Université de Nice-Sophia Antipolis

École doctorale des sciences et technologies
de l'information et de la communication

Thèse d'habilitation à diriger des recherches

«Quelques expériences de calculs et de raisonnements à l'aide de machines»

par

Loïc Pottier

Soutenue le 27 novembre 2003
devant le jury:

Gérard Boudol	examinateur
Gilles Dowek	rapporteur
Herman Geuvers	rapporteur
André Hirschowitz	examinateur
Emmanuel Kounalis	président
Roger Marlin	examinateur
Marie-Françoise Roy	rapporteur

à Nicolas, Anaïs, Héloïse et Lydie.

Merci

aux rapporteurs d'avoir bien voulu manifester de l'intérêt pour le modeste travail présenté dans ce mémoire,

aux membres du jury de s'être déplacés pour la soutenance, parfois de loin,

aux collègues, chercheurs, enseignants et administratifs de l'INRIA, de l'université de Nice et d'ailleurs, pour leurs travaux, avis, conseils, critiques, aide et encouragements depuis une quinzaine d'années.

Chapitre 1
Introduction

A défaut d'apprendre, les machines (ou plutôt les programmes d'ordinateurs) sont maintenant capables de calculer et de raisonner.

Calculer avec des nombres, mais aussi avec des symboles. L'évolution des calculatrices de lycée montre que petit à petit, avec les progrès des composants électroniques et ceux de l'algorithmique, des notions mathématiques de plus en plus abstraites sont traitées mécaniquement: nombres, polynômes, fonctions, etc.

Raisonner sur des booléens, mais aussi sur des énoncés logiques du premier ordre, voire dans la logique usuelle des mathématiques (l'ordre supérieur). Dans ce domaine, on n'en est pas encore au même point que pour le calcul. Le raisonnement, même en mathématique, est encore une notion qui relève plus du savoir-faire que de la science. Pourtant une démonstration est un objet formel comme un autre, et peut donc à ce titre être l'objet d'un calcul par machine. Nul doute que dans un proche(?) avenir, les calculatrices permettront de vérifier, voire d'effectuer des démonstrations.

Apprendre, c'est une autre paire de manches: les récents progrès en apprentissage statistique ne doivent pas cacher la difficulté du problème de l'apprentissage symbolique, pour lequel on n'a pas encore de théorie mathématique bien établie et suffisamment puissante.

Au départ de ma thèse, en 85, c'est le troisième point qui m'intéressait.

Après quelques mois et une communication à une conférence[1], sous l'impulsion d'André Galligo, Jacques Morgenstern et Jacques Chazarain, j'ai changé de sujet de recherche, pour m'intéresser aux algorithmes de Knuth-Bendix et de Buchberger. Je n'avais pas trouvé d'outils théoriques ne serait-ce que pour aborder et énoncer les problèmes de l'apprentissage automatique. Et je n'avais pas trouvé non plus de pistes à suivre dans cette direction. Les seuls travaux théoriques sur lesquels on[2] avait attiré mon attention étaient ceux de G.Plotkin et G.Huet sur l'anti-unification et la généralisation de clause (J.P. Jouannaud m'a par la suite aiguillé sur le problème de la généralisation associative-commutative, traité dans ma thèse[3]).

J'ai alors commencé à travailler à l'intersection du calcul formel et de la théorie de la réécriture, ce que j'ai continué ensuite jusqu'en 96[4]. Il s'agissait de découvrir et d'améliorer des algorithmes de calcul mathématique, principalement en algèbre commutative et en géométrie algébrique, ainsi que de mettre au point des logiciels permettant de tester ces algorithmes, et de manipuler des objets mathématiques de plus en plus abstraits (polynômes, anneaux, catégories, etc).

En calcul formel, cela a donné lieu aux travaux suivants:

1. ***«Greater Easy Common Divisor and Standard Basis Completion Algorithms»*** avec A.Galligo et C.Traverso, ISSAC 88, LNCS.

---

1. **«Généralisation d'exemples et de contre-exemples en calcul des propositions»",** 2ème Congrès d'Intelligence Artificielle de Marseille, déc. 86, actes édités chez Hermès.

   On utilise dans ce travail des bases standard (i.e. de Groebner) de polynômes booléens pour décomposer des formules, et les recomposer pour les généraliser.

2. Gérard Berry et Gilles Kahn.

3. «**Algorithmes de complétion et généralisation en logique du premier ordre**», thèse de doctorat, Université de Nice-Sophia Antipolis, février 1989.

4. C'était l'époque du projet Safir à l'INRIA Sophia Antipolis, ainsi que du GRECO de calcul formel et du PRC Math-Info, structures nationales du CNRS regroupant mathématiciens et informaticiens, dont les réunions annuelles proposaient des exposés vraiment très intéressants. J'ai beaucoup appris à ces occasions.

On propose une amélioration de l'algorithme de Buchberger consistant à garder une trace des factorisations partielles des coefficients des polynômes au cours des calculs pour s'en reservir dans la suite et accélérer ainsi l'éxécution.

2. '«***The design of Sisyphe: a system for doing symbolic and algebraic computations***» avec A.Galligo et J.Grimm, Design and Implementation of Symbolic Computation Systems, DISCO 90, LNCS.

   Il s'agit d'une description d'un prototype de système de calcul formel développé à l'INRIA Sophia, d'abord dans le projet MIAOU, puis ensuite dans le projet SAFIR.

3. «***The extensions of the SYSIPHE computer algebra system : ulysse and athena***» avec Christèle Faure, André Galligo et José Grimm, DISCO 92, actes publiés dans LNCS.

   Introduction de techniques de réécriture pour la simplification d'expression symboliques.

4. «***Computation of toric Grobner bases, successive minima of lattices, and integer points in polytopes***», actes du Posso Worshop on Software, mars 95.

5. «***A sharp lower bound on the time-space product for reversing a finite sequence***» avec José Grimm et Nicole Rostaing-Schmidt, conférence en mémoire de Jacques Morgenstern, mai 95. Publié dans «***Optimal time and minimum space-time product for reversing a certain class of programs***», Second International Workshop on Computational Differentiation, Santa Fé, EUA, 96.

   À partir de résultats de A. Griewank, J.Abbott et A.Galligo, on montre une borne inférieure sur l'espace-temps nécessaire pour inverser une suite u_0, $u_1 = f(u_0)$, ..., $u_n = f(u_{n-1})$.

6. «***Euclide's algorithm in dimension n***» ISSAC 96, ACM Press, juillet 1996.

   On étend à la dimension n l'algorithme d'Euclide de calcul du pgcd. Dans ce cas le résultat est une base de Graver du réseau engendré par les vecteurs de départ (en dimension 1, c'est un générateur de l'idéal engendré par les nombres de départ).

7. «***Computation of toric Grobner bases***», international worshop on tests sets in integer programming, Berlin, Germany, oct. 96.

8. «***Algorithms for Hilbert bases***», in invited session on Hilbert bases (org. M.Henk), International Symposium on Mathematical Programming (ISMP 97), Lausanne, Aug. 97.

En réécriture, cela a donné lieu aux travaux suivants:

1. «***Term generalization modulo equations***», actes du workshop UNIF 89, Lambrecht RFA, juillet 89

2. «***Bornes et algorithme de calcul des générateurs des solutions de systèmes diophantiens linéaires***» Comptes Rendus de l'Académie des Sciences de Paris, septembre 90.

   On y démontre une borne simplement exponentielle pour les générateurs des solutions de systèmes diophantiens linéaires. Indépendamment, Bernd Sturmfels en obtient aussi une en 90. Ce travail résulte au départ d'un problème soulevé par A.Boudet, E.Contejean, H.Devie dans l'équipe de J.P.Jouannaud à Orsay, dans le contexte de l'unification AC.

3. «***The complement problem in associative-commutative theories***», avec E.Kounalis et D.Lugiez, International Symposium on Mathematical Foundations of Computer Science, MFCS 91, LNCS, 91.

   On utilise les propriétés du conducteur d'un semi-groupe d'entiers dans le contexte du filtrage modulo AC.

4. «***Minimal solutions of linear diophantine systems: Bounds and algorithms***» International Congress on Rewriting Techniques and Applications, RTA 91, LNCS, Como, Italy, 91.

   Les systèmes diophantiens linéaires interviennent ici dans le contexte de la réécriture AC.

Un travail plus analytique, mais toujours sur les polynômes, provenant d'un problème concret de M.Berthod:

«***Optimization of positive generalized polynomials under lp constraints***» avec L.Baratchart et M.Berthod. Journal of Convex Analysis, 98.

L'utilisation de la réécriture en calcul formel que j'ai essayé de promouvoir dans le projet Safir[5] était fondée sur la remarque suivante: en mathématiques, de nombreuses preuves sont équationnelles, et la réécriture modulo est un candidat naturel pour effectuer ces preuves. Elle offre de plus une formalisation élégante, qui n'existe pas dans les systèmes de calcul formel: la simplification des expressions dans Maple, Mathematica, pour ne citer qu'eux, reste une opération mystérieuse qui n'est susceptible d'aucune étude formelle.

C'est d'ailleurs un phénomène plus général dans les logiciels à caractère scientifique et même pédagogiques: on dirait que pour eux les mathématiques ne se réduisent qu'au calcul, et que les démonstrations font partie d'un autre monde (celui du papier et du crayon ou du tableau noir et de la craie). Dans le plus évolué des systèmes de calcul formel, Axiom, on trouve des groupes, des corps, des modules, des catégories. Mais ce ne sont pas ceux qu'on rencontre en mathématiques, ou plutôt ce ne sont que leurs ombres: il ne reste que leurs opérations, et les axiomes qui les caractérisent ont disparu. Pire, aucun système de calcul formel ne permet d'énoncer un théorème: il leur manque le langage logique le plus élémentaire (quantificateurs et implications).

Ce sont ces réflexions qui m'ont conduit à partir de 96 à travailler sur le deuxième point que j'évoquais au début de cette introduction: le raisonnement. C'est le système Coq et la théorie des constructions inductives qui ont constitué le contexte des travaux suivants:

1. «***Extraction dans le CCI***», Journées Francophones des Langages Applicatifs, Pontarlier, 2001.
2. «***Quotients in the CIC***», Theorem Proving in Higher Order Logics: 14th International Conference, TPHOLs'01, University of Edinburgh ed., 2001.
3. «***Élimination des quantificateurs sur les réels en Coq***», avec Assia Mahboubi, Journées Francophones des Langages Applicatifs, 2002, et "*Proofs of polynomial inequalities in Coq*", exposé au séminaire "Verification and constructive algebra", Dagstuhl, janvier 2003.
4. «***Mathematical quotients and quotient types in Coq***», avec Laurent Chicli et Carlos Simpson, Actes du workshop TYPES 2002, à paraître dans LNCS.

Parallèlement j'ai aussi commencé à travailler sur un autre aspect des logiciels scientifiques: l'interface. Dans deux directions:

– la reconnaissance de formules mathématiques écrites dans le plan[6]:
    1. «***Optical formula recognition***», avec S.Lavirotte , Proceedings of the Fourth International Conference on Document Analysis and Recognition (ICDAR'97), volume 1, pages 357-361, Ulm, Allemagne. Août 1997. IEEE Computer Society Press.
    2. «***Mathematical formula recognition***» international workshop on retrodigitalization of mathematical documents, Essen, Germany, dec. 97.
    3. «***Mathematical formula recognition***» avec Stéphane Lavirotte, Electronic Imaging'98. Document Recognition V, volume 3305, pages 44-52, San José, Etats-Unis. Janvier 1998, SPIE - The International Society for Optical Engineering.
    4. «***On-Line Handwritten Formula Recognition using Hidden Markov Models and Context Dependent Graph Grammars***» avec Andreas Kosmala, Stéphane Lavirotte, Gerhard Rigoll. Proceedings of the Fifth International Conference on Document Analysis and Recognition (ICDAR'99), pages 107-110, Bangalore, Inde. Septembre 1999.

5. en particulier avec la thèse de Christèle Faure que j'ai co-encadrée avec André Galligo.
6. avec la thèse de Stéphane Lavirotte, que j'ai dirigée.

– le développement du système Pcoq et la traduction de preuves formelles en langage naturel:

«Mathematics and proof presentation in Pcoq», avec Ahmed Amerkad, Yves Bertot et Laurence Rideau, Workshop Proof Transformation and Presentation and Proof Complexities (PTP'01), Sienne, 2001.

La suite de ce mémoire est constituée de six des articles précédents, dont quatre ont été traduits en français, et d'une description d'une implémentation :

1. Une borne simplement exponentielle pour les système diophantiens linéaires.
2. L'algorithme d'Euclide en dimension n
3. Une borne inférieure optimale de l'espace-temps nécessaire à inverser une suite.
4. Algèbre en théorie des types.
5. Élimination des quantificateurs sur les réels dans Coq.
6. Quotients mathématiques et types quotients dans Coq.
7. Reconnaissance de formules planes.

Ils sont choisis parce que les résultats qu'ils présentent avaient quelque originalité (1, 2, 3, 6, 7), ou bien le travail qu'ils exposent était excitant et suivait des voies nouvelles (4, 5).

Le premier résultat est une note aux CRAS sur une borne simplement exponentielle pour les solutions minimales de systèmes diophantiens linéaires. J'en étais bien content à l'époque: le résultat était nouveau et concis, et sa preuve était simple et originale. Il était le fruit non pas du hasard, mais répondait à un problème ouvert que j'avais entrepris de résoudre: «y-a-t-il une borne simplement exponentielle pour les solutions minimales de systèmes diophantiens linéaires?». Ce problème à l'époque intéressait la petite communauté des chercheurs de l'unification AC, et avait des réponses dans des cas particuliers (Huet, Lambert, Romeuf, Contejean et Devie). Il est arrivé ensuite ce qui arrive souvent lorsqu'on s'intéresse à un sujet «chaud»: un résultat similaire avait été trouvé par Bernd Sturmfels la même année, avec une technique de preuve plus standard.

Pour replacer ces travaux dans le contexte de ce mémoire,

– les trois premiers travaux concernent le calcul (algorithmique, complexité, calcul formel),
– le cinquième concerne le calcul et le raisonnement (calcul de preuves),
– le quatrième et le sixième concernent le raisonnement (formalisation des mathématiques),
– le septième concerne le contexte des expériences (interfaces homme-machine), mais aussi le calcul.

Il y a un an, j'ai de nouveau regardé le problème de l'apprentissage automatique, avec quelques idées qui me semblaient nouvelles. Je compte continuer dès que possible à y travailler. Il faudra bien qu'un jour on puisse programmer des machines sans être obligé de tout leur dire, et que leur bêtise artificielle devienne une intelligence naturelle.

Chapitre 2

Une borne simplement exponentielle pour les systèmes diophantiens linéaires.[1]

2.1 Introduction.

Nous donnons des bornes supérieures simplement exponentielles pour la taille des générateurs des solutions entières positives d'un système d'équations diophantiennes linéaires homogènes (une équation diophantienne a ses coefficients entiers). Nous en déduisons alors des bornes similaires pour les générateurs des solutions d'un système diophantien linéaire $A\,x \leqslant b$.

2.2 Notations

Dans la suite $\mathbb{Z}^n$ sera plongé dans $\mathbb{R}^n$ et $\{e_1, ..., e_n\}$ sera sa base canonique.

Soit $x = (x_1, ..., x_n)$ un vecteur de $\mathbb{R}^n$. On définit $\|x\| = \sqrt{\sum_{i=1}^n x_i^2}$
$\|x\|_1 = \sum_{i=1}^n |x_i|$
$\|x\|_\infty = \sup_{1 \leq i \leq n} \{|x_i|\}$.

Soit A une matrice à m lignes et n colonnes. On lui étend ces définitions en la considérant comme un vecteur de $\mathbb{R}^{nm}$, et on définit de plus $\|A\|_l = \sup\{\|A\,x\| \mid \|x\| = 1\}$.

On note $B_n(s)$ la boule fermée de $\mathbb{R}^n$ centrée en 0 et de rayon s et $K_n(s)$ le nombre de points de $\mathbb{Z}^n$ qu'elle contient (on en donne en une majoration dans la section suivante).

Si E est un sous-espace de $\mathbb{R}^n$ et x un vecteur, on note $d(x, E)$ la distance euclidienne de x à E : $d(x, E) = \inf_{y \in E} \{\|x - y\|\}$.

Pour y dans $\mathbb{R}^n$, on note C_y le cube de volume 1 défini par

$$z \in C_y \Leftrightarrow z = y + \sum_{i=1}^{n} \lambda_i e_i, \text{ avec } \forall i \in [1, n],\ \lambda_i \in [0, 1]$$

2.3 Borne des générateurs des solutions positives d'un système $A\,x = 0$

Soit A une matrice d'entiers à m lignes et n colonnes. Elle définit un système S d'équations linéaires homogènes dont les solutions dans $\mathbb{Z}^n$ constituent le noyau H de A.

Les solutions positives de S constituent M, la trace de H sur $\mathbb{N}^n$.

Soit $\preceq$ l'ordre partiel sur $\mathbb{N}^n$ tel que $(x_1, ..., x_n) \preceq (y_1, ..., y_n)$ ssi $\forall i, 1 \leq i \leq n, x_i \leq y_i$.

M est alors engendré (comme sous-monoïde de $\mathbb{N}^n$) par ses éléments non nuls minimaux pour $\preceq$, qui de plus sont en nombre fini. On appelle «générateurs de M» ces éléments minimaux, et on note B_M leur ensemble.

1. publié dans "***Bornes et algorithme de calcul des générateurs des solutions de systèmes diophantiens linéaires***" Comptes Rendus de l'Académie des Sciences de Paris, septembre 90.

Si M n'est pas nul, il est connu[2] qu'il existe un générateur x de M de taille ($\|x\|_\infty$ par exemple) majorée par une expression simplement exponentielle (du type $\|A\|_l{}^m$). Mais, jusqu'à présent[3], les bornes majorant les tailles de tous les générateurs de M sont toutes, à notre connaissance, au moins doublement exponentielles (du type $\|A\|_\infty{}^{2^m}$) :

- *L.B.Treybig «Bounds in piecewise linear topology», Trans.AMS, v.201, 1975.* (résultat général sur le système $A\,x = b$).
- *G.Huet «An algorithm to generate the basis of solutions to homogeneous linear diophantine equation»', Information Processing Letters, vol.3, No.7, 1978,*
  J.L.Lambert «Une borne pour les générateurs des solutions entières positives d'une équation diophantienne linéaire.» Comptes Rendus de l'Académie des Sciences de Paris, t.305, Série I, pp39-40, 1987.
  la borne donnée pour $m = 1$, s'étend à $m > 1$. Les algorithmes proposés dans le cas $m = 1$ procèdent par énumération des vecteurs majorés par la borne.
- *J.F.Romeuf «Solutions of a linear diophantine syste»', UNIF'89, actes du troisième "Workshop" international sur l'unification, Lambrecht, RFA 89 :*
  représentation des solutions minimales par un automate, cas $n = 1$, 2, borne double exponentielle en n dans le cas général.
- *E.Contejean, H.Devie «Résolution de systèmes linéaires d'équations diophantiennes», UNIF'89, actes du troisième "Workshop" international sur l'unification, Lambrecht, RFA 89,* (algorithme de calcul des générateurs)
  et *L.Baratchart et L.Pottier «Un résultat sur les systèmes d'addition de vecteurs», rapport interne, INRIA Sophia Antipolis, février 1989* (borne sur les sorties de l'algorithme précédent).

On donne ici une borne simple exponentielle.

2.3.1 Borne des générateurs de M

Théorème 2.1. *Tout élément $x = (x_1, ..., x_n)$ de B_M vérifie :*

$$\sum_{i=1}^{n} x_i \leq K_m(\|A\|_l \sqrt{n-1})$$

Démonstration. Soient $x = (x_1, ..., x_n)$ un élément de M non nul, D la droite vectorielle porté par x, et $p = \|x\|_1 = \sum_{i=1}^{n} x_i$.
Nous aurons besoin d'un lemme :

Lemme 2.2. *Il existe une suite d'éléments de N^n $y^0, ..., y^p$ vérifiant :*

- $y^0 = 0 \prec y^1 \prec ... \prec y^p = x$.
- $\forall i \in [0, p-1], \exists j, y^{i+1} = y^i + e_j$
- $\forall i \in [0, p]$,
  $d(y^i, D) \leq \sqrt{n-1}$,
  $\|y^i\|_1 = i$,
  et $C_{y^i} \cap [0, x] \neq \emptyset$.

2. I.Borosh et L.B.Treybig «Bounds on positive integral solutions of linear diophantine equations», Proc. AMS v.55, n.2, mars 1976.

3. la note a été écrite et publiée en 1990...

Preuve :
Construisons la suite par récurrence.
$y^0 = 0$ convient clairement.
Supposons y^k construit, $0 \leq k \leq p-1$. Soit $y^k = \sum_i y_i^k e_i$.
$[0,x] \cap C_{y^k}$ est l'ensemble des z qui s'écrivent $\lambda \sum_i x_i e_i$ avec $0 \leq \lambda \leq 1$ et

$$\forall i \in [1,n],\ y_i^k \leq \lambda x_i \leq y_i^k + 1$$

Il est par hypothèse non vide, c'est un segment, et soit $z_k = \lambda_k \sum_i x_i e_i$ sa borne où λ est maximum.

x est non nul, donc il existe un j tel que

$$\lambda_k = \frac{y_j^k + 1}{x_j} = \inf f_{i|x_i \neq 0}\{\frac{y_i^k + 1}{x_i}\}$$

Soit $y^{k+1} = y^k + e\,j$. On a alors

$$\forall i \in [1,n],\ y_i^{k+1} \leq \lambda_k x_i \leq y_i^{k+1} + 1$$

Le point z_k appartient donc aux deux cubes C_{y^k} et $C_{y^{k+1}}$. Il est de plus sur une face commune aux deux cubes, face qui contient y^{k+1}. Comme une face d'un cube unité de $\mathbb{R}^n$ est un cube unité de $\mathbb{R}^{n-1}$, et qu'un tel cube a un diamètre (plus grande distance entre deux points du cube) égal à $\sqrt{n-1}$, et comme z_k est sur la droite D, on a bien $d(y^{k+1}, D) \leq \sqrt{n-1}$.

On a clairement $y^k \prec y^{k+1}$, et $\|y^{k+1}\|_1 = 1 + \|y^k\|_1 = k+1$.

Enfin, si $k = p-1$, $y^p = x$, car $y^p \preceq x$ et $\|y^p\|_1 = p = \|x\|_1$.

□

Prenons maintenant la suite (y^k) du lemme.

Tout y^k s'écrit (dans $\mathbb{R}^n$) $y^k = y'^k + y''^k$ où y'^k est la projection orthogonale de y^k sur D, et y''^k est la projection orthogonale de y^k sur l'orthogonal de D.

D'après le lemme, on a $\|y''^k\| \leq \sqrt{n-1}$. Ainsi, en identifiant un vecteur avec ses coordonnées dans la base canonique de $\mathbb{R}^n$ ou de $\mathbb{R}^m$ selon le cas :

$$\|A\,y^k\| = \|A\,y'^k + A\,y''^k\| = \|0 + A\,y''^k\| \leq \|A\|_l \|y''^k\| \leq \|A\|_l \sqrt{n-1}$$

Les vecteurs $A\,y^k$ sont donc dans $B_m(\|A\|_l \sqrt{n-1})$, qui contient au plus $K_m(\|A\|_l \sqrt{n-1})$ vecteurs à coefficients entiers.

Supposons maintenant $p > K_m(\|A\|_l \sqrt{n-1})$. Il existe donc (lemme des tiroirs) i et j, $i > j$ avec $A\,y^i = A\,y^j$.

Soit $z = y^i - y^j$. On a alors $A\,z = 0$, donc z est dans H. De plus on a $0 \prec z \prec x$, et $z \in M$. Donc $x \notin B_M$, ce qui achève la preuve du théorème. □

2.3.2 Expression de $K_m(s)$

Comme le diamètre d'un cube unité de $\mathbb{R}^m$ est égal à $\sqrt{m}$, les cubes C_z où $z \in B_m(s) \cap \mathbb{Z}^n$ sont inclus dans la boule $B_m(s + \sqrt{m})$. Ainsi le volume de cette boule majore $K_m(s)$.

En utilisant l'expression du volume d'une boule de $\mathbb{R}^m$, et en remarquant que si r est le rang de A on peut choisir r équations d'un système sans changer ses solutions, on a :

Corollaire 2.3. *Soient S un système linéaire homogène à n indéterminées, à coefficients dans $\mathbb{Z}$, de rang r, et A la matrice associée à r équations indépendantes de S. Alors tout générateur $(x_1, ..., x_n)$ des solutions de S à coefficients positifs vérifie :*

$$\text{si } r = 2d+1 : \quad \sum_{i=1}^{n} x_i \leq \frac{2^r \pi^d d!}{r(2d)!}(\sqrt{n-1}\|A\|_l + \sqrt{r})^r$$

$$\text{si } r = 2d : \quad \sum_{i=1}^{n} x_i \leq \frac{2\pi^d}{r(d-1)!}(\sqrt{n-1}\|A\|_l + \sqrt{r})^r$$

Remarque 2.4. En utilisant par exemple l'inégalité $\|A\|_l \leq \sqrt{n\,r}\|A\|_\infty$, on obtient des bornes qui dépendent explicitement des coefficients de A, en restant simplement exponentielles.

2.4 Borne des générateurs des solutions de $A\,x \geq b$

On considère maintenant un système d'inéquations linéaires à coefficients entiers. Il s'écrit $A\,x \geq b$, où A est une matrice comme précédemment, x est un vecteur de n inconnues, et b est dans $\mathbb{Z}^m$. Soit $\mathcal{S}$ l'ensemble de ses solutions dans $\mathbb{Z}^n$. Alors :

Corollaire 2.5. *Il existe deux parties finies $\mathcal{S}_1$ et $\mathcal{S}_2$ de $\mathbb{Z}^n$ telles que :*

$$x \in \mathcal{S} \Leftrightarrow x = x_1 + x_2 + ... + x_k,\ avec\ x_1 \in \mathcal{S}_1,\ et\ x_2, ..., x_k \in \mathcal{S}_2$$

$$et\ \ \forall x \in \mathcal{S}_1 \cup \mathcal{S}_2,\ \|x\|_1 \leq K_m(\sqrt{n+m}\sqrt{\|A\|^2 + m + \|b\|^2})$$

Démonstration. Par introduction de nouvelles variables, on se ramène à des systèmes $A'x = 0$ comme précédemment dont on projette certains générateurs. □

Chapitre 3
L'algorithme d'Euclide en dimension n[1]

3.1 Introduction.

On présente ici un algorithme qui est une extension naturelle en dimension n de l'algorithme d'Euclide de calcul du plus grand commun diviseur de deux entiers. Etant donné un sous-groupe H de $\mathbb{Z}^n$, donné par un système générateur, cet algorithme calcule la réunion des bases des monoïdes obtenus par intersection de H avec les 2^n orthants de $\mathbb{Z}^n$. Comme conséquence, on peut l'utiliser par exemple pour calculer les solutions minimales d'un système diophantien linéaire, la base du monoïde des points entiers d'un cône convexe simplicial rationnel, la série de Hilbert d'une algèbre graduée, ou encore les points entiers d'un simplexe rationel.

C'est un algorithme de *complétion*, i.e. du type de ceux de Buchberger (calcul de bases de Grobner) et de Knuth-Bendix (calcul d'un système de réécriture convergent), eux aussi apparentés à l'algorithme d'Euclide.

3.2 L'algorithme d'Euclide dans $\mathbb{Z}$

L'algorithme d'Euclide effectue des divisions euclidiennes successives :

$$a_1 = q_1 a_2 + a_3,\ a_2 = q_2 a_3 + a_4,\ , ..., a_{n-1} = q_{n-1} a_n + 0$$

pour finalement obtenir le pgcd a_n de a_1 et a_2, comme dernier reste non nul de ces divisions.

Plus généralement, étant donnés $g_1, ..., g_n$ des générateurs d'un sous-groupe H de $\mathbb{Z}$, si on les divise entre eux, en les remplaçant par les restes des divisions, tant que cela est possible, on obtient une base de H, i.e. le pgcd des g_i. Remarquons que ce pgcd est aussi, en valeur absolue, une base du monoïde $H \cap \mathbb{N}$.

L'algorithme d'Euclide apparaît alors comme un algorithme de *complétion*, comme celui de Buchberger, qui effectue des divisions sur des polynômes en plusieurs variables, et comme celui de Knuth-Bendix, qui complète des règles de réécriture.

3.3 Généralisation à $\mathbb{Z}^n$

Définition 3.1. *Un vecteur non nul v divise un vecteur v' si pour tout $i \in [1; n]$, $v_i v'_i \geq 0$ et $|v_i| \leq |v'_i|$*

Le *reste* de la division de v' par v est $v' - q\,v$, où q est le plus grand entier naturel tel que $q\,v$ divise v' si v divise v', et $q = 0$ sinon.

Notons que ce reste se trouve dans le même orthant de $\mathbb{Z}^n$ que v', ainsi que v si q est non nul.

Le reste $R(v, F)$ de la division d'un vecteur v par une famille de vecteurs F est obtenu par divisions successives par les vecteurs de F ou par leurs opposés. Il n'est en général pas unique, et dépend de l'ordre dans lequel les vecteurs de F sont utilisés.

1. publié en anglais dans ***"Euclide's algorithm in dimension n"*** ISSAC 96, ACM Press, juillet 1996.

Autrement dit, la relation de division par F n'est en général pas *confluente*. Et c'est pour la rendre confluente qu'il est nécessaire de compléter F, selon le principe partagé par les algorithmes de Buchberger et de Knuth-Bendix, que nous reprenons ici.

3.4 Algorithme d'Euclide en dimension n.

On construit incrémentalement, à partir d'un système générateur G d'un sous-groupe H de Z^n, une famille F de vecteurs de H, en ajoutant à chaque étape à F les restes non nuls des divisions par F des sommes et différences de vecteurs de F (sommes et différences qui sont les analogues des *S-polynômes* de Buchberger et des *paires critiques* de Knuth et Bendix). Une dernière étape divise les vecteurs entre eux.

- **Procédure Completion(G)**
  $F := G \cup -G$
  SD$:= \{v+v' | v, v' \in F\} - \{0\}$
  Tant que SD$\neq\emptyset$
    soit $v \in$ SD
    SD$:=$ SD$- \{v, -v\}$
    $v := R(v, F)$
    si $v \neq 0$
      SD$:=$ SD$\cup \{v+v' | v' \in F\} \cup \{-v+v' | v' \in F\}$
      $F := F \cup \{v, -v\}$
  Rendre F
- **Procédure Réduction(F)**
  Tant qu'il existe v et v' de F différents, v' divisant v
    $F := F - \{v\} \cup \{R(v, \{v'\})\}$
  Rendre F
- **Procédure BaseDeHilbert(G)**
  Rendre Reduction(Completion(G))

Théorème 3.2.

i. les trois procédures précédentes terminent toujours.

ii. $v \in H \Leftrightarrow R(v, Completion(G)) = 0$

iii. pour tout orthant $\mathcal{O}$ *de* Z^n, $\mathcal{O} \cap BaseDeHilbert(G)$ *est la base du monoïde* $\mathcal{O} \cap H$.

Démonstration.

i. La procédure Réduction termine car diviser un vecteur de F diminue strictement la somme des valeurs absolues des coordonnées des vecteurs de F.

   La procédure Completion construit une suite de vecteurs dont chaque élément ne peut être divisé par les précédents. Si cette suite est infinie, elle contient une sous-suite dont les premières coordonnées sont positives et croissantes (ou bien négatives et décroissantes). De même pour les autres coordonnées: elle contient donc une sous-suite de vecteurs divisant leur successeur, d'où une contradiction (l'argument est identique à ceux du théorème de la base de Hilbert, et du lemme de Dixon).

ii. Soit F:=Completion(G). Soit v un vecteur de H irréductible par F. F est générateur de H (il contient G), il contient les opposés de ses éléments, donc on peut écrire $v = v_1 + ... + v_p$ où les v_i sont dans F.

Notons u^+ la partie positive d'un vecteur u, et u^- l'opposé de sa partie négative, de sorte que $u = u^+ - u^-$.

Si $v^+ = v_1^+ + ... + v_p^+$, alors $v^- = v_1^- + ... + v_p^-$, et v est divisible par les v_i ce qui contredit l'hypothèse que v est irréductible.

Donc il existe deux vecteurs, disons v_1 et v_2, tels que $(v_1 + v_2)^+ \neq v_1^+ + v_2^+$. Comme $F=$ Completion(G), les sommes et différences de vecteurs de F se réduisent à 0 par division par F. Ainsi on peut écrire $v_1 + v_2 = v_{p+1} + ... + v_q$ où les v_j sont dans F, ce qui donne $v = v_3 + ... + v_q$. On a de plus $(v_1 + v_2)^+ = v_{p+1}^+ + ... + v_q^+$.

Comme $(v_1 + v_2)^+ \neq v_1^+ + v_2^+$, la somme des coordonnées de $(v_1 + v_2)^+$ est strictement plus petite que la somme des coordonnées de v_1^+ et v_2^+. Ainsi dans l'écriture $v = v_3 + ... + v_q$, la somme des coordonnées des parties positives des vecteurs de la somme est strictement plus petite que dans l'écriture $v = v_1 + ... + v_p$, de même que pour les parties négatives. En itérant ce raisonnement, on trouve donc $v = 0$.

Tout vecteur de H se réduit donc à 0 par F. Inversement, il est clair qu'un vecteur qui se réduit à 0 par F est somme de vecteurs de F donc est dans H

iii. Soit B:=BaseDeHilbert(G). La procédure Reduction conserve la propriété (ii), donc un vecteur est dans H si et seulement si il se réduit à 0 par $B\,S$. Les restes des divisions successives d'un vecteur restent dans son orthant, et ne font intervenir que des diviseurs dans son orthant, donc un vecteur de $H \cap \mathcal{O}$ s'écrit comme somme de vecteurs de $B \cap \mathcal{O}$. Ainsi $B \cap \mathcal{O}$ est un système générateur du monoide $H \cap \mathcal{O}$. Les vecteurs de B ne pouvant se diviser entre eux, il s'en suit que $B \cap \mathcal{O}$ est la base de $H \cap \mathcal{O}$

□

3.5 Améliorations de l'algorithme.

On peut améliorer l'algorithme en évitant d'ajouter à F les réductions de certaines sommes et différences, de manière analogue aux critères 1 et 2 de Buchberger, et aux critères "d'occurrences incomparables" et de la "middle rule" de l'algorithme de Knuth et Bendix.

Critère 1 Si v et v' sont dans le même orthant, alors $v + v'$ se réduit à 0 par division par v puis v' (de même, si v et v' sont dans deux orthants opposés, $v - v'$ se réduit à 0). Il n'est donc pas besoin d'ajouter à F leur somme (resp. leur différence).

Critère 2 Soient v_1, v_2, et v_3 dans F tels que $v_1 - v_3$ et $v_3 - v_2$ soient dans le même orthant. Alors il n'est pas nécessaire d'ajouter à F $v_1 - v_2$ si on y ajoute les restes de $v_1 - v_3$ et $v_3 - v_2$ par la division par F. En effet, cet ajout permet alors de réduire $v_1 - v_2$ à 0.

Plus généralement:

la différence $v_1 - v_2$ est inutile s'il existe dans F un vecteur v_3 différent de v_1 et v_2 tel que $v_1 - v_3$ et $v_3 - v_2$, ou $v_1 + v_3$ et $-v_3 - v_2$, sont dans le même orthant.

la somme $v_1 + v_2$ est inutile s'il existe dans F un vecteur v_3 différent de v_1 et v_2 tel que $v_1 - v_3$ et $v_3 + v2$, ou $v_1 + v_3$ et $v_2 - v_3$, sont dans le même orthant.

En pratique, ces critères sont très efficaces, et évitent de nombreuses divisions.

3.6 Quelques applications.

1. Soit un système diophantien linéaire $A\,x = 0$, , $x \geq 0$, où A est une matrice à coefficients entiers, et x est un vecteur entier. Ses solutions minimales sont par définition celles qui ne sont pas sommes d'autres solutions (voir [2] pour des algorithmes de calcul).

Soit G une base du noyau de l'application $x \mapsto A\,x$ de $\mathbb{Z}^n$ dans $\mathbb{Z}^m$. Alors BaseDeHilbert(G) $\cap\, \mathbb{N}^n$ est l'ensemble des solutions minimales de S

2. Soit C un cône convexe rationnel et simplicial de $\mathbb{R}^n$: C est l'enveloppe convexe de demi-droites $\mathbb{R}_+a_1$, ..., $\mathbb{R}_+a_n$, où les a_i sont dans $\mathbb{Z}^n$. Soit A la matrice carré dont les colonnes sont les a_i, et H le sous-groupe de $\mathbb{Z}^n$ engendré par la famille G des colonnes de la matrice $\det(A)A^{-1}$.

   L'image par l'application $x \mapsto \frac{A}{\det(A)}x$ de BaseDeHilbert(G) $\cap$ $\mathbb{N}^n$ est la base du monoide des points entiers de C.

3. Soit S un simplexe rationnel de $\mathbb{R}^n$: S est l'enveloppe convexe de $n+1$ points indépendants a_1, ..., a_{n+1}, à coordonnées rationnelles. Plongeons S dans $\mathbb{R}^{n+1}$ par l'application ϕ: $x \mapsto (x, 1)$ de $\mathbb{Z}^n$ dans $\mathbb{Z}^{n+1}$: $\phi(S)$ engendre un cône convexe C, qui est rationnel et simplicial.

   Les éléments de la base des points entiers de C qui ont 1 comme dernière coordonnée se projètent alors en les points entiers de S (par la projection associée à ϕ: $(x, y) \mapsto x$).

4. On peut généraliser ce qui précède au cas de sous-groupes discrets de $\mathbb{R}^n$.

   Dans ce cas, la procédure Complétion ne terminera pas : les monoïdes $H \cap \mathcal{O}$ ne sont plus nécessairement de type fini. Mais leurs intersections avec tout compact est finie. Donc si dans la procédure, on choisit dans l'ensemble SD toujours le vecteur de plus petite norme (euclidienne, ou infinie par exemple), on est sûr d'énumérer toute la base du monoïde en question.

   Le même raisonnement s'applique au problème du calcul de la base des points entiers d'un cône simplicial. Des résultats d'Arnol'd et Lachaud[3] permettent d'approximer les normales aux faces d'un cône simplicial par les faces de la voile du cône (bord de l'enveloppe convexe des points entiers non nuls du cône).

   Comme les sommets de la voile d'un cône simplicial sont en particulier dans la base des points entiers du cône, l'algorithme d'Euclide en dimension n devrait fournir une méthode d'approximation de nombres irrationnels.

   C'est en effet ce qui se passe en dimension 1, ou l'algorithme d'Euclide fournit les coefficients du développement en fraction continue du quotient des deux premiers nombres, et où ces coefficients peuvent être obtenus géométriquement comme longueurs des faces des voiles de deux cônes du plan.

   Peut-on généraliser cela en dimension n, pour donner un sens aux fractions continues dans ce contexte? Une autre question est de comprendre si le fait important est d'utiliser les sommets de la voile, ou bien une base des points entiers du cône (ces deux notions coincident dans le plan, mais pas en general).

5. La base de Hilbert de H est une base de Groebner universelle pour l'idéal torique de H, i.e. l'idéal engendré par les différences de monômes $X^{v^+} - X^{v^-}$, où v parcourt H.

6. Une implémentation de cet algorithme peut être téléchargée sur le Web à l'adresse :
   `http://www-sop.inria.fr/croap/personnel/Loic.Pottier/Bastat/bastat-demo.html`

---

2. Evelyne Contejean and Hervé Devie. *Solving systems of linear diophantine equations*. In *UNIF'89*, Lambrecht, RFA, 1990.

Eric Domenjoud. *Outils pour la Déduction Automatique dans les Théories Associatives-Commutatives*. PhD thesis, Université de Nancy I, 1991.

Loïc Pottier. *Gröbner bases of toric ideals*. Rapport de recherche 2224, INRIA, Mars1994.

A Schrijver. *Theory of Linear and Integer Programming*. Wiley-Interscience, 1986.

3. Gilles Lachaud. *Polyèdre d'Arnol'd et voile d'un cône simplicial: analogues du théorème de Lagrange. Comptes rendus de l'Académie des Sciences de Paris*, t.317, Série I, p. 711-716 1993.

Chapitre 4

Une borne inférieure optimale de l'espace-temps nécessaire à inverser une suite.[1]

On montre que l'espace-temps nécessaire à inverser une suite u_0, ..., u_p donnée par une récurrence $u_{i+1} = f(u_i)$ est asymptotiquement supérieur à $\frac{p(\ln p)^2}{(\ln 4)^2}$, et que cette borne inférieure est optimale.

4.1 Introduction

Considérons le problème suivante: on veut afficher dans l'ordre inverse une suite $u_0...u_p$, en utilisant une mémoire donnée (r registres). La suite est donnée par son premier terme u_0 et une fonction f telle que $u_{i+1} = f(u_i)$. Chaque élément u_{i+1} ne peut être calculé que grâce au précédent u_i, pris dans un registre. On suppose que le calcul de $f(u_i)$ prend une unité de temps, et qu'un registre prend une unité d'espace.

Ce problème a été étudié dans le contexte des compromis mémorisation/recalcul en différentiation automatique de programmes par A.Griewank[2] , J.Abbott and A.Galligo[3] :

J.Morgenstern[4] a donné une preuve élégante d'un théorème de Baur and Strassen[5], qui dit qu'une fraction rationnelle f et ses dérivées partielles peuvent être calculées dans un temps inférieur à 5 fois le temps nécessaire à calculer la fonction f elle-même. Cette méthode de calcul utilise des valeurs de f en des points donnés dans un ordre inverse de celui du calcul de f.

Le temps et l'espace sont très liés dans les programmes: si on peut garder des résultats intermédiaires en mémoire, on pourra s'en reservir plus tard et économiser ainsi le temps nécessaire à les recalculer. C'est pourquoi on s'intéresse aux bornes inférieures du produit espace-temps en algorithmique (voir le travail de Borodin[6]).

A partir des travaux de Griewank, Abbott et Galligo, qui s'intéressent aux algorithmes sans boucle du type «diviser pour régner» pour inverser une suite, on montre ici que l'espace-temps nécessaire à **tout** algorithme sans boucle pour inverser une suite u_0, ..., u_p donnée par une récurrence $u_{i+1} = f(u_i)$ est asymptotiquement supérieur à $\frac{p(\ln p)^2}{(\ln 4)^2}$, et cette borne inférieure est optimale.

---

1. écrit avec José Grimm et Nicole Rostaing, et publié dans "***A sharp lower bound on the time-space product for reversing a finite sequence***" avec José Grimm et Nicole Rostaing-Schmidt, conférence en mémoire de Jacques Morgenstern, mai 95. Ainsi que dans "***Optimal time and minimum space-time product for reversing a certain class of programs***», Second International Workshop on Computational Differentiation, Santa Fé, EUA, 96.

2. A. Griewank «*Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation*», Optimization methods and softwares,Volume 1, pages 35-54, 1992

3. J. Abbott and A. Galligo «*Reversing a finite sequence*», preprint, décembre 1991.

4. J. Morgenstern «*How to compute fast a function and all its derivatives, a variation on the theorem of Baur-Strassen*», Sigact News, Volume 16, pages 60-62, 1985

5. W. Baur and V. Strassen «*The complexity of partial derivatives*», Theoretical Comp. Sci., Volume 22, pages 317-330, 1983

6. A. Borodin, «*Time-Space Tradeoffs (Getting Closer to the Barrier?)*», Algorithms and Computation, ISAAC'93, Springer-Verlag, pages 209-220, 1993.

4.2 Modèle d'algorithme.

On s'intéresse ici aux algorithmes sans boucle (en anglais *straight line programs*). Pour le problème qui nous intéresse, ce sont des suites d'instructions où une instruction est soit $R_i := f(R_j)$, soit print(R_i) où R_i dénote le $i$ème registre. On suppose qu'au début tous les registres contiennent la valeur u_0 et que la valeur du registre R_0 ne change pas.

Un algorithme sans boucle est une *inversion* de la suite $u_0 ... u_p$ s'il imprime u_p, ..., u_0 dans cet ordre. À une étape d'un algorithme, un registre est dit utile s'il sera utilisé par la suite, avant que sa valeur ne change. Étant donné un algorithme sans boucle, pour tout indice j, soit (x_{ij}) la liste des indices des éléments de la suite contenus dans les registres utiles, en ordre croissant, juste avant que la valeur u_j ne soit imprimée.

Commencons avec un exemple: l'inversion d'une suite de longueur 5 avec 3 registres.

Le tableau suivant représente un algorithme sans boucle qui inverse la suite. À gauche, un registre y est représenté par un o, surmonté de son numéro. Il est placé sur la valeur de la suite qu'il contient. Chaque ligne représente l'état des registres quand un élément de la suite est imprimé. À droite on trouve l'algorithme lui-même.

				$R_1 := f(R_0)$
				$R_1 := f(R_1)$
				$R_2 := f(R_1)$
				$R_2 := f(R_2)$
	1		2	3
.	.	o	.	o o print(R_3)
				$R_3 := f(R_0)$
	3	1		2
.	o	o	.	o print(R_2)
				$R_2 := f(R_1)$
	3	1	2	
.	o	o	o	print(R_2)
	3	1		
.	o	o		print(R_1)
	3			
.	o			print(R_3)
.				print(R_0)

Le temps de calcul est égal à 7. Dans ce cas, les listes (x_{ij}) sont $(2, 4, 5)$, $(1, 2, 4)$, $(1, 2, 3)$, $(1, 2)$ et (1)., et sont représentées par la partie gauche du tableau ci-dessus.

Définition 4.1. *Le temps de calcul d'un algorithme sans boucle est le nombre d'instructions faisant intervenir f, et le temps d'inversion de la suite u_0, ..., u_p est le temps de calcul minimum des inversions de $u_0, ..., u_p$*

Le temps d'une inversion I est $t_I := \sum_{ij} t_{ij}$, où t_{ij} est le temps nécessaire pour calculer R_i à l'étape j:

$t_{ij} = x_{ij} - y_{ij}$

$y_{ij} = \sup\left(x_{i-1,j}, \sup_k \{x_{k,j+1} | x_{k,j+1} \leq x_{ij}\}\right)$

$y_{ip} = x_{i-1,p}$

$y_{0j} = 0$

Avec ces notations, étant donné (x_{ij}), on peut construire l'inversion ainsi:

pour chaque j, de p à 0, pour chaque i tel que $y_{ij} = \sup_k (x_{k,j+1})$, soit K l'indice du registre qui contient $u_{x_{k,j+1}}$. On ajoute t_{ij} instructions de la forme $R_K := f(R_K)$, sauf si $K = 0$. Dans ce cas, on ajoute $R_s := f(R_0)$, suivie par $t_{ij} - 1$ instructions $R_s := f(R_s)$, où R_s est un registre inutile (il y en a au moins un, celui qui contient u_{j+1}).

Il reste alors à calculer un certain nombre n de valeurs correspondants à $y_{ij} = x_{i-1,j}$. Si E est l'ensemble des registres non utilisés à ce moment, il contient au moins n éléments, donc pour chaque valeur pas encore calculée, on peut choisir un nouveau registre R_K dans E, et ajouter $R_K := f(R_s)$, suivie par $t_{ij} - 1$ instructions $R_K := f(R_K)$, où R_s est le registre qui contient $u_{x_{i-1,j}}$.

La donnée des (x_{ij}) suffit donc pour construire une inversion.

4.3 Inversions optimales

Une inversion est optimale quand son temps de calcul est minimum.

Lemme 4.2. *Pour toute inversion I il existe une inversion $I' := (x'_{ij})$ telle que (x'_{1j}) décroît quand j décroît, et telle que $t_{I'} \leqslant t_I$*

Démonstration. Soit $I := (x_{ij})$ une inversion. Si (x_{1j}) décroît, c'est terminé. Sinon, il existe un j minimum et un i maximum tels que

$$x_{1,j+1} \leq x_{i,j+1} < x_{1j} \leq x_{i+1,j+1} \quad (1)$$

Considérons d'abord le cas où $x_{1j} = x_{i+1,j+1}$. Soit $I' := I$, sans $x_{i,j+1}$. Les temps de calcul sont inchangés pour $k \geq j$. Les autres temps de calcul ne peuvent être plus grands, puisqu'il y a une valeur en moins à garder dans les registres.

Sinon, soit $I' := I$ sauf pour $x'_{i,j+1} := x_{1j}$. Soit $\delta = x_{ij} - x_{i,j+1}$. Comme x_{1j} a été calculé à partir de $x_{i,j+1}$ avec un temps δ, on gagne δ instructions pour calculer x_{ij} pour I'. On perd au plus δ instructions à décaler le registre contenant $x_{i,j+1}$ de δ vers la droite (dans le tableau qui représente les suites (x_{ij})).

Ainsi on a trouvé I' telle que $t_{I'} \leqslant t_I$. On peut répéter ce procédé, tant que $i \geqslant 1$. Après celà, soit la condition du lemme est satisfaite, soit (1) est vraie pour un j plus grand. Il suffit alors de recommencer ce procédé pour tous les j.

□

Soit (C_q) la condition $x_{1p} = x_{1,p-1} = \ldots = x_{1q} = q$. Cette condition dit que le premier registre reste constant jusqu'à ce que sa valeur soit imprimée.

Lemme 4.3. *Soit I une inversion telle que (x_{1j}) décroît, et soit $q := x_{1p}$. Alors il existe une inversion I' satisfaisant (C_q), avec $t_{I'} \leqslant t_I$.*

Démonstration. Soit $I' := (x'_{ij})$ telle que $x'_{ij} = q$, si $j \geq q$ et $x_{ij} \leq q$, et $x'_{ij} = x_{ij}$ sinon (si plus d'un x_{ij} est inférieur ou égal à q, certains indices doivent être renommés).

On découpe les temps t_{ij} en trois parties: $t_{ij} = t^1_{ij} + t^2_{ij} + t^3_{ij}$ ainsi:
- si $x_{ij} \leq q$, alors $t^1_{ij} := t_{ij}, t^2_{ij} := t^3_{ij} := 0$.
- si $x_{ij} > q, y_{ij} \geq q$, alors $t^1_{ij} := t^2_{ij} := 0, t^3_{ij} := t_{ij}$.
- si $x_{ij} > q, y_{ij} < q$, alors $t^1_{ij} := 0, t^2_{ij} := q - y_{ij}, t^3_{ij} := x_{ij} - q$.

Et on définit $t_k := \sum t^k_{ij}$, la somme étant prise sur $j \geq q - 1$. Si on définit $t_4 = \sum_{i,j<q-1} t_{ij}$, le temps de calcul total de l'inversion est $t_1 + t_2 + t_3 + t_4$.

Puisqu'on a changé x_{ij} seulement pour $j \geq q$, on a $t'_4 = t_4$. Clairement $0 = t'_2 \leq t_2$. Il n'est pas difficile de montrer $t'_3 = t_3$. Finallement, $t'_1 = 2q - 1 \leq t_1$: quelque soit la manière, le temps de calcul de u_{q-1} si le premier registre est localisé en q est au moins $q - 1$. À cela, on doit ajouter le temps de calcul de u_q .

□

Maintenant soit I une inversion. Avec le lemme précédent, on peut la remplacer par une inversion I' satisfaisant (C_q) et de temps de calcul inférieur ou égal.

L'inversion $I_1' := (x_{ij})_{j<q}$ est une inversion de $u_0 ... u_{q-1}$ avec r registres.

L'inversion $I_2' := (x_{ij})_{j\geq q}$ peut être considérée comme une inversion de $u_q ... u_p$ avec $r-1$ registres.

Définition 4.4. *I est appelée une inversion $\ll$ diviser pour régner $\gg$ (DR-inversion) de $u_0, ..., u_p$ avec r registres, si I satisfait (C_q), et I_1, I_2 sont soit triviales ($q = 0$) soit des DR-inversions.*

En itérant les lemmes précédents on obtient:

Théorème 4.5. *Il existe une inversion optimale de $u_0, ..., u_p$ avec r registres qui est une DR-inversion.*

4.3.1 Exemple

En appliquant le lemme 3.3 sur l'exemple précédent on obtient l'inversion:

1 2 3 . . o . o o	R_1:=f(R_0)
	R_1:=f(R_1)
1 3 2 . . o o o	R_2:=f(R_1)
	R_2:=f(R_2)
	R_3:=f(R_2)
	print(R_3)
1 3 . . o o	R_3:=f(R_1) (*)
	print(R_2) (*)
	print(R_3)
1 . . o	print(R_1)
	R_3:=f(R_0)
	print(R_3)
3 . o	print(R_0)

Et le temps de calcul est toujours égal à 7. On obtient une DR-inversion et appliquant le lemme 3.3 de nouveau, en échangeant les lignes (*).

4.4 Temps optimal.

Le théorème 3.5 montre qu'on peut se réduire aux DR-inversions si on cherche à minimiser le temps d'inversion d'une suite pour un nombre de registre donné.

Définition 4.6. *Soit* $M(r,k) = \begin{pmatrix} r+k-1 \\ r \end{pmatrix} = \frac{(r+k-1)!}{r!(k-1)!}$

Le temps optimal dans le théorème suivant est démontré par Abbott et Galligo dans le cas particulier des DR-inversion:

Théorème 4.7. *Soit p un entier, et k tel que*

$$M(r,k) - 1 \leq p \leq M(r,k+1) - 1 \tag{4.1}$$

Soit

$$T(r,p) = k(p+1) - M(r+1,k) \tag{4.2}$$

Alors $T(r,p)$ *est le temps minimal d'une inversion de la suite* $u_0 ... u_p$ *avec* r *registres.*

De plus, une DR-inversion est optimale si et seulement si son indice q *(* $q=x_{1p}$*) vérifie*

$$M(r,k-1) \leq q \leq M(r,k) \tag{4.3}$$

$$M(r-1,k)-1 \leq p-q \leq M(r-1,k+1)-1 \tag{4.4}$$

Démonstration. Soit $f(q) = q + T(r,q-1) + T(r-1,p-q)$, et E l'ensemble des q satisfaisant (4.3) et (4.4). Il suffit de montrer que f est minimum ssi q est dans E, et que la valeur minimale de f est $T(r,p)$.

Soit I une DR-inversion, I_1 et I_2 comme précédemment. Par hypothèse de récurrence, les temps de calcul de I_1 et I_2 sont $T(r,q-1)$ et $T(r-1,p-q)$, donc le temps de calcul de I est $f(q)$. Comme I est optimale, q est dans E et son temps de calcul est $T(r,p)$.

La relation clé est $M_r^{k+1} = M_{r-1}^{k+1} + M_r^k$. En particulier, cela implique que $T(r,p)$ est bien défini, même si k n'est pas complètement défini par (4.1). Cela montre aussi que E n'est pas vide ssi k satisfait (4.1). En fait, E est un intervalle $[q_0, q_1]$.

La fonction $f(q)$ est linéaire par morceaux et continue. Si p n'est pas un entier, il existe un unique $k_{r,p}$ tel que (4.1) soit satisfait. Définisons $k_1 = k_{r,q-1}$ et $k_2 = k_{r-1,p-q}$. Alors $f(q) = (1 + k_1 - k_2)q + X$, avec $X = k_2(p-1) - M_{r+1}^{k_1} - M_r^{k_2}$. Sur E, on a $k_1 = k_{r,p} - 1$ et $k_2 = k_{r,p}$, de sorte que $f(q) = T(r,p)$. Sur tout intervalle ne contenant pas d'entier, f est différentiable, et sa dérivée est $1 + k_1 - k_2$. Il est clair que $f'(q) > 0$ si $q > q_1$ et $f'(q) < 0$ si $q < q_0$ (puisque $q > q_1$ est équivalent à $k_1 > k_{r,p} - 1$ ou $k_2 < k_{r,p}$). Cela montre que $f(q) > T(r,q)$ pour chaque q qui n'est pas dans E, et la continuité de f donne la conclusion.

□

4.5 Borne inférieure.

Théorème 4.8. *Pour un programme sans boucle inversant la suite* $u_0, ..., u_p$ *avec* r *registres en un temps* T*, on a*

$$(r+1)\,T = \Omega(p \ln^2 p)$$

De plus, $(r+1)\,T$ *est asymptotiquement plus grand que* $\frac{p \ln^2 p}{\ln^2 4}$

Cette borne inférieure est oprimale, car il existe des suites de longueur p *arbitrairement grande pour lesquelles* $(r+1)T$ *est équivalent à* $\frac{p \ln^2 p}{\ln^2 4}$.

Démonstration. On donne les grandes lignes de la preuve. En utilisant les notations du théorème précédent, on a $(r+1)T \geq (p+1)r(k-1)$ et $M_r^{k+1} \leq \frac{(r+k)^{r+k}}{r^r k^k}$. Soit k' tel que

$$f(f,k') = \frac{(r+k')^{r+k'}}{r^r\, k'^{k'}} = p+1$$

Alors $k' \leq k$. Maintenant, le problème est réduit à minimiser $r(k'-1)$ sachant $f(r,k') = p+1$, quand r varie. Une étude soigneuse mais élémentaire de ce problème de minimisation contrainte montre que le minimum est atteint pour $r = k' + \frac{\ln 2}{1-\ln 2} + \epsilon(k')$ avec $\lim_{p\to\infty} \epsilon(k') = 0$, et que la valeur minimale de $r(k'-1)$ est équivalente à $\frac{p \ln^2 p}{\ln^2 4}$. L'étude des suites de longueur $p = M_r^r - 1$ montre que cette borne inférieure est optimale. □

Chapitre 5
Algèbre en théorie des types.[1]

Dans le cadre de l'ARC[2] «Calcul formel certifié», de 1998 à 2000, j'ai développé une bibliothèque d'algèbre en Coq, qui fait maintenant partie des contributions au système[3]. Le but de ce travail était d'une part de développer des mathématiques de base en vue de preuves d'algorithmes de calcul formel et de développements de mathématiques plus avancées (géométrie algébrique), et d'autre part de rechercher comment représenter efficacement et confortablement les structures algébriques avec la théorie des types en général, et le calcul des constructions inductives en particulier.

Je me suis inspiré du livre «Algebra» de S.Lang, de la théorie SETS de la distribution standard de Coq, et du développement de la théorie des catégories en Coq par A.Saïbi. Ce développement a été grandement facilité par l'utilisation de l'interface Ctcoq, tant pour la conduite des preuves que pour la formalisation de notions souvent complexes en syntaxe Coq, et beaucoup plus compréhensibles lorsqu'elles sont affichées par PPML (le pretty-printer de Centaur/Ctcoq).

5.1 Contenu de la bibliothèque.

5.1.1 Ensembles

Comme le veut l'intuition la plus répandue, on identifie ensembles et types: un type est l'ensembles des termes qui l'ont pour type.

L'égalité par contre est différente pour les ensembles: ce n'est pas forcément l'égalité de Leibnitz, ce peut être n'importe quelle relation d'équivalence. Cela permet de simuler les types quotients, qui n'existent pas dans le CCI: pour passer au quotient, il suffit de changer l'égalité par la relation d'équivalence du quotient. Comme cela est fait dans le développement de Saïbi par exemple, on appellera sétoïde la structure constituée d'un type et d'une relation d'équivalence.

5.1.1.1 Sétoïdes, quotients.

C'est un type muni d'une relation d'équivalence:

```
Record Setoid: Type := {
  Carrier:>Type;
  Equal: (relation Carrier);
  Prf_equiv:> (equivalence Equal) }.
```

La fonction `Carrier` définit une coercion[4] de `Setoid` vers les types, ce qui permet d'identifier un sétoïde avec son type sous-jascent et d'écrire `(x:E)` dans

1. ce travail a fait l'objet d'une présentation orale à l'atelier du réseau de recherche européen «Types» à Göteborg en 2000.

2. action de recherche coordonnée, financées par la direction scientifique de l'INRIA

3. téléchargeable à `http://pauillac.inria.fr/coq/contribs-fra.html`

4. les coercions de Coq sont un mécanisme qui permet d'alléger l'écriture des termes. Une coercion est une fonction d'un type vers un autre qui permet de voir un terme du premier type comme ayant le second type. Cela permet une forme de sous-typage, puisque les coercions, une fois déclarées dans l'environnement, n'ont pas à être écrites dans les termes, elles sont généralement déterminées par le système. Par exemple, si `f:A->B` est une coercion, et `g: B->C` une fonction, on pourra écrire `[x:A](g x)` en lieu et place de `[x:A](g (f a))`.

Toutefois, afin d'assurer cette détermination sans ambiguïté, toute fonction ne peut devenir une coercion (voir la documentation de Coq pour plus de précisions).

`(E:Setoid)(x:E)(Equal x x).`
Désormais, «ensemble» sera synonyme de «sétoïde».

5.1.1.2 Applications.

Une application entre deux ensembles est une fonction compatible avec l'égalité.

```
Definition fun_compatible:
     [f:A ->B] (x, y:A) (Equal x y) ->(Equal (f x) (f y)).

Record Map: Type := {
  Ap:> A ->B;
  Map_compatible_prf:> (fun_compatible Ap)}.
```

Par la coercion Ap, une application (map en anglais) peut-être vue comme une fonction, et on pourra écrire par exemple `(f x)` dans
`(E,F:Setoid)(f:(Map E F))(x:E)(Equal (f x) (f x))`
Avec l'égalité extensionnelle[5], les applications de E dans F forment un sétoïde: `(MAP E F)`

5.1.1.3 Parties, sous-types.

Comme cela est fait dans la bibliothèque standard de Coq (le développement de Kahn, Huet et Prost dans theories/Sets), une partie d'un ensemble est définie comme sa fonction caractéristique, i.e. un prédicat. Mais ici les ensembles sont des sétoïdes, on va donc se restreindre aux prédicats compatibles avec l'égalité:

```
Definition pred_compatible : (E -> Prop) -> Prop :=
   [P : E -> Prop] (x, y : E) (P x) -> (Equal y x) -> (P y).

Record Predicate : Type := {
  Pred_fun: E -> Prop;
  Pred_compatible_prf: (pred_compatible Pred_fun)}.
```

En disant que deux parties sont égales ssi elles ont les mêmes éléments, on fait du type `(Predicate E)` un sétoïde: `(part_set E)`. C'est l'ensemble des parties d'un ensemble, sur lequel on définit les opérations booléennes usuelles: intersection, union, complément, différence, singleton, union/intersections de familles de parties d'un ensemble.
L'appartenance est ainsi définie:

```
Definition in_part := [E : Setoid] [x : E] [A : (Predicate E)] (Pred_fun A x).
```

Notons que dans les opérations booléennes sur les parties d'un ensembles, on se donne l'axiome du tiers-exclu, qui n'est pas prouvable dans le CCI: `(P:Prop)P\/~P`.

On voudrait aussi considérer une partie d'un ensemble comme un ensemble. Comme il n'y a pas de sous-types dans le CCI, il faut donc ruser. Une façon assez simple de représenter un sous-ensemble d'un ensemble (sétoïde) E est de se donner un type F et une fonction i de F dans E. Cela donne un sétoïde dont le support est le type F et la relation d'équivalence $=_F$ est donnée par $x =_F y \Leftrightarrow i(x) =_E i(y)$. Le sous-ensemble de E qu'on définit ainsi est alors l'image de la fonction i.

```
Definition subtype_image_equal : F -> F -> Prop :=
    [x : F] [y : F] (Equal (i x) (i y)).
Lemma subtype_image_equiv: (equivalence subtype_image_equal).
Definition subtype_image_set : Setoid := (Build_Setoid subtype_image_equiv).
```

5. deux fonctions sont extensionnellement égales quand leurs valeurs en chaque point sont égales.

On définit une structure pour ce genre de sétoïdes:

```
Record subtype_image : Type := {
  subtype_image_carrier: Type;
  subtype_image_inj:> subtype_image_carrier -> E }.

Definition set_of_subtype_image : subtype_image -> Setoid.:=
  [S : subtype_image] (subtype_image_set (!subtype_image_inj S)).
```

Finalement, on peut voir une partie P de E comme un sous-ensemble de E ainsi: c'est l'ensemble des couples (x, p) où x est de type E et p est une preuve de $P(x)$, deux couples étant égaux ssi ils ont même première coordonnée:

```
Definition part := (!Build_subtype_image subtype subtype_elt).
Coercion set_of_subtype_image : subtype_image>-> Setoid.
Coercion part : Predicate >-> subtype_image.
```

Les deux dernières coercions permettant de voir ne partie de E comme un sétoïde. On pourra écrire `(x:A)` dans:

```
(E:Setoid)(A:(part_set E))(x:A)(Equal x x).
```

Mais on ne pourra pas voir directement un élément d'une partie A de E comme un élément de E, car le système des coercions de Coq n'accepte pas que

```
subtype_elt : (E:Setoid; P:(Predicate E))(subtype P)->E
```

soit une coercion: son type d'arrivée est une variable.

Par exemple on ne pourra écrire:

```
(E:Setoid)(A:(part_set E))(x:A)(in_part x A).
```

il faudra écrire:

```
(E:Setoid)(A:(part_set E))(x:A)(in_part (subtype_elt x) A).
```

Finalement, on arrive à un point où on peut parler d'ensembles et de sous-ensembles dans le contexte des ensembles quotients (les sétoïdes) assez confortablement, avec toutefois un désagrément, qui est l'alourdissement des notations dû au fait que la fonction `subtype_elt` ne peut être une coercion.

5.1.1.4 Injections, surjections, bijections.

Sans surprise, on définit ces notions ainsi:

```
Definition injective: (MAP A B) ->Prop :=
  [f:(MAP A B)] (x, y:A) (Equal (f x) (f y)) ->(Equal x y).
Definition surjective: (MAP A B) ->Prop :=
  [f:(MAP A B)] (y:B)(exT ? [x:A](Equal y (f x))).
Definition bijective: (MAP A B) ->Prop :=
  [f:(MAP A B)](injective f) /\ (surjective f).
```

Toutefois, une application bijective n'admet pas forcément de réciproque. Plus généralement, une application surjective n'admet pas forcément de section: ce n'est pas parce qu'on peut prouver que tout élément a un antécédent qu'on peut le construire effectivement. Il faudrait pour cela se donner l'axiome du choix. Mais on s'est déjà donné le tiers-exclu, et, comme on le montre dans le chapitre 6, section 5, le tiers exclu et l'axiome du choix sont contradictoires dans Coq.

Il faut donc se faire à ce constat, du moins si on se plie à la discipline de Coq, car si on oublie la sorte `Set`, tous ces problèmes d'inconsistance mutuelle d'axiomes de mathématiques classiques semblent disparaître[6]. On reviendra dans le chapitre 6 sur ce problème des sections des applications surjectives, dans le cas des quotients.

5.1.1.5 Images, images réciproques, restrictions.

Ces notions sont intéressantes car elles mélangent applications (fonctions compatibles) et parties d'un ensemble vues comme sétoïdes, et consistent en un test assez représentatif de la souplesse des formalisations choisies pour ces notions.

6. à vérifier...

5.1.1.6 Théorème: il n'y a pas de surjection de E dans $\wp(E)$.

Il s'agit ici de formaliser le théorème de Cantor, ce qui se fait facilement: en supposant une surjection f de E dans $\wp(E)$, on définit l'ensemble $X = \{x: E | x \notin f(x)\}$, puis on prend un x tel que $f(x) = X$, et on montre que $x \in X \Leftrightarrow x \notin X$.

5.1.1.7 Théorème de Cantor-Bernstein.

On montre que s'il y a une injection de A dans B et une injection de B dans A, alors il y a une bijection de A dans B.

5.1.2 Catégories.

Il s'agit ici de définir les notions de base de la théorie des catégories. Pour un développement beaucoup plus poussé, on pourra consulter la thèse d'A.Saïbi, et sa contribution au système Coq, qui est disponible depuis la version 7 de Coq.

5.1.2.1 Définition des catégories, foncteurs.

Une catégorie est donnée par

- un type (celui de ses objets),
- un ensemble (sétoïde) de morphismes entre deux objets A et B: `(Hom A B)`,
- une opération de composition des morphismes associative, un morphisme identité jouant le rôle d'élément neutre pour la composition.

Cela donne le type suivant en Coq:

```
Record category: Type := {
  Ob:>Type;
  Hom: Ob -> Ob ->Setoid;
  Hom_comp: (a, b, c:Ob)(MAP (cart (Hom b c) (Hom a b)) (Hom a c));
  Hom_id: (a:Ob)(Hom a a);
  Hom_comp_assoc_prf: (Hom_comp_assoc Hom_comp);
  Hom_comp_unit_l_prf: (Hom_comp_unit_l Hom_comp Hom_id);
  Hom_comp_unit_r_prf: (Hom_comp_unit_r Hom_comp Hom_id) }.
```

Par exemple, la catégorie des ensembles a pour objets les sétoïdes, et pour morphismes les applications:

```
Definition SET: category.
 Apply (!Build_category
            Setoid
            MAP
            [E, F, G:Setoid](uncurry (!comp_map_map_compatible E F G)) Id);
 Red; Simpl; Unfold Map_eq; Auto with algebra.
Defined.
```

5.1.2.2 Sous-catégorie, sous-catégorie pleine.

On restreint le type des objets, et/ou l'ensemble des morphismes.

5.1.3 Semi-groupes, monoïdes, groupes.

Les structures algébriques sont obtenues par raffinements successifs de structures de base: un semi-groupe est un sétoïde avec une loi de composition interne associative, un monoïde est un semi-groupe avec un élément neutre, etc. Si on veut par exemple qu'un corps puisse être vu à la fois comme un type, un ensemble, un groupe commutatif, un anneau, de manière transparente dans l'écriture, il faut rendre compte dans la formalisation de cet *héritage* des structures.

La méthode qu'on a choisie utilise deux outils de Coq: les «record», et les coercions. Un record est un type inductif avec un seul constructeur dont les arguments sont nommés. Une coercion est une fonction entre deux types qui permet de voir le premier comme un sous-type du second.

Un autre problème posé par la formalisation des structures algébriques est celui du partage de sous-structure: par exemple un anneau est donné par un groupe commutatif (avec l'addition) et un monoïde (avec la multiplication), et quelques autres choses. Ces deux structures ne sont pas indépendantes: elles ont toutes les deux le même sétoïde sous-jacent. Pour rendre compte de ce partage de sous-structures à un niveau arbitraire il faut pouvoir paramétrer les structures algébriques: dire qu'on se donne par exemple un groupe sur un sétoïde donné. Mais il faut aussi pouvoir se donner un groupe dans l'absolu (sans avoir à spécifier l'ensemble sous-jascent).

Tout ceci conduit à une hiérarchie des structures dédoublée: une sans paramètre, l'autre avec paramètre, le passage de l'une à l'autre étant assuré par des coercions.

Pour préciser les choses voici le début de la hiérarchie:

```
Record sgroup_on[E:SET]: Type := {
 sgroup_law_map: (law_of_composition E);
 sgroup_assoc_prf: (associative sgroup_law_map) }.

Record sgroup: Type := {
 sgroup_set:> Setoid;
 sgroup_on_def:> (sgroup_on sgroup_set) }.

Coercion Build_sgroup : sgroup_on >->sgroup.
```

Une structure `sgroup_on` est paramétrée par un ensemble (sétoïde). Une structure sgroup est donnée par un sétoïde et une structure `sgroup_on` sur celui-ci. Les coercions permettent alors de passer de l'une à l'autre, ainsi que de voir un semi-groupe (paramétré ou non) comme un sétoïde:

$$\begin{array}{ccc} & & \text{Setoid} \\ & & \uparrow \\ \text{sgroup} & \longleftrightarrow & \text{sgroup\_on} \end{array}$$

Ajoutons les monoïdes:

```
Record monoid_on[A:sgroup]: Type := {
  monoid_unit: A;
  monoid_unit_r_prf: (unit_r (sgroup_law_map A) monoid_unit);
  monoid_unit_l_prf: (unit_l (sgroup_law_map A) monoid_unit) }.

Record monoid: Type := {
  monoid_sgroup:> sgroup;
  monoid_on_def:> (monoid_on monoid_sgroup) }.

Coercion Build_monoid : monoid_on >-> monoid.
```

Le graphe des coercions devient:

$$\begin{array}{ccc} & & \text{Setoid} \\ & & \uparrow \\ \text{sgroup} & \longleftrightarrow & \text{sgroup\_on} \\ \uparrow & & \\ \text{monoid} & \longleftrightarrow & \text{monoid\_on} \end{array}$$

Notons que le graphe des coercions que maintient Coq est transitif et non ambigu. On n'a pas, dans les graphes ci-dessus, représenté les arêtes transitives.

Passons maintenant aux groupes:

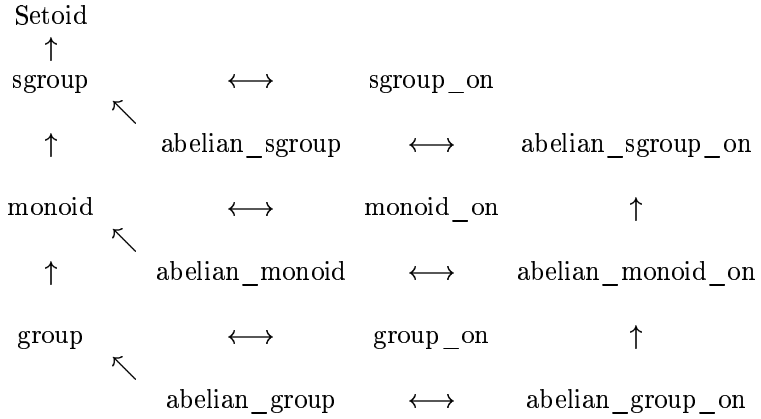
```
Record group_on[G:monoid]: Type := {
 group_inverse_map: (Map G G);
 group_inverse_r_prf: (inverse_r (sgroup_law_map G) (monoid_unit G)
                                 group_inverse_map);
 group_inverse_l_prf: (inverse_l (sgroup_law_map G) (monoid_unit G)
```

```
                                    group_inverse_map) }.
Record group: Type := {
 group_monoid:> monoid;
 group_on_def:> (group_on group_monoid) }.
Coercion Build_group : group_on >->  group.
```

Les structures abéliennes posent un nouveau problème: celui de l'héritage multiple. En effet, un groupe abélien peut-être défini comme un groupe qui a en plus la propriété d'être abélien, ou bien comme un monoïde abélien qui en plus est un groupe. Le fait qu'on ait deux façons de définir une structure (paramétrée ou non) va nous permettre d'utiliser ces deux définitions: par exemple, abelian_group va hériter de group, tandis que abelian_group_on va hériter de abelian_monoid_on. On obtient comme ceci un héritage multiple à peu de frais.

```
Record abelian_sgroup_on[A:sgroup]: Type := {
  abelian_sgroup_com_prf: (commutative (sgroup_law_map A)) }.

Record abelian_sgroup: Type := {
  abelian_sgroup_sgroup:> sgroup;
  abelian_sgroup_on_def:> (abelian_sgroup_on abelian_sgroup_sgroup) }.

Coercion Build_abelian_sgroup : abelian_sgroup_on>-> abelian_sgroup.

Record abelian_monoid_on[M:monoid]: Type := {
  abelian_monoid_abelian_sgroup:> (abelian_sgroup_on M) }.

Record abelian_monoid: Type := {
  abelian_monoid_monoid:> monoid;
 abelian_monoid_on_def:> (abelian_monoid_on abelian_monoid_monoid) }.

Coercion Build_abelian_monoid : abelian_monoid_on>-> abelian_monoid.

Record abelian_group_on[G:group]: Type := {
  abelian_group_abelian_monoid:> (abelian_monoid_on G) }.

Record abelian_group: Type := {
  abelian_group_group:> group;
  abelian_group_on_def:> (abelian_group_on abelian_group_group) }.

Coercion Build_abelian_group : abelian_group_on>->abelian_group.
```

On obtient alors le graphe de coercion suivant:



5.1.3.1 Morphismes de groupes, catégorie des groupes, etc.

Comme on l'a fait avec les structures, les morphismes vont être définis par raffinement successifs: un morphisme de groupe est un morphisme de monoïde, qui est un morphisme de semi-groupes. De même pour les catégories: la catégorie des groupes est définie à partir de celle des monoïdes, etc.

5.1.3.2 Sous-structures (sous-groupes, etc).

Soit G un semi-groupe. Un sous-semi-groupe de G sera donné par une partie H de G vérifiant que la loi de composition de G est interne à H.

```
Record subsgroup: Type := {
 subsgroup_part: (Predicate G);
 subsgroup_prop: (x, y:G) (in_part x subsgroup_part) ->
                          (in_part y subsgroup_part)
                          -> (in_part (sgroup_law ? x y) subsgroup_part) }.
...
Coercion sgroup_of_subsgroup : subsgroup>-> sgroup.
Coercion subsgroup_part : subsgroup>-> Predicate.
```

En restreignant la loi de G à H, on montre que H est un semi-groupe.

En ajoutant les coercions appropriées, un sous-semi-groupe pourra être vu comme une partie de G et comme un semi-groupe.

On procède de même pour les sous-monoïdes et les sous-groupes, en utilisant dans ce cas un héritage: un sous-groupe est un sous-monoïde, qui est un sous-semi-groupe, etc.

5.1.3.3 Structures libres (groupe libre, etc).

Prenons l'exemple du monoïde libre en gendré par un ensemble V (un sétoïde).

Son type support sera le type inductif suivant:

```
Mutual Inductive FM: Type :=
  Var: V ->FM
| Law: FM -> FM -> FM
| Unit: FM.
```

C'est-à-dire les expressions formelles construites avec les éléments de V, une opération binaire `Law` et une constante `Unit`.Pour faire de ce type un setoïde, on quotiente par la relation d'équivalence suivante, définie par les axiomes de monoïde:

```
Mutual Inductive eqFM: FM -> FM ->Prop :=
      eqFM_Var: (x, y:V) (Equal x y) ->(eqFM (Var x) (Var y))
    | eqFM_law:
        (x, x', y, y':FM) (eqFM x x') -> (eqFM y y') ->
         (eqFM (Law x y) (Law x' y'))
    | eqFM_law_assoc:
        (x, y, z:FM)(eqFM (Law (Law x y) z) (Law x (Law y z)))
    | eqFM_law0r: (x:FM)(eqFM (Law x Unit) x)
    | eqFM_law0l: (x:FM)(eqFM (Law Unit x) x)
    | eqFM_refl: (x:FM)(eqFM x x)
    | eqFM_sym: (x, y:FM) (eqFM x y) ->(eqFM y x)
    | eqFM_trans: (x, y, z:FM) (eqFM x y) -> (eqFM y z) ->(eqFM x z).
```

Il reste ensuite à montrer que ce sétoïde est bien un monoïde, et qu'il vérifie la propriété universelle du monoïde libre engendré par V:

$$\forall M\colon \mathrm{MONOID}, \forall f\colon V \to M, \exists \tilde{f}\colon \mathrm{Hom}(\mathrm{FreeMonoid}, M), f = \tilde{f} \circ \mathrm{Var}$$

$$\begin{array}{ccc} V & \longrightarrow_f & M \\ & \searrow_{\mathrm{Var}} & \uparrow \tilde{f} \\ & & \mathrm{FreeMonoid} \end{array}$$

On a défini de cette façon le monoïde libre, le monoïde abélien libre, le groupe libre et le groupe abélien libre.

5.1.3.4 Groupes quotients, noyaux, co-noyaux.

Sous-groupes distingués, groupe quotient.

On a défini les noyaux et co-noyaux(images) de morphismes pour les monoïdes et les groupes.

5.1.3.5 Décomposition canonique d'un morphisme de groupes.

Tout morphisme de groupe $f: G \to G'$ se décompose canoniquement ainsi:

$$\begin{array}{ccc} G & \longrightarrow_f & G' \\ \downarrow & & \uparrow \\ \dfrac{G}{\mathrm{Ker}\, f} & \longrightarrow & \mathrm{coKer}\, f \end{array}$$

5.1.3.6 Sous-structures engendrées par une partie.

Prenons l'exemple du monoïde engendré par une partie A d'un monoïde M. C'est le plus petit sous-monoïde de M qui contient A. Mais cette définition n'est pas très constructive. Plus constructivement, on peut le voir comme l'ensemble des expressions construites avec les éléments de A, la loi de M, et son élément neutre, ensuite évaluées dans M. C'est donc le monoïde libre engendré par A, envoyé par la fonction $f \mapsto \tilde{f}$ de la propriété universelle du monoïde libre appliquée au monoïde M et à l'injection canonique de A dans M:

```
Definition generated_monoid: (submonoid M) :=
   (image_monoid_hom (FM_lift (inj_part A))).
```

Voilà une définition rapide!

On peut montrer que c'est bien le plus petit sous-monoïde de M qui contient A:

```
Lemma generated_monoid_minimal:
  (M:MONOID) (A:(part_set M)) (H:(submonoid M))
     (included A H) ->(included (generated_monoid A) H).
```

On procède de même pour le sous-groupe engendré par une partie d'un groupe.

5.1.3.7 Monoïde des endomorphismes d'un ensemble.

5.1.3.8 Opération d'un monoïde sur un ensemble.

C'est un morphisme d'un monoïde vers le monoïde des endomorphismes d'un ensemble.

5.1.4 Anneaux.

Un anneau est une double structure sur un sétoïde: un groupe abélien, et un monoïde (les anneaux seront considérés comme unitaires).

```
Record ring_on[R:abelian_group]: Type := {
   ring_mult_sgroup: (sgroup_on R);
   ring_mult_monoid: (monoid_on ring_mult_sgroup);
   ring_monoid:> (monoid_on ring_mult_monoid);
   ring_dist_r_prf:
     (dist_r (sgroup_law_map ring_mult_sgroup) (sgroup_law_map R));
   ring_dist_l_prf:
     (dist_l (sgroup_law_map ring_mult_sgroup) (sgroup_law_map R)) }.
```

```
Record ring: Type := {
   ring_group:> abelian_group;
   ring_on_def:> (ring_on ring_group) }.

Coercion Build_ring : ring_on >-> ring.
```

Anneaux commutatifs:

```
Record cring_on[R:ring]: Type := {
   cring_com_prf: (commutative (sgroup_law_map (ring_mult_monoid R))) }.

Record cring: Type := {
   cring_ring:> ring;
   cring_on_def:> (cring_on cring_ring) }.

Coercion Build_cring : cring_on >-> cring.
```

Il nous faut ajouter une coercion qui voit un anneau commutatif comme un monoïde abélien pour la multiplication, la coercion `cring ->abelian_monoid` que le système déduit concerne l'addition:

```
Definition cring_monoid: cring ->abelian_monoid.
...
```

On définit ensuite naturellement les morphismes d'anneaux et les catégories des anneaux et des anneaux commutatifs.

5.1.5 Modules.

Un module sur un anneau R est un groupe abélien avec une opération externe de R.

Sans surprise, on définit la catégorie des modules, les sous-modules, le module libre, le sous-module engendré par une partie (comme pour le groupe engendré, en utilisant la propriété universelle du module libre), et enfin le module des morphismes de deux modules.

5.1.6 Algèbres.

On définit une algèbre sur un anneau R comme un module sur R avec une application bilinéaire (la multiplication):

```
Variable R:CRING.

Record algebra_on[Mod:(MODULE R)]: Type := {
   algebra_bilinear_map: (Hom_module Mod (Hom_module Mod Mod)) }.

Record algebra: Type := {
   algebra_carrier:> (module R);
   algebra_on_def:> (algebra_on algebra_carrier) }.
Coercion Build_algebra : algebra_on >-> algebra.
```

Puis on définit les algèbres associatives avec unité, qu'on peut voir comme des anneaux (on définit alors cette coercion).

5.1.7 Idéaux d'un anneau.

Un idéal d'un anneau R est un sous-groupe de R absorbant pour la multiplication.

5.1.7.1 Idéal engendré par une partie.

On peut voir R comme un R module. L'idéal engendré par un partie A de R est alors le sous-module de R engendré par A.

On vérifie que c'est le plus petit idéal contenant A.

5.1.7.2 Anneaux intègres.

On définit un anneau intègre comme un anneau commutatif tel que $x{\neq}0, y{\neq}0 \Rightarrow x\,y{\neq}0$.

5.1.8 Corps.

Un corps est un anneau muni d'un inverse pour la multiplication. L'application inverse est définie sur tout le corps, mais ses propriétés concernent les éléments non nuls:

```
Record field_on[R:ring]: Type := {
   field_inverse_map: (Map R R);
   field_inverse_r_prf:
     (x:R) ~ (Equal x (monoid_unit R)) ->
     (Equal (ring_mult x (Ap field_inverse_map x)) (ring_unit R));
   field_inverse_l_prf:
     (x:R) ~ (Equal x (monoid_unit R)) ->
     (Equal (ring_mult (Ap field_inverse_map x) x) (ring_unit R));
   field_unit_non_zero: (~ (Equal (ring_unit R) (monoid_unit R))::Prop) }.

Record field: Type := {
   field_ring:> ring;
   field_on_def:> (field_on field_ring) }.

Coercion Build_field : field_on >-> field.
```

Corps commutatifs:

```
Record cfield: Type := {
   cfield_ring: cring;
   cfield_on_def:> (field_on cfield_ring) }.

Coercion cfield_ring : cfield >-> cring.
```

5.1.8.1 Corps des fractions d'un anneau intègre.

On définit le corps des fractions d'un anneau intègre qui vérifie $1{\neq}0$, avec comme type support pour les fractions:

```
Record fraction: Type := {
   num: R;
   den: R;
   den_prf: ~ (Equal den (monoid_unit R)) }.
```

Une fraction est donc constiturie d'un numérateur, d'un dénominateur et d'une preuve que celui-ci est non nul.

Pour pouvoir définir l'inverse d'une fraction, on doit de plus supposer que l'égalité à 0 dans l'anneau est décidable:

```
Variable zero_dec:
   (x:R){(Equal x (monoid_unit R))}+{~ (Equal x (monoid_unit R))}.
```

5.1.9 Nombres.

5.1.9.1 $\mathbb{Z}$: l'anneau commutatif intègre des entiers.

À partir du type Z de la librairie de Coq (entiers en base 2), on construit l'anneau intègre ZZ. Puis on définit l'injection canonique de ZZ dans un groupe quelconque G:

```
Definition Z_to_group: (Hom (ZZ::GROUP) G).
...
```

et l'injection canonique de ZZ dans un anneau quelconque R:

```
Definition Z_to_ring: (Hom (ZZ::RING) R).
...
```

ainsi que sa propriété caractéristique.

5.1.9.2 $\mathbb{Q}$: le corps des nombres rationnels.

Il est construit comme le corps des fractions de l'anneau intègre des entiers ZZ:

```
Definition Q := (fraction_cfield Z_one_diff_zero Zzero_dec).
```

5.1.9.3 Le corps des «nombres complexes» construit à partir d'un corps commutatif R vérifiant: $\forall x, y \in R, x^2 + y^2 = 0 \Rightarrow x = 0$ et $y = 0$.

Soit R un corps commutatif. On définit les complexes sur R comme des couples (partie réelle, partie imaginaire):

```
Variable R:CFIELD.

Record Ctype: Type := {
   real:> R;
   imag: R }.
```

Avec les opérations d'addition et de multiplication attendues, on en fait un anneau commutatif.

On définit l'inverse ainsi:

$$\frac{1}{z} = \frac{\bar{z}}{z\,\bar{z}}$$

Pour montrer ses propriétés, on a besoin de l'hypothèse suivante:

```
Hypothesis sum_of_square:
           (x, y:R)
           (Equal
              (sgroup_law R (ring_mult x x) (ring_mult y y)) (monoid_unit R))
           ->(Equal x (monoid_unit R)) /\ (Equal y (monoid_unit R)).
```

C'est-à-dire $\forall x, y \in R, x^2 + y^2 = 0 \Rightarrow x = 0$ et $y = 0$.

Finalement, on obtient le corps commutatif CC(R).

5.1.10 Ensembles finis.

On a ici repris le développement de Kahn, Huet et Prost sur les ensembles finis, qui fonctionne avec l'égalité de Leibnitz, en l'adaptant aux sétoïdes.

5.1.10.1 Cardinaux finis, parties finies d'un ensemble, ensembles finis.

Les cardinaux finis sont des entiers naturels et sont définis sur les parties d'un ensemble par récurrence, en utilisant la fonction qui ajoute un élément à une partie:

```
Definition add_part := [A:(part_set E)] [x:E](union A (single x)).

Mutual Inductive cardinal: (part_set E) -> nat ->Prop :=
     cardinal_empty: (A:(part_set E)) (Equal A (empty E)) ->(cardinal A O)
```

```
| cardinal_add:
    (A, B:(part_set E)) (n:nat) (cardinal B n) ->
    (x:E) ~ (in_part x B) -> (Equal A (add_part B x)) ->(cardinal A (S n))
```

.

On montre en particulier que:

- le cardinal d'une partie est unique,
- le cardinal de $A \cup B$ est égal à la somme des cardinaux de A et B moins le cardinal de $A \cap B$,
- quelques principes de récurrence adaptés aux cardinaux finis.

On dit qu'un ensemble est fini ssi sa partie maximale a un cardinal fini:

```
Definition finite := [A:Setoid](exT ? [n:nat](cardinal (full A) n)).
```

5.1.10.2 Principe de Dirichlet (lemme des tiroirs).

On montre un des outils privilégiés des démonstrations sur les ensembles finis, cas particulier du théorème de Ramsey:

«si on met 5 chaussettes dans 4 tiroirs, il y a un tiroir avec au moins deux chaussettes»

```
Lemma tiroirs:
  (E, F:Setoid)
  (f:(MAP E F))
  (n:nat) (Chaussettes:(part_set E)) (cardinal Chaussettes n) ->
  (m:nat) (Tiroirs:(part_set F)) (cardinal Tiroirs m) ->
  (lt m n) -> ((x:E) (in_part x Chaussettes) ->(in_part (f x) Tiroirs))
 ->
  (ExT [x:E] (ExT [y:E] ~ (Equal x y) /\ (Equal (f x) (f y)))).
```

5.1.10.3 Applications entre ensembles finis.

On montre diverses propriétés des applications entre ensembles finis, par exemple:
si une application entre deux ensembles de même cardinal est surjective (resp. injective), alors elle est injective (resp. surjective).

5.2 Usage, travaux similaires, futur.

Cette bibliothèque a eu jusqu'à présent au moins deux usages :

1. elle a servi de base au travail de Laurent Chicli dans son développement de topologie et de géométrie algébrique, dont le but était de définir les shémas affines, pour l'étude de la méthode d'Horace. Dans ce cadre, elle s'est avérée tout à fait utile, avec toutefois deux inconvénients:
   - l'absence de coercion entre un sous-ensemble et son sur-ensemble, problème technique qui pourrait être résolu par une amélioration de l'algorithme d'inférence des coercions de Coq.
   - la lourdeur dans la définition des fonctions entre ensembles: il faut à chaque fois démontrer que la fonction qu'on définit est compatible avec les égalites des ensembles, ce qui est contraire à la pratique mathématique, où on dispose de quotients: une fois qu'on travaille dans un ensemble quotient, on oublie la relation d'équivalence et on ne travaille plus qu'avec l'égalité de Leibnitz, ce qui permet de définir les fonctions comme des abstractions quelconques, puiqu'elles sont automatiquement compatibles avec l'égalité de Leibnitz.

C'est pour cela que l'on s'est intéressé, à la suite de ce développement d'algèbre basé sur les sétoïdes, aux types quotients dans le CCI (voir chapitre 6). Néanmoins, on n'a pas, pour l'instant de types quotients suffisamment souples pour qu'il soit raisonnable de reprendre ce développement dans ce cadre: les quotients sont axiomatiques, donc inaccessibles aux calculs, et de plus ne peuvent être que strictement plus faibles que les quotients mathématiques dans l'état actuel de la théorie de Coq (ils ne peuvent contenir de fonction de choix dans les classes d'équivalence sous peine d'incohérence avec l'imprédicativité de Set).

2. elle a servi de base aux développements de Frédérique Guilhot dans le cadre de son travail de formalisation des mathématiques de premier cycle avec André Hirschowitz, et de la géométrie du lycée. Dans le premier cas, cela a concerné le calcul matriciel et vectoriel, le calcul sur les nombres complexes. Dans le second, il s'agissait de la géométrie plane. Ces deux domaines sont formalisables correctement avec la bibliothèque d'algèbre présentée, modulo le développement de tactiques de réécriture adaptées aux sétoïdes.

En Coq, il y a, à ma connaissance, un travail similaire[7], qui est celui de l'équipe de Nijmegen qui a démontré le théorème de d'Alembert-Gauss: tout polynôme non constant a une racine dans $\mathbb{C}$.

Ils ont pour cela développé une librairie des structures algébriques similaire à celle présentée ici, mais avec une définition constructive des sétoïdes: au lieu de donner une égalité/relation d'équivalence, ils donnent une relation de séparation, et définissent ensuite l'égalité à l'aide de cette relation. Ils s'interdisent ensuite tout usage d'axiome non constructif, comme le tiers exclu.

C'est une différence fondamentale avec notre développement, dans lequel on a voulu au contraire suivre au mieux la pratique mathématique, même si l'absence de quotients et de sous-types dans le CCI a conduit à des lourdeurs difficilement acceptables.

Dans le futur, notre bibliothèque devrait se développer dans une direction plus algorithmique avec le calcul polynomial (on a déjà presque les polynômes, il suffit de définir l'algèbre libre). Le calcul matriciel devrait aussi y être intégré, avec le calcul vectoriel.

7. voir `http://www.cs.kun.nl/gi/projects/fta/`

Chapitre 6

Élimination des quantificateurs sur les réels dans Coq[1]

6.1 Introduction

Le système Coq permet maintenant de faire des preuves formelles en analyse réelle. Mais les preuves dans ce domaine sont encore très pénibles, car très peu automatisées, en particulier celles qui manipulent les inégalités. Dans le cas des équations et inéquations linéaires, la tactique Fourier permet au système de faire lui-même les preuves, mais cette tactique ne s'applique pas dans le cas de systèmes d'équations et d'inéquations polynômiales. Pourtant il existe pour ce cas des procédures de décision efficaces et simples, au moins en degré faible et avec un nombre de variables ne dépassant pas la dizaine.

Le but du travail qu'on présente ici est d'implémenter en Ocaml une telle procédure sous la forme d'une tactique de Coq. Elle pourra être très utile pour les utilisateurs de Coq qui travailleront avec des nombres réels. On peut pour s'en convaincre constater que de très nombreuses preuves faites dans la bibliothèque Reals de Coq reviennent à montrer qu'un système d'inéquations polynômiales n'a pas de solution (par exemple dans les calculs de limites, de continuité).

La méthode qu'on a choisie[2] est une méthode d'élimination des quantificateurs dans les corps réels clos, basée sur le principe de Tarski-Seidenberg, et est exposée dans le livre de J. Bochniak, M. Coste, M-F. Roy «*Géométrie Algébrique Réelle*» Springer-Verlag (1986), pages 15-19. Basée sur une démonstration de Hormander[3], qui lui-même l'attribue à Cohen et Tarski, elle a l'avantage d'être assez simple à programmer, relativement efficace, et de ne faire intervenir que des théorèmes d'analyse réelle élémentaire (essentiellement le théorème des valeurs intermédiaires, par ailleurs déjà prouvé en Coq[4]).

En pratique la tactique suit les modèles des tactiques Omega et Fourier. On commence par traduire les objets mathématiques de Coq qui interviennent dans les hypothèses et dans le but qu'on veut prouver dans des types de données de Ocaml, puis on effectue le calcul de la procédure de décision en Ocaml. Si celle-ci réussit, on construit alors à partir d'une trace des calculs effectués une preuve dans le formalisme de Coq, qui est ensuite vérifiée par le système.

Si cette tactique s'avère utile, on pourra ensuite envisager d'utiliser la technique de la réflexion pour programmer et vérifier dans Coq lui-même la procédure de décision utilisée (ce qui reviendrait à prouver dans Coq la procédure elle-même alors qu'ici on génère une preuve pour chaque entrée). Mais ce n'est pas encore à l'ordre du jour, d'autant que la programmation de la méthode de Hormander a fait apparaître de nombreuses améliorations algorithmiques possibles : l'algorithme n'est de fait pas encore stabilisé.

---

1. avec Assia Mahboubi, Journées Francophones des Langages Applicatifs, 2002, et "*Proofs of polynomial inequalities in Coq*", exposé au séminaire«Verification and constructive algebra», Dagstuhl, janvier 2003.

2. En 2000, on avait testé, lors du stage ENS Lyon de F.R. Sinot, une autre méthode d'élimination des quantificateurs dans les réels, celle de Kreisel et Krivine, mais, bien qu'assez simple à mettre en oeuvre, elle était vraiment trop lente. Elle nécessitait en outre quelques corrections non triviales dans son énoncé original pour assurer la terminaison de la procédure. Les procédures efficaces d'élimination des quantificateurs dans les réels comme la décomposition cylindrique ont été écartées car elle nécessitent une implémentation assez lourde mais surtout des démonstrations qui font intervenir des mathématiques non triviales, comme les extensions algébriques, qui ne sont pas encore formalisées en Coq, et qui prendront du temps pour l'être.

3. L. Hörmander: *The analysis of linear partial differental operators, vol.2*, Springer-Verlag (1986).

4. M. Hirschowitz, M. Chiaverini: *Preuve en Coq du théorème des valeurs intermédiaires*, projet de maîtrise MIM, UNSA, printemps 2000.

Le plan de ce chapitre est le suivant : on commence par exposer la procédure d'élimination des quantificateurs de Hormander, qui permet essentiellement de calculer tous les signes possibles d'une famille de polynômes en plusieurs variables, et on donne un exemple d'application de cette méthode. Puis on décrit quelles traces des calculs on utilise. Ensuite, on montre comment utiliser cette trace des calculs pour effectuer l'élimination des quantificateurs universels. Enfin on montre comment reconstruire une preuve Coq à partir de ces traces des calculs.

6.2 Tableaux des signes d'une famille de polynômes.

Soit $f = \{f_1, ..., f_s\}$ une famille de polynômes de $\mathbb{R}[X_1, ..., X_n]$. On définit le signe d'un réel par $-$, 0 ou $+$ selon qu'il est négatif, nul ou positif. Un tableau de signes w de f, pour des valeurs $x_1, ..., x_{n-1}$ des $n-1$ premières variables, est donné par une subdivision $y_1 < ... < y_p$ de $]-\infty; +\infty[$ telle que pour tout intervalle $I \in \{]-\infty; y_1[, [y_1],]y_1; y_2[, [y_2], ...,]y_p; +\infty[\}$, pour tout $j \in \{1...s\}$, et pour tout $x \in I$, $f_j(x_1, ..., x_{n-1}, x)$ a pour signe $w(I, j)$. Lorsque le tableau n'a qu'une seule colonne, tous les polynômes de la famille ont un signe constant sur $]-\infty; +\infty[$

Par exemple, un tableau de signes pour $\{X_3^2 + X_1X_3 + X_2, X_1^2 - 4X_2\}$ est

	$]-\infty; y_1[$	$[y_1]$	$]y_1; y_2[$	$[y_2]$	$]y_2; +\infty[$
$X_3^2 + X_1X_3 + X_2$	$+$	0	$-$	0	$+$
$X_1^2 - 4X_2$	$+$	$+$	$+$	$+$	$+$

il donne les signes de ces deux polynômes pour X_3 variant entre $-\infty$ et $+\infty$,

avec $y_1 = \dfrac{-x_1 - \sqrt{x_1^2 - 4x_2}}{2}$, et $y_2 = \dfrac{-x_1 + \sqrt{x_1^2 - 4x_2}}{2}$.

Il n'y a que deux autres tableaux possibles pour ces polynômes :

	$]-\infty; +\infty[$
$X_3^2 + X_1X_3 + X_2$	$+$
$X_1^2 - 4X_2$	$-$

	$]-\infty; y_1[$	$[y_1]$	$]y_1; +\infty[$
$X_3^2 + X_1X_3 + X_2$	$+$	0	$+$
$X_1^2 - 4X_2$	0	0	0

avec $y1 = \dfrac{-x_1}{2}$.

On a donc calculé dans cet exemple le signe du trinôme du second degré en fonction du signe de son discriminant et dans ce cas simple, on sait exprimer les y_i en fonction des x_j.

Le but de ce qui suit est de calculer tous les tableaux de signes d'une famille f, correspondant aux valeurs possibles des $n-1$ premières variables.

6.2.1 Cas d'une variable: $\mathbb{Z}[X]$.

On traite ici le cas de polynômes de $\mathbb{Z}[X]$, car ce qui suit s'étend immédiatement à la fois à $\mathbb{Z}[X]$ et à $\mathbb{Z}[X_1, ..., X_{n-1}][X_n]$. En effet, l'outil principal qu'on utilise est la pseudo-division euclidienne.

Supposons les polynômes de f non identiquement nuls et f_s non constant, de degré maximal (si ce n'est pas le cas, tous les polynômes sont constants, leur tableau de signes est trivial).

Soient $a_1, ..., a_{s-1}$ les coefficients dominants de $f_1, ..., f_{s-1}$ et a_s celui de f_s'. Par pseudo-division euclidienne, on obtient:

$$\forall i, 1 \leq i \leq s-1,\ c_i f_s = q_i f_i + g_i, \deg(g_i) < \deg(f_i),\ c_s f_s = q_s f_s' + g_s, \deg(g_s) < \deg(f_s')$$

avec $\forall i, c_i > 0$ et c_i divise une puissance de a_i.

Soient w' le tableau de signe de $\{f_1, ..., f_{s-1}, f_s', g_1, ..., g_s\}$ et $x_1 < ... < x_N$ sa subdivision.

Soient $z_1 < ... < z_p$ les x_i qui sont racines de $f_1, ..., f_{s-1}$ ou f_s'. Des équations de pseudo-division, on déduit qu'en un z_i racine de f_j (resp f_s'), f_s a le signe de g_j (resp g_s). Entre les z_i, le théorème des valeurs intermédiaires donne éventuellement des racines de f_s, une au plus par intervalle, f_s' y étant non nulle et de signe constant.

Par exemple, supposons que z_1 est racine de f_j (soit $w'([z_1], j) = 0$), z_2 est racine de f_k (soit $w'([z_2], k) = 0$), et que f'_s est positive sur $]-\infty; x_1[$ (soit $w'(]-\infty; x_1[, s) = +$). Alors

- Si $w'([z_1], s+j) = +$ et $w'([z_2], s+k) = -$, alors il existe un unique $y \in]z_1; z_2[$ tel que $f_s(y) = 0$. En effet f'_s ne s'annule pas sur $]z_1; z_2[$, donc comme elle est continue, elle garde un signe constant. Or $g_j(z_1) > 0$ et $g_k(z_2) < 0$, donc $f_s(z_1) > 0$ et $f_s(z_2) < 0$ et comme f_s est une fonction continue strictement monotone sur $]z_1; z_2[$, le théorème des valeurs intermédiaires assure le résultat.
- Si $w'([z_1], s+j) = +$ et $w'([z_2], s+k) \neq -$, alors on peut déduire que f_s ne s'annule pas sur $]z_1; z_2[$, car elle est strictement monotone et continue.
- Si $w'([z_1], s+j) = +$, alors il existe un unique $y \in]-\infty; z_1[$ tel que $f_s(y) = 0$. En effet f'_s est positive en $-\infty$, non nulle avant z_1, $g_j(z_1)$ est positif donc $f_s(z_1)$ aussi. Finalement f_s est un polynôme strictement croissant sur $]-\infty; z_1[$, positif strictement en z_1 et le théorème des valeurs intermédiaires assure le résultat.
- Si $w'([z_1], s+j) \neq +$, alors on peut déduire que f_s ne s'annule pas sur $]-\infty; z_1[$.

On peut voir que tous les autres cas se traitent de la même manière.

Soient $y_1 < ... < y_M$ la réunion des racines de f_s données par le théorème des valeurs intermédiaires et des z_i non racines de f'_s. Ces points donnent la subdivision du tableau w. Les signes de f_1, ..., f_{s-1} sont tirés directement de w'. Ceux de f_s sont déduits de ceux de f'_s et des g_j dans w'.

En itérant ce processus, on se ramène fatalement à une famille de polynômes constants, dont le tableau de signes est trivial. En effet, en appliquant la transformation qui fait passer de $\{f_1, ..., f_s\}$ à $\{f_1, ..., f_{s-1}, f'_s, g_1, ..., g_s\}$, soit le degré maximum des polynômes décroît, soit il stagne mais alors le nombre de polynômes qui ont ce degré décroît.

6.2.2 Cas de plusieurs variables: $\mathbb{R}[X_1]...[X_n]$.

La méthode générale est basée sur celle du cas à une variable, en ajoutant le traitement de la nullité éventuelle des coefficients dominants non constants. En effet, les coefficients des polynômes sont maintenant des polynômes en $X_1, ..., X_{n-1}$, donc pouvant être nuls ou non selon les valeurs de ces variables.

On effectue les calculs en gardant en mémoire une famille $\mathcal{N}$ de polynômes qu'on suppose non nuls. Ce seront en fait des facteurs des coefficients dominants qui interviennent dans les pseudo-divisions. Au départ, $\mathcal{N}$ est vide.

Lorsque tous les polynômes de f sont constants en X_n, on calcule les tableaux de signes possibles de f pour la variable suivante (X_{n-1}). De ceux-ci on retient les colonnes de signes où les polynômes de f qui divisent un des polynômes de $\mathcal{N}$ sont effectivement non nuls.

Lorsqu'un des polynômes de f est non constant, on suit la procédure suivante :

* soit c le premier coefficient dominant non constant de $f_1, ..., f_s$, qui ne divise pas un des polynômes de $\mathcal{N}$, i.e. qui peut être nul quand les polynômes de $\mathcal{N}$ sont non nuls; soit $f_i = c\,x^d + r$ le polynôme dont c est le coefficient dominant; on traite alors les deux cas :
  - $\{f_1, ..., f_{i-1}, f_i, f_{i+1}, ..., f_s\}$ en ajoutant les facteurs sans carrés [5] de c à $\mathcal{N}$.
  - $\{f_1, ..., f_{i-1}, r, f_{i+1}, ..., f_s, c\}$ dont on ne garde que les tableaux de signes ou c est identiquement nul.
* soit il n'y a pas de tel c. Dans ce cas lorsque les polynômes de $\mathcal{N}$ sont non nuls, tous les coefficients dominants des polynômes de f sont non nuls, et on peut appliquer alors la méthode à une variable (dérivation et pseudo-divisions) car les hypothèses sont vérifiées.

5. si $P = \prod (h_i)^i$, avec les h_i non constants en X_n, premiers entre eux, de contenu 1 - i.e. leurs coefficients sont premiers entre eux- et sans facteurs carrés - i.e. premiers avec leur dérivée -, alors les facteurs sans carrés de P sont les h_i, plus les facteurs sans carrés de son contenu.

6.2.3 Améliorations

On accélère considérablement l'algorithme en factorisant les polynômes avec la factorisation sans carrés qui a l'avantage d'être rapide à calculer (uniquement des pgcd de polynômes à plusieurs variables, que l'on calcule avec l'algorithme des sous-résultants). Il est alors immédiat de calculer les tableaux des signes de f si on connaît ceux de la famille de ses facteurs.

L'implémentation qu'on a faite en Ocaml traite les polynômes de $\mathbb{Z}[X_1, ..., X_n]$ [6]. En effet, dans ce cas, tous les calculs ne produisent que des polynômes dans cet anneau.

6.2.4 Un exemple

Soit $f = \{XY^3 + Y^2\}$, polynôme de $\mathbb{R}[X][Y]$. Au départ, $\mathcal{N} = \{\}$.

La factorisation sans carrés de f donne deux facteurs, qui constituent la nouvelle famille $f = \{XY + 1, Y\}$. Etudions ses coefficients dominants non constants :

$X \neq 0$: maintenant $\mathcal{N} = \{X\}$, et tous les coefficients dominants de f sont non nuls. Appliquons la méthode à une variable :

$f_1 = Y$, $f_2 = XY + 1$ puis $f_2' = X$, $g_1 = 0$, $g_2 = 1$ La factorisation sans carré élimine les polynômes constants: la famille courante f devient $\{Y, X\}$.
Les coefficients dominants sont tous non nuls : on peut dériver et on obtient la famille $\{X, 1, 0, 0\}$. En factorisant, on est ramené à la famille $\{X\}$, constante en Y. On passe donc à la variable suivante, X. En dérivant on obtient $\{1, 0\}$, dont le tableau de signes est

1	+
0	0

D'où celui de $\{X\}$ qui est

X	−	0	+

Il y a donc trois tableaux de signes possibles pour $\{X\}$ considérée maintenant comme famille de polynômes (constants) en Y:

X	−

X	0

X	+

On n'en retient que les cas où X est non nul, puisque X divise un des polynômes de $\mathcal{N} = \{X\}$. Restent donc deux tableaux:

X	−

X	+

En remontant à la famille $\{Y, X\}$, on a donc deux tableaux:

Y	−	0	+
X	−	−	−

Y	−	0	+
X	+	+	+

Puis à $\{XY + 1, Y\}$, qui a deux tableaux de signes:

$XY + 1$	+	+	+	0	−
Y	−	0	+	+	+

$XY + 1$	−	0	+	+	+
Y	−	−	−	0	+

ce qui achève le cas $X \neq 0$.

$X = 0$: $\mathcal{N} = \{\}$ et la famille courante devient $\{1, Y\}$ puis $\{Y\}$ après la factorisation sans carrés. Après calcul pour la variable suivante X, on obtient un unique tableau de signes : $\{Y : [-, 0, +]\}$, qui est valide, puisque ses polynômes ne divisent aucun polynôme de $\mathcal{N}$. D'où un tableau de signes pour $\{XY + 1, Y\}$ quand X est nul: $\{XY + 1 : [+, +, +], Y : [-, 0, +]\}$.

6. On utilise la bibliothèque `Big_int` de Ocaml.

Finalement on a trois tableaux de signes pour la famille $\{XY+1, Y\}$:

$XY+1$	$+$	$+$	$+$	0	$-$
Y	$-$	0	$+$	$+$	$+$

$XY+1$	$-$	0	$+$	$+$	$+$
Y	$-$	$-$	$-$	0	$+$

$XY+1$	$+$	$+$	$+$
Y	$-$	0	$+$

D'où trois tableaux de signes possibles pour la famille de départ $\{XY^3+Y^2\}$:

XY^3+Y^2	$+$	0	$+$	0	$-$

XY^3+Y^2	$-$	0	$+$	0	$+$

XY^3+Y^2	$+$	0	$+$

correspondant aux cas $X<0$, $X>0$ et $X=0$.

6.3 Trace des calculs.

Pour pouvoir construire la preuve qu'un ensemble de tableaux de signes correspond bien à ceux d'une famille de polynômes, on va utiliser une trace des calculs qui ont permis de produire ces tableaux de signes. Comme on l'a vu, l'algorithme fonctionne par étapes :

- on tranforme une famille f de polynômes en une famille f', soit par factorisation, soit par dérivation et divisions euclidiennes, soit en passant à la variable suivante,
- on calcule récursivement les tableaux de signes de f',
- à partir de ceux-ci on construit les tableaux de signes de f,
- l'arrêt a lieu lorsque tous les polynômes sont constants, donnant lieu à un tableau de signe trivial.

Une trace des calculs est donc une suite de traces élémentaires correspondant aux étapes de calculs. Chaque étape donne lieu éventuellement à des informations qui décrivent les résultats des calculs intermédiaires effectués, pour éviter de les refaire lors de la construction de la preuve :

Dérivation: on stocke dans la trace les paramètres c_i, q_i des pseudo-divisions $c_i f_s = q_i f_i + g_i$, $c_s f_s = q_s f'_s + g_s$.

Factorisation: on stocke les décompositions des polynômes de f en produits de polynômes de f' et d'un entier.

Une trace du calcul des signes d'une famille f en la variable v à partir de ceux de f' est finalement un des termes suivants :

- **Choix**(f, v, l) où l est la liste de traces correspondant aux différents tableaux possibles de la famille f;
- **Derivation**(f, v, w, I, t) où w est le tableau des signes de f, I une liste des paramètres des pseudo-divisions, et t est la trace de f';
- **Factorisation**(f, v, w, I, t) où w est le tableau des signes de f, I une liste des factorisations de f en fonction de f', et t est la trace de f';
- **Cas**(f, v, w, t) où w est le tableau des signes de f, et t est la trace de f'. f est constituée de constantes pour la variable v, w n'a qu'une colonne, prise parmi les tableaux possibles de f', qui est ici égale à f, pour la variable $v-1$. La liste des traces de type *Cas* parcourt toutes les colonnes w possibles pour f';
- **Constantes**(f, w) où w est le tableau des signes de f.

6.4 Élimination des quantificateurs.

La trace d'un calcul permet d'éliminer les quantificateurs universels dans une formule de la forme

$$\forall X_n, ..., X_{p+1},\ f_1 \#_1 0, ..., f_s \#_s 0$$

où $\#_i \in \{ <, >, \leq, \geq, = \}$. En effet il suffit, pour chaque tableau de signes w de f compatible avec les équations ou inéquations $f_1 \#_1 0$, ..., $f_s \#_s 0$, de récupérer par un parcours en profondeur de la trace des calculs qui l'ont produit, les premiers tableaux de signes concernant la variable X_{p+1}.

Par exemple pour la formule $\forall X_4,\ X_1 X_4^2 + X_2 X_4 + X_3 > 0$, on obtient les tableaux de signes suivants:

X_1	0
X_3	+
X_2	0

X_1	+
$4X_1X_3 - X_2^2$	+

On a donc l'équivalence

$$(\forall X_4, X_1 X_4^2 + X_2 X_4 + X_3 > 0) \Leftrightarrow (X_1 = 0 \wedge X_3 > 0 \wedge X_2 = 0) \vee (X_1 > 0 \wedge 4X_1X_3 - X_2^2 > 0)$$

6.5 Construction des preuves pour Coq.

En Coq, les preuves sont des termes comme les autres. Dans ce langage typé, basé sur l'isomorphisme de Curry-Howard, le type d'une preuve p est le théorème t qu'elle démontre: on écrit alors p: t. La construction des preuves dans une déduction logique élémentaire est très simple, elle se fait par application d'une fonction à ses arguments. En effet, dans Coq, on identifie une implication logique $A \to B$ à une fonction qui prend une preuve de A et qui rend une preuve de B. Ainsi, par exemple, si on a une preuve $H1$ de A, une preuve $H2$ de B, et une preuve H de $A \to B \to A \wedge B$, et bien $(H\ H1\ H2)$ est une preuve de $A \wedge B$. Toutes les preuves de Coq ne sont évidemment pas des applications [7] mais pour nos besoins ce sera suffisant: les preuves qu'on construira seront des applications successives de théorèmes d'analyse élémentaire.

Leur type en Ocaml est le suivant:

```
type preuve = Theo of string
  | Preuve of preuve list
;;
```

par exemple :

`Preuve [Theo "(A,B:Prop)A/\B->A"; Theo "1>0/\0<1"]`

est une preuve de `1>0` (on omet ici, comme en Coq, les arguments implicites).

La traduction de ces preuves en preuves Coq est immédiate.

6.5.1 Méthode.

Etant données une famille $f = \{f_1, ..., f_s\}$ de polynômes de $\mathbb{R}[X_1, ..., X_n]$, et des hypothèses de signes sur ces polynômes H_1: $f_1 \#_1 0, ..., H_s$: $f_s \#_s 0$ où $\#_i \in \{ <, >, \leq, \geq, =, \neq \}$ le but est de construire une contradiction, i.e. une preuve de la formule $\Phi = H_1 \wedge ... \wedge H_s \Rightarrow$ False. Pour cela on utilisera une preuve de la formule :

$$\begin{aligned} \Psi = \forall x_1, ..., x_n,\ & \\ & \text{signe}(f_1(x_1, ..., x_n)) = S_{11} \wedge ... \wedge \text{signe}(f_s(x_1, ..., x_n)) = S_{1s} \\ & ... \\ \vee\ & \text{signe}(f_1(x_1, ..., x_n)) = S_{p1} \wedge ... \wedge \text{signe}(f_s(x_1, ..., x_n)) = S_{\text{ps}} \end{aligned}$$

où les signes S_{ij} sont tirés des tableaux de signes possibles de la famille f.

7. il y des abstractions, des points fixes, des traitement par cas,...

En effet il suffit de vérifier que les signes des hypothèses H_1, ..., H_s sont incompatibles avec les familles de signes $S_{i1}, ..., S_{is}$.

On obtiendra une preuve de Ψ à partir des différents tableaux de signes de f. Soit w un tableau de signes de f pour la variable X_n. Sa signification est la formule :

$$\begin{array}{ll} \Psi_{fw} = & \exists x_1, ..., x_{n-1}, \exists z_1, ..., z_p, \\ & z_1 < z_2 \wedge ... \wedge z_{p-1} < z_p \\ \wedge & \text{signe}_{w(I_{1,1})}(f_1(x_1, ..., x_{n-1}, I_1)) \\ ... & ... \\ \wedge & \text{signe}_{w(I_{1,s})}(f_s(x_1, ..., x_{n-1}, I_1)) \\ ... & \\ \wedge & \text{signe}_{w(I_{2p+1,1})}(f_1(x_1, ..., x_{n-1}, I_{2p+1})) \\ ... & ... \\ \wedge & \text{signe}_{w(I_{2p+1,s})}(f_s(x_1, ..., x_{n-1}, I_{2p+1})) \end{array}$$

où I_1, ..., I_{2p+1} est la subdivision de $]-\infty; +\infty[$ associée à z_1, ..., z_p, et les prédicats $\text{signe}_+ = \text{Pos}$, $\text{signe}_0 = \text{Nul}$, $\text{signe}_- = \text{Neg}$ sont définis par $\text{Pos}(h, I) \Leftrightarrow \forall x \in I, h(x) > 0$, etc. Si on connaît $x_1, ..., x_{n-1}$ et $z_1, ..., z_p$, il suffit, pour obtenir une de preuve de Ψ_{fw}, d'avoir, pour chaque I_k et chaque i, une preuve de $\text{signe}_{w(I_k,i)}(f_i(x_1, ..., x_{n-1}), I_k)$. Autrement dit un tableau de signes doit contenir pour chacun de ses signes, une preuve de ce signe : une case d'un tableau contiendra désormais un couple (signe, preuve).

Comme on l'a vu, l'algorithme de calcul des tableaux de signes procède par transformation successives de la famille f. On construit les tableaux de signes de f à partir de ceux de sa transformée f'. Pour les preuves des signes de ces tableaux, le procédé est analogue : on construira une preuve d'un signe $w(I, i)$ de f_i dans un intervalle I à partir de preuves de signes dans le tableau correspondant de la famille f'. Pour cela on utilisera soit le théorème des valeurs intermédiaires, soit des théorèmes élémentaires sur le signe d'un polynôme en fonction de celui de ses facteurs, soit des théorèmes élémentaires sur les unions d'intervalles.

Pour obtenir les preuves d'existence des valeurs $x_1, ..., x_{n-1}$ et $z_1, ..., z_p$, on utilisera encore le théorème des valeurs intermédiaires et les valeurs et subdivision associées au tableau f'.

Enfin on construira, toujours à partir des preuves qui concernent f', la preuve que les tableaux de signes de f sont les seuls possibles.

On va maintenant décrire sur un exemple la construction des preuves dans le cas d'une variable. Le cas multi-variables n'est pas encore complètement programmé, mais ne devrait pas poser de problème particulier.

6.5.2 Cas d'une variable, exemple.

Prenons l'exemple du polynôme $X^2 - 1$. L'algorithme donne la trace suivante :

```
Derivation
   ([|X^2-1|], 1,
    | + 0 - 0 + |,
    [1*(X^2-1) = X*X+(-1)],
    Factorisation
     ([|X; -1|], 1,
      | - 0 + |
      | - - - |,
      [X=1*X; -1=-1],
      Derivation
       ([|X|], 1,
        | - 0 + |,
        [1*X=X*1+0],
        Constantes ([|1; 0|],
                    | + |
                    | 0 |))))
```

Construisons maintenant les preuves pas à pas, en remontant dans la trace:

- les constantes 1 et 0: leur tableau de signe, avec les preuves:

  `1 | (+,p1) |`
  `0 | (0,p2) |`
  où
  `p1 = Theo "(Pos 1 ]-inf;+inf[)"`
  `p2 = Theo "(Nul 0 ]-inf;+inf[)"`
  en considérant que les théorèmes utilisés sont des théorèmes de base. On utilise à dessein dans les preuves une syntaxe proche de celle de Coq.

  Avec ces preuves, la formule $\Psi_{fw} =$

$$\mathrm{Pos}(1,]-\infty;+\infty[) \wedge \mathrm{Nul}(0,]-\infty;+\infty[)$$

  a pour preuve le terme
  `Preuve [Theo "(A1,A2:Prop)A1->A2->A1/\A2"; p1; p2]`

- la famille $\{X\}$: son tableau de signes avec leurs preuves:

  `X | (-,p3) (0,p4) (+,p5) |`
  avec

```
p3 = Preuve [and31;
           Preuve [Exists_prf;
                   Preuve [Preuve_val_interm1; Theo "(Derivee X 1)";
p1]]]
p4 = Preuve [and32;
           Preuve [Exists_prf;
                   Preuve [Preuve_val_interm1; Theo "(Derivee X 1)";
p1]]]
p5 = Preuve [and33;
           Preuve [Exists_prf;
                   Preuve [Preuve_val_interm1; Theo "(Derivee X 1)";
p1]]]
```

  où

```
and31 = Theo "(A1,A2,A3:Prop)A1/\A2/\A3 -> A1"
and32 = Theo "(A1,A2,A3:Prop)A1/\A2/\A3 -> A2"
and33 = Theo "(A1,A2,A3:Prop)A1/\A2/\A3 -> A3"

theo_val_interm1 =
     Theo "(f,f':Pol)(Derivee f f') -> (Pos f' ]-inf;+inf[)
              -> (Exists x (Neg f ]-inf;x[)/\(Nul f [x])/\(Pos f
]x;+inf[))"
```

  `Preuve_val_interm1` est une version particulière du théorème des valeurs intermédiaires,

  si `p` est une de preuve de `Theo "(Exists x (P x))"`, alors `Preuve [Exists_val; p]` donne un `y` tel que `(P y)`, et `Preuve [Exists_prf; p]` est une preuve de `(P y)`.

  Enfin, `(Derivee a b)` indique que b est la dérivée de a modulo un facteur strictement positif.

  On a alors

$$\begin{array}{ll} \Psi_{fw} = & \exists z_1, \\ & \mathrm{Neg}(X,]-\infty;z_1[) \\ \wedge & \mathrm{Nul}(X,[z_1]) \\ \wedge & \mathrm{Pos}(X,]z_1;+\infty[) \end{array}$$

  dont une preuve est :
  `p6 = Preuve [theo_val_interm1; Theo "(Derivee X 1)"; p1]`

- la famille $\{X, -1\}$: son tableau de signes est

```
      X | (-,p3) (0,p4) (+,p5) |
     -1 | (-,p7) (-,p8) (-,p9) |
```

où

```
z1= Preuve [Exists_val; p6]
p7 = Preuve [Theo "(x:R)(Neg f ]-inf;+inf[) -> (Neg f ]-inf;x[)";
             z1;
             Theo "(Neg -1 ]-inf;+inf[)"]
p8 = Preuve [Theo "(x:R)(Neg f ]-inf;+inf[) -> (Neg f [x])";
             z1;
             Theo "(Neg -1 ]-inf;+inf[)"]
p9 = Preuve [Theo "(x:R)(Neg f ]-inf;+inf[) -> (Neg f ]x;+inf[)";
             z1;
             Theo "(Neg -1 ]-inf;+inf[)"]
```

ainsi

$$\begin{array}{ll} \Psi_{fw} = & \exists z_1, \\ & \mathrm{Neg}(X,]-\infty; z_1[) \\ \wedge & \mathrm{Nul}(X, [z_1]) \\ \wedge & \mathrm{Pos}(X,]z_1; +\infty[) \\ \wedge & \mathrm{Neg}(-1,]-\infty; z_1[) \\ \wedge & \mathrm{Neg}(-1, [z_1]) \\ \wedge & \mathrm{Neg}(-1,]z_1; +\infty[) \end{array}$$

qui a pour preuve

```
Preuve [Exists_couple;
     z1;
     Preuve [Theo "(A1,A2,A3,A4,A5,A6:Prop)A1->...->A6->A1/\.../\A6";
             p3;p4;p5;p7;p8;p9]]
```

où `Exists_couple` construit une preuve de `(Exists x (P x))` à partir d'un `y` et d'une preuve de `(P y)`.

- enfin $X^2 - 1$: son tableau de signes avec les preuves :

```
   X^2-1 | (+,p10) (0,p11) (-,p12) (0,p13) (+,p14) |
```

avec

```
theo_val_interm2 =
Theo "(f,f':Pol)(z:R)(Derivee f f') -> (Neg f' ]-inf;z[) ->
Neg(f,[z])
      -> (Exists x (x<z)/\(Pos f ]-inf;x[)/\(Nul f [x])/\(Neg f
]x;z[))"
```

qui est une version du théorème des valeurs intermédiaires,

```
p15 = Preuve [Theo "(f,g,q,r,c:Pol)(x:R) c>0 -> Nul(g,x) -> Neg(r,x) ->
                  -> c*f=q*g+r -> Neg(f,x)";
           z1; Theo "1>0"; p4; p8; Theo "1*(X^2-1)=X*X-1"]
```

qui est une preuve que $X^2 - 1$ est négatif en `z1`,

```
p20 = Preuve [theo_val_interm2;
           z1;
           Theo "(Derivee X^2-1 X)";
           p3;
           p15]
p10 = Preuve [and42; Preuve [Exists_prf; p20]]

p11 = Preuve [and43; Preuve [Exists_prf; p20]]

p16 = Preuve [and44; Preuve [Exists_prf; p20]]
```

En appliquant de nouveau la version suivante du théorème des valeurs intermédiaires, on obtient les autres preuves du tableau :

```
theo_val_interm3 =
Theo "(f,f':Pol)(z:R)(Derivee f f') -> (Pos f' ]z;+inf[) ->
Neg(f,[z])
       -> (Exists x (z<x)/\(Neg f ]z;x[)/\(Nul f [x])/\(Pos f
]x;+inf[))"

p21 = Preuve [theo_val_interm3;
              z1;
              Theo "(Derivee X^2-1 X)";
              p5;
              p15]
p17 = Preuve [and42; Preuve [Exists_prf; p21]]
p13 = Preuve [and43; Preuve [Exists_prf; p21]]
p14 = Preuve [and44; Preuve [Exists_prf; p21]]
```

et enfin,

```
p12 = Preuve [Theo "(f:Pol)(x,y,z:R)x<y -> y<z ->
                  Neg(f,]x;y[)->Neg(f,[y])->Neg(f,]y;z[)
                  -> Neg(f,]x;z[)";
              Preuve [and41; Preuve [Exists_prf; p20]];
              Preuve [and41; Preuve [Exists_prf; p21]];
              p16;
              p15;
              p17]
```

finalement

$$\begin{aligned}\Psi_{fw} = \;& \exists z_2, z_3 \\ & z_2 < z_3 \\ \wedge \quad & \mathrm{Pos}(X^2-1,]-\infty; z_2[) \\ \wedge \quad & \mathrm{Nul}(X^2-1, [z_2]) \\ \wedge \quad & \mathrm{Neg}(X^2-1,]z_2; z_3[) \\ \wedge \quad & \mathrm{Nul}(X^2-1, [z_3]) \\ \wedge \quad & \mathrm{Pos}(X^2-1,]z_3; +\infty[)\end{aligned}$$

avec sa preuve

```
Preuve [Exists_couple;
     z2;
     Preuve [Exists_couple;
             z3;
             Preuve [Theo "(A1,A2,A3,A4,A5,A6:Prop)A1->...->A6-
>A1/\.../\A6";
                     Preuve [Theo "(x,y,z:R)x<y->y<z->x<z";
                             Preuve [and41; Preuve [Exists_prf;
p20]];
                             Preuve [and41; Preuve [Exists_prf;
p21]]];
                     p10;p11;p12;p13;p14]]]
```

où

```
z2 = Preuve [Exists_val; p20]
z3 = Preuve [Exists_val; p21]
```

6.6 Travail futur. A l'heure où cet article est écrit, les choses en sont au stade suivant :

- L'algorithme d'élimination des quantificateurs est écrit en Ocaml et fonctionne sur des exemples simples mais non triviaux, comme l'équation du troisième degré. Il fonctionne aussi sur des exemples tirés des preuves sur les réels de la bibliothèque de Coq. Enfin il fonctionne sur l'équation du quatrième degré, mais échoue à reconstruire la trace complète des calculs par manque de place mémoire (>1 Giga). Un tel algorithme général ne produit en effet pas sur cette entrée le calcul optimal qu'on sait effectuer pour ce problème[8].
- La construction de la preuve Coq dans le cas général est effectuée: la tactique, appelée «Tarski», fonctionne. Reste à prouver en Coq les théorèmes sur les réels utilisés.

A l'avenir, l'algorithme devrait être amélioré en utilisant par exemple des sous-résultants. Il y a en effet un grand nombre de chaînes de pseudo-divisions lors des calculs, et adapter les simplifications par division que permet l'algorithme des sous-résultants permettra de ralentir la croissance des coefficients intermédiaires des polynômes intervenant dans les calculs, que ce soit des coefficients entiers ou polynomiaux. Cela fait partie du sujet de la thèse que commence Assia Mahboubi.

8. D. Lazard «*Quantifier Elimination: Optimal Solution for Two Classical Examples*», J. Symbolic Computation (1988) **5**, 261-266.

Chapitre 7

Quotients mathématiques et types quotients dans Coq [1]

Il n'y a pas de types quotients dans le calcul des constructions inductives. La solution qui est sans doute la plus simple est de représenter un ensemble quotient par un type muni d'une relation d'équivalence. Mais alors on perd les avantages de l'égalité de Leibnitz (remplacement des égaux par des égaux), lorsqu'on travaille dans un ensemble quotient. On étudie ici quels types quotients sont possibles et impossibles à axiomatiser dans le CCI, et comment ils se comparent aux quotients mathématiques classiques.

Dans sa thèse[2], Hofmann propose une extention du calcul des constructions avec des types quotients qu'il montre consistante, mais note qu'elle n'est pas suffisante pour rendre compte de l'isomorphisme naturel qui existe entre les deux ensembles $E \to \frac{F}{R}$ et $\frac{E \to F}{S}$, où R est une relation d'équivalence sur F et S est définie par $f S g \Leftrightarrow \forall x \in E, f(x) R g(x)$.

On montre ici que dans la théorie de `Coq`, l'existence de cet isomorphisme, même non constructive, entraîne l'inconsistance de la théorie. La raison profonde semble en être l'imprédicativité de la sorte Set.

7.1 Introduction

Soient deux ensembles E et F, R une relation d'équivalence sur F, et π_R la surjection canonique de F sur $\frac{F}{R}$. Sur $E \to \frac{F}{R}$ on définit une relation S par $f S g \Leftrightarrow \forall x \in E\, f(x) R\, g(x)$. Par définition de S l'application qui compose une fonction $f\colon E \to F$ avec π_R est compatible avec S, et il y a donc un morphisme naturel θ tel que le diagramme suivant commute:

$$\begin{array}{ccc} & {\scriptstyle \pi_R o\,.} & \\ E \to F & \longrightarrow & E \to \frac{F}{R} \\ \downarrow \pi_S & \nearrow \theta & \\ \frac{E \to F}{S} & & \end{array}$$

Tableau 7.1.

En théorie des ensembles (où l'axiome du choix et l'extentionnalité des fonctions sont données), ce morphisme est clairement injectif:

$$\theta(\pi_S(f)) = \theta(\pi_S(g)) \Rightarrow \pi_R o\, f = \pi_R o\, g \Leftrightarrow \forall x \in E, f(x) R g(x) \Rightarrow f S g \Rightarrow \pi_S(f) = \pi_S(g)$$

Il est aussi surjectif: une fonction f de $E \to \frac{F}{R}$ est l'image par θ de $\pi_S(g)$ pour toute fonction g qui envoie x sur un élément de $[f(x)]_R$.

1. avec Laurent Chicli et Carlos Simpson, Actes du workshop TYPES 2002, accepté pour publication dans LNCS.

2. Martin Hofmann. «Extentional concepts in intentional type theory» Phd thesis, LFCS Edinburgh, 1995.

Dans le calcul des constructions inductives (CCI en abrégé), étendu avec les types quotients de Martin Hofmann, l'application θ peut être définie (on doit toutefois utiliser l'axiome d'extentionnalité des fonctions), on peut montrer qu'elle est injective, mais pas surjective. Le but de ce chapitre est de montrer qu'on ne peut pas étendre ces types quotients pour obtenir cette surjectivité: les deux types $E \to \frac{F}{R}$ et $\frac{E \to F}{S}$ ne sont pas et ne peuvent pas être rendus isomorphes dans le CCI, à cause de la coexistence de l'imprédicativité et de l'élimination forte de la sorte `Set`.

La suite de ce chapitre est organisée comme suit.

Le problème est posé en détails dans la section 2: on y décrit les types quotients que l'on considère, la différence en Coq entre la proposition disant qu'un objet existe et la donnée constructive de celui-ci, et finalement on montre l'équivalence du problème avec l'existence d'une section pour la surjection canonique d'un enxemble quotient (π_R).

Dans la section 3, on montre que la donnée constructive d'une section est contradictoire. On décrit quelques résultats connus sur les quotients dans le CCI: une adaptation d'un résultat de Diaconescu par Lacas et Werner[3].

Mais notre problème est équivalent à *l'existence* d'une section pour les quotients. On montre dans la section 4 l'inconsistance de cette hypothèse. La première étape consiste à montrer l'existence d'une fonction `Proptobool` de `Prop` vers `bool` qui envoie les propositions vraies sur `true` et les autres sur `false`. La deuxième étape est la suivante: dans un contexte où `False` (de type `Prop`) est le but à prouver, on utilise l'élimination de l'existence de `Proptobool` et ainsi on peut utiliser `Proptobool` pour construire un tiers-exclu dans `Set`, ce qui implique `False`[4].

Avec la même méthode, mais indépendemment du concept de type quotient, on verra dans la section 5 une autre conséquence de l'imprédicativité de `Set`: le tiers exclu et l'axiome du choix tous les deux dans `Prop` conduisent à une contradiction.

Toutes les preuves ont été formalisées en Coq. On en donne ici le code Coq correspondant aux définitions, énoncés et preuves, au fur et à mesure qu'ils sont introduits.

7.2 Types quotients en Coq.

7.2.1 Les types quotients.

Les types quotients que l'on considère ont été étudiés par Barthe[5], Boutin[6] et Hofmann. Ils sont une formalisation quasi-directe de la pratique mathématique des quotients, amputée toutefois du choix d'un représentant dans une classe d'équivalence[7].

Pour un type E et une relation d'équivalence R, un quotient consiste en:

- un type `quo` pour les éléments du quotient,
- une fonction `class: E -> quo` pour la surjection canonique,
- deux propositions caractérisant l'égalité des classes,
- une construction des fonctions qui passent au quotient,
- finalement, la surjectivité de la fonction `class`, énoncée comme une proposition (pour pallier partiellement a l'absence de choix d'un représentant dans une classe d'équivalence).

3. S.Lacas and B.Werner. «*Which choices imply the Excluded Middle?*» pre-print, 1999.

4. voir la section 3.2

5. G.Barthe. «*Extensions of pure type systems*» In M.Dezani-Ciancaglini and G.Plotkin, editors. Proceedings of TLCA'95, volume 902, pages 16–31. Springer-Verlag, 1995.

6. S.Boutin. «*Réflexions sur les quotients*» Thèse de doctorat, Université de Paris 7, 1997.

7. Dans sa thèse («Représentation d'algèbres non libres en théorie des types.» Université Paris-Sud, 2001), P.Courtieu étudie les quotients définis par une relation d'équivalence décidable: dans ce cas on peut choisir constructivement un représentant dans chaque classe d'équivalence, et on retrouve la pratique mathématique classique des quotients. Mais, malheureusement, toutes les relations d'équivalence ne sont pas décidables.

En Coq, cela donne:

```
   Record type_quotient [E : Type; R : (Relation E);
                         p : (Equivalence R)] : Type := {
   quo:> Type;
   class:> E->quo;
   quo_comp: (x,y:E)(R x y)->(class x) == (class y);
   quo_comp_rev: (x,y:E)(class x)==(class y)->(R x y);
   quo_lift: (F:Type)(f:E->F)(compatible[8] R f)->quo->F;
   quo_lift_prop:
     (F:Type)(f:E->F)(H:(compatible R f))
     (x:E)((comp (quo_lift F f H) class) x)== (f x);
   quo_surj: (c:quo)(ExT x:E | c==(class x))}.

Axiom quotient :
   (E:Type) (R:(Relation E))(p:(Equivalence R))(type_quotient p).
```

Avec ces types quotients , on peut définir la relation S et le morphisme θ en Coq (l'extentionalité des fonctions est nécessaire, mais est connue pour être consistante avec les types quotients, voir par exemple la thèse d'Hofmann).

```
   Variable E,F:Type.
Variable R:(Relation E); p:(Equivalence R).

Definition S:=[f,g:F->E](x:F)(R (f x) (g x)).

Lemma Sequiv: (Equivalence S).

Axiom Extensionality :
   (F,G:Type)(f,g:F->G)((x:F)(f x) == (g x))->f == g.

Lemma ps:(compatible S [f:F->E](comp (class (quotient p)) f)).

Definition theta:=(!quo_lift ? ? ? (quotient Sequiv) ? ? ps).
```

7.2.2 Prop et Set.

Le bas de la hiérarchie d'univers de Coq consiste en les deux sortes Prop et Set. La première, Prop, est censée représenter le type des propositions logiques. La seconde, Set, représente le type des objets sur lesquels porte le discours logique: les nombres entiers, les listes, les programes, etc.

Comme les éléments de Set représentent des types de données, on veut qu'ils soient non-dégénérés: $0 \neq 1$ par exemple. Pour prouver que 0 est différent de 1, on doit autoriser (d'un point de vue technique) l'élimination de Set vers Type. Les éliminations de Set vers Prop et de Set vers Set suivent par «cumulativité des univers».

De plus, les éléments de Prop sont des propositions, on peut donc les utiliser soit en logique intuitionniste, soit en logique classique. Pour un mathématicien classique, le fait que des preuves puissent être égales ou différentes est indifférent. D'autre part, de nombreux axiomes classiques entraînent l'indifférence des preuves (i.e. toutes les preuves d'une même proposition sont égales).

---

8. `Definiton compatible:=[E:Type;R:(Relation E);F:Type;f:E->F]`
   `(x,y:E)(R x y)->(f x)==(f y).`

Pour ces raisons, et pour pouvoir garder la possibilité de raisonner en logique classique, le système doit interdire toute règle qui permettrait de montrer que deux preuves sont différentes. C'est pourquoi il n'y a pas d'élimination de Prop vers Set ni vers Type en Coq (en effet, dans ce cas on pourrait distinguer les deux preuves de `True ∨ True` en raisonnant par cas et en les envoyant l'une sur le booléen true et l'autre sur false).

L'existence d'un objet x vérifiant un prédicat P est définie en Coq comme une proposition :

```
Inductive exT [A : Type; P : A->Prop]  : Prop :=
   exT_intro : (x:A)(P x)->(ExT P)
```

on pourra effectuer une élimination de l'existence de x (et disposer ainsi de x dans le contexte) pour construire un terme de sorte Prop, mais pas Set ni Type. Affirmer l'existence d'un objet est donc plus faible que de le construire. Dans le premier cas, on ne pourra que prouver des propriétés qui le concernent, alors que dans le second on pourra l'utiliser pour construire de nouveaux objets.

Dans la suite cette différence est essentielle, et à chaque fois qu'on parlera de l'existence d'un objet, on sous-entendra cette existence affirmée comme une proposition, i.e. dans Prop.

D'autre part, l'élimination forte est autorisée pour les objets inductifs de Set, et c'est elle, associée à l'imprédicativité de Set, qui va permettre de construire différents paradoxes.

7.2.3 L'existence d'une section pour les types quotients.

Revenons à notre problème initial. Si on prend $E = \frac{F}{R}$, la surjectivité du morphisme θ revient à l'existence d'une section de π_R, i.e. une fonction $s: \frac{F}{R} \to F$ telle

$$\pi_R \circ s = Id_{\frac{F}{R}}$$

Autrement dit, s choisit un représentant dans chaque classe d'équivalence de R.

Inversement, s'il existe une fonction $s: \frac{F}{R} \to F$ telle que $\pi_R \circ s = Id_{\frac{F}{R}}$, alors pour tout E, l'application θ correspondante est surjective. En effet, pour tout $f: E \to \frac{F}{R}$ on a:

$$\theta(\pi_S(s \circ f)) = \pi_R \circ (s \circ f) = (\pi_R \circ s) \circ f = f.$$

Dans la suite, on s'intéressera à la première déduction, qui s'écrit en Coq ainsi:

```
  Hypothesis theta_surj : (f : F ->  (quotient p))
 (EXT g : (quotient Sequiv) | (theta g) == f).

  Lemma step1: (EXT  s:(quo (quotient p))->E |
(comp (class (quotient p)) s) == (Id (quotient p))).
```

La surjectivité de θ nous amène donc à la question d'étendre les types quotients par l'existence d'une section de la surjection canonique. Dans la suite on rappelle rapidement des résultats de Lacas et Werner, où une telle section est donnée constructivement, et on montre que cette hypothèse est contradictoire. C'est seulement dans la section 4 que l'on supposera uniquement *l'existence* d'une section pour en déduire une contradiction.

7.3 Se donner constructivement une fonction de choix conduit à une contradiction.

On suppose ici qu'avec la définition des quotients, on a en plus une section à la surjection canonique, c: soit $s: \text{quo} \to E$ telle que $\forall x: \text{quo}, c(s(x)) = x$.

7.3.1 L'astuce de Diaconescu

En adaptant l'astuce de Diaconescu, Lacas et Werner montrent que, si on se donne une section de la surjection canonique des quotients, on peut pour une proposition donnée P, construire un terme de type `{P}+{~P}`, c'est-à-dire la décidablité. Ce qui revient à montrer le tiers-exclu dans Set. Ils procèdent ainsi.

Prenons $E = \text{bool}$. On définit R par:

$$x\,R\,y \Leftrightarrow x = y \vee P,$$

R est une relation d'équivalence, et si on considère le quotient $\frac{E}{R}$, on a

$$\begin{array}{rcl} s(c(\text{true})) =_{bool} s(c(\text{false})) & \Leftrightarrow & c(\text{true}) = c(\text{false}) \\ & \Leftrightarrow & \text{true}\,R\,\text{false} \\ & \Leftrightarrow & P \end{array}$$

Comme l'égalité dans bool est décidable, on en déduit la décidabilité de P, i.e. `{P}+{~P}`.

7.3.2 Inconsistance du tiers exclu dans Set.

En généralisant la construction de Lacas-Werner, on obtient une preuve de `(P:Prop){P}+{~P}`, i.e. le tiers-exclu fort, dans la sorte `Set`.

Pour obtenir la contradiction annoncée, on utilise le fait que le tiers exclu dans Set est contradictoire dans `Coq`.

Il existe deux voies pour montrer ce dernier résultat. La première[9] est une adaptation d'un résultat de Barbanera et Berardi, qui montre que le tiers-exclu associé à l'axiome du choix dans le calcul des constructions conduit à l'indifférence aux preuves. On code alors le paradoxe de Russell. La seconde[10] se base sur les travaux de Coquand et Hurkens et utilise un codage du paradoxe de Burali-Forti.

Il est troublant que ces deux paradoxes différents puissent être employés pour prouver le même résultat. Notons que dans les deux cas, l'imprédicativité de `Set` est essentielle.

7.4 L'existence d'une section conduit à une contradiction.

Cette fois-ci, on suppose seulement l'existence d'une section de la surjection canonique pour les quotients:

$$\exists s{:}\,\text{quo} \to E, \forall x{:}\,\text{quo}, c(s(x)) = x \qquad (i)$$

Ici l'astuce de Diaconescu ne fonctionne plus. On utilise une autre astuce (due à Simpson, simplifiée par Chicli), qui consiste à construire une fonction `Proptobool:Prop->bool` vérifiant

$$\forall P{:}\,\text{Prop}, P \leftrightarrow \text{Proptobool}(P) = \text{true} \qquad (ii)$$

On montre ensuite que ceci conduit à une contradiction.

7.4.1 De Prop vers bool.

9. L.Pottier. «*Quotients dans le CCI*» Rapport de Recherche INRIA RR-4053.
`http://www-sop.inria.fr/rapports/sophia/RR-4053.html`
(aussi «*Quotients in the CIC*», Theorem Proving in Higher Order Logics: 14th International Conference, TPHOLs'01, University of Edinburgh ed., 2001).
Voir aussi `la librairie standard de Coq/Logic/Berardi.v`.

10. H.Geuvers. «*Inconsistency of classical logic in type theory*» Note 2002.
`http://www.cs.kun.nl/~herman/note.ps.gz`

7.4.1.1 Un quotient particulier.

Considérons le type $P2$, fait de deux copies de `Prop`:

```
Inductive P2 : Type :=
p1: Prop ->  P2
| p2: Prop ->  P2 .
```

On définit une relation binaire sur $P2$ ainsi: deux propositions équivalentes d'une même copie de `Prop` sont en relation, les propositions vraies de $p1$ et les propositions vraies de $p2$ sont en relation.
En Coq, cela donne:

```
Inductive R : P2 -> P2 ->  Prop :=
R_12: (x, y : Prop) x -> y ->  (R (p1 x) (p2 y))
| R_21: (x, y : Prop) x -> y ->  (R (p2 x) (p1 y))
| R_11: (x, y : Prop) (iff x y) -> (R (p1 x) (p1 y))
| R_22: (x, y : Prop) (iff x y) -> (R (p2 x) (p2 y)) .

Lemma Requiv: (Equivalence R).
```

R est une relation d'équivalence, on peut construire le type quotient QP2:

```
Definition QP2 := (quotient Requiv).
```

7.4.1.2 Quelques propriétés.

On veut montrer l'existence de la fonction Proptobool, c'est-à-dire un énoncé dans Prop, donc on peut éliminer l'hypothèse (i) d'existence d'une section pour le quotient QP2: on dispose maintenant de la section s dans le contexte.

Considérons la fonction $f = s \circ c$, de $P2$ dans lui-même. Elle vérifie les propriétés suivantes:

- $\forall x{:}\, P2, x\, R\, f(x)$: (1)

$$\begin{aligned} \forall x{:}\, P2, x\, R f(x) &\Leftrightarrow c(x) = c(f(x)) \\ &\Leftrightarrow c(x) = c(s(c(x))) \\ &\Leftrightarrow c(x) = c(x) \text{ par } (i) \end{aligned}$$

- $\forall x, y{:}\, P2, x\, R\, y \Leftrightarrow f(x) = f(y)$: (2)

$$\begin{aligned} \forall x, y{:}\, P2, x\, R\, y &\Leftrightarrow c(x) = c(y) \\ &\Leftrightarrow s(c(x)) = s(c(y)) \text{ car, par } (i), s \text{ est injective} \end{aligned}$$

On définit maintenant deux fonctions `in_p1` et `in_p2`:

```
Definition in_p1: P2 ->  bool :=
[x:P2]Cases x of (p1 p)=>true  | (p2 p)=> false end.

Definition in_p2 : P2 ->  bool :=
   [x:P2]Cases x of (p1 p)=>false | (p2 p)=> true end.
```

Enfin, la propriété suivante nous servira: si un élément $p_1(P)$ est en relation avec un élément $p_2(Q)$ alors P est vraie.

$$\forall P, Q{:}\,\mathrm{Pr}\,o\,p, p_1(P) R\, p_2(Q) \to P. \quad (3)$$

Sa preuve est rapide en Coq, par inversion de l'hypothèse $p_1(P) R\, p_2(Q)$.

7.4.1.3 L'existence de la fonction `Proptobool`.

La fonction `Proptobool` est définie ainsi, à partir de f, donc de s:

```
Proptobool= Cases (in_p2 (f (p1 True))) of
                  true  => [p : Prop] (in_p2 (f (p1 p)))
                | false => [p : Prop] (in_p1 (f (p2 p)))
```

Pour montrer son existence en Coq, on écrira donc d'abord:

```
Exists Cases (in_p2 (f (p1 True))) of
       true  => [p : Prop] (in_p2 (f (p1 p)))
     | false => [p : Prop] (in_p1 (f (p2 p)))
    end.
```

Il reste à montrer que cette fonction vérifie la propriété (ii):$\forall P$: Prop, $P \leftrightarrow \mathrm{Proptobool}(P) =$ true

On raisonne par cas sur `(in_p2 (f (p1 True)))`:

- `(in_p2 (f (p1 True))) = true`: alors Proptobool $=$ in_$p2 \circ f \circ p1$: Prop $\to$ bool
  Soit P une proposition. Si P est vraie, on a $P \leftrightarrow$ True, et donc $p1(P) R\, p1$(True).
  Par (2), on en déduit $f(p1(P)) = = f(p1(\mathrm{True}))$.
  Ainsi, Proptobool$(P) =$ in_$p2(f(p1(P))) =$ in_$p2(f(p1(\mathrm{True}))) =$ true.
  Inversement, si Proptobool$(P) =$ true, on a in_$p2(f(p1(P))) =$ true, et en raisonnant par cas sur $f(p1(P))$, on obtient $f(p_1(P)) = = p_1(Q)$ ou $f(p_1(P)) = = p_2(Q)$. Le premier cas conduit par réduction de in_$p2$ à une absurdité false $=$ true. Dans le second cas, grâce à (1), on obtient $p_1(P) R\ f(p_1(P))$, et donc $p_1(P) R\ p_2(Q)$, et par (3) on a une preuve de P.
- `(in_p2 (f (p1 True))) = false`: le raisonnement est similaire.

En conclusion, on vient de montrer dans Coq:

```
Lemma existence_decision:
(EXT Proptobool:Prop->bool |
       (P:Prop)( P <-> (Proptobool P) = true)).
```

7.4.2 Inconsistance.

Comme au auparavant, pour obtenir une contradiction, on va montrer le tiers-exclu dans Set.

Mais pas directement: on a en hypothèse l'existence de la fonction Proptobool, énoncé de Prop, donc insuffisant pour montrer le tiers exclu dans Set `(P:Prop){P}+{~P}` , qui est dans Set.

Ce qu'on cherche à montrer est une contradiction, i.e. False.

On commence par éliminer l'hypothèse d'existence de Proptobool, on dispose alors de la fonction Proptobool dans le contexte avec sa propriété `(P:Prop)( P <-> (Proptobool P) = true)`. On effectue alors une coupure avec `(P:Prop)({P}+{~P})`. Le tiers-exclu dans Set premet de montrer False, il reste alors à monter le tiers-exclu dans Set.

On le fait par analyse de cas sur `(Proptobool P)`.

La preuve en Coq est alors:

```
   Lemma incoherence: False.
Generalize existence_decision.
Intros yH.
Elim yH; Intros Proptobool H; Clear yH.
Cut (P:Prop){P}+{~P}.
Exact paradoxe.
Intros P.
Generalize (refl_equal ? (Proptobool P)).
Pattern -1 (Proptobool P); Case (Proptobool P).
Intros H0.
Case (H P).
Intros H1 H2.
Left; Exact (H2 H0).
Intros hyp; Right.
Unfold not;Intros p.
Case (H P).
Intros H0 H1.
Rewrite (H0 p) in hyp; Inversion hyp.
Qed.
```

7.5 Inconsistance de l'axiome du choix et du tiers exclu dans Prop.[11]

Les techniques de preuve de la section précédente peuvent être utilisées hors du contexte des quotients.

Considérons l'axiome du choix unique suivant:

```
Axiom choice :
(A,B : Type)(R:A->B->Prop)
((x : A)  (EXT y : B |  (R x y))) ->
((x : A) (y1, y2 : B) (R x y1) -> (R x y2) ->  y1 == y2) ->
(EXT  f : (A -> B) | (x : A)  (R x (f x))).
```

Il affirme qu'étant donnés deux types A et B, si pour tout x dans A il existe un unique y dans B tel que xRy, alors il existe une fonction f de A vers B, telle que pour toit x dans A, $xR\,f(x)$.

Avec cet axiome, allié au tiers exclu dans Prop: (P:Prop)P $\vee$ $\sim$ P, on peut montrer de nouveau l'existence de la fonction Proptobool.

On prend pour R la relation suivante:

```
Definition R:Prop->bool->Prop:=
[P:Prop][x:bool]P<-> b=true.
```

Il suffit maintenant de prouver

$$\forall P\colon \mathrm{Prop}, \exists! b\colon \mathrm{bool}, P \leftrightarrow b = \mathrm{true}$$

Ce qui se fait facilement avec le tiers exclu: il suffit de distinguer les cas P et $\sim P$, et de choisir $b = \mathrm{true}$ dans le premier cas, et $b = \mathrm{false}$ dans le second.

On obtient alors la fonction Proptobool en appliquant l'axiome du choix.

En Coq cela donne:

11. Ce résultat est dû à L.Chicli.

```
   Lemma existence_decision:
 (EXT f : Prop ->  bool |  (P : Prop)  (iff P (f P) == true)).
Apply (!AC ? ? [P : Prop] [b : bool]  (iff P b == true)).
Intros P; Elim (EM P).
Intros p; Exists true; Split; Intuition.
Intros p; Exists false; Split; Intuition.
Inversion H.
Intros x y1 y2.
Elim (EM x).
Intros xx H0 H;Elim H0;Elim H;Intros;Transitivity true;Intuition.
Case y1; Case y2; Intuition.
Elim (H (H3 (refl_eqT bool true))).
Qed.
```

La contradiction est alors obtenue avec les résultats de la section précédente, à partir de l'existence de la fonction Proptobool.

Chapitre 8
Reconnaissance de formules planes[1]

8.1 Introduction

Le contexte dans lequel on se place ici est celui de la reconnaissance automatique de formules mathématiques: à partir de l'ensemble des caractères qui constituent une formule et de leur rectangle englobant, on veut reconstruire l'arbre de syntaxe représentant cette formule. On suppose donc que l'étape de reconnaissance des caractères, imprimés ou manuscrits, est faite.

Pourquoi veut-on reconnaître automatiquement des formules mathématiques tracés sur une feuille? Tout simplement pour enfin s'affranchir des moyens actuels d'écrire des formules de mathématiques avec une machine, qui utilisent soient le clavier, soit la souris, soit les deux, et qui sont extrêmement lents! Car contrairement à l'écriture linéaire où, selon les personnes, l'écriture au clavier est un peu plus rapide ou un peu plus lente qu'avec un crayon et un papier, l'écriture d'une formule mathématique avec LaTeX, Maple, et même $\mathrm{TEX}_{\mathrm{MACS}}$, est beaucoup plus lente qu'avec un papier et un crayon...

La méthode qu'on propose est en deux étapes:

1. on construit un graphe étiqueté avec comme noeuds les caractères et leurs rectangles, et comme arcs, des relations de proximité, de places relatives (en haut, à droite, etc): c'est ce qu'on appelle le graphe géométrique de la formule.
2. à l'aide d'une grammaire de graphes avec contextes, on réécrit ce graphe tant qu'on peut, chaque règle transformant un groupe de noeuds en un unique noeud contenant un arbre de syntaxe représentant le groupe de noeuds réécrits.

Après ces étapes, si on obtient un noeud, il contient l'arbre de syntaxe de la formule reconnue.

La grammaire de graphes que l'on utilise est elle-même le résultat d'un calcul de type complétion par paires critiques, qui élimine les ambiguïtés de la grammaire originelle qu'on a fourni.

8.2 Les données de départ.

On suppose qu'un processus de reconnaissance de caractères nous a fourni, pour chaque caractère de la formule :

- le caractère lui-même.
- les coordonnées planes des sommets du rectangle le contenant.

1. avec S.Lavirotte, Proceedings of the Fourth International Conference on Document Analysis and Reco gnition (ICDAR'97), volume 1, pages 357-361, Ulm, Allemagne. Août 1997. IEEE Comp uter Society Press. On ne présente ici que la partie qui concerne le formalisme utilisé et ses propriétés. L'article précédent décrit aussi le système mettant en oeuvre ce formalisme et quelques expériences.

- la taille de la police de caractères utilisée.
- le point de base du caractère.

Par exemple, pour la formule $(a^2 + b)$, on obtient:

```
  Symbol     Bounding Box      Baseline  Size
2     1246,454,1258,472   1246,472    7
b     1316,461,1330,490   1316,490   10
+     1275,466,1302,492   1275,489   10
a     1224,471,1243,490   1224,490   10
```

8.3 Graphes géométriques.

Ce sont des graphes dont les noeuds seront des caractères et leurs positions et propriétés comme décrit à la section précédente, et dont les arcs sont des relations de proximité géométrique.

Chaque objet est représenté par un terme ou par un ensemble fini de termes.

L'ensemble $T(F, V)$ des termes est défini par récurrence avec un ensemble F de symboles d'arité fixée, et un ensemble de variables V: les variables sont des termes, et si f est un symbole de F d'arité n, et si t_1, ..., t_n sont des termes, alors $f(t_1, ..., t_n)$ est un terme. On note Var(t) l'ensemble des variables d'un terme t. On utilisera des majuscules pour les symboles de F et des minuscules pour les variables.

Les noeuds, arcs, et graphes sont représentés ainsi:

- un **noeud** est un terme $V(t, v, i)$ où:
  - t est son type lexical, par exemple «Opérateur», «Variable», «Chiffre»,etc.
  - v est sa valeur, typiquement une expression mathématique sous forme de terme, par exemple Plus(Mult$(2, y), x)$.
  - i est un identificateur, distinguant plusieurs occurrences d'une même expression.
- un **arc** est un terme $E(t, v_1, v_2)$ où:
  - v_1 et v_2 sont des noeuds,
  - t est un type d'arc, i.e. un terme $L(d, w)$, d étant une direction géométrique (par exemple «Gauche», «Haut», etc), et w étant un poids, donnant la proximité relative de deux symboles dans le plan.
- un **graphe** est un ensemble fini d'arcs:
  $\{E(t_1, v_{11}, v_{2,1}), ..., E(t_n, v_{1n}, v_{2,n})\}$. L'ensemble $\{v_{ij}\}$ étant l'ensemble des noeuds du graphe. Pour simplifier, on suppose que les graphes sont ici connexes et ont au moins un arc [2].

8.4 Construction du graphe à partir des données.

Les données de départ forment les noeuds du graphe. Il faut construire les arcs.

Cela se fait en regardant, pour chaque caractère, ses voisins dans le plan, dans les différentes directions: *left* (**l**), *right* (**r**), *top* (**t**), *bottom* (**b**), *top-left* (**tl**), *top-right* (**tr**), *bottom-left* (**bl**), *bottom-right* (**br**). Un type dárc supplémentair concerne les inclusions (la racine carrée par exemple) **:i**.

Il y a toutefois un compromis à trouver dans les relations de proximité qui déterminent si un arc va être mis entre deux noeuds: si le graphe a trop d'arcs ou s'il en a trop peu, il donnera trop peu d'information pour pouvoir reconstruire l'arbre de la formule.

2. ainsi chaque noeud apparaît au moins dans un arc. Ce n'est pas une restriction: on peut ajouter un noeud générique connecté à chacun des autres noeuds.

Pour affiner ces informations, on utilise, comme dans les langages de chaînes de caractères, une notion d'unité lexicale, et des règles de syntaxe qui déterminent si un type de lien peu intervenir ou non entre deux catégories lexicales données.

Voici par exemple une partie de ces règles:

```
                              List of forbidden
Reg-expr  Type    outgoing edges    incoming edges
[0-9]     Digit         ()               ()
"Sigma"   Sum     '(tl bl tr br)      '(tr br)
...
```

8.5 Grammaires de graphes.

Les grammaires de graphes sont un formalisme utile pour décrire les manipulations structurelles de données multi-dimensionnelles. Elle ont été introduites par Pfaltz et Rosenfeld[3] pour des problèmes d'images. Elles ont étudiées d'un point de vue théorique par exemple par Courcelle[4], Raoult et Voisin[5], et d'un point de vue pratique par exemple par Bunke[6], Fahmy et Blostein[7].

Pour une synthèse, on peut consulter les articles de Fahmy et Blostein[8] et Nagl[9].

Les grammaires de graphes sont utilisées pour lire ou générer des graphes. Une grammaire de graphe est donnée par un graphe de départ, et un ensemble de règles de production. Une règle permet de remplacer un sous-graphe par un autre. Elle peut décrire aussi comment ce nouveau sous-graphe va se connecter à son contexte.

Nous allons utiliser des grammaires de graphes sensibles au contexte. Dans notre cas, une règle va transformer un sous-graphe dont le contexte (le reste du graphe) vérifie une certaine propriété syntaxique, en un noeud unique (représentant une sous-formule reconnue).

Pour ces grammaires, les symboles terminaux seront les caractères reconnus sur la feuille où est écrite la formule, et les symboles termnaux seront les sous-formules reconnues.

---

3. Pfaltz, J., and Rosenfeld, A. « Web grammars.»
Proc. First international Joint Conference on Artificial
Intelligence (Washington, 1969), pp. 609–619.

4. Courcelle, B. « An axiomatic definition of context-free rewriting and its application
to nlc graph grammars.» Tech. rep., Universit\é de Bordeaux, Feb. 1987.

5. Raoult, J.-C., and Voisin, F.« Set-theoretic graph rewriting.»Tech. rep., IRISA, Apr. 1992.

6. Bunke, H.« Graph grammars as a generative tool in image understanding.»
In Graph Grammars and their Application to Computer Science
(Oct. 1982), H.Ehrig, M.Nagl, and G.Rozenberg, Eds., vol.153 of
Lecture Notes in Computer Sciences, Springer-Verlag, pp.8–19.

7. Fahmy, H., and Blostein, D.«Graph grammar processing of uncertain data.»
In Advances in Structural and Syntactic Pattern Recognition
(Aug. 1992), H.Bunke, Ed., vol.5 of Machine Perception and Artificial
Intelligence, World Scientific, pp.373–382.
et
Fahmy, H., and Blostein, D.« A graph grammar for high-level recognition of music notation.»
In Proceedings of First International Conference on Document
Analysis and Recognition (Saint Malo, France, Oct. 1991), vol.1, IEEE
Computer Society, pp.70–78.

8. Fahmy, H., and Blostein, D.« A survey of graph grammars: Theory and applications.»
In Proceedings of the 11th International Conference on Pattern
Recognition (Sept. 1992), vol.The Hague, Netherlands, IEEE Computer
Society.

9. Nagl, M.« A tutorial and bibliographical survey on graph grammars.»
In Workshop on Graph-Grammars and their Application to Computer
Science and Biology (Bad Honnef, Nov. 1978), vol.Lecture Notes in
Computer Science, pp.70–126.

Leur forme précise est la suivante:

- une **règle** est un terme $V \leftarrow G, C$ où:
  - V est un noeud, appelé la «production» de la règle.
  - G est un graphe, appelé le «motif» de la règle.
  - C est un ensemble fini de graphes, appelé le «contexte» de la règle. On précisera ce point dans la suite.
- une **grammaire** est un ensemble fini de règles.

Étant donné un graphe représentant une formule, une règle remplace un sous-graphe qui est filtré par son motif, par l'instance appropriée de sa production, si son contexte est satisfait.

Rappelons les notions de substitution et de remplacement sur les termes:

- une **substitution** est un endomorphisme de $T(F, V)$, i.e. une application σ vérifiant $\forall f \in F, \forall\ t_1, ..., t_n,\ \sigma f(t_1, ..., t_n) = f(\sigma t_1, ..., \sigma t_n)$. Une susbtitution est caractérisée par ses valeurs sur les variables.
- un terme t **filtre** un terme t', ce qui est noté $t \leq t'$ ssi il existe une substitution σ telle que $\sigma t = t'$.

Le filtrage des ensembles finis de termes est défini par:

$$\{t_1, ..., t_n\} \leq \{t'_1, ..., t'_m\} \Leftrightarrow \exists \sigma\ \{\sigma t_1, ..., \sigma t_n\} = \{t'_1, ..., t'_m\}$$

8.6 Les contextes des règles.

Un des principaux problèmes avec les grammaires et les règles de réécriture est l'existence d'ambiguïtés: deux règles peuvent réécrire un objet en deux objets différents. La suppression des ambiguïtés peut être faite à l'aide de priorités, d'analyse de cas sur les motifs des règles, ou par complétion du type Knuth-Bendix. Ces techniques sont difficilement adaptables à notre problème, c'est pourquoi nous allons utiliser des contextes dans les règles: étant donnée une grammaire avec des ambiguïtés, notre but est d'ajouter des contraintes de contexte aux règles pour éliminer les ambiguïtés, automatiquement si possible.

Précisons les choses.

Une ambiguïté ne peut arriver que quand les motifs des deux règles peuvent se superposer. Sinon, les deux règles ne peuvent que s'appliquer à des sous-graphes sans noeud commun, et donc commutent.

- Deux graphes G_1 et G_2 peuvent se superposer ssi il existe σ_1 et σ_2 telle que $\sigma_1 G_1$ et $\sigma_2 G_2$ ont un noeud commun. On note $S(G_1, G_2)$ l'ensemble des couples de ces substitutions, qu'on appelle **superpositions** de G_1 et G_2.
- Etant données deux règles $r_i = V_i \leftarrow G_i, C_i$, $i = 1, 2$, l'ensemble $A(r_1, r_2)$ des **ambiguïtés** de r_1 et r_2 est défini comme le sous-ensemble de $S(G_1, G_2)$ formé des couples (σ_1, σ_2) tels que les deux règles s'appliquent au graphe $\sigma_1 G_1 \cup \sigma_2 G_2$,
  i.e. $\forall i = 1, 2, \forall H \in C_i$, il n'y a pas de substitution τ telle que $\tau_{|\mathrm{Var}(G_i)} = \sigma_{i|\mathrm{Var}(G_i)}$ and $\tau H \subset \sigma_1 G_1 \cup \sigma_2 G_2$.

L'ensemble $S(G_1, G_2)$ peut être infini, mais les superpositions «minimales» sont en nombre fini, comme on le montre maintenant.

Définissons d'abord un pré-ordre sur les substitutions:

Définition 8.1.

$$(\sigma_1, \sigma_2) \leq (\sigma'_1, \sigma'_2) \Leftrightarrow \exists \tau, \sigma'_1 = \tau \circ \sigma_1,\ \sigma'_1 = \tau \circ \sigma_1$$

Proposition 8.2. *Étant donnés deux graphes G_1 et G_2, il existe un sous-ensemble fini S de $S(G_1, G_2)$ tel que:*

$$\forall c \in S(G_1, G_2), \exists c' \in S, c' \leqslant c.$$

Démonstration. Pour chaque superposition des squelettes des deux graphes, on prend l'unificateur principal de chaque paire de noeuds superposés. □

Proposition 8.3. *Étant donnés deux graphes G_1 et G_2, il existe un ensemble fini minimal $S_0(G_1, G_2)$ de superpositions de G_1 et G_2, unique à renommage des variables près, et minorant (pour $\leq$) l'ensemble $S(G_1, G_2)$.*

À cause des contextes, l'ensemble des ambiguïtés de deux règles a une structure plus compliquée que celle de l'ensemble des superpositions de leurs motifs. La propriété suivante va nous aider à le décrire:

Proposition 8.4. *Soient $r = V \leftarrow G, C$ une règle, $r' = V \leftarrow G, \emptyset$ la même règle avec le contexte vide, G_1, G_2 deux graphes, tels que $G_1 \rightarrow_{r'} G_2$ and $G_1 \not\rightarrow_r G_2$.*
Alors, pour toute substitution ρ, $\rho G_1 \rightarrow_{r'} \rho G_2$ and $\rho G_1 \not\rightarrow_r \rho G_2$.

Soient deux règles r_1 et r_2 avec des motifs G_1 et G_2, et une superposition (σ_1, σ_2) de $S(G_1, G_2)$. Si on enlève les contextes, elles s'appliquent toutes les deux à $\sigma_1 G_1 \cup \sigma_2 G_2$. Supposons maintenant que (σ_1, σ_2) n'est pas une ambiguïté de r_1 et r_2. Alors une des règles (avec leurs contextes) ne s'applique pas à $\sigma_1 G_1 \cup \sigma_2 G_2$. La dernière proposition implique que pour toute substitution τ, la superposition $(\tau \circ \sigma_1, \tau \circ \sigma_2)$ n'est pas une ambiguïté de r_1 et r_2.
Plus brièvement, on a:

Proposition 8.5.
Si $c \in S(G_1, G_2) \setminus A(r_1, r_2)$ et $c \leq c'$ alors $c' \in S(G_1, G_2) \setminus A(r_1, r_2)$.

Cela signifie que le complémentaire de $A(r_1, r_2)$ dans $S(G_1, G_2)$ a la même structure agréable de cône que $S(G_1, G_2)$.
En particulier , on a:

Corollaire 8.6. *Si les superpositions minimales des motifs de deux règles ne sont pas des ambiguïtés des ces règles, alors ces deux règles n'ont pas d'ambiguïté.*

On va exploiter maintenant cette propriété pour définir une méthode qui, étant donnée une grammaire avec ambiguïtés, ajoute des contextes aux règles pour obtenir une nouvelle grammaire sans ambiguïté.

8.7 Construction des contextes.

Soit $\mathcal{G} = \{r_1, ..., r_n\}$ une grammaire de graphes, avec $r_i = V_i \leftarrow G_i, \emptyset$.
Soient $(\sigma_{k1}^{ij}, \sigma_{k2}^{ij}), k = 1...a_{ij}$ un ensemble minimal de superpositions de G_i et G_j. Comme les contextes sont vides, ces superpositions coïncident avec les ambiguïtés des règles.
Supposons que pour chaque superposition minimale, i.e. quand deux règles peuvent s'appliquer simultanément, on ait une méthode pour choisir la bonne règle à appliquer. C'est équivalent à donner une fonction C telle que $C(i, j, k) \in \{1, 2\}$: si $C(i, j, k) = 1$, cela siginifie que l'on veut éviter l'application de la règle r_j dans l'ambiguïté $(\sigma_{k1}^{ij}, \sigma_{k2}^{ij})$. Obtient cela en ajoutant le conexte $\sigma_{k1}^{ij} G_1$ à la règle r_j.

En faisant cela pour toutes les règles et leurs superpositions minimales, on définit une nouvelle grammaire $\mathcal{G}' = \{r'_1, ..., r'_n\}$, où:

$$r'_i = V_i \leftarrow G_i, \bigcup_{j=1...n} \{\sigma^{ij}_{k2} G_j | k = 1...a_{ij}, C(i,j,k) = 2\}$$

Par le corollaire 1, on a alors:

Proposition 8.7. *La grammaire $\mathcal{G}'$ n'a pas d'ambiguïté.*

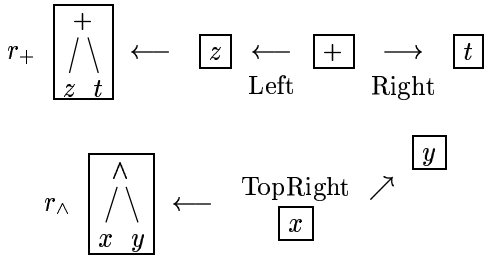
Illustrons tout cela par un exemple.
Prenons $\mathcal{G} = \{r_+, r_\wedge\}$, la première règle pour l'addition $z + t$:

$$\begin{aligned} r_+ = \; & V(\text{Expr}, +(z,t), i_+) \leftarrow \\ & \{E(\{\text{Left}, w\}, V(\text{Add}, +, i_+), V(\text{Expr}, z, i_z)), \\ & E(\{\text{Right}, w\}, V(\text{Add}, +, i_+), V(\text{Expr}, t, i_t))\}, \\ & \emptyset \end{aligned}$$

la deuxième pour la puissance x^y:

$$\begin{aligned} r_\wedge = \; & V(\text{Expr}, \wedge(x,y), i_x) \leftarrow \\ & \{E(\{\text{TopRight}, w\}, V(\text{Expr}, x, i_x), V(\text{Expr}, y, i_y))\}, \\ & \emptyset \end{aligned}$$

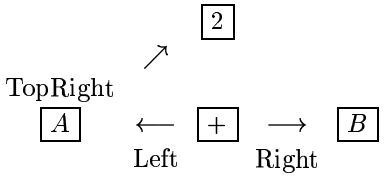
Graphiquement:



Supposons qu'on ait la formule $A^2 + B$, donnée par le graphe

$$\begin{aligned} \mathcal{F} = \; & \{E(\{\text{TopRight}, w\}, V(\text{Expr}, A, 1), V(\text{Expr}, 2, 1)), \\ & E(\{\text{Left}, w\}, V(\text{Add}, +, 3), V(\text{Expr}, A, 1)), \\ & E(\{\text{Right}, w\}, V(\text{Add}, +, 3), V(\text{Expr}, B, 2))\} \end{aligned}$$

Graphiquement:



Il y a une ambiguïté, car on peut appliquer les deux règles. Si on applique r_+, les noeuds A, $+$ and B sont contractés en un unique noeud, et on obtient le graphe de $(A+B)^2$:

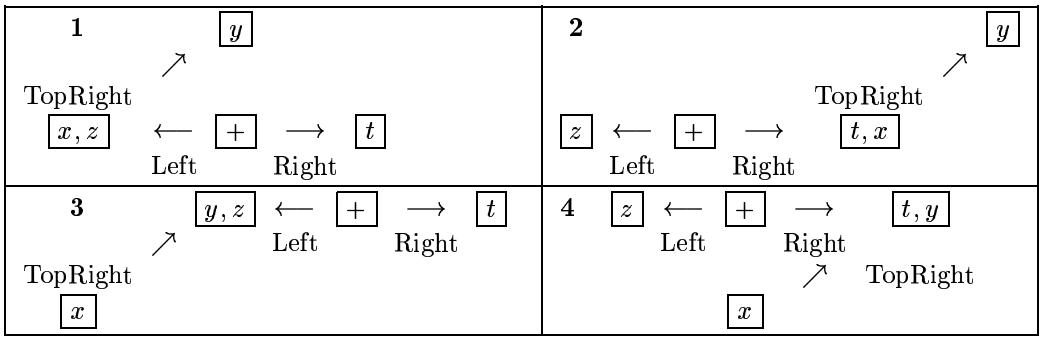
$$\mathcal{F} = \{E(\{\text{TopRight}, w\}, V(\text{Expr}, +(A,B), 3), V(\text{Expr}, 2, 1))\}$$

Si on applique $r_\wedge$, on obtient le graphe de la formule $(A^2) + B$:

$$\begin{aligned} \mathcal{F} = \; & \{E(\{\text{Left}, w\}, V(\text{Add}, +, 3), V(\text{Expr}, \wedge(A,2), 1)), \\ & E(\{\text{Right}, w\}, V(\text{Add}, +, 3), V(\text{Expr}, B, 2))\} \end{aligned}$$

le bon choix est clairement ici d'appliquer $r_\wedge$.

Voici les quatre superpositions minimales de r_+ et $r_\wedge$:



Le graphe $\mathcal{F}$ précédent est un cas particulier de la première.

La fonction C qui indique comment résoudre chaque ambiguïté est elle assez empirique, et suit différents critères:

- la priorité usuelle en syntaxe linéaire de l'opérateur décrit. Par exemple le produit a une priorité supérieure à la somme.
- l'information graphique, importante pour déterminer la bonne priorité à donner à une règle. Dans notre exemple, $\wedge$ a une priorité plus grande que $+$, mais cela ne veut pas dire que la règle de $\wedge$ s'appliquera toujours avant celle $+$ (par exemple dans la superposition 3). Dans le cas de $\wedge$, il y a des parenthèse implicites sur l'argument en exposant et cela est vrai pour les opérateurs comme $x^{(y+z)}, x_{(y+z)}$, $\sum_{(i=1)}^{(n^2)} x_i$, $\frac{(x+y)}{(z+t)}$, ...

Avec ces critères, on peut résoudre la plupart des cas.

En particulier les 4 superpositions sont résolues et conduisent à la grammaire non ambiguë $\mathcal{G}'$ (les contextes sont en gras):

$$
\begin{aligned}
r_+ = \; & V(\mathrm{Expr}, +(z,t), i_+) \leftarrow \\
& \{E(\{\mathrm{Left}, w\}, V(\mathrm{Add}, +, i_+), V(\mathrm{Expr}, z, i_z)), \\
& E(\{\mathrm{Right}, w\}, V(\mathrm{Add}, +, i_+), V(\mathrm{Expr}, t, i_t))\}, \\
& \boldsymbol{\{\{E(\{\mathrm{TopRight}, w\}, V(\mathrm{Expr}, z, i_z), V(\mathrm{Expr}, u1, i_{u1}))\},} \\
& \boldsymbol{\{E(\{\mathrm{TopRight}, w\}, V(\mathrm{Expr}, t, i_t), V(\mathrm{Expr}, u2, i_{u2}))\}\}}
\end{aligned}
$$

$$
\begin{aligned}
r_\wedge = \; & V(\mathrm{Expr}, \wedge(x,y), i_x) \leftarrow \\
& \{E(\{\mathrm{TopRight}, w\}, V(\mathrm{Expr}, x, i_x), V(\mathrm{Expr}, y, i_y))\}, \\
& \boldsymbol{\{\{E(\{\mathrm{Left}, w\}, V(\mathrm{Add}, +, i_+), V(\mathrm{Expr}, x, i_x)),} \\
& \boldsymbol{E(\{\mathrm{Right}, w\}, V(\mathrm{Add}, +, i_+), V(\mathrm{Expr}, u1, i_{u1}))\},} \\
& \boldsymbol{\{E(\{\mathrm{Left}, w\}, V(\mathrm{Add}, +, i_+), V(\mathrm{Expr}, u2, i_{u2})),} \\
& \boldsymbol{E(\{\mathrm{Right}, w\}, V(\mathrm{Add}, +, i_+), V(\mathrm{Expr}, y, i_y))\}\}}
\end{aligned}
$$

Cette grammaire fonctionne alors parfaitement sur des formules comme $A^2 + B$, $A + B^2$, A^{a+b}, $x^{a+b} + y^{c+d}$, $x^{a+b^2} + y^{c^{i+j^2}+d}$, etc.

8.8 Construction de la formule.

On dit qu'une règle $r = V \leftarrow G, C$ **réécrit** un graphe G_1 en un graphe G_2, et on note cela $G_1 \rightarrow_r G_2$ ssi il existe une substitution σ, un sous-graphe G' de G_1 (i.e. $G' \subset G_1$), tels que:

- $\sigma G = G'$.
- pour tout graphe H dans le contexte C, il n'y a pas de substitution τ telle que $\tau_{|Var(G)} = \sigma_{|Var(G)}$ et $\tau H \subset G_1$.

- G_2 est obtenu en contractant G' en σV, i.e. en enlevant de G_1 tous les arcs de G' et en remplaçant dans G_1 tous les noeuds de G' par le noeud σV.

Cela donne un algorithme de lecture du type «bottom-up». Trois raisons motivent ce choix, bien que la perception humaine indiquerait le contraire: on a souvent une vue globale d'une formule avant de lire ses composants.

D'abord, une feuille peut contenir plusieurs formules, et dans ce cas un algorithme «top-down» échouerait.

De plus un algorithme «top-down» conduirait à une explosion combinatoire, comme la littérature l'indique.

Enfin, les paramètres géométriques d'une formule déduits de ses composants sont donnés par des fonctions très difficiles à inverser. En particulier, le rectangle englobant d'une formule ne peut être séparé en sous-rectangles comme on peut le faire en dimension 1 quand on coupe une chaîne de caractères en sous-chaînes dans un algorithme «top-down».

Dans notre cas, une analyse rapide montre que notre algorithme est au pire quadratique en temps.

8.9 Exemples.

Ces deux exemples ont été reconnus avec l'implémentation réalisée par Stéphane Lavirotte:

$$\frac{\frac{x^{-201}+y^{523}}{a\ b\ c\ e}}{(x^2+y^2)(x^3+y^3)}$$

$$(\frac{1}{x^2+1})^{(n)} = (-1)^n.n!\frac{\sum_{0\leq 2p\leq n}(-1)^p C_{n+1}^{2p+1} X^{n-2p}}{(X^2+1)^{n+1}}$$

Ce mémoire a été écrit avec TEXMACS (`www.texmacs.org`).

Table des matières

1 Introduction 7

2 Une borne simplement exponentielle pour les systèmes diophantiens linéaires. . 11

2.1 Introduction. 11
2.2 Notations 11
2.3 Borne des générateurs des solutions positives d'un système $A\,x = 0$ 11
2.3.1 Borne des générateurs de M 12
2.3.2 Expression de $K_m(s)$ 13
2.4 Borne des générateurs des solutions de $A\,x \geq b$ 14

3 L'algorithme d'Euclide en dimension n 15

3.1 Introduction. 15
3.2 L'algorithme d'Euclide dans $\mathbb{Z}$ 15
3.3 Généralisation à $\mathbb{Z}^n$ 15
3.4 Algorithme d'Euclide en dimension n. 16
3.5 Améliorations de l'algorithme. 17
3.6 Quelques applications. 17

4 Une borne inférieure optimale de l'espace-temps nécessaire à inverser une suite. 19

4.1 Introduction 19
4.2 Modèle d'algorithme. 20
4.3 Inversions optimales 21
4.3.1 Exemple 22
4.4 Temps optimal. 22
4.5 Borne inférieure. 23

5 Algèbre en théorie des types. 24

5.1 Contenu de la bibliothèque. 24
5.1.1 Ensembles 24
5.1.1.1 Sétoïdes, quotients. 24
5.1.1.2 Applications. 25
5.1.1.3 Parties, sous-types 25
5.1.1.4 Injections, surjections, bijections. 26
5.1.1.5 Images, images réciproques, restrictions. 26
5.1.1.6 Théorème: il n'y a pas de surjection de E dans $\wp(E)$. 27
5.1.1.7 Théorème de Cantor-Bernstein 27
5.1.2 Catégories. 27
5.1.2.1 Définition des catégories, foncteurs. 27
5.1.2.2 Sous-catégorie, sous-catégorie pleine. 27
5.1.3 Semi-groupes, monoïdes, groupes. 27
5.1.3.1 Morphismes de groupes, catégorie des groupes, etc. 30
5.1.3.2 Sous-structures (sous-groupes, etc). 30
5.1.3.3 Structures libres (groupe libre, etc). 30
5.1.3.4 Groupes quotients, noyaux, co-noyaux. 31
5.1.3.5 Décomposition canonique d'un morphisme de groupes. 31

5.1.3.6 Sous-structures engendrées par une partie. . . . 31
5.1.3.7 Monoïde des endomorphismes d'un ensemble. . . . 31
5.1.3.8 Opération d'un monoïde sur un ensemble. . . . 31
5.1.4 Anneaux. . . . 31
5.1.5 Modules. . . . 32
5.1.6 Algèbres. . . . 32
5.1.7 Idéaux d'un anneau. . . . 32
5.1.7.1 Idéal engendré par une partie. . . . 33
5.1.7.2 Anneaux intègres. . . . 33
5.1.8 Corps. . . . 33
5.1.8.1 Corps des fractions d'un anneau intègre. . . . 33
5.1.9 Nombres. . . . 34
5.1.9.1 $\mathbb{Z}$: l'anneau commutatif intègre des entiers. . . . 34
5.1.9.2 $\mathbb{Q}$: le corps des nombres rationnels. . . . 34
5.1.9.3 Le corps des «nombres complexes» construit à partir d'un corps commutatif R vérifiant: $\forall x, y \in R, x^2 + y^2 = 0 \Rightarrow x = 0 \text{ et } y = 0$ . . . 34
5.1.10 Ensembles finis. . . . 34
5.1.10.1 Cardinaux finis, parties finies d'un ensemble, ensembles finis. . . . 34
5.1.10.2 Principe de Dirichlet (lemme des tiroirs). . . . 35
5.1.10.3 Applications entre ensembles finis. . . . 35
5.2 Usage, travaux similaires, futur. . . . 35

6 Élimination des quantificateurs sur les réels dans Coq . . . 37

6.1 Introduction . . . 37
6.2 Tableaux des signes d'une famille de polynômes. . . . 38
6.2.1 Cas d'une variable: $\mathbb{Z}[X]$. . . . 38
6.2.2 Cas de plusieurs variables: $\mathbb{R}[X_1]...[X_n]$. . . . 39
6.2.3 Améliorations . . . 40
6.2.4 Un exemple . . . 40
6.3 Trace des calculs. . . . 41
6.4 Élimination des quantificateurs. . . . 42
6.5 Construction des preuves pour Coq. . . . 42
6.5.1 Méthode. . . . 42
6.5.2 Cas d'une variable, exemple. . . . 43
6.6 Travail futur. . . . 47

7 Quotients mathématiques et types quotients dans Coq . . . 48

7.1 Introduction . . . 48
7.2 Types quotients en Coq. . . . 49
7.2.1 Les types quotients. . . . 49
7.2.2 `Prop` et `Set`. . . . 50
7.2.3 L'existence d'une section pour les types quotients. . . . 51
7.3 Se donner constructivement une fonction de choix conduit à une contradiction. . . . 51
7.3.1 L'astuce de Diaconescu . . . 52
7.3.2 Inconsistance du tiers exclu dans Set. . . . 52
7.4 L'existence d'une section conduit à une contradiction. . . . 52
7.4.1 De Prop vers bool. . . . 52
7.4.1.1 Un quotient particulier. . . . 53
7.4.1.2 Quelques propriétés. . . . 53
7.4.1.3 L'existence de la fonction `Proptobool`. . . . 54
7.4.2 Inconsistance. . . . 54
7.5 Inconsistance de l'axiome du choix et du tiers exclu dans Prop. . . . 55

8 Reconnaissance de formules planes . . . 57

8.1 Introduction . . . 57

8.2 Les données de départ. 57
8.3 Graphes géométriques. 58
8.4 Construction du graphe à partir des données. 58
8.5 Grammaires de graphes. 59
8.6 Les contextes des règles. 60
8.7 Construction des contextes. 61
8.8 Construction de la formule. 63
8.9 Exemples. 64

Résumé:

Ce mémoire concerne le calcul et le raisonnement à l'aide de machines Il est constituée de six articles, dont quatre ont été traduits en français, et d'une description d'une implémentation :

1. Une borne simplement exponentielle pour les système diophantiens linéaires.
2. L'algorithme d'Euclide en dimension n
3. Une borne inférieure optimale de l'espace-temps nécessaire à inverser une suite.
4. Algèbre en théorie des types.
5. Élimination des quantificateurs sur les réels dans Coq.
6. Quotients mathématiques et types quotients dans Coq.
7. Reconnaissance de formules planes.

Abstract:

This document deals with computing and reasoning with machines. It contains six papers, four of them translated from english, and a description of an implementation:

1. A simply exponential bound for linear diophantine systems.
2. Euclide's algorithm in dimension n.
3. An optimal lower bound for time-space necessary to reverse a finite sequence.
4. Algebra in type theory.
5. Elimination of quantifiers with real numbers in Coq.
6. Mathematical quotients and quotient types in Coq.
7. Planar formulas recognition.