l2)s1 + l3 s2 c1 - b,
       2     2
    (l2  - l3 )s1 + 2l3 l2 s2 c1 - a l3 s2 + b l3 c2 - b l2,
    a s1 - b c1 + l3 s2, b s1 + a c1 - l3 c2 - l2,
                                            2     2    2    2
    (2a l2 s2 - 2b l2 c2 - 2b l3)c1 + (- l2  + l3  - a  + b )s2 + 2b a c2,

                                                         2     2    2    2
      (2b l2 s2 + 2a l2 c2 + 2a l3)c1 - 2b a s2 + (- l2  - l3  - a  + b )c2
     +
      - 2l3 l2,
                                         2
    (a l3 s2 + b l3 c2 + b l2)c1 + (- l3 c2 - l3 l2)s2 - b a,
                                       2  2     2    2
    (b l3 s2 - a l3 c2 - a l2)c1 + l3 c2  - l3  + a ,
      2    2                                2     2
    (a  + b )c1 - b l3 s2 - a l3 c2 - a l2, s2  + c2  - 1,
                   2     2    2    2
    2l3 l2 c2 + l2  + l3  - a  - b ]
                         Time: 0.03 (IN) + 1.20 (EV) + 0.22 (OT) = 1.45 sec

T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T

   newTriangulation

                         2     2       2     2
   [[close= {b,a,l3,l2,s2  + c2  - 1,s1  + c1  - 1},open= []],
                   2     2                2     2
    [close= {b,a,l2  - l3 ,l3 c2 + l2,s2,s1  + c1  - 1},open= [l3]],

     [
       close =
           4         2    2    2   2     4       2    2   2
         {l2  + (- 2l3  + a  + b )l2  + l3  + (- a  - b )l3 , l3 c2 + l2,
            2     2
          s2  + c2  - 1, l3 s2 c1 - b, l3 s2 s1 + (- l3 c2 - l2)c1 + a}
       ,
```

```
 open= [l3,s2]]
,
                    2     2
[close= {b,a,l3,l2,s2  + c2  - 1,(l3 c2 + l2)s1 + l3 s2 c1 - b},open= [l2]],

[
  close =
      2    2                2     2
    {a  + b , 2l3 l2 c2 + l2  + l3 , b l3 s2 + a l3 c2 + a l2,
                  2  2              2    2
     2b l3 s2 c1 + l3 c2  + 2l3 l2 c2 + l2  - b ,
     (l3 c2 + l2)s1 + l3 s2 c1 - b}
  ,
                    2     2
 open= [b,a,l3,l2,l2  - l3 ]]
,

[
  close =
           2    2    2    2     2
    {l3, l2  - a  - b , s2  + c2  - 1,
                    2     2
     (2l3 l2 c2 + l2  + l3 )c1 - b l3 s2 - a l3 c2 - a l2,
     (l3 c2 + l2)s1 + l3 s2 c1 - b}
  ,
         2    2
 open= [a  + b ,l2]]
,

[
  close =
       2    2    2        2     2
    {l3  - a  - b , l2, s2  + c2  - 1,
                    2     2
     (2l3 l2 c2 + l2  + l3 )c1 - b l3 s2 - a l3 c2 - a l2,
     (l3 c2 + l2)s1 + l3 s2 c1 - b}
  ,
 open= [l3,c2]]
,

[
  close =
                  2     2    2    2    2     2
    {2l3 l2 c2 + l2  + l3  - a  - b , s2  + c2  - 1,
                    2     2
     (2l3 l2 c2 + l2  + l3 )c1 - b l3 s2 - a l3 c2 - a l2,
     (l3 c2 + l2)s1 + l3 s2 c1 - b}
  ,
         2    2         2     2    2    2
 open= [a  + b ,l3,l2,l2  - l3  + a  + b ]]
]
          Time: 0.02 (IN) + 5.80 (EV) + 0.40 (OT) + 2.45 (GC) = 8.67 sec
```

```
-- solution too much complicated to be checked

T := WuWenTsunTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

                      Time: 235.42 (EV) + 0.65 (OT) + 57.40 (GC) = 293.47 sec

-- solution too much complicated to be printed and checked

T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T
   [
                    2     2    2    2    2     2
     {2l3 l2 c2 + l2  + l3  - a  - b , s2  + c2  - 1,
                     2     2
      (2l3 l2 c2 + l2  + l3 )c1 - b l3 s2 - a l3 c2 - a l2,
      (l3 c2 + l2)s1 + l3 s2 c1 - b}
     ]
             Time: 0.02 (IN) + 13.17 (EV) + 0.07 (OT) + 4.90 (GC) = 18.15 sec

-- (too much) simple solution

lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]

             Time: 0.02 (IN) + 15.20 (EV) + 0.38 (OT) + 8.37 (GC) = 23.97 sec

-- lps is [gb]

T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T
   [
                    2     2    2    2
     {2l3 l2 c2 + l2  + l3  - a  - b ,

           2  2  2     4         2     2     2   2     4        2     2   2
        4l3 l2 s2  + l2  + (- 2l3  - 2a  - 2b )l2  + l3  + (- 2a  - 2b )l3
      +
         4     2 2    4
        a  + 2b a  + b
      ,
        2     2                         2       2    3    2
      (2a  + 2b )l2 c1 - 2b l3 l2 s2 - a l2  + a l3  - a  - b a,
        2     2                         2       2      2    3
      (2a  + 2b )l2 s1 + 2a l3 l2 s2 - b l2  + b l3  - b a  - b }
     ,

      2    2                2     2                    2       2
     {a  + b , 2l3 l2 c2 + l2  + l3 , 2b l3 l2 s2 + a l2  - a l3 ,
        2  3     2  2           4           2     2    2       4
      (4b l2  - 4b l3 l2)c1 + a l2  + (- 2a l3  - 4b a)l2  + a l3 ,
           3        2        4       2     2  2     4
      (4b l2  - 4b l3 l2)s1 - l2  + (2l3  - 4b )l2  - l3 }
     ,
```

```
       2    2    2   2     2      2    2             2    2
   {l3,l2  - a  - b ,s2  + c2  - 1,(a  + b )c1 - a l2,(a  + b )s1 - b l2},

      2    2    2        2     2        2    2
    {l3  - a  - b , l2, s2  + c2  - 1, (a  + b )c1 - b l3 s2 - a l3 c2,
      2    2
     (a  + b )s1 + a l3 s2 - b l3 c2}
    ,
                          2     2
   {b,a,l2 + l3,c2 - 1,s2,s1  + c1  - 1},
                          2     2
   {b,a,l2 - l3,c2 + 1,s2,s1  + c1  - 1},
                2     2      2     2
   {b,a,l3,l2,s2  + c2  - 1,s1  + c1  - 1}]
                    Time: 30.03 (EV) + 0.33 (OT) + 12.35 (GC) = 42.72 sec

-- solution too much complicated to be checked
-----------------------------------------------------------------------------
--% (not so) Easy example 3
-----------------------------------------------------------------------------
```

Voici un premier exemple où l'une des implantations de nos quatre méthodes *échoue*. Plus précisément, nous n'obtenons pas de résultat avec notre implantation de la méthode de Wu au bout d'une heure. Même l'opération

$$\mathsf{medialSet}(lp, initiallyReduced?, initiallyReduce)$$

ne donne une réponse dans ce délai. Les initiaux des différents polynômes apparaissant dans les ensembles Q successifs de l'algorithme medialSet *grossissent* très vite et rendent inaccessible la découverte d'un polynôme univarié en `e`. Par suite, notre implantation de la méthode n'arrive même pas à calculer un ensemble caractéristique de l'idéal associé à `lp` et par conséquent n'est pas capable de *scinder les calculs.* A l'opposé, les méthodes de Wang et Lazard qui, elles, *scindent vite* obtiennent rapidement un résultat. Pour ces deux méthodes, comme pour celle de Kalkbrener, le résultat obtenu est analogue à celui de l'algorithme *lexTriangular* et se vérifie sans peine.

```
ls : List Symbol := [a,b,c,d,e];
R := Integer;

lp := [p1,p2,p3,p4,p5]

     2        2                    2     3               3     4
   [a  + a, (b  + b + 1)a + b, c b  + (c  + 1)b + c, d c  + (d  + 1)c + d,
      4     5
    e d  + (e  + 1)d + e]

gb := reverse groebner lp

          6       5        4    5   5        3    5   5        2    5      5
   (16)  [e  + e,(e  + 1)d,d  - e ,(e  + 1)c,c  - e ,(e  + 1)b,b  - e ,a - e ]
                        Time: 0.05 (IN) + 0.48 (EV) + 0.12 (OT) = 0.65 sec
T := LXATSET(R,E,V,P);
lts := lexTriangular(gb)
```

```
           5      4      3      2
   (17)  [{e  + 1,d  + 1,c  + 1,b  + 1,a + 1},{e,d,c,b,a}]
                         Time: 0.03 (IN) + 0.40 (EV) + 0.55 (OT) = 0.98 sec

T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T

                     2
   [[close= {e,d,c,b,(b  + b + 1)a + b},open= []],
             5      4      3      2       2
    [close= {e  + 1,d  + 1,c  + 1,b  + 1,(b  + b + 1)a + b},open= [c,b]]]
                         Time: 0.03 (IN) + 0.63 (EV) + 0.08 (OT) = 0.75 sec

lps : List GPS := [[lexSaturate(ts.close,ts.open)$T]$GPS for ts in ltqas]

                     5      4      3      2
   (21)  [{e,d,c,b,a},{e  + 1,d  + 1,c  + 1,b  + 1,a + 1}]
                         Time: 0.10 (IN) + 1.35 (EV) + 0.23 (OT) = 1.68 sec

T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T

                      5      4      3      2
   (24)  [{e,d,c,b,a},{e  + 1,d  + 1,c  + 1,b  + 1,a + 1}]
                       Time: 9.08 (EV) + 0.07 (OT) + 12.87 (GC) = 22.02 sec

T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

           5      4      3      2
   (27)  [{e  + 1,d  + 1,c  + 1,b  + 1,a + 1},{e,d,c,b,a}]
                                     Time: 0.38 (EV) + 0.03 (OT) = 0.42 sec
```

Nous illustrons maintenant l'explosion des calculs avec notre implantation de la méthode de Wu sur cet exemple. Pour cela nous donnons tout d'abord les 3 premiers couples `[bs,rs]` calculés par l'algorithme medialSet. Puis, pour quelques *tours de boucle* supplémentaires, nous donnons les monômes de tête des polynômes de ces ensembles.

```
rs := lp
     2        2                   2     3               3     4
   [a  + a, (b  + b + 1)a + b, c b  + (c  + 1)b + c, d c  + (d  + 1)c + d,
       4     5
    e d  + (e  + 1)d + e]

rec := basicSet(rs,initiallyReduced?)$T
            4     5               3     4              2     3            2
   [bas= {e d  + (e  + 1)d + e,d c  + (d  + 1)c + d,c b  + (c  + 1)b + c,a  + a}
     ,
           2
    top= [(b  + b + 1)a + b]]

bs := (rec::RBT).bas;
rs := (rec::RBT).top;
rs := rewriteSetWithReduction(rs,bs,initiallyReduce,initiallyReduced?)$T;
rs := removeDuplicates concat(rs,members(bs));
```

```
rec := basicSet(rs,initiallyReduced?)$T
   [
     bas =
          4     5                3     4                 2     3
       {e d  + (e  + 1)d + e, d c  + (d  + 1)c + d, c b  + (c  + 1)b + c,
         5
        (e  - e + 1)d c b a - e d c b}
     ,
          2
    top= [a  + a]]

bs := (rec::RBT).bas;
rs := (rec::RBT).top;
rs := rewriteSetWithReduction(rs,bs,initiallyReduce,initiallyReduced?)$T;
rs := removeDuplicates concat(rs,members(bs));
rec := basicSet(rs,initiallyReduced?)$T
   [
     bas =
          4     5                3     4
       {e d  + (e  + 1)d + e, d c  + (d  + 1)c + d,

                 22      17      12      7     2       18      13      8    3
              ((e   + 4e   + 6e   + 4e  + e )d + e   + 3e   + 3e  + e )c
            +
                 10     5  7        10     5  3
              (- e   - e )d  + (- e   - e )d
         *
            b
        +
              10    5  4 3
          (- e   - e )d c
        }
     ,
           5                              2     3
    top= [(e  - e + 1)d c b a - e d c b,c b  + (c  + 1)b + c]]



                                      22        50 3 2    4
[leadingMonomial(b) for b in bs] = [e  d c b,6e  d c ,e d ]
                                      26 3 2     3
[leadingMonomial(r) for r in rs] = [e  d c a,d c ]

                                       162 3     4
[leadingMonomial(b) for b in bs] =  [e   d c,e d ]
                                        104 3     22        50 3 2
[leadingMonomial(r) for r in rs] =  [5e   d c a,e  d c b,6e  d c ]

                                         398 3
[leadingMonomial(b) for b in bs] =  [21e   d ]
                                         272 3    192 3   162 3     4
[leadingMonomial(r) for r in rs] =  [14e   d a,9e   d b,e   d c,e d ]

                                         805 2
```

```
[leadingMonomial(b) for b in bs] =  [21e   d ]
                                           666 2       586 2     556 2     398 3
[leadingMonomial(r) for r in rs] =  [1029e   d a,1134e   d b,42e   d c,21e   d ]


                                           2016
[leadingMonomial(b) for b in bs] =   [441e    d]
                                           1483        1403        1373        805 2
[leadingMonomial(r) for r in rs] =   [147e    d a,252e    d b,21e    d c,21e   d ]
```

Nous terminons cet exemple en montrant les *grandes étapes* du calcul de la décomposition obtenue par notre implantation de la méthode de Lazard. Notons que la normalisation permet sur cet exemple de conserver des initiaux *simples*.

```
T := LazardTriangularSet(R,E,V,P);

                                                   Time: 0.02 (IN) = 0.02 sec
ts : T := [[]]

   (32)  {}
                                     Time: 0.43 (IN) + 0.18 (OT) = 0.62 sec
lts := [ts]

   (33)  [{}]
                                                                Time: 0 sec
lts := intersect([p5],lts)$T

             4     5
   (34)  [{e d  + (e  + 1)d + e},{e,d}]
                           Time: 0.03 (IN) + 0.02 (EV) + 0.05 (OT) = 0.10 sec
lts := intersect([p4],lts)$T

             4     5             3       5
   (35)  [{e d  + (e  + 1)d + e,e c  + (- e  - 1)c + e},{e,d,c}]
                           Time: 0.03 (IN) + 0.07 (EV) + 0.02 (OT) = 0.12 sec
lts := intersect([p3],lts)$T

       4     5             3       5               2     5
   [{e d  + (e  + 1)d + e,e c  + (- e  - 1)c + e,e b  + (e  + 1)b + e},
    {e,d,c,b}]
                           Time: 0.05 (IN) + 0.07 (EV) + 0.03 (OT) = 0.15 sec
lts := intersect([p2],lts)$T

   [
        4     5             3       5               2     5
     {e d  + (e  + 1)d + e, e c  + (- e  - 1)c + e, e b  + (e  + 1)b + e,
        5
      (e  - e + 1)a - e}
     ,
    {e,d,c,b,a}]
                           Time: 0.02 (IN) + 0.17 (EV) + 0.07 (OT) = 0.25 sec
lts := intersect([p1],lts)$T

           5      4      3      2
   (38)  [{e  + 1,d  + 1,c  + 1,b  + 1,a + 1},{e,d,c,b,a}]
                           Time: 0.02 (IN) + 0.10 (EV) + 0.02 (OT) = 0.13 sec
```

```
--------------------------------------------------------------------------------
--% Caprasse (from the PoSSo data base)
--------------------------------------------------------------------------------
```

Voici un autre exemple de dimension 0, de difficulté *moyenne* et qui n'est pas du type *shape lemma*. Comme sur le précédent, notre implantation de la méthode de Wu *échoue*.

Notre implantation de la méthode de Wang donne une solution nécessitant un travail supplémentaire pour être aussi satisfaisante que celle de *lexTriangular* (et ceci en un temps *moyen*).

Notre implantation de la méthode de Kalkbrener donne une solution *plus exploitable* et un temps *canon* par rapport aux autres méthodes.

Enfin, notre implantation de la méthode de Lazard donne, quant à elle, la *meilleure décomposition* (elle découvre plus de factorisations que celle de Kalkbrener ou *lexTriangular* et produit des ensembles triangulaires normalisés) en un temps *moyen*.

```
ls : List Symbol := [x,y,z,t];
R := Integer;

lp := [p1,p2,p3,p4]
                       2
   [(2t y - 2)x + z y  - z,
        3             2        2               3      2
    - z x  + (4t y + 4)x  + (4z y  + 4z)x + 2t y  - 10y  - 10t y + 2,
      2
    (t  - 1)x + 2t z y - 2z,
       3      2                 2     3               2      2
    (- z  + (4t  + 4)z)x + (4t z  + 2t  - 10t)y + 4z  - 10t  + 2]

T := LXATSET(R,E,V,P);
gb := reverse groebner lp

      20        16        14         12         8         6        4
   [27t   - 1719t   - 8000t   - 11746t   + 11746t  + 8000t  + 1719t  - 27,
      14        12        10        8        6        4        2
   (27t   + 243t   + 711t   + 559t   - 559t   - 711t   - 243t   - 27)z,

          6         4          2          2         18        16         14
     (9600t  + 22400t  - 22400t  - 9600)z  - 270t   + 513t   + 17190t
    +
           12          10          8          6          4         2
     46826t   - 34270t   - 186100t  - 12190t  + 143174t  + 29540t  - 4413
    ,

           2          4          4          2         18        16          14
     (16000t  - 16000)z  + (64000t  - 64000)z  + 2943t   + 297t   - 187344t
    +
             12          10          8           6           4          2
     - 890936t   - 1370330t   - 138430t  + 1330344t  + 1063936t  + 224387t
    +
     - 34867
    ,

        9       7          4         2          3
     24z  - 384z  + (1664t  + 1792t  + 2688)z
    +
           12          10          8          6          4          2
     (- 783t   - 6750t   - 18537t  - 11140t  + 14735t  + 12098t  + 4233)z
    ,
```

```
                      8           6
      6144000y - 48000t z  + 768000t z
    +
                5          3             2          19         17
      (- 3328000t  - 3584000t  - 5376000t)z  - 627615t   - 190431t
    +
               15             13             11           9            7
      40085460t   + 198035588t   + 321282190t   + 48665150t  - 311914460t
    +
                 5           3
      - 241936588t  - 48825575t  + 1570281t
    ,

                   7       5           4       2        3
      12288x - 128z  + 1792z  + (- 1920t  - 1280t  + 5248)z
    +
              12         10         8         6         4        2
      (- 2079t   - 17334t   - 44865t  - 28980t  + 24591t  + 29642t  + 22641)z
    ]
             Time: 0.03 (IN) + 5.98 (EV) + 0.35 (OT) + 15.52 (GC) = 21.88 sec
lts := lexTriangular(gb)
   [
      2       9      7       3                   8        6         2
     {t  - 1, z  - 16z  + 256z  - 256z, 128y - t z  + 16t z  - 256t z  + 128t,
            7      5      3
      96x - z  + 14z  + 16z  - 128z}
     ,
      4      2       4       2       2      2             2
    {3t  + 10t  + 3,3z  + (12t  + 12)z  - 16t ,y + t,4x + (3t  + 5)z},
      4      2       4       2       2      2             2
    {3t  + 10t  + 3,3z  + (12t  + 12)z  - 16t ,y + t,4x + (3t  + 5)z},
      4      2       4       2       2      2             2
    {3t  + 10t  + 3,3z  + (12t  + 12)z  - 16t ,y + t,4x + (3t  + 5)z},
     6     4     2             5      3
    {t  - 9t  - 9t  + 1,z,6y + t  - 10t  - 5t,x}]
              Time: 0.02 (IN) + 0.68 (EV) + 0.18 (OT) + 1.30 (GC) = 2.18 sec

T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T
               2                              2
   [[close= {t  - 1,z,t y + 1,(2t y - 2)x + z y  - z},open= [t]],

    [
      close =
          2       8      6       2
        {t  - 1, z  - 16z  + 256z  - 256, t y - 1,
           3             2          2
         (z  - 8z)x + (- 4t z  + 8t)y - 4z  + 8}
       ,
                 2
     open= [t,z,z  - 8]]
    ,


    [
      close =
          2
        {t  + 1, z,

               4         3        2    5     3          4      2      2    4
           (t z  + (- 2t  - 6t)z  + t  - 6t  + 5t)y - z  + (6t  + 2)z  - 5t
```

```
          +
               2
             6t  - 1
          ,
            2
         (t  - 1)x + 2t z y - 2z}
       ,
      open= [t]]
     ,
     [
       close =
             4      2
         {3t  + 10t  + 3,

                       6            4            2            4
             (5643333t  + 19923975t  + 8959203t  + 981585)z
          +
                          6             4             2             2             6
             (- 24266082t  - 86182515t  - 38768772t  - 4245975)z  - 104788796t
          +
                          4             2
             - 365170284t  - 164001716t  - 17973924
          ,

                 4         3        2    5     3           4      2      2     4
             (t z  + (- 2t  - 6t)z  + t  - 6t  + 5t)y - z  + (6t  + 2)z  - 5t
          +
               2
             6t  - 1
          ,
            2
         (t  - 1)x + 2t z y - 2z}
       ,

       open =
              2       2         2          2           2            2
         [t, t  - 1, t  + 1, 10t  + 3, 91t  + 30, 820t  + 273, 4921t  + 1641,
                  2         2          2
          (29523t  + 9837)z  + 59050t  + 19686]
       ]
     ,
     [
       close =
            4      2
         {t  - 10t  + 1, z,

                 4         3        2    5     3           4      2      2     4
             (t z  + (- 2t  - 6t)z  + t  - 6t  + 5t)y - z  + (6t  + 2)z  - 5t
          +
               2
             6t  - 1
          ,
            2
         (t  - 1)x + 2t z y - 2z}
       ,
              2      2      2        2        2          2           2
      open= [t,t  - 5,t  - 1,t  + 1,10t  - 1,99t  - 10,980t  - 99,9701t  - 980]]
     ]
```

```
                  Time: 47.98 (EV) + 0.57 (OT) + 39.58 (GC) = 88.13 sec
```

```
lps : List GPS := [[lexSaturate(ts.close,ts.open)$T]$GPS for ts in ltqas]
```

```
      2
   [{t  - 1,z,y + t,x},
```

```
      2      8      6       2                   7      5      3
    {t  - 1,z  - 16z  + 256z  - 256,y - t,96x - z  + 14z  + 16z  - 128z},
      2
    {t  + 1,z,y + t,x},
       4      2      4       2       2      2                2
    {3t  + 10t  + 3,3z  + (12t  + 12)z  - 16t ,y + t,4x + (3t  + 5)z},
      4     2
    {t  - 10t  + 1,z,y - t,x}]
               Time: 0.07 (IN) + 4.20 (EV) + 0.22 (OT) + 2.63 (GC) = 7.12 sec

-- no redundant varieties

varietySplit?(lps,lp)$GPS

   (20)  true
              Time: 0.03 (IN) + 8.82 (EV) + 0.05 (OT) + 6.92 (GC) = 15.82 sec

T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T

   [
        4      2            2         2         2
     {3t  + 10t  + 3, (3279t  + 1089)z  - 13124t  - 4380,
        2     2            2    3         2
      (z  + 5t  - 1)y - t z  + t  - 5t, (t  - 1)x + 2t z y - 2z}
     ,
      6     4     2         3            2           2
    {t  - 9t  - 9t  + 1,z,(t  - 5t)y - 5t  + 1,x}, {t  - 1,z,t y + 1,x},

       2       8      6       2
     {t  - 1, z  - 16z  + 256z  - 256, t y - 1,
        3               2           2
      (z  - 8z)x + (- 4t z  + 8t)y - 4z  + 8}
     ]
                          Time: 0.02 (IN) + 4.02 (EV) + 0.18 (OT) = 4.22 sec

lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]

       4      2      2     2                   2
   [{3t  + 10t  + 3,z  + 2t  + 2,y + t,4x + (3t  + 5)z},
      6     4     2            5      3           2
    {t  - 9t  - 9t  + 1,z,6y + t  - 10t  - 5t,x}, {t  - 1,z,y + t,x},
      2      8      6       2                   7      5      3
    {t  - 1,z  - 16z  + 256z  - 256,y - t,96x - z  + 14z  + 16z  - 128z}]
                          Time: 0.02 (IN) + 2.93 (EV) + 0.12 (OT) = 3.07 sec

-- no redundant varieties

varietySplit?(lps,lp)$GPS

   (25)  true
                          Time: 7.88 (EV) + 0.02 (OT) + 1.50 (GC) = 9.40 sec
```

```
T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T

      2      2                    2                   2
   [{t  + 3,z  - 4,y + t,x - z}, {t  + 1,z,y + t,x}, {t  - 1,z,y + t,x},
            8      6       2                    7      5      3
    {t + 1,z  - 16z  + 256z  - 256,y + 1,96x - z  + 14z  + 16z  - 128z},
      4      2                   2       2
    {t  - 10t  + 1,z,y - t,x}, {3t  + 1,3z  + 4,y + t,x + z},
            8      6       2                    7      5      3
    {t - 1,z  - 16z  + 256z  - 256,y - 1,96x - z  + 14z  + 16z  - 128z}]
                      Time: 108.55 (EV) + 0.15 (OT) + 25.07 (GC) = 133.77 sec

-- Zero Dimensional Lazard triangular sets
-- are they own saturated ideals. None
-- of them may be contained in another one.

lps : List GPS := [[ts::LP]$GPS for ts in lts];
varietySplit?(lps,lp)$GPS

   (30)  true
             Time: 0.02 (IN) + 12.15 (EV) + 0.02 (OT) + 3.25 (GC) = 15.43 sec

-----------------------------------------------------------------------------
--% Cyclic 5
-----------------------------------------------------------------------------
```

Cet exemple permet d'illustrer l'intérêt des trois méthodes fondées *explicitement sur une notion de pgcd*. Plus précisément, l'algorithme *lexTriangular* et les méthodes de Kalkbrener et Lazard, calculent des *pgcd de polynômes modulo un ensemble triangulaire*, cette notion étant légèrement différente d'un cas à l'autre comme nous l'avons déjà vu. Ceci permet de découvrir certaines factorisations de polynômes à coefficients dans une extension du corps de base et de provoquer des scindages de deuxième espèce c'est-à-dire du type :

$$I \longmapsto (I + a, I + b) \text{ avec } ab \in I$$

où I est un idéal de $\mathbf{P}_n$. Les méthodes de Wang et Wu, dans leurs *versions bruts* n'utilisant pas la factorisation sans facteurs carrés dans $\mathbf{P}_n$ ne provoquent que des scindages de première espèce c'est-à-dire du type :

$$I \longmapsto (I + p, I : p)$$

où I est un idéal de $\mathbf{P}_n$ et p un polynôme de $\mathbf{P}_n$. Nos implantations des méthodes de Wang et Wu *échouent* sur cet exemple.

De plus, comme dans l'exemple précédent, la normalisation (utilisée dans *lexTriangular* et dans la méthode de Lazard) permet ici d'obtenir des ensembles triangulaires *assez simples*.

Notons que les méthodes de Kalkbrener et Lazard peuvent découvrir des factorisations que ne découvre pas *lexTriangular*. Par exemple, elles découvrent une factorisation du second polynôme modulo le premier, dans le dernier ensemble triangulaire de la sortie de *lexTriangular* ci-dessous.

```
ls : List Symbol := [a,b,c,d,e];
R := Integer;
```

```
lp := [p5,p1,p2,p3,p4]

   [e d c b a - 1, a + b + c + d + e, (b + e)a + c b + d c + e d,
    ((c + e)b + e d)a + d c b + e d c, (((d + e)c + e d)b + e d c)a + e d c b]

T := LXATSET(R,E,V,P);
gb := reverse groebner lp;
lts := lexTriangular(gb);
      5                  2           2
   [{e  - 1,d - e,c - e,b  + 3e b + e ,a + b + 3e},
      5            2           2
    {e  - 1,d - e,c  + 3e c + e ,b + c + 3e,a - e},

       5       6       5     2 4     3 3     4 2
     {e  - 1, d  + 4e d  + 5e d  + 5e d  + 5e d  + 4d + e,
               5      2 4      3 3      4 2
      5c + 8e d  + 30e d  + 30e d  + 25e d  + 30d + 22e,
               5      2 4      3 3      4 2
      5b - 2e d  - 10e d  - 15e d  - 10e d  - 10d - 8e,
               5      2 4      3 3      4 2
      5a - 6e d  - 20e d  - 15e d  - 15e d  - 15d - 9e}
     ,

       10       5         2        6              7       2          6
     {e   + 123e  + 1, 55d  + (- 2e  - 233e)d - 8e  - 987e , 55c + e  + 144e,
             6                       6
      55b + e  + 144e, 55a + 55d - 2e  - 233e}
     ]
                         Time: 0.02 (IN) + 0.25 (EV) + 0.18 (OT) = 0.45 sec

T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T
      5                  2           2
   [{e  - 1,d - e,c - e,b  + 3e b + e ,a + b + 3e},
      5            2           2                    2
    {e  - 1,d - e,c  + 3e c + e ,(4c + e)b + e c - e ,a + b + c + 2e},

       5       2           2                         2
     {e  - 1, d  + 3e d + e , (13d + 5e)c + 5e d + 2e ,
                           2              2       2     3
      ((d + 3e)c - e d + 2e )b + (2e d - e )c + 3e d + e , a + b + c + d + e}
     ,

       10       5              5                   6
     {e   + 123e  + 1, (439204e  + 3571)d + 167761e  + 1364e,
           5            6              6             7    2
      ((66e  - 1)d + 27e  - 2e)c + (27e  - 2e)d + 11e  - e ,
                           2              2       2     3
      ((d + 3e)c - e d + 2e )b + (2e d - e )c + 3e d + e , a + b + c + d + e}
     ,

       5       4      3    2 2    3     4
     {e  - 1, d  + e d  + e d  + e d + e ,
           2 3      3 2       4                3 3       4 2
      (276e d  + 16e d  + 181e d + 133)c - 165e d  - 117e d  + 16d - 260e,
        2                2             2            2        2    2     3
      (c  - d c + e d - e )b + (d - e)c  + (e d - 2e )c + e d  + e d - e ,
      a + b + c + d + e}
     ,
```

```
   10       5
 {e   + 123e  + 1,

          94030751020073942391409155330154842587565168631767494302004169431
          11991564642569736430639516155403107315724664653211696649835607187
          39645176022111272449295251
        *
            5
           e
       +
        7645281878642074485535133205944172751143011384808768193177375809620
        0956820603208268302880843838729926953428088797082523678357212407888
        72451903860114461144
    *
       d
   +
     -
         137189016858942574813987960473276725913793500449015015431633921489
         426427977458850029545499585044983724193922911803790583217672598429
         809600464229122907959008
       *
          6
         e
   +
     -
         111543159356079653280855561338741544643564060217965054123173813260
         324732261763692337866823574725792820776071912157276632224335911924
         4897932753117196435737
       *
         e
   ,

                     5                        4
         566001535115592076d  + 483423024042566379e d
       +
                     2 3                      3 2
         483423024042567759e d  + 483423024042566459e d
       +
                     4                       5
         483423024042567284e d - 82578511073025032e
    *
       c
   +
                     5                       2 4
     1481811256617236851e d  + 1265617907887698986e d
   +
                     3 3                     4 2
     1265617907887698161e d  + 1265617907887698401e d
   +
                     5                     6
     1265617907887699066e d - 216193348729539165e
   ,
    2              2            2            2        2    2     3
  (c  - d c + e d - e )b + (d - e)c  + (e d - 2e )c + e d  + e d - e ,
  a + b + c + d + e}
 ]
                   Time: 1702.22 (EV) + 0.80 (OT) + 105.97 (GC) = 1808.98 sec
```

```
lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]
```

```
    5                  2           2
 [{e  - 1,d - e,c - e,b  + 3e b + e ,a + b + 3e},
    5              2           2
  {e  - 1,d - e,c  + 3e c + e ,b + c + 3e,a - e},
```

```
      5      2           2
    {e  - 1,d  + 3e d + e ,c + d + 3e,b - e,a - e},

       10       5             6                6                6
     {e   + 123e  + 1, 55d + e  + 144e, 55c + e  + 144e, 55b + e  + 144e,
                6
      55a - 3e  - 377e}
     ,

       5       4      3    2 2    3     4       4 2      3 3
     {e  - 1, d  + e d  + e d  + e d + e , c - e d , b - e d ,
           3 3    4 2
      a + e d  + e d  + d + e}
     ,

       10       5              6                6                6
     {e   + 123e  + 1, 55d - 3e  - 377e, 55c + e  + 144e, 55b + e  + 144e,
               6
      55a + e  + 144e}
     ]
                          Time: 5.13 (EV) + 0.22 (OT) + 1.22 (GC) = 6.57 sec

-- no redundant varieties

varietySplit?(lps,lp)$GPS

   (22)  true
           Time: 0.03 (IN) + 66.32 (EV) + 0.23 (OT) + 29.32 (GC) = 95.90 sec

T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T
   [
       10       5             6                6                6
     {e   + 123e  + 1, 55d + e  + 144e, 55c + e  + 144e, 55b + e  + 144e,
                6
      55a - 3e  - 377e}
     ,

       5       4      3    2 2    3     4       4 2      3 3
     {e  - 1, d  + e d  + e d  + e d + e , c - e d , b - e d ,
           3 3    4 2
      a + e d  + e d  + d + e}
     ,
      5      2           2
    {e  - 1,d  + 3e d + e ,c + d + 3e,b - e,a - e},

       10       5              6                6                6
     {e   + 123e  + 1, 55d - 3e  - 377e, 55c + e  + 144e, 55b + e  + 144e,
               6
      55a + e  + 144e}
     ,
      5            2           2
    {e  - 1,d - e,c  + 3e c + e ,b + c + 3e,a - e},
```

```
       5                  2           2
    {e  - 1,d - e,c - e,b  + 3e b + e ,a + b + 3e}]
            Time: 0.02 (IN) + 38.05 (EV) + 0.17 (OT) + 27.93 (GC) = 66.17 sec

-- this last decomposition is excatly the
-- list of the saturated ideals associated
-- to Kalkbrener's decomposition, so the
-- verification is unnecessary.

-------------------------------------------------------------------------------
--% Singular points of a curve (a test-example communicated by Dominique Duval)
-------------------------------------------------------------------------------
```

Voici un exemple, *apparemment simple*, et instructif. Ici encore, notre implantation de la méthode de Wu *échoue*.

Celle de Wang retourne rapidement un résultat, mais celui-ci n'est pas *très satisfaisant*. En effet, des calculs supplémentaires sont nécessaires pour avoir une *description claire* de l'ensemble des solutions.

Notre implantation de la méthode de Kalkbrener retourne aussi rapidement un résultat qui, lui, nécessite moins de travail pour être aussi *lisible* que celui retourné par *lexTriangular*.

Enfin, notre implantation de la méthode de Lazard retourne un résultat comparable à celui de *lexTriangular* mais en un temps qui n'est pas satisfaisant. Nous pensons que ceci est dû aux calculs de normalisation dans les calculs pgcd. Cet exemple constitue donc une piste pour essayer d'optimiser notre implantation de la méthode de Lazard. On remarquera que cette dernière est aussi performante que *lexTriangular* si on lui *soumet* directement la base de Gröbner lexicographique. En effet, dans ce cas, on est ramené essentiellement à un unique calcul de pgcd de deux polynômes univariés à coefficients dans une extension simple. Nous pensons que ce n'est pas la normalisation qui soit *en cause* sur cet exemple (comme dans d'autres où notre implantation de la méthode de Lazard est moins *performante* que celle de Kalkbrener) mais son recours trop fréquent.

```
ls : List Symbol := [y,x]; R := Integer;

lp :=[p1,p2,p3]

      5     4     2         3      3    2      2      3          4    2
   [y  - x y  + (x  - x - 1)y  + (- x  + x  + x)y  + (- x  + x)y + x  - x ,
       4          3       2          2      2          3
    - y  + (2x - 1)y  + (- 3x  + 2x + 1)y  + (- 3x  + 1)y + 4x  - 2x,
       4      3      2           2      3     2          3
    5y  - 4x y  + (3x  - 3x - 3)y  + (- 2x  + 2x  + 2x)y - x  + x]

T := LXATSET(R,E,V,P);
gb := reverse groebner lp
       6     4     3     2       2            3    2
   [2x  - 5x  + 2x  + 2x  - x, (x  + x - 1)y - x  - x  + x,
     2     5     4     3
    y  - 4x  - 2x  + 8x  - 3x]
                         Time: 0.03 (IN) + 0.70 (EV) + 0.08 (OT) = 0.82 sec
lts := lexTriangular(gb)

             2          2         4     3    2
   (15)  [{x  + x - 1,y  - x},{2x  - 2x  - x  + x,y - x}]
```

```
                         Time: 0.02 (IN) + 0.40 (EV) + 0.07 (OT) = 0.48 sec

T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T

             2            2
   [[close= {x  + x - 1,x y  + x - 1},open= [x,6x - 5]],

     [
       close =
           6     4     3     2
         {2x  - 5x  + 2x  + 2x  - x,

                7      6     5    4      3     2                  8      7
             (16x  + 26x  + 6x  - x  - 22x  - 6x  + 5x + 2)y - 12x  - 16x
           +
                 6      5      4      3     2
             - 12x  - 20x  + 26x  + 18x  - 6x  - 4x
           }
       ,

       open =
                   3     2
         [6x - 5, 3x  + 5x  - 3x - 2,
              7      6     5    4      3     2
          16x  + 26x  + 6x  - x  - 22x  - 6x  + 5x + 2]
       ]
     ]
                                     Time: 2.32 (EV) + 0.18 (OT) = 2.50 sec

lps : List GPS := [[lexSaturate(ts.close,ts.open)$T]$GPS for ts in ltqas]

           2          2         4     3    2
   (19)  [{x  + x - 1,y  - x},{2x  - 2x  - x  + x,y - x}]
                         Time: 0.08 (IN) + 1.72 (EV) + 0.10 (OT) = 1.90 sec
-- lps is the lextriangular result

T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T

       3     2              2                       2
   [{2x  - 2x  - x + 1,(3448x  - 17x - 1711)y - 3431x  - 13x + 1724},
     2                     2
    {x  + x - 1,(426x - 265)y  + 691x - 426}, {x,y}]
                         Time: 0.02 (IN) + 2.42 (EV) + 0.07 (OT) = 2.50 sec

lps : List GPS := [[saturate(ts)$T]$GPS for ts in lts]

            3     2                   2          2
   (23)  [{2x  - 2x  - x + 1,y - x},{x  + x - 1,y  - x},{x,y}]
                         Time: 0.03 (IN) + 1.12 (EV) + 0.03 (OT) = 1.18 sec

-- Then, it is easy to check that lps
```

```
-- is correct for solving lp.

T := LazardTriangularSet(R,E,V,P)
lts := zeroSetSplit(lp)$T

            2          2         3
   (25)  [{x  + x - 1,y  - x},{2x  - x,y - x},{x - 1,y - 1}]
                       Time: 247.15 (EV) + 0.07 (OT) + 3.50 (GC) = 250.72 sec

-- Then, it is easy to check that lps
-- is correct for solving lp.

zeroSetSplit(gb)$T

            2          2          4     3    2
   (27)  [{x  + x - 1,y  - x},{2x  - 2x  - x  + x,y - x}]
                           Time: 0.02 (IN) + 0.10 (EV) + 0.03 (OT) = 0.15 sec


-----------------------------------------------------------------------------
--% Example f633 (et f744, f855) (from Jean-Charles FAUGERE)
-----------------------------------------------------------------------------
```

Voici un exemple tiré d'un problème réel en *compression d'image*, de dimension 2 sur le corps des complexes et où une application numérique était nécessaire. Seules nos implantations des méthodes de Wang et Lazard donnent une réponse sur cet exemple. Si celle de Wang conclut plus vite que celle de Lazard, elle donne un résultat qui n'est pas exploitable directement, du fait des polynômes apparaissant dans les inégalités.

Celle de Lazard a permis une exploitation numérique. qui était d'autant plus commode que, dans cette décomposition, seuls les polynômes univariés n'étaient pas de degré principal 1. La décomposition produite par Wang n'a pas cette propriété.

Remarquons aussi que, sur cet exemple, notre implantation de la méthode de Lazard conclut plus vite si on lui donne le système `lp` à traiter plutôt que sa base lexicographique.

Notons enfin que des systèmes plus compliqués, appelés respectivement f744 et f855, provenant encore du même problème de compression d'image, ont pu être résolus grce à notre implantation de la méthode de Lazard. Mais, cette fois-ci, il a fallu partir des bases lexicographiques. De plus, sur ces sytèmes, les trois autres méthodes ont échoué, y compris en partant des bases lexicographiques. Notre implantation de la méthode de Lazard a résolu les systèmes f744 et f855 à partir de leurs bases lexicographiques en 5 minutes et une heure respectivement. Les variétés associés sont de dimension 1.

```
ls : List Symbol := [U6,U5,U4,U3,U2,u6,u5,u4,u3,u2];
R := Integer
lp := [p1,p2,p3,p4,p6,p7,p8,p9,p10]

   [2u6 + 2u5 + 2u4 + 2u3 + 2u2 + 1,

        (8u6 + 8u5)U5 + (8u6 + 8u5 + 8u4)U4 + (8u6 + 8u5 + 8u4 + 8u3)U3
      +
        (8u6 + 8u5 + 8u4 + 8u3 + 8u2)U2 - 13
      ,
    2U6 + 2U5 + 2U4 + 2U3 + 2U2 + 1,

        (8u5 + 8u4 + 8u3 + 8u2)U6 + (8u5 + 8u4 + 8u3 + 8u2)U5
      +
        (8u4 + 8u3 + 8u2)U4 + (8u3 + 8u2)U3 + 8u2 U2 - 13
      ,
    u2 U2 - 1, u3 U3 - 1, u4 U4 - 1, u5 U5 - 1, u6 U6 - 1]

lg := groebner lp;

T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T

   [
     [
       close =
             2
         {4u2  + 13u2 + 4, (8u2 + 10)u3 - 28u2 - 11, 2u4 + 2u3 + 2u2 + 1,
          u6 + u5, u2 U2 - 1, 8u3 U3 + (8u3 + 8u2)U2 + 4u3 + 4u2 - 11,
          2U4 + 2U3 + 2U2 + 1, 2u5 U5 + 2u5 U4 + 2u5 U3 + 2u5 U2 + u5 - 2,
          2U6 + 2U5 + 2U4 + 2U3 + 2U2 + 1}
        ,
      open= [u2,4u2 + 5,28u2 + 11,u5]]
     ,

     [
       close =
             2
         {4u2  + 13u2 + 4,
               3       2                         4       3       2
          (16u2  - 26u2  + 32u2 + 32)u3 + 40u2  - 26u2  + 17u2  + 20u2,

                                 2                  2       2
            (((13u2 + 4)u3 + 8u2  + 4u2)u4 + 8u2 u3  + (8u2  + 4u2)u3)u5
          +
                      2           2       2
            8u2 u3 u4  + (8u2 u3  + (8u2  + 4u2)u3)u4
          ,
          2u6 + 2u5 + 2u4 + 2u3 + 2u2 + 1, u2 U2 - 1, u3 U3 - 1, u4 U4 - 1,
          (8u4 + 8u3 + 8u2 + 4)U5 + (8u3 + 8u2 + 4)U4 + (8u2 + 4)U3 + 4U2 + 13,
          2U6 + 2U5 + 2U4 + 2U3 + 2U2 + 1}
       ,

       open =
         [u2, u2 + 2, 2u2 + 1, 49u2 + 20, u4, (196u2 + 80)u4 + 557u2 + 196,
          (25828u2 + 8912)u4 + 75029u2 + 25828]
       ]
     ,

     [
       close =
         {(u2 + 2)u3 + 2u2,
```

```
                2                 2         3         2                 4
          (52u2  + 120u2 + 32)u3  + (84u2  + 201u2  + 84u2)u3 + 32u2
        +
                3       2
          120u2  + 52u2
       *
        u4
     +
            2           3         3         2           2
      (32u2  + 64u2)u3  + (64u2  + 120u2  + 32u2)u3
     +
            4       3       2
      (32u2  + 56u2  + 20u2 )u3
     ,

                           2                  2       2
      (((13u2 + 4)u3 + 8u2  + 4u2)u4 + 8u2 u3  + (8u2  + 4u2)u3)u5
     +
                2           2       2
      8u2 u3 u4  + (8u2 u3  + (8u2  + 4u2)u3)u4
     ,
    2u6 + 2u5 + 2u4 + 2u3 + 2u2 + 1, u2 U2 - 1, u3 U3 - 1, u4 U4 - 1,
    (8u4 + 8u3 + 8u2 + 4)U5 + (8u3 + 8u2 + 4)U4 + (8u2 + 4)U3 + 4U2 + 13,
    2U6 + 2U5 + 2U4 + 2U3 + 2U2 + 1}
  ,
                                               2               2
 open= [u2,u2 + 2,4u2 - 3,4u2 + 5,2u2  + u2 + 2,16u2  + 8u2 + 49]]
,

[
 close =
   {
                2                2         3         2                 3
          (20u2  + 56u2 + 32)u3  + (32u2  + 120u2  + 64u2)u3 + 64u2
        +
               2
          32u2
       *
          2
        u4
     +
                2                 3         3         2           2
          (52u2  + 120u2 + 32)u3  + (84u2  + 201u2  + 84u2)u3
        +
                4         3       2
          (32u2  + 120u2  + 52u2 )u3
       *
        u4
     +
            2           4         3         2           3
      (32u2  + 64u2)u3  + (64u2  + 120u2  + 32u2)u3
     +
            4       3       2   2
      (32u2  + 56u2  + 20u2 )u3
     ,

                           2                  2       2
      (((13u2 + 4)u3 + 8u2  + 4u2)u4 + 8u2 u3  + (8u2  + 4u2)u3)u5
     +
                2           2       2
      8u2 u3 u4  + (8u2 u3  + (8u2  + 4u2)u3)u4
     ,
    2u6 + 2u5 + 2u4 + 2u3 + 2u2 + 1, u2 U2 - 1, u3 U3 - 1, u4 U4 - 1,
    (8u4 + 8u3 + 8u2 + 4)U5 + (8u3 + 8u2 + 4)U4 + (8u2 + 4)U3 + 4U2 + 13,
    2U6 + 2U5 + 2U4 + 2U3 + 2U2 + 1}
  ,
```

```
      open =
        [u2, u3,
             2              2       3       2                 3      2
         (5u2  + 14u2 + 8)u3  + (8u2  + 30u2  + 16u2)u3 + 16u2  + 8u2 , u4,
         2u4 + 2u3 + 2u2 + 1,
                            2                    2       2
         ((13u2 + 4)u3 + 8u2  + 4u2)u4 + 8u2 u3  + (8u2  + 4u2)u3]
      ]
    ]
                          Time: 25.32 (EV) + 0.80 (OT) + 6.38 (GC) = 32.50 sec
```

```
T := LazardTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T
```

```
   [
     {
             2               2        3        2                 3       2   2
        ((20u2  + 56u2 + 32)u3  + (32u2  + 120u2  + 64u2)u3 + 64u2  + 32u2 )u4
      +
               2                 3        3        2          2
           (52u2  + 120u2 + 32)u3  + (84u2  + 201u2  + 84u2)u3
         +
               4        3       2
           (32u2  + 120u2  + 52u2 )u3
       *
          u4
      +
             2          4        3        2          3
        (32u2  + 64u2)u3  + (64u2  + 120u2  + 32u2)u3
      +
             4       3       2   2
        (32u2  + 56u2  + 20u2 )u3
      ,

                     2       2
        ((8u2 + 16)u3  + (8u2  + 10u2)u3)u5
      +
                     2       2                     2                         3
        ((8u2 + 16)u3  + (8u2  + 20u2 + 8)u3 + 16u2  + 8u2)u4 + (8u2 + 16)u3
      +
             2              2       3       2
        (16u2  + 30u2 + 8)u3  + (8u2  + 14u2  + 5u2)u3
      ,
                2       2                                  2
     ((4u2 + 8)u3  + (4u2  + 5u2)u3)u6 + ((- 5u2 - 4)u3 - 8u2  - 4u2)u4,
     u2 U2 - 1, u3 U3 - 1,

               2          4        3        2          3
           (32u2  + 64u2)u3  + (64u2  + 120u2  + 32u2)u3
         +
               4       3       2   2
           (32u2  + 56u2  + 20u2 )u3
       *
          U4
      +
             2               2        3        2                 3       2
        ((20u2  + 56u2 + 32)u3  + (32u2  + 120u2  + 64u2)u3 + 64u2  + 32u2 )u4
      +
             2               3        3        2          2
        (52u2  + 120u2 + 32)u3  + (84u2  + 201u2  + 84u2)u3
      +
             4        3       2
        (32u2  + 120u2  + 52u2 )u3
      ,
```

```
             3        2          5         4        3        2          4
        (40u2  + 112u2  + 64u2)u3  + (144u2  + 374u2  + 224u2  + 32u2)u3
      +
              5        4        3       2   3
        (168u2  + 406u2  + 265u2  + 52u2 )u3
      +
             6        5       4       3   2
        (64u2  + 144u2  + 96u2  + 20u2 )u3
   *
      U5
  +
               3        2                4
        (- 20u2  - 96u2  - 144u2 - 64)u3
      +
               4        3        2                  3
        (- 52u2  - 290u2  - 496u2  - 264u2 - 32)u3
      +
               5        4        3        2          2
        (- 32u2  - 304u2  - 688u2  - 464u2  - 96u2)u3
      +
                5        4        3       2           5        4       3
        (- 128u2  - 464u2  - 392u2  - 96u2 )u3 - 128u2  - 128u2  - 32u2
   *
      u4
  +
           3        2           5          4        3        2          4
    (- 32u2  - 128u2  - 128u2)u3  + (- 64u2  - 248u2  - 272u2  - 64u2)u3
  +
           5        4        3        2          3
    (- 32u2  - 184u2  - 342u2  - 184u2  - 32u2)u3
  +
           5        4        3       2   2           5        4       3
    (- 64u2  - 272u2  - 248u2  - 64u2 )u3  + (- 128u2  - 128u2  - 32u2 )u3
  ,

             2          3         3       2          2
        (40u2  + 32u2)u3  + (104u2  + 84u2  + 16u2)u3
      +
             4       3       2
        (64u2  + 64u2  + 16u2 )u3
   *
      U6
  +
          2               2        3        2                 3       2
    ((20u2  + 56u2 + 32)u3  + (32u2  + 120u2  + 64u2)u3 + 64u2  + 32u2 )u4
  +
         2               3        3        2          2
    (52u2  + 120u2 + 32)u3  + (84u2  + 201u2  + 84u2)u3
  +
         4        3       2
    (32u2  + 120u2  + 52u2 )u3
  }
,
{2u3 + 2u2 + 1, (24u2  - 20u2  + 8u2 - 32)u4 + 98u2  + 65u2  + 40u2 + 16,
      4     3       2                       4        3       2
 (30u2  - u2  - 10u2  - 32u2 - 32)u5 - 196u2  - 130u2  - 80u2  - 32u2,
      3       2                       3       2
 (40u2  + 52u2  + 56u2 + 32)u6 + 98u2  + 65u2  + 40u2 + 16, u2 U2 - 1,
 (2u2 + 1)U3 + 2,
      3       2                       3       2
 (98u2  + 65u2  + 40u2 + 16)U4 + 24u2  - 20u2  + 8u2 - 32,
       4        3       2                  4     3       2
 (196u2  + 130u2  + 80u2  + 32u2)U5 - 30u2  + u2  + 10u2  + 32u2 + 32,
      3       2                       3       2
 (98u2  + 65u2  + 40u2 + 16)U6 + 40u2  + 52u2  + 56u2 + 32}
,
```

```
{(u2 + 2)u3 + 2u2,
      3       2                    3      2
 (16u2  + 40u2  + 65u2 + 98)u4 - 32u2  + 8u2  - 20u2 + 24,
      3       2                     4      3      2
 (32u2  + 80u2  + 130u2 + 196)u5 + 32u2  + 32u2  + 10u2  + u2 - 30,
      3       2                    3      2
 (16u2  + 40u2  + 65u2 + 98)u6 + 32u2  + 56u2  + 52u2 + 40, u2 U2 - 1,
 2u2 U3 + u2 + 2,
      3      2                     3      2
 (32u2  - 8u2  + 20u2 - 24)U4 - 16u2  - 40u2  - 65u2 - 98,
      4       3      2                    3      2
 (32u2  + 32u2  + 10u2  + u2 - 30)U5 + 32u2  + 80u2  + 130u2 + 196,
      3       2                    3      2
 (32u2  + 56u2  + 52u2 + 40)U6 + 16u2  + 40u2  + 65u2 + 98}
,

    2
{4u2  + 13u2 + 4, 2u3 + 4u2 + 1, u5 + u4,
 (4u4 - 4u2)u6 - 4u2 u4 - 13u2 - 4, 4U2 + 4u2 + 13, U3 - 2u2 - 6,
 u4 U4 - 1, u4 U5 + 1, (4u4 - 4u2)U6 + (4u2 + 13)u4 + 4}
,

    2
{4u2  + 13u2 + 4, 2u3 + 4u2 + 1, u4 - u2, u6 + u5, 4U2 + 4u2 + 13,
 U3 - 2u2 - 6, 4U4 + 4u2 + 13, u5 U5 - 1, u5 U6 + 1}
]
                    Time: 79.30 (EV) + 1.17 (OT) + 29.28 (GC) = 109.75 sec

zeroSetSplit(lg)$T;

          Time: 0.02 (IN) + 87.63 (EV) + 1.23 (OT) + 64.65 (GC) = 153.53 sec

--------------------------------------------------------------------------------
--% Basic operations tests for domain constructors of category TSETCAT
--% ROBOT ROMIN (Tomas Recio, Santander)
--------------------------------------------------------------------------------
```

Voici un autre problème reél (bras manipilateur, en robotique). Ici, seule notre implantation de la méthode de Lazard ne donne pas de décomposition à partir du système initial `lp`. Cependant, elle en donne une à partir de la base lexicographique.

Les implantations de Wang, Wu et Lazard donnent des solutions trop *encombrantes* pour être reproduites ici. Nous ne donnons donc que les temps pour nos quatre méthodes. De plus. les décompositions produites sont trop compliquées pour notre *moulinette de vérification habituelle*.

Malgré tout, grce aux outils propres à la méthode de Lazard, nous pouvons tirer des informations sur la décomposition produite, comme le montre les calculs ci-après.

```
ls : List Symbol := [s1,c1,s2,c2,s3,c3,l2,l3,a,b,c,d];
R := Integer;
lp := [f1,f2,f3,f4,f5,f6,f7]

                                                                 2     2
   [- d s1 - a, d c1 - b, l2 c2 + l3 c3 - d, 2s2 + l3 s3 - c, s1  + c1  - 1,
      2     2         2     2
    s2  + c2  - 1, s3  + c3  - 1]

lg := groebner lp;
```

```
              Time: 0.02 (IN) + 1.27 (EV) + 0.50 (OT) + 1.67 (GC) = 3.45 sec

T := WangTriangularSet(R,E,V,P);
ltqas := zeroSetSplitIntoTriangularSystems(lp)$T;

                        Time: 19.92 (EV) + 1.20 (OT) + 6.53 (GC) = 27.65 sec

T := WuWenTsunTriangularSet(R,E,V,P);
lts := zeroSetSplit(lp)$T;

             Time: 0.02 (IN) + 14.07 (EV) + 1.45 (OT) + 6.53 (GC) = 22.07 sec

T := TableRegularChain(R,E,V,P);
lts := zeroSetSplit(lp)$T

   [
      2    2    2
     {a  + b  - d ,

          4  4      4  2       4   4          3  2         3   3
        (l3 l2  - 8l3 l2  + 16l3 )c3  + (16d l3 l2  - 64d l3 )c3
      +
                 4      2       2   4       4      2     2        2   2
          (- 2l3  + (2c  + 8)l3 )l2  + (8l3  + (8c  - 8d  - 32)l3 )l2
        +
               2  2
          96d l3
       *
            2
          c3
      +
                 3          2            2      3
        ((- 16d l3  + (- 16d c  + 64d)l3)l2  - 64d l3)c3
      +
          4        2       2    4     2        4      2  2     2 2      2   2
        (l3  + (- 2c  - 8)l3  + c  - 8c  + 16)l2  + (8d l3  + 8d c  - 32d )l2
      +
          4
        16d
      ,
           2        2  2      2   2                    2    2        2     2
      2c l3 l2 s3 + (l3 l2  - 4l3 )c3  + 8d l3 c3 + (- l3  - c  + 4)l2  - 4d ,
      l2 c2 + l3 c3 - d, 2s2 + l3 s3 - c, d c1 - b, d s1 + a}
     ]
                       Time: 60.00 (EV) + 0.22 (OT) + 20.10 (GC) = 80.32 sec

T := LazardTriangularSet(R,E,V,P)
lts := zeroSetSplit(lg)$T ;

                      Time: 0.02 (IN) + 68.43 (EV) + 57.00 (GC) = 125.45 sec

[dimension(ts)$T for ts in lts]
```

```
         [5,4,4,4,4,4,4,3,3,3,3,3,3,3,3,3,3,3,2,2]

ts1 := first lts

     2    2    2
   {a  + b  - d ,

          4  4      4  2       4   4          3  2         3   3
       (l3 l2  - 8l3 l2  + 16l3 )c3  + (16d l3 l2  - 64d l3 )c3
     +
                  4      2       2   4       4      2     2        2   2
             (- 2l3  + (2c  + 8)l3 )l2  + (8l3  + (8c  - 8d  - 32)l3 )l2
           +
                2  2
             96d l3
      *
          2
        c3
     +
                3           2             2      3
       ((- 16d l3  + (- 16d c  + 64d)l3)l2  - 64d l3)c3
     +
          4        2       2    4     2        4      2  2     2 2      2   2
       (l3  + (- 2c  - 8)l3  + c  - 8c  + 16)l2  + (8d l3  + 8d c  - 32d )l2
     +
          4
       16d
     ,
          2        2  2      2   2                   2    2       2     2
    2c l3 l2 s3 + (l3 l2  - 4l3 )c3  + 8d l3 c3 + (- l3  - c  + 4)l2  - 4d ,
    l2 c2 + l3 c3 - d,
        2          2  2      2   2                  2    2       2     2
    4c l2 s2 + (- l3 l2  + 4l3 )c3  - 8d l3 c3 + (l3  - c  - 4)l2  + 4d ,
    d c1 - b, d s1 + a}

lts := rest lts;

[containedInClosure?(us,ts1)$T for us in lts]

   [true, false, true, true, false, true, false, false, false, false, true,
    true, true, false, false, true, false, false, false]

lts := [us for us in lts | not containedInClosure?(us,ts)$T];


ts2 := first lts

   {d, b, a,

          4  4      4  2       4   4
       (l3 l2  - 8l3 l2  + 16l3 )c3
     +
```

```
             4      2       2   4       4      2        2   2   2
      ((- 2l3  + (2c  + 8)l3 )l2  + (8l3  + (8c  - 32)l3 )l2 )c3
     +
         4        2       2    4     2        4
      (l3  + (- 2c  - 8)l3  + c  - 8c  + 16)l2
     ,
          2         2  2      2   2        2    2       2
    2c l3 l2 s3 + (l3 l2  - 4l3 )c3  + (- l3  - c  + 4)l2 , l2 c2 + l3 c3,
       2          2  2      2   2      2    2       2    2     2
    4c l2 s2 + (- l3 l2  + 4l3 )c3  + (l3  - c  - 4)l2 , s1  + c1  - 1}
                                   Time: 0.02 (IN) + 0.15 (OT) = 0.17 sec

lts := rest lts;
[containedInClosure?(us,ts2)$T for us in lts]

   [false,true,true,true,true,false,false,false,false,false]

lts := [us for us in lts | not containedInClosure?(us,ts2)$T];
ts3 := first lts

    2    2    2        2       2     2    2     2               2
   {a  + b  - d , l3, (c  - 4)l2  + 4d , s3  + c3  - 1, 4d c2 + (c  - 4)l2,
    2s2 - c, d c1 - b, d s1 + a}

lts := rest lts;
[containedInClosure?(us,ts3)$T for us in lts]

   [false,false,false,false,false]

ts4 := first lts
                2     2        2    2              2     2
   {d,b,a,l3,l2,s3  + c3  - 1,4c2  + c  - 4,2s2 - c,s1  + c1  - 1}

lts := rest lts;
[containedInClosure?(us,ts4)$T for us in lts]

    [false,false,false,false]

ts5 := first lts

                   2     2                 2     2
   {d,c + 2,b,a,l3,s3  + c3  - 1,c2,s2 + 1,s1  + c1  - 1}

lts := rest lts;
[containedInClosure?(us,ts5)$T for us in lts]

   [false,false,false]

ts6 := first lts
                   2     2                 2     2
   {d,c - 2,b,a,l3,s3  + c3  - 1,c2,s2 - 1,s1  + c1  - 1}

lts := rest lts
```

```
            2             2     2                               2     2
   [{d,c,b,a,l3  - 4,l2 + 2,s3  + c3  - 1,2c2 - l3 c3,2s2 + l3 s3,s1  + c1  - 1}
     ,
            2             2     2                               2     2
    {d,c,b,a,l3  - 4,l2 - 2,s3  + c3  - 1,2c2 + l3 c3,2s2 + l3 s3,s1  + c1  - 1}
     ]
newlts := [ts,ts2,ts3,ts4,ts5,lts.1,lts.2];

-----------------------------------------------------------------------------
--% Neural network (see Kalkbrener thesis and Gao-Chou)
-----------------------------------------------------------------------------
```

Nous terminons par un exemple de dimension 1 *qui n'a l'air de rien*. Aucune des implantations de nos quatre méthodes n'en vient à bout. Sa base lexicographoque nécessite du reste 18 secondes de GB. A partir de cette base, notre implantation de la méthode de Lazard produit la *jolie* décomposition suivante.

```
ls : List Symbol := [x,y,z,a]
R := Integer


lp := [p1,p2,p3]


           2      2             2     2            2    2
   (10)  [z x  + z y  - a z + 1,y x  + (z  - a)y + 1,(y  + z  - a)x + 1]

gb := reverse lexBase([lp]$GPS);

    GB time: 18.433332
                                     Time: 0.03 (IN) + 3.95 (EV) = 3.98 sec

T := LazardTriangularSet(R,E,V,P)
lts := zeroSetSplit(gb)$T

   [
       7      5    4     2 3      2      3         2
     {2z  - 5a z  - z  + 4a z  + 2a z  + (- a  - 1)z - a ,
         2      6      4    3   2 2                2    2
      a y  + (2z  - 3a z  - z  + a z  + a z - 1)y + a z  - a ,
                  6      4    3    2 2
      a x + a y + 2z  - 3a z  - z  + a z  + a z - 1}
     ,

         7     6     2 5     3      3    2 2
     {4a z  - 4z  - 4a z  + (a  - 4)z  + a z  - a z - 1,
             2 6      5      3      4     2 3      4      2        2
      4y - 4a z  + 8a z  + (4a  - 4)z  - 4a z  + (- a  + 6a)z  - 6z + a ,
             2 6      5      3      4     2 3      4      2        2
      4x - 4a z  + 8a z  + (4a  - 4)z  - 4a z  + (- a  + 6a)z  - 6z + a }
     ,

      6     3              5      2     3     4
     {a  - 54a  + 54, 90z - a  + 42a , 30y  + (a  - 72a)y - 30,
```

```
             4        2           5       2
  45x + (- a  + 72a)y  - 45y + 2a  - 114a }
 ,
  3             2  3                    2          2
{2a  - 27,9z + 2a ,y  - a y - 1,9x + 6a y  - 9y - 4a },
    3                        3      2          2
{a,2z  + 1,y - z,x - z}, {a,z  - 1,y  + z y + z ,x + y + z}]
                  Time: 102.60 (EV) + 0.28 (OT) + 33.35 (GC) = 136.23 sec
```

7 Conclusions

Que peut-on dire tout d'abord de la façon dont nous avons mené nos expériences ? Nos quatre méthodes ont été implantées dans un cadre tel qu'elles utilisent les mêmes outils de base : polynômes, opérations de réduction, ensembles triangulaires. Ce point était crucial car on sait que dans ces méthodes on passe une majeure partie du temps à réduire des polynômes. De plus, elles ont été testées sur de nombreux exemples (qui n'ont pas tous été rapportés ici) et les calculs ont été vérifiés dans la plupart des cas. Pour chacune des méthodes autres que celle de Lazard, les optimisations les plus fréquemment préconisées ont été implantées. Le cas de la méthode de Lazard est un peu à part car elle n'avait pas encore été implantée. De plus nous pensons en avoir une implantation correcte seulement depuis quatre mois. Il reste donc beaucoup à faire au niveau des optimisations. Dans ce qui précède, l'exemple appelé `curvel` et qui nous a été communiqué par Dominique Duval (points singuliers d'une courbe) est donné en 240 sec. En passant cet exemple *à la moulinette* dans le chapitre 3, nous l'avons résolu 100 plus rapidement en échangeant l'ordre de deux opérations. De même, l'exemple appelé `Liu et Lorenz` était hors de portée (même en une journée) lors des tests comparatifs. Pour le résoudre, il nous fallait partir de la base lexicographique et utiliser la factorisation. En rédigeant le même chapitre 3, nous avons pu le résoudre (certes en 10 heures) sans utiliser aucun de ces artifices.

Passons aux enseignements que l'on peut tirer de cette comparaison. Tout d'abord rappelons que les décompositions produites ne sont pas exactement de même nature.

- Pour la méthode de Wang, on obtient des systèmes triangulaires et non pas des ensembles. Ceci rend parfois difficile la lecture d'une décomposition (s'il y a beaucoup d'inéquations). Il peut y avoir des systèmes inconsistants, mais cela est rare
- Pour la méthode de Wu, on obtient des ensembles triangulaires dont on doit lire les zéros réguliers, ce qui est plutôt agréable. Cependant, sur des exemples *un peu gros*, les décompositions s'allongent rapidement. De plus, certaines quasi-composantes peuvent être inconsistantes
- Pour la méthode de Kalkbrener, les décompositions sont très souvent lisibles, même sur des gros exemples. De plus chaque ensemble triangulaire donne une quasi-composante consistante. Cependant, chacun des zéros de la variété affine de départ n'est pas nécessairement un zéro régulier de l'un des ensembles triangulaires produits. Chacune des quasi-composante doit être remplacée par sa fermeture pour récupérer toute la variété. De plus les décompositions peuvent être redondantes
- Pour la méthode de Lazard, on ne retrouve aucun des inconvénients des méthodes précédentes. En effet, les décompositions ne s'allongent pas autant que chez Wu. En outre, il suffit de lire les zéros réguliers et chaque ensemble triangulaire donne une quasi-composante consistante. De plus ces quasi-composantes sont incomparables pour l'inclusion et la normalisation facilite l'exploitation des résultats. Elle permet du reste de vérifier si une quasi-composante obtenue est contenue dans la variété de départ.

Sur un exemple donné, notons que les décompositions de Wang et Wu sont souvent semblables. D'autre part les méthodes de Kalkbrener et Lazard, qui sont mieux adaptées à la découverte de scindages de deuxième espèce (voir proposition II.20 p. 62), s'en sortent globalement mieux. Notons que celle de Kalkbrener est plus performante dans les cas où F engendre un idéal premier de *grosse* dimension (`Robot Romin`) tandis que celle de Lazard prend l'avantage dans la plupart des autres cas. En effet, cette dernière *scinde* plus que celle de Kalkbrener du fait qu'elle résout au sens des zéros réguliers et du fait qu'elle travaille explicitement modulo l'ensemble triangulaire courant. Ce dernier point facilite la découverte de factorisation de polynômes univariés à coefficients dans une tour d'extensions.

Passons aux temps de calcul. Sur tel ou tel *petit exemple* chaque méthode est à son tour la plus efficace. Nous retiendrons qu'en général sur les moyens ou gros exemples, surtout en dimension positive, c'est la méthode de Lazard qui s'en sort le mieux (exemples Caprasse, cyclic-5, f633, neural, Liu et Lorenz). Et même si sur certains moyens ou gros exemples, une autre méthode est plus rapide que celle de Lazard, la décomposition la plus exploitable est celle obtenue par la méthode de Lazard appliquée à la base lexicographique. Certes, il lui reste des progrès à faire sur certains exemples en particulier, ceux du type *shape lemma* où le fait d'obtenir des ensembles normalisés se paye. Enfin, quand chacune de ces méthodes échoue, la seule façon d'obtenir un résultat est souvent de lancer la méthode de Lazard sur la base lexicographique.

En conclusion, il semble que l'on ait intérêt dans la majorité des cas à procéder comme suit : lancer la méthode de Lazard, et si un résultat n'est pas obtenu rapidement, recommencer à partir de la base lexicographique.

Index

Bibliographie

[ALM97] P. Aubry, D. Lazard, and M. Moreno Maza. On the theories of triangular sets. *submitted to J. Symb. Comp.*, 1997.

[AM97a] P. Aubry and M. Moreno Maza. Triangular sets for solving polynomial systems : A comparison of four methods. Technical Report LIP6/009, LIP6, Université Paris 6, Paris, 1997.

[AM97b] P. Aubry and M. Moreno Maza. Triangular sets for solving polynomial systems : a comparative implementation of four methods. *Presented at CoCoA V and submitted to J. Symb. Comp.*, 1997.

[Aug93] Daniel Augot. *Étude algébrique des mots de poids minimum des codes cycliques, méthodes d'algèbre linéaire sur les corps finis.* PhD thesis, Université Paris 6, 1993. Thèse de Doctorat.

[BF91] J. Backelin and R. Frberg. How we proved that there are exactly 924 cyclic 7-roots. In S. M. Watt, editor, *Proc. ISSAC'91*, pages 103–111. ACM, 1991.

[BGK86] W. Boege, R. Gebauer, and H. Kredel. Some examples for solving systems of algebraic equations by calculating groebner bases. *J. Symb. Comp.*, 2 :83–98, 1986.

[Buc65] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal.* PhD thesis, Innsbruck, 1965.

[CG90a] S.C. Chou and X.S. Gao. Methods for mechanical geometry formula deriving. In *Proc. ISAAC'90*, pages 265–270, Tokyo, Japan, 1990.

[CG90b] S.C. Chou and X.S. Gao. On the parametrization of algebraic curves. Technical Report TR-90-18, Univ. of Texas, Dept. of Comp. Sci., Austin, Texas, USA, 1990.

[CG90c] S.C. Chou and X.S. Gao. Ritt-Wu's decomposition algorithm and geometry theorem proving. In *Proc. CADE-10*, pages 202–220, Kaiserslautern, Germany, 1990.

[CG91a] S.C. Chou and X.S. Gao. Computations with parametric equations. In *Proc. ISAAC'91*, pages 122–127, Bonn, Germany, 1991.

[CG91b] S.C. Chou and X.S. Gao. On the dimension of an arbitrary ascending chain. *Chinese Bull. of Sci.*, 1991. to appear.

[CG92] S.C. Chou and X.S. Gao. Solving parametric algebraic systems. In *Proc. ISAAC'92*, pages 335–341, Berkeley, California, 1992.

[CG93] S.C. Chou and X.S. Gao. A zero structure theorem for differential parametric systems. *J. Symb. Comp.*, 1993. submitted.

[Cho88] S.C. Chou. *Mechanical Geometry Theorem Proving.* D. Reidel Publ. Comp., Dordrecht, 1988.

[Cho90] S.C. Chou. Automated reasoning in geometries using the characteristic set method and Gröbner basis method. In *Proc. ISSAC'90*, pages 255–260, Tokyo, Japan, 1990.

[CLO91] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms.* Spinger-Verlag, 1991.

[DDD85] J. Della Dora, C. Discrescenzo, and D. Duval. About a new method method for computing in algebraic number fields. In *Proc. EUROCAL 85 Vol. 2*, volume 204 of *Lect. Notes in Comp. Sci.*, pages 289–290. Springer-Verlag, 1985.

[DR94a] D. Duval and J.C. Reynaud. Sketches and computation -i : basic definitions and static evaluation. In *Math. Strut. in Comp. Science*, volume 4, pages 185–238. Cambridge University Press, 1994.

[DR94b] D. Duval and J.C. Reynaud. Sketches and computation -ii : dynamic evaluation and applications. In *Math. Strut. in Comp. Science*, volume 4, pages 239–271. Cambridge University Press, 1994.

[DST93] J. Davenport, Y. Siret, and E. Tournier. *Calcul Formel.* Masson, 1993. 2^e édition révisée.

[Duv87] D. Duval. *Questions Relatives au Calcul Formel avec des Nombres Algbriques.* Universit de Grenoble, 1987. Thèse d'État.

[Duv94] D. Duval. Algebraic Numbers : an Example of Dynamic Evaluation. *J. Symb. Comp.*, 18(5) :429–446, November 1994.

[Eur92] European Commission. *PoSSo - Polynomial System Solving Research Project.* Esprit Scheme Project No. 6846, 1992. Has been extended to the FRISCO project.

[Eur96] European Commission. *FRISCO - A Framework for Integrated Symbolic/Numeric Computation.* Esprit Scheme Project No. 21 024, 1996. European research project to investigate and develop technologies which can be used to deliver highly efficient versatile polynomial solvers to industrial users.

[Fau94] J.C. Faugère. *Résolution des systèmes d'équations algébriques.* Université Paris 6, 1994. Thèse de Doctorat.

[FMR96] J.C. Faugère, F. Moreau de Saint Martin, and F. Rouiller. Design of nonseparable bidimensional wavelets and filter banks using Gröbner bases techniques. *IEEE Trans. in Signal Processing*, 1996. preprint.

[Gia87] P. Gianni. Properties of Gröbner basis under specializations. *Lecture Notes in Computer Science*, 378 :293–297, 1987.

[GM90] G. Gallo and B. Mishra. Efficient algorithms and bounds for Wu-Ritt characteristic sets. In *Proc. MEGA'90*, pages 119–142, 1990.

[GM91] G. Gallo and B. Mishra. Wu-Ritt characteristic sets and their complexity. In J.E. Goodman, R. Pollack and W. Steiger, editor, *Discrete and Computational*

Geometry : Papers from the DIMACS Special Year, volume 6 of *Dimacs Series in Discrete Mathematics and Theoretical Computer Science*, pages 111–136. American Mathematical Society and Association for Computing Machinery, 1991.

[GM92] G. Gallo and B. Mishra. Recent progress in characteristic set computation : Complexity and open problems. In Wu Wen-Tsün and Cheng Min-De, editors, *Proc. of the Int. Workshop on Math. Mechanisation*, pages 28–37, Beijing, China, 1992. Institute of Systems Science, Academia Sinica.

[Gom92] Teresa Gomez Diaz. *Quelques applications de l'évaluation dynamique.* Université de Limoges, 1992. Thèse de Doctorat.

[GTZ88] P. Gianni, B. Trager, and G. Zacharias. Gröbner Bases and Primary Decomposition Of Polynomial Ideals. *J. Symb. Comp.*, 6 :149–167, 1988.

[Hea79] A.C. Hearn. Non-modular computation of polynomial gcds using trial division. In Springer, editor, *Proc. EUROSAM'79, Marseille*, volume LNCS 72, pages 227–239, 1979.

[JS92] Richard D. Jenks and Robert S. Stutor. *AXIOM, The Scientific Computation System.* Springer-Verlag, 1992. AXIOM is a trade mark of NAG Ltd, Oxford UK.

[Kal87] M. Kalkbrener. Solving systems of algebraic equations by using Gröbner basis. *Lecture Notes in Computer Science*, 378 :282–292, 1987.

[Kal91] M. Kalkbrener. *Three contributions to elimination theory.* PhD thesis, Johannes Kepler University, Linz, 1991.

[Kal93] M. Kalkbrener. A generalized euclidean algorithm for computing triangular representations of algebraic varieties. *J. Symb. Comp.*, 15 :143–167, 1993.

[Kal95] M. Kalkbrener. *Algorithmic properties of polynomial rings.* Dep. of math., Swiss Federal Institute of Technology, Zurich, 1995. Habilitation Thesis.

[KH85] H.P. Ko and M.A. Hussain. A study of Wu's method – a method to prove certain theorems in elementary geometry. In *Proc. of 1985 Congressus Numerantium*, 1985.

[Ko88] H.P. Ko. Geometry theorem proving by decomposition of quasi-algebraic sets : an application of the Ritt-Wu principle. *Artificial Intelligence*, 37 :95–122, 1988.

[KV95] J. Kovacevic and M. Vetterli. "Nonseparable two- and three-dimensional wavelets". *IEEE Transactions on Signal Processing*, 43(5) :1269–1272, May 1995.

[KW90] D. Kapur and H.K. Wan. Refutational proofs of geometry theorems via characteristic set computation. In *Proc. ISSAC'90*, pages 277–284, Tokyo, Japan, 1990.

[Laf74] J-P. Lafon. *Les formalismes fondamentaux de l'algèbre commutative.* Hermann, Paris, 1974.

[Laz91a] D. Lazard. A new method for solving algebraic systems of positive dimension. *Discr. App. Math*, 33 :147–160, 1991.

[Laz91b] D. Lazard. Systems of algebraic equations (algorithms and complexity). In Eisenbud and Robbiano, editors, *Cortona proceedings.* Cambridge University Press, 1991.

[Laz92a] D. Lazard. Solving zero-dimensional algebraic systems. *J. Symb. Comp.*, 15 :117–132, 1992.

[Laz92b] D. Lazard. Terminology for triangular and characteristic sets. Technical report, Université Paris 6, 1992. PoSSo report.

[Liu89] Z.J. Liu. An algorithm on finding all isolated zeros of polynomial equations. *MM Research Preprints*, 4(63-76), 1989.

[Loo82] R. Loos. Generalized polynomial remainder sequences. In *Symbolic and Algebraic Computation*, pages 115–137. Spinger-Verlag, 1982.

[Mor96] M. Moreno Maza. Notes de cours de calcul formel et de programmation en AXIOM-XL. Université Paris 6, 1996.

[Mor97a] M. Moreno Maza. An implementation of the lextriangular algorithm in AXIOM. Technical report, NAG Ltd, Oxford, UK, 1997. FRISCO task 3.3.2.5.

[MR95] M. Moreno Maza and R. Rioboo. Polynomial gcd computations over towers of algebraic extensions. In *Proc. AAECC-11*, pages 365–382. Springer, 1995.

[Reb97] François Rebufat. *Placement de Polygones Généralisés dans le Plan.* PhD thesis, Université Paris 6, 1997. Thèse de Doctorat.

[Rio91] R. Rioboo. *Quelques aspects du calcul exact avec les nombres rels.* Université Paris 6, 1991. Thèse de Doctorat.

[Rio92] R. Rioboo. Real algebraic closure of an ordered field, implementation in axiom. In *Proc. ISSAC'92*, pages 206–215. ISSAC, ACM Press, 1992.

[Rit32] J.F. Ritt. *Differential equations from an algebraic standpoint*, volume 14. American Mathematical Society, New York, 1932.

[Rit66] J.F. Ritt. *Differential Algebra.* Dover Publications, Inc., New York, 1966.

[Rou96] Fabrice Rouiller. *Algorithmes efficaces pour l'étude des zéros réels des systèmes polynomiaux.* PhD thesis, Universié de Rennes I, 1996. Thèse de Doctorat.

[Sei56] A. Seidenberg. Some remarks on Hilbert's Nullstellensatz. *Archiv der Mathematik*, 7 :235–240, 1956.

[SZ67] P. Samuel and O. Zariski. *Commutative algebra.* D. Van Nostrand Company, INC., 1967.

[Val95] Annick Valibouze. Cours de calcul formel. Université Paris 6, 1995.

[van91] B.L. van der Waerden. *Algebra.* Springer-Verlag, 1991. seventh edition.

[Wan91] D. M. Wang. On Wu's method for solving systems of algebraic equations. Technical Report RISC-LINZ Series no 91-52.0, Johannes Kepler University, Austria, 1991.

[Wan92a] D. M. Wang. An implementation of the characteristic set method in Maple. In *Proc. DISCO'92*, Bath, England, 1992.

[Wan92b] D. M. Wang. Some improvements on Wu's method for solving systems of algebraic equations. In Wu Wen-Tsün and Cheng Min-De, editors, *Proc. of the Int. Workshop on Math. Mechanisation*, Beijing, China, 1992. Institute of Systems Science, Academia Sinica.

[Wan93a] D. M. Wang. An elimination method based on Siedenberg's theory and its applications. In F. Eysette and A. Galligo, editors, *Comptutational Algebraic Geometry*, pages 301–328. Birkhäuser Boston, Inc., 1993.

[Wan93b] D. M. Wang. An elimination method for polynomial systems. *J. Symb. Comp.*, 16 :83–114, 1993.

[Wu84a] W. T. Wu. Basic principles of mechanical theorem proving in elementary geometries. *J. Sys. Sci. and Math. Scis*, 4 :207–235, 1984.

[Wu84b] W. T. Wu. Some recent advances in mechanical theorem-proving of geometries. *Contemporary Math.*, 29 :235–241, 1984.

[Wu86] W. T. Wu. On zeros of algebraic equations – an application of Ritt principle. *Kexue Tongbao*, 31(1) :1–5, 1986.

[Wu87] W. T. Wu. A zero structure theorem for polynomial equations solving. *MM Research Preprints*, 1 :2–12, 1987.

[Wu89] W. T. Wu. On a projection theorem of quasi-varieties in elimination theory. *MM Research Preprints*, 4 :40–53, 1989.

[Yun76] D.Y.Y. Yun. On Square-free Decomposition Algorithms. In *Proc. ISSAC'76*, pages 65–70, 1976.

[YZ91] L. Yang and J. Zhang. Searching dependency between algebraic equations : an algorithm applied to automated reasoning. Technical Report IC/89/263, International Atomic Energy Agency, Miramare, Trieste, Italy, 1991.

Résumé

Cette thèse est consacrée à la résolution des systèmes d'équations polynomiales à l'aide d'ensembles triangulaires. Une première partie consiste en la présentation de deux algorithmes permettant de calculer des pgcd de polynômes à coefficients dans une tour d'extensions simples. Le premier, réalisé avec Renaud Rioboo, s'applique aux tours algebriques. Le second, est une adaption du précédent au cas plus général des tours séparables. Ces algorithmes ont permis l'implantation efficace de deux méthodes proposées par Daniel Lazard pour résoudre les systèmes d'équations algébriques à l'aide d'ensembles triangulaires. Ces programmes ont permis la résolution automatique de problèmes inaccessibles auparavant. La deuxième de ces méthodes n'était qu'esquissée ; aussi une seconde partie est consacrée aux développements théoriques nécessaires à une implantation correcte. De plus, un travail d'unification théorique et de comparaison expérimentale avec les methodes voisines de Wu Wen-Tsün, Dongming Wang et Michael Kalkbrener a été réalisé avec Philippe Aubry.

Abstract

This thesis is dedicated to polynomial system solving by means of triangular sets. A first part presents two algorithms to compute polynomial gcds over tower of simple extensions. The first one was designed with Renaud Rioboo and applies to algebraic towers. The second one is a generalization of the previous one to the most general case of seperable towers. These algorithms lead to an efficient implementation of two methods suggested by Daniel Lazard to solve polynomial systems by means of triangular sets. These programs solved problems that were previously unreachable. The second method was only sketched by its author. So, a second part of this thesis presents the necessary developements to describe a right implementation. Moreover, a theorical and unified presentation, together with an experimental comparison with similar methods due to Wu Wen-Tsün, Dongming Wang and Michael Kalkbrener were realized with Philippe Aubry and are reported in a third part of this document.