Outils effectifs en asymptotique et applications

par Joris van der Hoeven

LIX
École Polytechnique
91128, Palaiseau
France

le 28 juin 1994

Résumé

Malgré l'omniprésence de développements asymptotiques en mathématiques, les définitions qui en sont données diffèrent d'un auteur à l'autre. Nous proposons ici un cadre général dans lequel on peut effectuer des développements asymptotiques, tout en leur gardant un sens. Il va s'agir d'établir des développements asymptotiques et d'en tirer profit numériquement. A titre d'application, nous donnons une nouvelle méthode de développement formel des fonctions exp-logs modulo un oracle déterminant le signe des constantes exp-logs. Pour réaliser ceci, nous présentons un certain nombre d'outils effectifs généraux, permettant de généraliser ce résultat ultérieurement. Plus précisement, ce papier est le "fil guidant" pour la mise au point d'une théorie plus complète.

Mots clés : Développement asymptotique, fonction exp-log, transsérie, anneau ordonné, anneau asymptotique, belordre, algorithme.

1 Introduction

En mathématiques, les développements asymptotiques se rencontrent partout. Pourtant, il y a une grande divergence dans la façon dont les différents auteurs traitent ce sujet et même en ce qui concerne la définition propre des développements asymptotiques. En effet, les développements les plus simples sont de la forme $a_0 + a_1 x + a_2 x^2 + \cdots$ et on peut très bien s'y restreindre dans un cadre limité. De façon plus générale, on peut considérer des développements asymptotiques par rapport à une échelle, comme dans [Bour 51]. Mais l'inconvénient de cette définition est qu'il faut préciser l'échelle, or souvent l'échelle naturelle à prendre dépend précisement essentiellement du procédé qui a donné suite au développement asymptotique. On a proposé beaucoup d'autres définitions, presque toutes avec des désavantages (comment traiter des développements en plusieurs variables, quels coefficients prendre, etc.).

Nous ne prétendons pas pouvoir résoudre tous ces problèmes. En revanche, nous donnons dans la suite une définition très générale (probablement englobant beaucoup d'anciennes définitions), qui donne lieu à des méthodes très générales de calcul numérique, ainsi que formel. Avant de présenter notre démarche, rappelons l'origine des développements asymptotiques. Initinalement, il a dû y avoir deux raisons à leur introduction :

- Pouvoir exprimer le comportement limite d'une fonction à partir des fonctions usuelles.
- En tirer profit pour calculer des valeurs approchées de la fonction quand on s'approche de cette limite.

La première raison soulève la question de ce qu'est une fonction usuelle. Il ne semble pas exister de définition convenable, du fait, encore, que la définition dépende de la nature du processus qui a donné lieu à la fonction. C'est pourquoi, notre définition sera vague de ce point de vue, et nous utiliserons juste des symboles formels pour les fonctions de base à partir desquelles nous faisons nos développements. Nous remarquons néanmoins qu'il existe des classes de fonctions, comme les transséries (voir la section 3.3), ou les fonctions analysables (voir [Ec 92]), qui ont de bonnes propriétés de clôture en ce qui concerne la résolution des équations algébrico-différentielles à différences. Si on veut vraiment une définition de fonction usuelle, il faudrait probablement chercher dans cette direction.

La deuxième raison conduit à s'interroger sur la fabrication d'un procédé de calcul à partir d'un développement asymptotique. Ce procédé devrait fournir les termes du développement un par un et les sommer jusqu'à ce que la précision requise soit obtenue. Ceci donne à nouveau lieu à une subtilité : comment savoir si on a atteint ou si on peut atteindre cette précision. Lorsque l'on dispose, en plus du développement, d'une majoration des termes d'erreur, on peut détecter si la précision requise est obtenue, sans nécessairement pouvoir assurer que le processus s'arrête. Lorsque l'on sait que le développement converge, alors on peut obtenir la précision voulue en itérant suffisamment longtemps, sans nécessairement savoir quand. A cause de cette subtilité, il est utile de pouvoir utiliser des algorithmes légèrements différents, en fonction du cas où l'on se trouve, et qui peuvent, si possible, même dans le pire des cas, donner des approximations relativement convenables.

Pour en revenir sur les développements asymptotiques formels : la plus grande partie de ce papier leur est consacrée. A titre d'application nous nous sommes attachés à donner un nouvel algorithme de comparaison de fonctions exp-logs modulo un oracle décidant du signe d'une constante exp-log. L'existence d'un tel algorithme a été montrée pour la première fois dans [DaGö 84]. Shackell a été le premier à en obtenir explicitement un (voir [Sh 90]), mais son algorithme ne semble pas très applicable dans la pratique. Récemment, Gonnet et Gruntz ont donné un algorithme

pratique, qu'ils ont implanté et qui a de bonnes performances (voir [GoGr 92]). Dans les sections 2 et 3 nous développons un certain nombre d'outils permettant d'éclaircir les techniques qui peuvent être employées pour obtenir des développements asymptotiques et ceci même dans des cadres plus généraux. Ces techniques conduisent à de nouveaux algorithmes. En particulier nous avons retrouvé de façon indépendante l'algorithme de Gonnet et Gruntz. Cependant, il faut préciser que dans le cas du problème de la comparaison des fonctions exp-logs, nous n'avons pas véritablement besoin de nos nouvelles techniques. Nous avons pourtant choisi ce problème comme exemple d'application, parce que c'est une des applications les plus faciles. Pour l'avenir nous pensons qu'il pourrait y en avoir d'autres, mais ceci nécessitera de prolonger l'étude entreprise.

En ce qui concerne l'organisation des différentes sections : la deuxième section est consacrée aux belordres. Nous résumons d'abord un certain nombre de résultats classiques et en établissons ensuite quelques nouveaux. Pour un traitement plus complet nous renvoyons vers [VdH *a]. La troisième section porte sur l'algèbre asymptotique, le domaine qui nous fournit une base théorique par la suite. On peut la considérer comme un remplacant de la théorie des corps de Hardy (voir [Har 10], [Har 11]), utilisée par nos prédécesseurs. En effet, l'algèbre asymptotique comprend l'étude algébrique des relations asymptotiques comme la dominance, la comparaison, etc. dans un cadre très général. Cependant, nous ne développons que les parties esssentielles pour la suite de cette théorie. Pour une étude plus profonde, nous renvoyons vers [VdH *b].

La quatrième section concerne les méthodes générales pour obtenir formellement des développements asymptotiques (section 4.3) et pour les exploiter numériquement (section 4.2). La section 5 contient l'application aux fonctions exp-logs et nous y établissons également quelques résultats de nature structurelle sur les fonctions exp-logs. Un algorithme de comparaison des constantes basé sur les même techniques et modulo la conjecture de Schanuel en théorie de nombres sera donné dans [VdH *c]. Pour une discussion sur l'intérêt de cette conjecture dans notre contexte, je renvoie vers [Sh 92].

Pour terminer, je tiens à remercier Maurice Pouzet, Bruno Salvy et Jean Marc Steyaert pour les discussions fructueuses que j'ai pu avoir avec eux. Je remercie également Fabrice Lefebvre de l'aide qu'il m'a apporté lors de l'installation de divers logiciels, qui m'ont servi à rédiger cet article.

2 Introduction aux belordres

2.1 Introduction

Les belordres constituent le premier outil dont nous auront besoin par la suite. Un *belordre* est une relation d'ordre[1] bien fondé sans antichaîne infinie. Ceci veut dire qu'il n'y a pas de suite infinie strictement décroissante, ni d'ensemble infini d'éléments deux à deux incomparables. Leur intérêt vient du fait que plus tard un développement asymptotique sera une somme indexée par les éléments d'un belordre. Les principaux résultats sur les belordres concernent ce que nous appelons des *constructions élémentaires*. Plus précisement, on montre comment fabriquer des nouveaux belordres à partir des anciens à l'aide de ces constructions. Plus tard, ces constructions correspondent naturellement aux opérations élémentaires sur les développements asymptotiques.

Les belordres apparaissent sous beaucoup de formes en mathématiques et sont fréquemment redécouverts. Le premier résultat classique est le lemme de Dickson ($\mathbb{N}^n$ est un belordre). Ensuite, il y a les théorèmes de Higman et de Kruskal, concernant les mots et les arbres. Pour une discussion assez complète nous renvoyons vers [Pouz *]. Dans la section 2.2 nous résumons les résultats dont nous aurons besoin dans la suite. Dans la section 2.3 nous montrons sous quelques réserves que des constructions élémentaires de la section 2.2 sont en faite effectuables par algorithme. Dans la section 2.4 nous montrons que l'algèbre Booléenne engendrée par les parties finales d'un belordre vérifie des propriétés intéressantes vis-à-vis les constructions élémentaires.

Nous aurons également besoin de quelques dérivées subtiles de la notion de belordre. Par exemple, pour pouvoir faire des calculs numériques à l'aide d'un développement asymptotique, la notion légèrement moins forte d'*ordre inductif* (qui est un ordre bien fondé, dont chaque élément possède qu'un nombre fini de successeurs) suffit. On montre qu'en particulier tout belordre est un ordre inductif. On obtient une autre variante de la notion de belordre en lui imposant d'être propre (voir la définition dans la section 2.2). Or, la chose essentielle que toutes ces variantes des belordres ont en commun est la stabilité vis-à-vis des constructions élémentaires.

[1] Nous convenons qu'une relation d'ordre est une relation réflexive, transitive et antisymétrique. Les ordres dans ce papier seront donc a priori partiels et quand on aura affaire à des ordres totaux, nous le préciserons.

Nous remarquons également que certaines personnes préfèrent lever la condition d'antisymétrie dans la définition d'un belordre. Ce point de vue n'est pas plus général, du fait qu'à un tel *quasi ordre* $\leq$ on peut canoniquement associer un ordre en identifiant des éléments x et y, tels que $x \leq y \leq x$.

Enfin, nous commettrons souvent l'abus de langage de confondre les ensembles avec leurs structures sous-jacentes. Par exemple, on dira souvent que E est un belordre, au lieu de dire que E est un ensemble muni d'une relation de belordre. La même remarque s'applique dans les autres sections lorsque l'on considère des ensembles muni d'une structure algébrique.

2.2 Rappels sur les belordres

Rappelons d'abord un peu de terminologie. Une *partie finale* d'un ordre $\leq$ sur E est une partie $F \subseteq E$, vérifiant $x \in F \ \wedge \ x \preceq y \Rightarrow y \in F$. On note par $(A) = \{y \in E | \exists x \in A \ \ x \leq y\}$ la partie finale *engendrée* par $A \subseteq E$. Ensuite, la réunion disjointe de deux ordres est l'ordre sur la réunion disjointe des ensembles sous-jacents, dont les deux sommants sont mutuellements incomparables et dont l'ordre sur chaque sommant est l'ordre induit. L'ordre produit de deux ordres sur E resp. F est l'ordre par composants sur $E \times F$. Soit maintenant $(E, \leq_E)$ un ensemble ordonné et soit E^* l'ensemble de *mots* à valeurs dans E. On munit E^* de sa relation d'ordre "naturel" $\leq_{E^*}$ définie par : $x_1 \cdots x_n \leq_{E^*} y_1 \cdots y_m$, ss'il existe une application φ strictement croissante de $\{1, \cdots, n\}$ dans $\{1, \cdots, m\}$, telle que $x_i \leq_E y_{\varphi(i)}$, pour tout $1 \leq i \leq n$. C'est l'ordre d'abritement de Higman. On peut considérer la relation d'équivalence $\sim$ sur E qui rend équivalents tout couple de mots identiques à une permutation des lettres près. L'ensemble $E^\diamond = E^*/\sim$ est l'ensemble de *multiensembles* d'éléments de E. Comme $\leq$ est compatible avec $\sim$ (c.à.d. que $x \leq y$ et $x \sim x'$ implique l'existence d'un $y' \sim y$, tel que $x' \leq y'$), cet ensemble est canoniquement muni d'un ordre $\leq_{E^\diamond}$ (une classe est inférieure à une autre, si un représentant de l'un est inférieure à un représentant de l'autre). Par abus de notation, nous noterons les éléments de $E^\diamond$ encore par des mots, bien qu'il soit sous-entendu que les lettres commutent. Nous remarquons qu'un élément de $E^\diamond$ peut toujours être représenté par $x_1 \cdots x_n \in E^*$, où $x_i < x_j$ implique $i < j$. Si E est muni d'un ordre total, cette représentation est canonique.

Notre objectif principal dans cette section est de fabriquer des belordres en utilisant des constructions élémentaires, comme la somme, le produit, etc. Donnons d'abord une caractérisation des belordres.

Proposition 1. *Soit E un ensemble ordonné. Les trois propositions suivantes sont équivalentes :*

(a) l'ordre sur E est un belordre.
(b) Toute partie finale de E est finiment engendrée.
(c) Toute suite à valeurs dans E admet une sous suite extraite croissante.

Démonstration. Soit F une partie finale d'un belordre et $G \subseteq F$ l'ensemble des éléments minimaux de F. G est une antichaîne, donc fini. De plus G engendre F, car un belordre est bien fondé. Réciproquement, si $x_1, x_2, \cdots$ est une antichaîne infinie ou une suite strictement décroissante infinie, pour un certain ordre, la partie finale engendrée par $\{x_1, x_2, \cdots\}$ n'est pas finiment engendrée. Ceci démondre (a) $\Leftrightarrow$ (b).

Soit maintenant une suite $\{x_1, x_2, \cdots\}$ à valeurs dans l'ensemble E muni d'un belordre. Extrayons une sous suite croissante $\{x_{i_1}, x_{i_2}, \cdots\}$ par le procédé suivant : On appelle F_n la partie finale engendrée par les x_k, avec $k > i_n$ et $x_k \succeq x_{i_n}$ en convenant que $F_0 = E$. On suppose par récurrence que la suite admet une infinité

de termes dans F_n. Or F_n admet un nombre fini de générateurs et nous pouvons donc choisir un générateur $x_{i_{n+1}}$, avec $i_{n+1} > i_n$ et tel que la suite admet une infinité de termes dans F_{n+1}. Réciproquement, il est évident que l'on ne peut pas extraire une sous suite croissante d'une antichaîne infinie ou d'une suite strictement décroissante infinie. Ceci démontre (b) $\Rightarrow$ (c) $\Rightarrow$ (a). $\heartsuit$

Nous avons trois corrolaires triviaux de la proposition précédente, qui sont également trivialement vrais, si on remplace belordre par ordre inductif.

Corollaire 1. *Tout ordre total bien fondé est un belordre.*

Corollaire 2. *Toute réunion disjointe de deux belordres est un belordre.*

Corollaire 3. *Tout ordre produit de deux belordres est un belordre.*

Corollaire 4. *Soit $(E, \leq)$ un ensemble belordonné et $\sim$ une relation d'équivalence compatible avec $\leq$. Alors $E/\sim$ est belordonné par son ordre quotient.*

Un résultat plus profond sur les belordres est le

Théorème 1. (Higman) *Soit $(E, \leq_E)$ un ensemble belordonné. Alors E^* est belordonné par $\leq_{E^*}$.*

Démonstration. Raisonnons par l'absurde et supposons que l'ensemble des antichaînes infinies (vues comme des suites) $\mathcal{A}$ soit non vide. On muni $\mathcal{A}$ de l'ordre (lexicographique) $\leq_{\mathcal{A}}$ suivant : $\{x_i\}_{i\in\mathbb{N}} \leq_{\mathcal{A}} \{y_i\}_{i\in\mathbb{N}}$, si pour un certain n, les x_i et y_i sont égaux ou incomparables, pour $0 \leq i < n$ et $x_n \leq_{E^*} y_n$. En choisissant d'abord x_0, puis x_1 et ainsi de suite on s'aperçoit qu'il existe un $x \in \mathcal{A}$, qui est minimal pour $\leq_{\mathcal{A}}$.

Or, considérons maintenant la suite $\{x_{i,1}\}_{i\in\mathbb{N}}$. On peut en extraire une sous suite croissante $\{x_{\varphi(i),1}\}_{i\in\mathbb{N}}$. Maintenant considerons la suite

$$x_0, \cdots, x_{\varphi(0)-1}, x'_{\varphi(0)}, x'_{\varphi(1)}, \cdots,$$

où $x'_{\varphi(k)}$ désigne le mot $x_{\varphi(k)}$ privé de sa première lettre. Cette suite est inférieure à x, donc pas dans $\mathcal{A}$. On vérifie qu'il existe alors un $0 \leq j \leq \varphi(0) - 1$ et un k, tel que $x'_{\varphi(k)} \leq_{E^*} x_j$. On peut supposer j minimal. Il s'en suit que la suite

$$x_0, \cdots, x_{j-1}, x'_{\varphi(k)}, x'_{\varphi(k+1)}, \cdots,$$

qui est également inférieure à x appartient à $\mathcal{A}$, d'où une contradiction. $\heartsuit$

Corollaire. *Soit $(E, \leq_E)$ un ensemble belordonné. Alors l'ordre $\leq_{E^\diamond}$ sur $E^\diamond$ est également un belordre.*

On laisse au soin du lecteur de vérifier que lorsque E est muni d'un ordre inductif, alors il en est de même pour E^* et $E^\diamond$.

Nous terminons cette sous-section par quelques définitions. Nous disons qu'un élément x d'un ordre E est *atteignable*, s'il existe $x_1 \leq \cdots \leq x_n = x$ dans E, tel que chaque x_{i+1} soit un successeur de x_i et tel que x_1 est un élément minimal de E. Nous qualifierons de *propre* tout ordre, dont chaque élément est atteignable. Nous laissons au lecteur le de vérifier que les constructions élémentaires conservent la propreté d'un ordre, avec cette petite réserve que les seuls ordres totaux propres sont ceux qui sont isomorphes à un ordre fini ou à $\mathbb{N}$. Remarquons également qu'un ordre propre n'est pas toujours bien fondé. Enfin, nous disposons d'une caractérisation des belordres propres :

Proposition 2. *Soit E un ensemble belordonné. Alors les propositions suivantes sont équivalentes :*

(a) l'ordre sur E est propre.
(b) Chaque élément non minimal de E est successeur d'un élément.
(c) Pour tout $x \in E$, x est parmi les éléments minimaux du complément d'une partie initiale finie de E.

Démonstration. L'équivalence entre (a) et (b) est triviale et est même vérifiée lorsque l'ordre sur E n'est que bien fondé. L'implication (c) $\Rightarrow$ (b) est facile en remarquant que si pour $x \in E$, on prend une partie initiale I finie vérifiant les hypothèses, alors x est successeur d'un élément de I. Finalement, supposons que (b) soit vérifié et soit $x \in E$. Considérons l'ensemble $I_x = \{y \in E | y < x\}$. x est clairement un élément minimal de $E - I_x$. Montrons par l'absurde que I_x est fini. Si I_x est infini, il existe un prédécesseur x' de x (nécessairement sans I_x), tel que $I_{x'}$ est infini, car les prédécesseurs de x formant une antichaîne sont en nombre fini. En répetant ce procédé et en appliquant l'axiome du choix, nous construisons une suite descendante infinie, contredisant les hypothèses. ♡

2.3 Sur les belordres algorithmiques

Dans la suite nous aurons besoin de pouvoir faire des calculs effectifs à partir des ordres inductifs et des belordres. Ainsi, on dit qu'un ordre inductif E est *algorithmique*, si la comparaison est effective et si on peut calculer l'ensemble fini des éléments minimaux $\mathcal{M}_E$ de E ainsi que l'ensemble fini de successeurs $\mathcal{S}_E(x) = \mathcal{M}_{(x)-\{x\}}$ de chaque élément $x \in E$ par algorithme. De plus, nous exigeons que l'ordre soit propre. Comme exemples nous avons tous les ordres que l'on obtient à l'aide des constructions élémentaires :

1. Tout ensemble fini E muni d'un ordre algorithmique. $\mathcal{M}_E$ et $\mathcal{S}_E(x)$ peuvent être calculés de façon exhaustive. Dans les cas courant, où toutes les éléments sont comparables, où quand $E = \{1, \cdots, n\}$, il est possible de donner (on le laisse au soin du lecteur) des algorithmes plus rapides.

2. L'ensemble $\mathbb{N}$, avec $\mathcal{M}_{\mathbb{N}} = \{0\}$ et $\mathcal{S}_{\mathbb{N}}(n) = \{n+1\}$. L'ordre sur $\mathbb{N}$ est l'ordre classique.

3. Si E et F sont des ordres inductifs algorithmiques, alors $E \amalg F$ en est un, avec $\mathcal{M}_{E\amalg F} = \mathcal{M}_E \cup \mathcal{M}_F$ et $\mathcal{S}_{E\amalg F}(x) = \mathcal{S}_E(x)$ (resp. $\mathcal{S}_F(x)$), si $x \in E$ (resp. F). On a $x \leq_{E\amalg F} y$, ssi x et y appartiennent au même composant et x y est inférieur à y.

4. Si E et F sont des ordres inductifs algorithmiques, alors $E \times F$ en est un, avec $\mathcal{M}_{E\times F} = \mathcal{M}_E \times \mathcal{M}_F$ et $\mathcal{S}_{E\times F}((x,y)) = \mathcal{S}_E(x) \times \{y\} \cup \{x\} \times \mathcal{S}_F(y)$. On a $(x,x') \leq_{E\times F} (y,y')$, ssi $x \leq_E y$ et $x' \leq_F y'$.

5. Si E est un ordre inductif algorithmique muni d'une relation d'équivalence $\sim$ algorithmique (pour tout $x \in E$, sa classe d'équivalence est finie et on sait calculer ses éléments par algorithme) compatible avec la relation d'ordre sur E, alors $E/\sim$ en est un. Pour calculer $\mathcal{M}_{E/\sim}$, on calcule $\{\overline{x} | x \in \mathcal{M}_E\}$ et on supprime les éléments redondants. Pour calculer l'ensemble de successeurs de $\overline{x}$, on calcule $\{\overline{y} | y \in S_E(x') \ \wedge \ x' \in \overline{x}\}$. Enfin, on supprime à nouveau les éléments redondants.

6. Si E est un ordre inductif algorithmique, alors E^* en est un. On a $\mathcal{M}_{E^*} = \{\varepsilon\}$ et l'ensemble de successeurs de $x_1 \cdots x_n$ est la réunion de l'ensemble de mots de la forme $x_1 \cdots x_{i-1} y_i x_{i+1} \cdots x_n$, avec $y_i \in \mathcal{S}_E(x_i)$ et l'ensemble de mots de la forme $x_1 \cdots x_i y x_{i+1} \cdots x_n$, avec $y \in \mathcal{M}_E$. Pour comparer deux mots $x = x_1 \cdots x_n$ et $y = y_1 \cdots y_m$, on compare par récurrence x_i avec y_j (avec $i = j = 1$ au début) et on augmente i et j, si $x_i \leq_E y_j$ et que i, sinon. Lorsque i atteint $n+1$, on a $x \leq_{E^*} y$. Sinon, on a $\neg(x \leq_{E^*} y)$. On justifiera cet algorithme par la proposition 3.

Pour qu'un belordre sur E soit *algorithmique*, il faut de plus que pour tout couple $(x,y) \in E^2$ on sache calculer l'ensemble $\mathcal{G}_E(x,y) = \mathcal{M}_{(x)\cap(y)}$ d'éléments minimaux de $(x) \cap (y)$. Par une récurrence facile on peut alors construire un algorithme qui fait ceci pour tout ensemble fini $F \subseteq E$ (c.à.d. de calculer l'ensemble $\mathcal{G}_E(F)$ des éléments minimaux de $\bigcap_{x\in F}(x)$). En vue des corrolaires et du théorème de la section précédente, on peut maintenant remplacer ordre inductif par belordre dans les exemples précédents. Montrons comment calculer les $\mathcal{G}_E(x,y)$ dans ces cas :

1. On peut à nouveau calculer $\mathcal{G}_E(x,y)$ de façon exhaustive.

2. On a $\mathcal{G}_{\mathbb{N}}(x,y) = \{\max(x,y)\}$.

3. On a $\mathcal{G}_{E\amalg F}(x,y) = \mathcal{G}_E(x,y)$ (resp. $\mathcal{G}_F(x,y)$), si $x,y \in E$ (resp. F) et $\mathcal{G}_{E\amalg F}(x,y) = \emptyset$ sinon.

4. On a $\mathcal{G}_{E\times F}((x,y),(x',y')) = \mathcal{G}_E(x,x') \times \mathcal{G}_F(y,y')$.

5. Pour calculer $\mathcal{G}_{E/\sim}(\overline{x},\overline{y})$, on procède de façon analogue au calcul des successeurs.

6. $\mathcal{G}_{E^*}(x_1\cdots x_n, y_1\cdots y_m)$ est l'ensemble de mots (dont on élimine les éléments redondants), qui sont résultat du processus non déterministe suivant : On commence par un mot $z = \varepsilon$ et des variables $i = j = 1$. Puis, de façon repetée, on choisit parmi les options suivantes : On rajoute x_i à z et on augmente i, ou bien on rajoute y_j à z et on augmente j, ou bien on choisit une lettre dans $\mathcal{G}_E(x_i,y_j)$, que l'on rajoute à z et on augmente i et j. Si i atteint $n+1$ (ou j atteint $m+1$), on rajoute $y_j\cdots y_m$ à z (resp. $x_i\cdots x_n$) et l'algorithme est terminé. Cet algorithme est également justifié par la proposition 2.

Pour terminer, justifions les algorithmes de comparaison et du calcul d'intersection de parties finales dans le cas de $E^\diamond$. Pour les autres exemples la justification est facile et laissée au soin du lecteur.

Proposition 3. *Les algorithmes de comparaison et du calcul d'intersection de deux parties finales, présentés dans les exemples précédents, sont corrects.*

Montrons d'abord que l'algorithme de comparaison entre $x = x_1\cdots x_n$ et $y = y_1\cdots y_m$ est correct. Nous supposons que l'ensemble des applications de $\{1,\cdots,n\}$ dans $\{1,\cdots,m\}$ est muni de l'ordre lexicographique. Il est clair que lorsque l'algorithme décide que $x \leq_{E^*} y$, alors on a effectivement $x \leq_{E^*} y$. Supposons maintenant que $x \leq_{E^*} y$ et soit φ une injection croissante minimale pour l'ordre lexicographique tel que $x_i \leq_E y_{\varphi(i)}$, pour chaque i. A cause de cette minimalité, on augmente j dans l'algorithme précisement quand $j = \varphi(i)$. L'algorithme décidera donc également que $x \leq_{E^*} y$.

Montrons maintenant que l'algorithme d'intersection de deux parties finales est correct. Déjà on vérifie aisément que les mots z fabriqués par l'algorithme sont tous supérieurs à x et y. Soit maintenant un mot $m_1\cdots m_p \in (x_1\cdots x_n)\cap(y_1\cdots y_m)$ et montrons qu'il y a un z avec $z \leq m$. Soient φ et ψ tels que $x_i \leq m_{\varphi(i)}$ et $y_j \leq m_{\psi(j)}$, pour tout i et j. Nous pouvons supposer sans perdre de généralité que $Im(\varphi)\cup Im(\psi) = \{1,\cdots,p\}$. Alors montrons en effectuant l'algorithme, que l'on aurait pu choisir un $z_1\cdots z_p$, avec $z_i \leq m_i$, pour tout i. En effet, supposons que l'on se trouve à la position (x_i,y_j,m_k,z_k). Si $x_i \leq m_k$ et $y_j \leq m_k$, alors on aurait pu choisir z_k dans $\mathcal{G}_E(x_i,y_j)$, avec $z_k \leq m_k$. De plus, dans l'algorithme, on aurait augmenté i et j (et k). Si juste un élément parmi x_i et y_j est inférieur à m_k, disons x_i, alors on prend $z_k = x_i$ et on augmente i (et k). A la fin de l'algorithme, nous

avons bien les inégalités voulues entre les lettres de z et m. ♡

Remarque. Du point de vue de l'efficacité, les algorithmes précédents sont loins d'être optimaux. L'algorithme de calcul d'intersection de deux parties finales de mots, peut être optimisé en supprimant les éléments redondants en cours de route.

En ce qui concerne les multiensembles, on peut combiner les exemples 5 et 6, mais on peut gagner en vitesse en supposant que l'ordre $\leq_E$ est prolongé en un ordre total effectif $\leq'_E$ (en particulier, on peut faire ceci pour les constructions élémentaires). Après, on travaille sur les représentations canoniques de mots par rapport à $\leq'_E$. Du coup, on peut simuler les algorithmes sur les mots, ce qui évite de passer au quotient.

2.4 Sur les belordres pondérés algorithmiques

Soit E un ensemble. On appelle *algèbre booléenne* sur E tout sous-ensemble de $\mathcal{P}(\mathcal{P}(E))$, stable par intersection et complémentaire dans E et contenant l'ensemble vide. Tout sousensemble $\mathcal{G}$ de $\mathcal{P}(E)$ engendre une telle algèbre, notée $(\mathcal{G})$, car l'intersection de deux algèbres booléennes en est également une. Réciproquement toute partie d'une algèbre booléenne qui l'engendre est appellée *partie génératrice*. Soit A un groupe abélien. On appelle *pondération* de E (par rapport à une algèbre booléenne $\mathcal{B}$) toute application μ de $\mathcal{B}$ dans A, vérifiant $\mu(\emptyset) = 0$ et $\mu(X \amalg Y) = \mu(X) + \mu(Y)$, pour tout X et Y dans $\mathcal{B}$. Nous rappelons que étant donné une pondération, on peut exprimer le poids d'une réunion d'un nombre fini d'ensembles $X_1, \cdots, X_n$ en fonction des poids des intersections de ces ensembles. Cette relation est donnée par la *formule de crible* :

$$\mu(X_1 \cup \cdots \cup X_n) = \sum_{1 \leq i_1 \leq \cdots \leq i_k \leq n} (-1)^k \mu(X_{i_1} \cap \cdots \cap X_{i_k}).$$

De façon duale, nous avons

$$\mu(X_1 \cap \cdots \cap X_n) = \sum_{1 \leq i_1 \leq \cdots \leq i_k \leq n} (-1)^k \mu(X_{i_1} \cup \cdots \cup X_{i_k}).$$

Une pondération importante sur tout ensemble E est l'application χ qui à chaque partie de E associe sa fonction caractéristique dans $\mathbb{Z}^E$. On vérifie aisément que toute pondération μ se factorise $\mu = \varphi \circ \chi$, où φ est un morphisme de groupes. Soit maintenant $\mathcal{G}$ une partie de $\mathcal{P}(E)$. Alors nous avons également le lemme suivant :

Lemme 1.

(a) Une application $\mu : \mathcal{G} \to A$ se prolonge (éventuellement en élargissant A) en une pondération par rapport à $(\mathcal{G})$ ssi pour tout $\xi \in \mathbb{Z}^{(\mathcal{G})}$ telle que $\sum_{X \in \mathcal{G}} \xi(X)\chi_X = 0$, on ait $\sum_{X \in \mathcal{G}} \xi(X)\mu(X) = 0$.

(b) Ce prolongement est unique (et à valeurs dans A), si l'intersection de tout couple d'éléments de $\mathcal{G}$ est réunion disjointe d'un nombre fini d'éléments de $\mathcal{G}$.

Démonstration. Supposons que μ se prolonge en une pondération sur $(\mathcal{G})$ et soit $\xi \in \mathbb{Z}^{(\mathcal{G})}$ telle que $\sum_{X\in\mathcal{G}} \xi(X)\chi_X = 0$. Le support de ξ engendre une algèbre booléenne, qui est également engendrée par un nombre fini de parties deux à deux disjoints (appellé *base*). En décomposant sur la base, on trouve $\sum_{X\in\mathcal{G}} \xi(X)\mu(X) = 0$.

Supposons réciproquement que la condition de la partie (a) du lemme est vérifiée. Déjà, nous pouvons (et devons) définir $\mu(Y) = \sum_{X\in\mathcal{G}} \xi(X)\mu(X)$, pour toute partie Y telle que $\chi_Y = \sum_{X\in\mathcal{G}} \xi(X)\chi_X$. Nous remarquons que grace à la formule de crible une telle écriture est toujours possible si la condition de la partie (b) du lemme est vérifiée.

Supposons maintenant qu'il existe une partie Y dans $(\mathcal{G})$, telle que l'on puisse écrire $n\chi_Y = \sum_{X\in\mathcal{G}} \xi(X)\chi_X$, pour un certain $n \in \mathbb{N}^*$. Alors placons nous dans le surgroupe abélien B de A, obtenu en adjoignant formellement tous les éléments de la forme a/m à A, avec $a \in A$ et $m \in \mathbb{N}^*$. Alors dans B nous pouvons (et devons) poser $\mu(Y) = (1/n)\sum_{X\in\mathcal{G}} \xi(X)\mu(X)$. Remarquons que $\mathcal{G} = \{\{1,2\},\{1,3\},\{2,3\}\}$ est un cas simple dans lequel cette situation arrive.

Supposons finalement qu'il existe une partie Y dans $(\mathcal{G})$, telle que l'on ne puisse pas écrire $n\chi_Y = \sum_{X\in\mathcal{G}} \xi(X)\chi_X$, pour aucun $n \in \mathbb{N}^*$. Alors la condition de la partie (a) du lemme est vérifiée pour l'ensemble $\mathcal{G} \cup \{Y\}$ en choisissant n'importe quelle valeur pour $\mu(Y)$. Le lemme est maintenant aisément obtenu en appliquant le lemme de Zorn. ♡

Si l'ensemble E est belordonné, nous pouvons canoniquement lui associer l'algèbre booléene $\mathcal{B}(E)$ engendrée par les parties finales de E. Si on dispose d'une pondération μ par rapport à $\mathcal{B}(E)$, nous disons que E est un *belordre pondéré*. Nous remarquons que la partie (b) du lemme précédent implique qu'une telle pondération est déterminée par les poids des parties finales. De plus, la pondération est dite algorithmique, si on sait évaluer le poids de toute partie finale (donnée par l'ensemble fini de ses éléments minimaux) par algorithme. Si de plus E et A sont algorithmiques (pour A ceci veut dire que toutes les opérations de A et le test d'égalité se font par algorithme), alors on dit que E est un *belordre pondéré algorithmique*. On laisse au lecteur le soin de vérifier que l'on sait alors évaluer le poids de tout élément de $\mathcal{B}(E)$ par algorithme.

Nous allons généraliser les constructions élémentaires aux belordres pondérés algorithmiques. Si E est un ensemble fini, il faut et suffit de connaître le poids de chaque élément. Dans le cas où $E = \mathbb{N}$, il faudrait en outre connaître le poids de $\mathbb{N}$ tout entier. Si E et F sont des belordres pondérés algorithmiques, on définit la pondération $\mu_{E\amalg F} = \mu_E + \mu_F$ sur $E \amalg F$ par

$$\mu_{E\amalg F}(X) = \mu_E(X \cap E) + \mu_F(X \cap F).$$

Il est clair que cette pondération est algorithmique. Supposons maintenant que A est un anneau algorithmique (c.à.d. qu'à nouveau toutes les opérations ainsi que le test d'égalité se font par algorithme) et E et F des belordres pondérés algorithmiques à

valeurs dans A. Soit $\mathcal{G}$ l'ensemble des parties de $E \times F$ de la forme $X \times Y$, où X et Y appartiennent respectivement à $\mathcal{B}(E)$ et $\mathcal{B}(F)$. Alors nous pouvons associer un poids à ces parties par

$$\mu_{E\times F}(X \times Y) = \mu_E(X)\mu_F(Y).$$

Montrons que ceci induit une pondération algorithmique sur $E \times F$ (que l'on note $\mu_E\mu_F$) :

Proposition 4. *L'application $\mu_{E\times F}$ définie ci-dessus se prolonge en une pondération algorithmique sur $E \times F$.*

Démonstration. Nous remarquons que les conditions des parties (a) et (b) du lemme sont trivialement vérifiées (pour la partie (a), on utilise la distributivité dans A). Il suffit donc de démontrer que toute partie finale est dans $(\mathcal{G})$ et que son poids est calculable par algorithme. Soit $(x_1, y_1), \cdots, (x_n, y_n)$ une antichaîne de $E \times F$. Alors nous savons calculer les poids $\mu_E(X)$ et $\mu_F(Y)$ de $X = (x_1) \cap \cdots \cap (x_n)$ et $Y = (y_1) \cap \cdots \cap (y_n)$. Or $Z = ((x_1, y_1)) \cap \cdots \cap ((x_n, y_n)) = X \times Y$, donc $\mu_{E\times F}(Z) = \mu_E(X)\mu_F(Y)$. En appliquant ceci, ainsi que la formule de crible, on peut calculer le poids de toute partie finale de $E \times F$. ♡

Passons maintenant aux belordres quotients. Si E est un belordre pondéré algorithmique et $\sim$ une relation d'équivalence algorithmique sur E, nous munissons $E/\sim$ d'une pondération $\mu_E/\sim = \mu_{E/\sim}$ en imposant que pour toute partie finale X de $E/\sim$ on ait

$$\mu_{E/\sim}(X) = \mu_E(\{a \in E | \overline{a} \in X\}).$$

De plus, à partir de l'ensemble des éléments minimaux de X on peut facilement reconstruire l'ensemble des éléments minimaux du préimage par la surjection canonique. Il s'en suit que $E/\sim$ est également un belordre pondéré algorithmique. Ceci est en particulier le cas pour l'ensemble $E^n/\sim$, considéré comme sous-belordre de $E^\diamond$. Nous remarquons par ailleurs que $E^n/\sim \in \mathcal{B}(E^\diamond)$, car $E^0/\sim \cup \cdots \cup E^n/\sim$ est le complémentaire de la partie finale des mots d'au moins $n+1$ lettres.

La dernière construction élémentaire qui nous sera utile dans ce papier est la construction multiensemble. Nous nous occupons de la construction ensemble de mots dans [VdH *a]. Nous supposons cette fois ci que la pondération est à valeurs dans un corps algorithmique K (définition laissée au lecteur). Formellement, nous définissons le poids d'un élément $x_1 \cdots x_n \in E^\diamond$ par

$$\mu_{E^\diamond}(x_1 \cdots x_n) = f_n \mu_E(x_1) \cdots \mu_E(x_n),$$

pour certains éléments $f_0, f_1, \cdots$ dans K. Nous allons donner un sens à cette définition formelle. Soit $\mathcal{G}$ l'ensemble des parties de la forme $X^\diamond Y$, où $X \in \mathcal{B}(E)$ et $Y \in \mathcal{B}(E^n/\sim)$. On peut associer un poids à des telles parties, si $\mu_E(X) \neq 0$, par

$$\mu_{E^\diamond}(X^\diamond Y) = \frac{f(\mu_E(X)) - (f_{n-1}\mu_E(X)^{n-1} + \cdots + f_0)}{\mu_E(X)^n} \mu_{E^n/\sim}(Y),$$

pour une certaine fonction $f : A \to A$ (qui correspond formellement à la fonction définie par $f(x) = f_0 + f_1 x + \cdots$). Cette formule reste valide, si on assimile la fraction à f_n, lorsque X est de poids nul. Montrons maintenant que cette formule induit une pondération algorithmique sur $E^\diamond$ (que l'on note par $f(\mu_E)$) :

Proposition 5. *L'application $\mu_{E^\diamond}$ définie ci-dessus, pour $f : A \to A$ et des constantes $f_0, f_1, \cdots \in A$ calculables par algorithme, se prolonge en une pondération algorithmique sur $E^\diamond$.*

Démonstration. Montrons d'abord que $(\mathcal{G}) = \mathcal{B}(E^\diamond)$. Pour cela, il suffit de montrer que la partie finale X engendrée par un élément $x = x_1 \cdots x_n \in E^\diamond$ est dans $(\mathcal{G})$. Notons par X_i l'ensemble des mots de $E^\diamond$ qui sont supérieurs à un sousmot de x de i lettres, sans être supérieur à un sousmot de $i+1$ lettres. Comme le supplémentaire $\overline{X}$ de X est donné par $\overline{X} = X_0 \cup \cdots \cup X_{n-1}$, il suffit de montrer que les X_i sont dans $(\mathcal{G})$, pour chaque $0 \leq i \leq n-1$. Notons par S_i l'ensemble de sousmots de i lettres de x. Pour $S \subseteq S_i$ notons par J_S l'ensemble d'indices j tel que $x_j x'$ est un sousmot de x, pour un certain $x' \in S$. Notons également par Y_S l'ensemble des mots y de i lettres tel que S soit précisement l'ensemble de sousmots de x à i lettres, inférieurs à x. Alors S_i est donné explicitement par

$$X_i = \bigcup_{S \in S_i} \left(\overline{\bigcup_{j \in J_S} (x_j)} \right)^\diamond Y_S.$$

Il est clair que le complémentaire de $\bigcup_{j \in J_S}(x_j)$ est dans $\mathcal{B}(E)$. Il est facile à voir que l'on peut former Y_S en partant des (y), avec $y \in S_i$ et en utilisant des complémentaires et des intersections dans $\mathcal{B}(E^i)$.

Montrons maintenant que la condition de la partie (a) du lemme est vérifiée. Supposons donc que l'on ait

$$\xi_1 \chi_{X_1^\diamond Y_1} + \cdots + \xi_k \chi_{X_k^\diamond Y_k} = 0,$$

avec les $X_i^\diamond Y_i$ dans $\mathcal{G}$. Supposons d'abord que les X_i sont vides. Alors on peut regrouper les termes Y_i dont les mots ont même longueur et la condition est vérifiée en appliquant la proposition précédente et en passant au quotient. Supposons maintenant que les X_i ne sont pas tous vides. Quitte à rajouter des termes de la forme $\emptyset^\diamond Y$ à la somme, nous pouvons supposer que les mots dans les Y_i ont tous même longueur. Soit maintenant I l'ensemble des indices pour lesquelles $X_i = X_m$, où X_m est un élément maximal pour l'inclusion parmis $X_1, \cdots, X_k$. Quitte à réordonner les indices, on peut admettre que $I = \{l+1, \cdots, k\}$. Supposons qu'il existe un mot y tel que $\xi_{l+1}\chi_{Y_{l+1}}(y) + \cdots + \xi_k \chi_{Y_k}(y) \neq 0$. Alors soit $x \in X_m^\diamond$ un mot de la forme $x = x_1^p \cdots x_l^p y$, tel que $x_i \notin X_i$, pour $1 \leq i \leq l$. Pour p suffisamment grand un tel mot ne peut appartenir aux $X_i^\diamond Y_i$, avec $1 \leq i \leq l$. On obtient donc une contradiction en écrivant

$$0 = \xi_1 \chi_{X_1^\diamond Y_1}(x) + \cdots + \xi_k \chi_{X_k^\diamond Y_k}(x) = \xi_{l+1} \chi_{X_{l+1}^\diamond Y_{l+1}}(x) + \cdots + \xi_k \chi_{X_k^\diamond Y_k}(x) \neq 0.$$

On a donc $\xi_{l+1}\chi_{Y_{l+1}} + \cdots + \xi_k\chi_{Y_k} = 0$, d'où $\xi_1\chi_{X_1^\diamond Y_1} + \cdots + \xi_k\chi_{X_l^\diamond Y_l} = 0$. En repètant notre argument, on retombe donc toujours sur le cas où les X_i sont vides.

Montrons ensuite que la condition de la partie (b) du lemme est vérifiée. Alors soient $(X_1, X_2) \in \mathcal{B}(E)^2$ et $(Y_1, Y_2) \in \mathcal{B}(E^n) \times \mathcal{B}(E^m)$ et soit x un mot de longueur $l \geq n + m$ dans $Z = X_1^\diamond Y_1 \cap X_2^\diamond Y_2$. Alors nous pouvons désigner un sousmot de n lettres de x qui est dans Y_1 et un sousmot de m lettres de x qui est dans Y_2. Ainsi, nous pouvons factoriser $y = y'y''$, avec $y' \in (X_1 \cap X_2)^\diamond$ et $y'' \in X_1^m Y_2 \cap X_2^n Y_1$. Finalement, l'ensemble des mots $x \in Z$ de longueur $l < n+m$ est donné explitement par $X_1^{l-n} Y_2 \cap X_2^{l-m} Y_1$.

Montrons enfin que l'on sait calculer les poids des éléments de $(\mathcal{G})$ par algorithme. Déjà on sait calculer les poids des éléments de $\mathcal{G}$ par définition. Sinon le poids du complémentaire d'un élément X de $(\mathcal{G})$ est donné par $\mu_{E^\diamond}(\overline{X}) = f(\mu_E(E)) - \mu_{E^\diamond}(X)$. Il suffit donc (en utilisant à nouveau la formule de crible) de montrer que l'intersection de deux parties de $\mathcal{G}$ est une réunion finie disjointe de parties de $\mathcal{G}$ et que cette décomposition est effectuable par algorithme. Ce que nous avons démontré tout à l'heure. ♡

3 Introduction à l'algèbre asymptotique

3.1 Définitions et exemples

Pour la définition des développements asymptotiques, nous aurons besoin de fixer un cadre formel, dans lequel on dispose des opérations habituelles $+$ et $\cdot$, ainsi que des relations asymptotiques, comme la comparaison asymptotique et le symbole petit o (soit, de façon équivalente, d'une relation de prépondérance). Ainsi, on est amené à considérer des anneaux munis d'une structure supplémentaire. Dans la suite nous supposerons les anneaux commutatifs et unitaires, bien que beaucoup de résultats peuvent s'étendre au cas non commutatif.

Premièrement, on appelle *anneau ordonné* tout anneau A muni d'une relation d'ordre $\leq$ vérifiant (pour tout $a, b, a', b' \in A$) :

AO1. $a \leq b \ \wedge \ a' \leq b' \Rightarrow a + a' \leq b + b'$,
AO2. $0 \leq a \ \wedge \ 0 \leq b \Rightarrow 0 \leq ab$.

Dans le cas, où l'ordre $\leq$ est total, on parle d'*anneau totalement ordonné*. Nous remarquons que l'ordre est déterminé par l'ensemble des éléments positifs.

Exemple 1. Tout anneau habituel de germes de fonctions peut être muni d'une relation d'ordre. Par exemple $C_\infty(\mathbb{R})$ (l'anneau de germes de fonctions continues à l'infini) est un anneau ordonné pour la relation d'ordre suivant :

$$\overline{f} \leq \overline{g} \Leftrightarrow (\exists X \in \mathbb{R} \ \forall x > X \quad f(x) \leq g(x)).$$

L'anneau de germes de fonctions à deux variables par rapport à la première variable qui tend vers l'infini $C_{\infty,\cdot}(\mathbb{R}^2)$ est aussi un anneau ordonné pour l'ordre suivant :

$$\overline{f} \ll \overline{g} \Leftrightarrow (\ \exists X \in \mathbb{R} \ \forall x > X \ \forall y \in \mathbb{R} \quad f(x,y) \leq g(x,y)).$$

Deuxièmement, un *anneau asymptotique* est un anneau A, muni d'une relation $\ll$ d'*ordre asymptotique* vérifiant pour tout a, b et c dans A :

OA1. $0 \ll a$,
OA2. $a \ll b \Rightarrow -a \ll b$,
OA3. $a \ll c \ \wedge \ b \ll c \Rightarrow a + b \ll c$,
OA4. $a \ll b \Rightarrow ac \ll bc$,
OA5. $a \ll b \ \wedge \ b \ll c \Rightarrow a \ll c$.

Nous remarquons que cette notation est un peu troublante au niveau des relations d'ordres, car $\ll$ n'est pas un ordre strict (on a toujours $0 \ll 0$). Remarquons également que lorsque A est un corps, la relation $\ll$ est entièrement déterminée par le sous-anneau de A des éléments $a \ll 1$.

Exemple 2. Soit A un anneau ordonné et B un sousanneau de A. Nous pouvons définir un ordre asymptotique par

$$a \prec\!\!\!\prec_B b \Leftrightarrow \forall \varepsilon > 0, \varepsilon \in B \quad a \leq \varepsilon b.$$

On peut également définir un ordre asymptotique par

$$a \preceq_B b \Leftrightarrow \exists \varepsilon > 0, \varepsilon \in B \quad a \leq \varepsilon b.$$

Exemple 3. Tout anneau habituel de germes de fonctions est un anneau asymptotique, pour la relation de prépondérance, ainsi que la relation de dominance. Par exemple $C_\infty(\mathbb{R})$ est un anneau asymptotique, pour la relation de prépondérance suivante :

$$\overline{f} \ll \overline{g} \Leftrightarrow (\forall \varepsilon > 0 \; \exists X \in \mathbb{R} \; \forall x > X \quad |f(x)| \leq \varepsilon |g(x)|).$$

Il l'est également pour la relation de dominance définie par

$$\overline{f} \preceq \overline{g} \Leftrightarrow (\exists \varepsilon > 0 \; \exists X \in \mathbb{R} \; \forall x > X \quad |f(x)| \leq \varepsilon |g(x)|).$$

De façon analogue, on définit des ordres asymptotiques (uniformes) sur l'anneau $C_{\infty,\cdot}(\mathbb{R}^2)$.

Exemple 4. L'algèbre commutative donne également naturellement lieu à des anneaux asymptotiques. Soit par exemple $\mathfrak{i}$ un idéal de A. Nous pouvons définir la valuation $\mathfrak{i}$-adique d'un élément a de A :

$$\nu_\mathfrak{i}(a) = \max\{n \in \mathbb{N} | a \in \mathfrak{i}^n\}.$$

Alors nous définissons une relation d'ordre asymptotique par

$$a \preceq_\mathfrak{i} b \Leftrightarrow \nu_\mathfrak{i}(a) \geq \nu_\mathfrak{i}(b).$$

Si l'on suppose de plus que $\nu_\mathfrak{i}(ab) = \nu_\mathfrak{i}(a) + \nu_\mathfrak{i}(b)$ et $\nu_\mathfrak{i}(a) = +\infty \Leftrightarrow a = 0$, on peut également définir une relation d'ordre asymptotique par

$$a \prec\!\!\!\prec_\mathfrak{i} b \Leftrightarrow (\nu_\mathfrak{i}(a) > \nu_\mathfrak{i}(b)) \; \vee \; a = b = 0.$$

Exemple 5. Il y a une autre façon de fabriquer des anneaux asymptotiques en algèbre commutative. Supposons à nouveau que $\mathfrak{i}$ est un idéal d'un anneau A. Alors nous pouvons définir la relation d'ordre asymptotique suivante :

$$a \ll_\mathfrak{i} b \Leftrightarrow a \in \mathfrak{i}b.$$

Cette relation est généralement plus fine que les relations $\prec_\mathfrak{i}$ et $\prec\!\!\!\prec_\mathfrak{i}$ de l'exemple précédent. Considérons par exemple l'anneau des séries formelles $K[[X, Y]]$, d'idéal maximal $\mathfrak{m} = (X, Y)$. Alors $X^2 \prec\!\!\!\prec_\mathfrak{m} Y$, bien que $X^2 \not\ll_\mathfrak{m} Y$. Plus généralement, si A est un anneau local, d'idéal maximal $\mathfrak{m}$, l'ordre asymptotique $\ll_\mathfrak{m}$ coïncide plus ou moins avec la relation de divisibilité stricte. En effet, on a $a \ll_\mathfrak{m} b \Leftrightarrow (b|_{\neq} a \; \vee \; a = b = 0)$, avec $b|_{\neq} a \Leftrightarrow (a) \subset_{\neq} (b)$. Plus généralement, le symétrique de la relation de divisibilité est un ordre asymptotique.

3.2 Anneaux des séries formelles généralisées

Supposons maintenant que X est un semigroupe commutatif (pas d'inverse) ordonné ($\leq$ compatible avec la multiplication) et que C est un anneau. Nous allons définir l'anneau des séries formelles généralisées $C[[X]]$.

Nous posons $A = C[[X]] = C^{[X]}$, où $C^{[X]}$ est l'ensemble des applications de C dans X à support belordonné (pour l'ordre induit par X). Intuitivement, un élément a de A correspond à la somme $\sum_{x\in X} a(x)x$. Remarquons que plus généralement, on peut donner un sens aux sommes $\sum_{i\in I} c_i x_i$, où I est un ensemble belordonné et $\{c_i\}$ resp. $\{x_i\}$ des familles à valeurs dans C resp. X vérifiant $i \leq j \Rightarrow x_i \leq x_j$. En effet, pour $x \in X$ fixé, les indices i tels que $x_i = x$ forment une antichaîne et ils sont donc en nombre fini. On définit donc bien une application a de X dans C par $a(x) = \sum_{x_i=x} c_i$. De plus, le support de cette application est belordonné pour l'ordre induit par X, car c'est un sousensemble de l'image d'un ensemble belordonné par une application croissante.

Donnons un peu de terminologie. Les éléments de X sont appelés des *monômes* (et X est inclu canoniquement dans A). Les éléments de C sont appelés des *coefficients* (et C est également canoniquement inclu dans A). Si $a \in A$ et $x \in X$, on dit que $a(x)$ est le *coefficient* de x dans a. Enfin, notons par A^o (resp. par A^O) le sous-anneau de A formé des éléments a tels que $x \in \mathbf{supp}\, a \Rightarrow x > 1$ (resp. $x \in \mathbf{supp}\, a \Rightarrow x \geq 1$). Les éléments de A^o (resp. A^O) sont appelés éléments *infinitésimaux* (resp. *bornés*).

Montrons maintenant que l'on peut munir A d'une structure d'anneau. La somme de deux éléments a et b dans A se définit comme étant

$$a + b = \sum_{i\in \mathbf{supp}\, a \amalg \mathbf{supp}\, b} c_i x_i,$$

où $c_i = a(x)$ (resp. $c_i = b(x)$) et $x_i = x$, si i est l'image de x par l'injection canonique de $\mathbf{supp}\, a$ (resp. $\mathbf{supp}\, b$) dans $\mathbf{supp}\, a \amalg \mathbf{supp}\, b$. De même le produit de a et b est défini par

$$ab = \sum_{(x,y)\in \mathbf{supp}\, a \times \mathbf{supp}\, b} a(x)b(y)xy.$$

La compatibilité de l'ordre sur X avec la multiplication assure que cette définition est correcte. On vérifie aisément que les lois d'anneau sont vérifiées pour l'addition et la multiplication définies ainsi.

L'hypothèse du support belordonné entraîne également que toute série formelle $f \in C[[t]]$ (au sens habituel) induit une application $\hat{f} : A^o \to A^O$. En effet, écrivons $f(t) = f_0 + f_1 t + f_2 t^2 + \cdots$. Alors on peut poser

$$\hat{f}(a) = \sum_{x_1^{k_1}\cdots x_n^{k_n} \in (\mathbf{supp}\, a)^\diamond} f_{k_1+\cdots+k_n} a(x_1)^{k_1} \cdots a(x_n)^{k_n} x_1^{k_1} \cdots x_n^{k_n}.$$

En fait, on vérifie que l'on obtient ainsi un morphisme d'algèbres. A titre d'application nous remarquons que tout élément de $1 + A^o$ est inversible, car il suffit de

considérer $f(t) = 1 - t + t^2 + \cdots$. En particulier, on en déduit que A est un corps, si X était un groupe totalement ordonné.

Nous pouvons naturellement munir l'anneau $C[[X]]$ d'un ordre asymptotique $\prec\!\!\!\prec$ par $a \prec\!\!\!\prec b \Leftrightarrow a \in A^o b$. De façon analogue, on définit un ordre asymptotique $\preceq$ par $a \preceq b \Leftrightarrow a \in A^O b$.

Lorsque C est un anneau ordonné, nous pouvons également munir A de la structure d'anneau ordonné. En effet, soit P l'ensemble des éléments a dont les coefficients des éléments minimaux du support sont positifs. Alors P induit un ordre sur A, pour lequel il est précisement l'ensemble des éléments positifs. Nous remarquons que cet ordre est total, si les ordres sur C et X l'étaient. Dans ce cas, on a de plus une décomposition en somme directe de A, car $A = A^\uparrow \oplus C \oplus A^\downarrow$, où $A^\uparrow$ resp. $A^\downarrow$ sont les éléments de A à support strictement plus grand resp. petit que 1. Pour tout élément $a \in A$ on peut donc écrire de façon unique $a = a^\uparrow + a^c + a^\downarrow$, avec $a^\uparrow \in A^\uparrow$, $a^c \in C$ et $a^\downarrow \in A^\downarrow$.

3.3 Le corps des transséries

Dans cette section nous donnons une définition alternative du corps des transséries (voir [Ec 92]). L'intérêt de notre approche est qu'elle se généralise relativement facilement (voir [VdH *b]). La construction se fait de façon algébrique par une combinaison de différents types de clôture : la définition de ln pour des éléments $1 + x$ et x, avec x infinitésimal, la clôture par le logarithme et l'exponentiation et enfin la clôture par limite inductive.

Nos différentes opérations de clôture supposent chaque fois que l'on dispose d'un corps $K = \mathbb{R}[[X]]$, où X est un groupe totalement ordonné (donc K est un corps ordonné). De plus on suppose que l'on dispose d'une application logarithme ln partielle. Nous remarquons qu'une telle application partielle définit naturellement une application partielle d'exponentiation. En effet, nous posons $\exp \ln x = x$, pour tout x tel que $\ln x$ soit défini. Au fur à mesure nous élargissons le corps K en définissant ln pour de plus en plus de valeurs. Nous démarrons notre construction par $\mathbb{R}[[X]] \approx \mathbb{R}((x))$, avec $X \approx \mathbb{Z}$ et nous supposons que ln n'est défini que sur $\mathbb{R}_+^*$ de façon habituelle. Considérons maintenant les 4 types de clôture suivants :

Type I. (clôture élémentaire) : D'abord supposons que pour tous les $x, y \in X$ pour lesquels ln est défini, on a $\ln(xy) = \ln x + \ln y$. Remarquons ensuite que tout élément $a \in K$ puisse s'écrire $a = (c + \varepsilon)x$, avec $c \in \mathbb{R}$, ε infinitésimal et $x \in X$. Nous supposons également que si $\ln a$ est défini, alors $\ln c$, $\ln x$ et $\ln(1 + \varepsilon/c)$ le sont également par l'identité

$$\ln a = \ln x + \ln c + \hat{l}\left(1 + \frac{\varepsilon}{c}\right),$$

où l est une série formelle à coefficients dans $\mathbb{R}$, donnée par $l(x) = x - x^2/2 + \cdots$. Sous ces hypothèses, la clôture de K est précisement le corps $K' = K$ dans lequel

$\ln a$ est défini par la formule précédente, pour tout $a = (c + \varepsilon)x$, tel que $c > 0$ et tel que $\ln x$ est défini (ε étant infinitésimal, $c \in \mathbb{R}$ et $x \in X$). Nous remarquons que dans la clôture $\ln(ab) = \ln a + \ln b$, pour tous les a et b tels que $\ln a$ et $\ln b$ soient définis.

Type II. (clôture par logarithme) : Nous supposons que l'on puisse librement décomposer le groupe X comme produit $X = \{\cdots, 1/x, 1, x, \cdots\}X'$, avec $x >_X 1$ et $x < y$, pour tout $y \in X'$ strictement positif. De plus, on suppose que les éléments de X' sont les seuls éléments de X pour lesquels ln soit défini. Alors nous rajoutons librement un élément l à X. C.à.d. que nous considérons $X'' = \{\cdots, 1/l, 1, l, \cdots\}X$. On peut naturellement munir X'' d'une relation d'ordre total compatible avec la multiplication, en postulant que tout l^n, avec $n \geq 1$ est plus petit que tout élément strictement plus grand que 1 de X. Alors le corps K est naturellement inclu dans le corps $K' = \mathbb{R}[[X'']]$ et on peut prolonger le logarithme défini sur K en postulant que pour tout $x^n y \in X$, avec $y \in X'$ on ait $\ln(x^n y) = nl + \ln y$. Nous remarquons que le corps K' vérifie l'hypothèse faite au départ, pour pouvoir effectuer la clôture.

Type III. (clôture par exponentiation) : Nous supposons que ln est défini pour tout élément positif de K et que $\ln x \in K^{\uparrow}$, pour $x \in X$. Soit $X' = K^{\uparrow}$, muni de sa structure de groupe totalement ordonné. Alors nous pouvons plonger $K = \mathbb{R}[[X]]$ dans $K' = \mathbb{R}[[X']]$ en envoyant tout élément $x \in X$ vers $\varphi(x) = \ln x$ dans X' et en prolongeant par linéarité. Plus généralement, ln peut être défini pour tout élément x de X' par $\ln x = \varphi(x)$. Nous remarquons que la clôture élémentaire de K' (lorsqu'elle est bien définie) vérifie à nouveau l'hypothèse de départ pour pouvoir effectuer la clôture par exponentiation.

Type IV. (clôture par limite inductive) : Supposons maintenant que l'on a une suite $K_0 = \mathbb{R}[[X_0]], K_1 = \mathbb{R}[[X_1]], \cdots$ de corps telle que K_{n+1} est chaque fois naurellement inclu dans K_n. Ceci veut dire que X_{n+1} prolonge X_n et que ln est défini (de la même façon) dans K_{n+1}, pour tout élément dans K_n pour lequel il l'est. Alors $K = C[[X]]$, avec $X = X_0 \cup X_1 \cup \cdots$ contient naturellement tout les corps K_n.

Nous remarquons que les clôtures de type II,III et IV préservent les conditions d'applicabilité de la clôture élémentaire. De plus, nous remarquons que notre anneau de départ vérifie ces conditions.

Continuons maintenant notre construction. En appliquant à $K_0 = \mathbb{R}[[\mathbb{Z}]]$ de façon répétitive la clôture par logarithme on construit une suite de corps $K_0 = \mathbb{R}[[X_0]] \subset K_1 = \mathbb{R}[[X_1]] \subset \cdots$ que l'on peut clôter successivement par limite inductive et clôture élémentaire. Notons par $\mathbb{R}_0[[[x]]] = \mathbb{R}[[X]]$ le corps obtenu. Nous remarquons

que X est le groupe multiplicatif des éléments de la forme

$$\ln_{\boldsymbol{n}} x = \ln_{0^{n_0}\dots k^{n_k}} = x^{n_0} \ln^{n_1} x \cdots \ln_k^{n_k} x, \text{ avec}$$
$$\ln_k x = \ln^{k \text{ fois}} \cdots \ln x.$$

où $\boldsymbol{n}$ est un élément du groupe additif libre engendré par les éléments de $\mathbb{N}$. De plus on remarque que ln est défini pour tout élément strictement positif et qu'il envoie X dans $\mathbb{R}[[[X]]]^\dagger$.

Ensuite, soit de façon récursive $\mathbb{R}_{n+1}[[[x]]]$ le corps obtenu en sousmettant $\mathbb{R}_n[[[x]]]$ d'abord à une clôture par exponentiation et ensuite à une clôture élémentaire. La suite ainsi obtenue peut être clôtée par limite inductive. Enfin, nous notons par $\mathbb{R}[[[x]]]$ le corps obtenu en clôtant élémentairement cette limite inductive. On l'appelle le corps des *transséries*.

Remarque. Le corps des transséries défini ici ne vérifie que "l'axiome de bon ordre" ABO. Originalement le corps de transséries introduit par Ecalle vérifie des axiomes de finitude plus forts (voir [Ec 92]). Il est possible de modifier la construction précédente àfin que ces axiomes plus forts soient vérifiés. Par exemple, "l'axiome de finitude des générateurs pour toute série portée" AFG est vérifié lorsque l'on remplace la construction des séries formelles par une construction plus faible, où on exige en outre que le support de toute série soit inclu dans un sous groupe multiplicatif finiment engendré. De façon alternative, en interprétant les transséries comme des expressions (éventuellement infinies), on montre que l'on obtient des sous-corps de $\mathbb{R}[[[X]]]$, si on impose aux transséries de vérifier des axiomes de finitude supplémentaires.

Remarque. Le corps des transséries est stable par beaucoup d'opérations, comme la composition, dérivation, intégration, etc. De plus, les opération algébriques correspondent aux opérations analytiques, si la transsérie a un sens analytique. Ce fait, facile à démontrer pour les expressions exp-logs, sera admis implicitement dans la suite. Dans des cas plus compliqués, il est parfois nécessaire de donner un sens aux transséries en utilisant le procédé de l'acceléro-sommation. Le lecteur intéressé pourra se reporter à [Ec 92] ou à [VdH *b], pour des transséries plus générales.

4 Sur les développements asymptotiques

4.1 Introduction

Dans le cadre le plus général, nous disons qu'un développement asymptotique est une somme formelle

$$\sum_{i\in I} c_i x_i, \tag{1}$$

où les a_i et les c_i sont dans un anneau asymptotique A pour $\prec\!\!\!\prec$ et où I est un belordre pour la relation d'ordre partiel $<$ définie par $x < y \Leftrightarrow x \succ\!\!\!\succ y \ \wedge\ x \not\prec\!\!\!\prec y$. On les qualifie de *développements asymptotiques théoriques*. Nous remarquons que cette définition permet aux coefficients c_i d'être absolument quelconques. Il est coûtume que les coefficients sont bornés sans tendre vers 0. Algébriquement, on appelle sous-anneau *régulier* de A, tout sous anneau, dont les éléments c non nuls vérifient $c \preceq 1$ et $c \not\prec\!\!\!\prec 1$. Ici $c \preceq 1 \Leftrightarrow \forall x \prec\!\!\!\prec y \quad cx \prec\!\!\!\prec y$. Si les c_i sont tous dans un tel sous-anneau, on dit que le développement est *régulier*. Nous remarquons que dans ce papier on n'a pas tellement besoin de la définition dans toute sa généralité ; A partir de la section 4.3 nous ne considérons que le cas, où $A = C[[X]]$. De plus, à partir de la section 5, C et X sont totalement ordonnés.

Bien que l'ensemble des développements asymptotiques théoriques possède une structure intéressante (voir [VdH *a]), dans la pratique il faut exiger en outre que le belordre soit algorithmique. Cependant, souvent, et notamment si l'ordre sur I n'est pas propre (voir aussi plus bas), il est nécessaire d'affaiblir la relation d'ordre partiel. Ainsi, par *développement asymptotique* nous entendons une somme formelle de la forme (1), où I est un belordre algorithmique, tel que $i \leq j \Rightarrow x_j \prec\!\!\!\prec x_i$. De plus, on demande que les c_i et x_i soient calculables par algorithme. Si I n'est qu'un ordre inductif algorithmique on parle de *développement asymptotique au sens faible*.

Le seule point qui pourrait éventuellement paraître non naturel dans notre définition est que celle-ci donne lieu à l'existence de développements "farfelus" du genre (pour $x \to \infty$)

$$1 \approx (1 - \frac{1}{x}) + (\frac{1}{x} - \frac{1}{x^2}) + (\frac{1}{x^2} - \frac{1}{x^3}) + \cdots,$$

ou pire encore (pour le belordre $\mathbb{N} \amalg \mathbb{N}$) :

$$\begin{aligned} 0 \ &\approx\ 1 + \frac{1}{x} + \frac{1}{x^2} + \frac{1}{x^3} + \cdots + \\ &-\ 1 - \frac{1}{x} - \frac{1}{x^2} - \frac{1}{x^3} - \cdots. \end{aligned}$$

Cette situation devient d'autant plus pénible lorsque l'on veut développer des fonctions comme

$$f(x) = e^{\frac{1}{x} + \frac{1}{e^{-x}}} - e^{\frac{1}{x}}.$$

Or ce genre d'annulations ne sont d'habitude pas génants pour le calcul numérique à partir des développements. En outre la détection des annulations ne peut se faire que dans un cadre algébrique, où on dispose de propriétés algébriques sur le développement en entier (par exemple dans le cas de la fonction f, on a une expression explicite pour f). On verra comment s'exploite ce genre de connaisances dans la section 4.3 et on y utilise crucialement la notion du belordre. Dans l'absence de tels connaisances algébriques, nous sommes obligés de permettre des développements asymptotiques farfelus.

Un avantage important de notre définition est qu'elle autorise des "semi doubles développements". Illustrons cette idée. Classiquement, on a le développement asymptotique suivant, pour $x \to \infty$:

$$e^{\frac{100}{x}+\frac{1}{\ln x}} \approx 1 + \frac{1}{\ln x} + \frac{1}{2\ln^2 x} + \frac{1}{6\ln^3 x} + \cdots.$$

Mais pour des "grandes valeurs" de x, comme $x = 100$, l'erreur commise quand on calcule $f(x)$ par son développement est très grande. Pour des valeurs encore plus grandes ceci s'arrange, mais l'erreur reste sensible. Dans ce cas, on est donc mené à envisager des "doubles développements" comme

$$\begin{array}{rcccccccc} f(x) & = & 1 & + & \dfrac{1}{\ln x} & + & \dfrac{1}{2\ln^2 x} & + & \cdots \\ & + & \dfrac{100}{x} & + & \dfrac{100}{x\ln x} & + & \dfrac{50}{x\ln^2 x} & + & \cdots \\ & + & \dfrac{5000}{x^2} & + & \dfrac{5000}{x^2\ln x} & + & \dfrac{2500}{x^2\ln^2 x} & + & \cdots \\ & + & \cdots & & & & & & \end{array}$$

C'est ici que l'on remarque que le belordre sousjacent est l'ordre produit sur $\mathbb{N}^2$ et non l'ordre lexicografique, qui n'est pas propre et donne lieu à "l'oubli" des termes non atteignables. Maintenant, pour obtenir une bonne approximation de $f(x)$ pour un x donné, on prendra un nombre fini de termes dans les deux directions et de préférence horizontalement. Dans la pratique, on tombe souvent sur ce genre de cas. Un exemple concret est la méthode d'analyse de singularités (voir [FlO 90]), qui nous fournit systématiquement ce style de développements. Enfin, on remarque que les développements asymptotiques en plusieurs variables conduisent toujours à de véritables doubles (et multiples) développements.

4.2 Evaluation numérique d'un développement asymptotique

Passons maintenant à l'algorithme d'évaluation numérique à l'aide d'un développement asymptotique au sens faible. Dans sa forme la plus élémentaire, on part d'un développement asymptotique au sens faible (1), d'une somme $S := 0$ et de deux

sous-ensembles $J, K \subseteq I$, initialement égaux à $\mathcal{M}_I$, resp. $\emptyset$. Ensuite, on détermine par récurrence le $i \in J$, pour lequel $|c_i x_i|$ est maximal. On rajoute $c_i x_i$ à la somme par $S := S + c_i x_i$ et on pose $K = K \cup \{i\}$ et puis $J := J \cup \mathcal{S}_I(i) - K$. L'algorithme s'arrête quand les $c_i x_i$ sont tous suffisamment petits.

Cette première version n'est pas très stable numériquement, car l'algorithme ne s'arrête pas nécessairement et surtout pas dans les cas où le développement est divergent. De plus, il peut se passer qu'accidentellement $c_i x_i$ soit très petit, mais que $c_j x_j$ soit grand pour un j dans la partie finale engendré par i. Voyons comment remédier à ces trois objections. Enfin, il faut donner un sens à "suffisamment petit", ce qui s'avère difficile.

Pour la première objection, prenons le point de vue des astrologues (voir [P 1893]). Pour des développements classiques, on fait l'hypothèse implicite que l'erreur commise quand on arrête le développement à un certain ordre est moindre que la valeur absolue du dernier terme. Dans notre cas, ceci mène à ne plus rajouter un successeur j de i à J, lorsque $|c_j x_j|$ est supérieur à $|c_i x_i|$. Ainsi, on peut correctement sommer la plupart des développements divergents.

En ce qui concerne la deuxième objection, l'algorithme devrait être capable de "regarder un peu en avance". Au lieu de ne rajouter que les successeurs de i à J, on pourrait donc également rajouter ses successeurs itérés. De même, on commence pas juste avec les éléments minimaux de I dans J, mais on y rajoute déjà leurs successeurs itérés.

La troisième objection mène à des développements asymptotiques *avec terme d'erreur*, où en plus du développement asymptotique on dispose pour chaque $i \in I$, d'une majoration

$$|\sum_{j \geq i} c_j x_j| \leq C_i x_i.$$

Dans ce cas, on arrête l'algorithme, si l'erreur totale $E_J = \sum_{j \in J} C_j h_j$ est inférieure à la précision requise, ou, en tenant compte de la première objection, ne descend plus suffisamment. Nous remarquons d'ailleurs qu'avec des techniques analogues à celles de la section suivante on pourrait donner des estimations d'erreur E_J pour $|\sum_{i \geq j, j \in J} c_i x_i|$ plus fins.

Même en employant ces trois améliorations simultanément, il peut se faire que l'on n'obtienne pas un résultat d'une précision convenable. Dans ce cas, le développement asymptotique ne comportait pas suffisamment de renseignements pour les applications numériques que l'on envisageait. C'est à ce moment qu'il faut repartir de la signification algébrique du développement et utiliser des transformations permettant d'obtenir la précision requise. Ici, des techniques de resommation pourront être utiles (voir [Ec 92]).

4.3 Linéarisation d'un développement asymptotique

Supposons maintenant que $A = C[[X]]$, où C est un anneau et X un groupe ordonné. Cet anneau est un anneau asymptotique pour chacune des relations $\prec\!\!\prec$ et $\preceq$ (voir la section 3.2). Dans l'introduction on a vu que notre définition de développement asymptotique pouvait donner suite à des développements "farfelus". Dans cette section, nous montrons comment on peut remédier à cette objection, si l'on dispose de suffisamment de renseignements algébriques "globaux" sur la somme formelle $f = \sum_{i \in I} c_i x_i$.

En effet, nous allons montrer comment *linéariser* un développement asymptotique. Par ceci, on entend que l'on peut réécrire

$$f = d_1 x_{i_1} + \cdots + d_n x_{i_n} + \sum_{i \in I'} c_i x_i,$$

pour tout $n \in \mathbb{N}$, avec $1 \leq j < k \leq n \Rightarrow x'_{i_j} \not\preceq x'_{i_k}$ et $1 \leq j \leq n, i \in I' \Rightarrow x'_{i_j} \not\preceq h_i$. Ici, $i_1, \cdots, i_n \in I$, $I' \subseteq I$ et les $d_1, \cdots, d_n$ sont des combinaisons linéaires des c_i. Eventuellement, nous autorisons également que la linéarisation s'arrête, c'est à dire que l'on a l'éciture précédente pour un certain n et $I' = \emptyset$. Or, pour pouvoir linéariser il faut justement pouvoir détecter des développements farfelus, c'est à dire des développements présentant des phénomènes d'annulation importants. C'est pour ceci que nous avons besoin des belordres pondérés algorithmiques.

Plus précisément, nous disons qu'un développement asympotique $\sum_{i \in I} c_i x_i$, avec les c_i resp. x_i dans C resp. X est *algorithmique*, si l'application μ définie (pour $J \subseteq I$) par

$$\mu(J) = \sum_{j \in J} c_i x_i$$

est une pondération algorithmique de I. Ceci suppose que l'on sache représenter de façon algorithmique le sous-groupe additif de A, engendré par les $\mu(J)$, où J est une partie finale de I. Alors nous avons le théorème suivant :

Théorème 2. *On peut linéariser tout développement asymptotique algorithmique.*

Démonstration. Considérons l'algorithme suivant : On commence par initaliser les variables $J := \mathcal{M}_I$ et $S := 0$. Par récurrence, on détermine un indice $j \in J$ tel que x_j soit maximal (pour $\prec\!\!\prec$) et le sous-ensemble $K \subseteq J$, d'indices k, avec $x_k = x_j$. On calcule la somme $s = \sum_{i \in K} c_i x_i$. Si cette somme est non nulle, on la rajoute à S par $S := S + s$ et on passe à l'étape suivante, en posant $J := J \cup \mathcal{S}_I(K) - K$. Eventuellement, on ne gardera que les éléments minimaux de J. Si $s = 0$, on teste si la somme $\sigma = \sum_{i \in (K) - (J-K)} c_i x_i$ est nulle. Si σ est non nulle, on passe à l'étape suivante comme auparavant. Si $\sigma = 0$, on passe à l'étape suivante en posant $J := J - K$.

Montrons maintenant, que si on arrête l'algorithme après un nombre suffisant d'étapes, on obtient une linéarisation du développement asymptotique à l'ordre n. Il

est facile à voir que les termes fournis par l'algorithme forment les premiers termes d'une linéarisation du développement. Il suffit donc de montrer que s ne peut être nulle une infinité de fois de suite. Montrons ceci par l'absurde. Quitte à enlever les premiers termes du développement, on peut supposer que $s = 0$ durant toute l'exécution de l'algorithme. Soit L la réunion des K que l'on rencontre lors de l'execution. L est manifestement une partie initiale de I. De plus, puisque la relation de belordre sur I est propre, il y a un moment pendant l'exécution, où J contient les éléments minimaux de $I - L$. Or à ce moment, σ devient nulle et $J \cap L$ devient vide l'itération d'après. Alors K ne peut plus être inclus dans L, ce qui contredit nos hypothèses. ♡

Bien que l'on vient de démontrer que l'algorithme nous fournit toujours des nouveaux termes, il n'y a en général pas moyen de prédire si l'algorithme s'arrête. Or ceci est plutôt une question de nature algébrique, car elle se ramène souvent à savoir si f appartient à un certain anneau.

Une autre question que l'on peut se poser est de linéariser jusqu'à un certain ordre. Il est clair que l'on peut généraliser l'algorithme pour qu'il "jette" tous les termes x_i, avec $x_i \prec\!\!\!\prec x'$ (ou $x_i \preceq x'$), pour un certain $x' \in X'$, où X' est un sous ensemble fini de X, appelé *frontière*. Une autre question qui se pose alors est de savoir s'il y a une expession explicite pour les termes que l'on vient de jeter. Plus précisement, si J est la partie finale de I d'indices i avec $x_i \prec\!\!\!\prec x'$, pour un certain $x' \in X'$, alors peut-on trouver l'ensemble des éléments minimaux de J ? Il s'en suit évidemment qu'alors on peut calculer $\mu(J)$. Montrons que ceci est le cas, lorsque pour toute partie dans $\mathcal{B}(I)$ on sait décider si elle contient un indice i, tel que $x_i \not\prec\!\!\!\prec x'$, pour tout $x' \in X'$.

Lemme 2. *Soit $X' \subseteq X$ une frontière. Si pour toute partie J dans $\mathcal{B}(I)$ on sait tester par algorithme s'ils existent $i \in J$ et $i \in X'$, avec $x_j \preceq x'$, alors on peut déterminer par algorithme les éléments minimaux i de I, avec $x_i \preceq x'$ pour un certain $x' \in X'$.*

Démonstration. Nous commençons par poser $J = \mathcal{M}(I)$ et $M = \emptyset$. Maintenant nous effectuons la récurrence suivante : nous déterminons d'abord les indices $j \in J$ tels que $x_j \preceq x'$, avec $x' \in X'$ et nous les transferrons vers M. Ensuite, nous testons pour tout $j \in J$ si $((J) - (J - \{j\}))$ contient des indices i, avec $x_i \preceq x'$ et $x' \in X'$. Si ceci est le cas, nous enlevons j de J. Si à l'issu de cette boucle J est vide, l'algorithme s'arrête et M est l'ensemble des éléments minimaux cherchés. Sinon on remplace J par son ensemble de successeurs et on continue. L'algorithme se justifie et on montre sa terminaison par la même technique que celle de la démonstration du théorème 2. ♡

5 Applications

5.1 Introduction

Dans cette section nous donnons quelques algorithmes de comparaison de fonctions exp-logs réelles, en supposant qu'il y a un oracle qui peut donner le signe d'une constante exp-log. Nous rappelons qu'une constante $exp - log$ est un élément de $\mathbb{R}$ obtenu à partir de $\mathbb{Q}$ en utilisant les opérations du corps, ainsi que le logarithme et l'exponentiation. L'existence d'un tel algorithme a été montré dans [DaGö 84] et pour une bonne introduction au problème nous renvoyons vers [Sh 90], où un tel algorithme était explicité pour la première fois. Un algorithme plus pratique a été trouvé par Gonnet et Gruntz (voir [GoGr 92]) et cet algorithme a été implanté avec succès. Nous remarquons que nous avons trouvé ce dernier algorithme de façon indépendante.

Nous présentons ici trois algorithmes différents de développement d'une expression exp-log (dont en particulier un algorithme de comparaison). Le premier donne un développement sous "forme normale". En l'occurence ceci veut dire que les fonctions représentées par les termes fournis par l'algorithme dépendent exclusivement de la fonction représentée par l'expression exp-log du départ. De plus les opérations de corps (des fonctions exp-logs) se traduisent naturellement par les opérations correspondantes sur les développements. Le calcul d'un développement sous forme normale peut faire exploser le calcul. Ainsi, nous donnons un deuxième algorithme qui ne présente plus ce désavantage, mais qui ne calcule plus les développements sous forme normale. Cet algorithme présente en outre l'avantage que l'on ne développe que les sous-expressions nécessaires lors du calcul. Le dernier algorithme est celui de Gonnet et Gruntz, qui est obtenu naturellement en optimisant l'algorithme de linéarisation d'un développement asymptotique. Dans tout les algorithmes nous nous servons implicitement du fait qu'il existe un algorithme pour tester si une fonction exp-log est nulle (voir [] et []).

Il est nécessaire de faire une remarque sur l'utilité des techniques présentées dans les sections 2 et 3. En fait, l'exemple des fonctions exp-logs étaient un peu mal choisi, car l'algorithme de Gonnet et Gruntz montre que le concept de *sous-expressions variant au plus vite* y est plus utile dans la pratique. Cette simplification est essentiellement due au fait que les transséries qui représentent des fonctions exp-logs vérifient l'axiome de finitude AFG (voir la section 3.3). Il en est de même pour la plupart des fonctions issues d'une source naturelle (voir [VdH *d]). En revanche, ceci n'est plus le cas pour les solutions de beaucoup d'équations aux différences, comme

$$f(x) = \frac{1}{x} + f(x^2) + f(e^{-\ln^2 x}).$$

On tombe dans une situation semblable, quand on considère certains développements

explicites, comme

$$f(x) = \sum_{n\in\mathbb{N}} \exp(-x^{n^2+3n+1}).$$

Ici les techniques de la section 2 pourraient être les seules à être complètement générales. Une autre application éventuelle est le traitement des transséries en plusieurs variables, où les ordres à considérer ne sont plus totaux. Enfin, nos techniques permettrons peut être de mieux comprendre la structure théorique des développements asymptotiques.

5.2 Algorithme de développement sous forme normale

Dans un premier temps nous allons nous intéresser à des expressions exp-logs bien définies (on ne prend des logarithmes que des expressions positives), qui ne font pas intervenir l'exponentiation. On les appelle expressions logarithmiques et elles (plus précisement : les éléments qu'elles représentent) forment un sous-corps de $\mathbb{R}_0[[[x]]]$. Nous allons d'abord donner un algorithme $DL(x)$, qui nous fournit le développement d'une telle expression f par rapport à l'échelle des fonctions $\ln_{\boldsymbol{n}} x$. Pour faire ceci, nous allons associer à f son développement asymptotique algorithmique correspondant et utiliser le théorème 2. Il faut donc associer un belordre pondéré algorithmique à f.

Si f est de la forme $c\ln_{\boldsymbol{n}} x$, on lui associe un singleton de poids f (ou l'ensemble vide, si $c = 0$). Supposons que f est de la forme $f_1 + f_2$. Soient (B_{f_1}, μ_{f_1}) et (B_{f_2}, μ_{f_2}) les belordres pondérés algorithmiques associés par récurrence à f_1 et f_2. Alors on associe la réunion disjointe $(B_{f_1} \amalg B_{f_2}, \mu_{f_1} + \mu_{f_2})$ de ces deux belordres pondérés à f. De même, si f est de la forme $f_1 f_2$, on considère le belordre pondéré algorithmique produit. En ce qui concerne l'opposé $f = -f'$, il suffit de prendre l'opposé de la pondération : $\mu_{f'}(X) = -\mu_f(X)$. Supposons maintenant que $f = 1/f'$. Traitons d'abord le cas où $f' = 1 + g$, avec $g \prec\!\!\prec 1$. Comme belordre pondéré algorithmique on prend $B_f = B_g^\diamond$, muni de $\mu_f = 1/(1 - \mu_g)$. Le cas général se ramène à ce cas en écrivant $f = (1/P)(1/(f'/P))$, où P est le terme principal de f' (calculable par récurrence). Finalement le cas $f = \ln f'$ se traite de la même façon : on suppose d'abord que $f' = 1 + g$ et en deuxième instance on écrit $\ln f = \ln P + \ln(f'/P)$, comme tout à l'heure (on remarque que $\ln P$ est une somme finie). De plus ceci nous donne une méthode de vérification si $\ln f$ est bien défini, en vérifiant le signe de P. En effet, en regardant (récursivement) le premier terme du développement de $f - g$, nous avons également un algorithme de comparaison de fonctions logarithmiques f et g. De même, nous avons un algorithme récursif pour tester si une fonctions domine une autre. Pour les fonctions de la forme $\ln_{\boldsymbol{n}} x$, c'est juste l'ordre lexicographique sur $\mathbb{N}^\diamond$. Pour des fonctions plus générales, on compare leurs monômes dominants.

Pour étendre notre étude aux fonctions exp-logs nous avons besoin de quelques autres algorithmes. D'abord, en notant $\mathbf{supp}\, f$ le support de f (l'ensemble sousjacent des monômes pour la pondération μ_f), nous allons démontrer qu'ils existent des

monômes d_f et $s_f \prec\!\!\prec 1$ vérifiant

$$\begin{cases} \forall x \in \mathbf{supp}\, f \;\; \exists n \geq 0 & x \succeq d_f s_f^n, \\ \forall n \geq 0 \;\; \exists x \in \mathbf{supp}\, f & x \preceq d_f s_f^n. \end{cases}$$

En effet, nous allons donner un algorithme établissant de tels d_f et s_f. Les propriétés énoncées en haut se vérifient aisément par récurrence.

Si f est de la forme $c \ln_{\boldsymbol{n}} x$, nous prenons $d_f = \ln_{\boldsymbol{n}} x$ et $s_f = 1$. Si f est de la forme $f_1 + f_2$, nous prenons pour d_f le plus petit (pour $\prec\!\!\prec$) de d_{f_1} et d_{f_2}, mettons d_{f_1}. Ensuite, nous testons si $\ln s_{f_2} \leq \ln(d_{f_1}/d_{f_2})$. Si ceci est le cas, on prend s_{f_1} pour s_f. Sinon, on prend s_{f_2}. Supposons maintenant que $f = -f'$. Alors nous prenons $d_f = d_{f'}$ et $s_f = s_{f'}$. Ensuite supposons que f est de la forme $f_1 f_2$. Alors nous posons $d_f = d_{f_1} d_{f_2}$ et $s_f = \min_{\prec\!\!\prec}(s_{f_1}, s_{f_2})$. Enfin, supposons que f est de la forme $1/(1+g)$, avec $g \prec\!\!\prec 1$ (le cas $f = 1/f'$ se ramène à ce cas en multipliant). Alors on pose $d_f = 1$ et $s_f = \min_{\prec\!\!\prec}(d_g, s_g)$. Le cas $f = \ln(1+g)$ se traite de façon similaire.

Déduisons maintenant de l'algorithme de calcul de d_f et s_f un algorithme pour calculer $f^\uparrow$, f^c et $f^\downarrow$ dans la décomposition $f = f^\uparrow + f^c + f^\downarrow$. Or, pour toute partie dans $\mathcal{B}(I)$ nous pouvons calculer son poids g. Du coup, nous pouvons tester si le support de g contient des éléments non infiniment grands. En effet, $g = g^\uparrow$ ssi $d_g \succ\!\!\succ 1$ et $\ln s_g \prec\!\!\prec \ln d_g$. Nous concluons par le lemme 2.

Généralisons maintenant ces algorithmes au cas des fonctions exp-logs générales. Nous allons faire ceci par récurrence : nous supposons que l'on a établi les algorithmes pour des fonctions exp-logs dans $\mathbb{R}_p[[[x]]]$ et nous allons montrer comment les fabriquer pour des fonctions dans $\mathbb{R}_{p+1}[[[x]]]$. La base de la récurrence vient d'être fournie.

Les constructions de B_f sont pareilles que dans la section précédente, pour les cas qui y sont traités. Reste à considérer le cas où $f = \exp(g)$. Nous pouvons décomposer $g = g^\uparrow + g^c + g^\downarrow$. Or $g^\uparrow$ est un élément de l'échelle de $\mathbb{R}_p[[[x]]]$ et nous pouvons écrire $f = e^{g^c} \exp(g^\downarrow) e^{g^\uparrow}$. On a $g^\downarrow \prec\!\!\prec 1$ et $\exp(g^\downarrow)$ se développe de façon analogue à $1/(1+x)$ et $\ln(1+g)$. Nous remarquons que le poids d'un singleton du belordre B_f peut toujours s'écrire comme produit d'une fonctions de la forme $\ln_{\boldsymbol{n}} x$ et de l'exponentiel d'une fonction de $\mathbb{R}_{p-1}[[[x]]]$ (en simplifiant $e^f e^g = e^{f+g}$ et $1/e^f = e^{-f}$). Cette remarque est utilisée pour développer le logarithme d'un monôme.

De même, le calcul de d_f et s_f se déroule de la même façon que tout à l'heure. Dans le cas où $f = \exp(g)$, nous décomposons à nouveau $g = g^\uparrow + g^c + g^\downarrow$ et nous posons $d_f = e^{g^\uparrow}$ et $s_f = s_{g^\downarrow}$. Enfin, compte tenu de cette extention, l'algorithme du calcul d'une décomposition se transporte sans problème au cas présent. En résumé nous obtenons le

Théorème 3. *Nous pouvons expliciter un algorithme de comparaison de fonctions exp-logs, sous réserve de pouvoir comparer des constantes exp-logs.* ♡

Remarque. En fait, les raisonnements et algorithmes précédents nous donnent plus. Que $f^\uparrow$ est une véritable fonction exp-log, et non pas une transsérie arbitraire, n'est pas trivial a priori. De plus, nous pouvons calculer cette fonction et il en est de même pour la troncation du développement de f à n'importe quel ordre.

5.3 Algorithme sans calcul des décompositions

Notre algorithme pour développer f n'est pas optimal du point de vue de la complexité. En effet, le fait de développer par rapport à une échelle fixe de $\mathbb{R}[[[x]]]$ nécessite le calcul des décompositions $f = f^\uparrow + f^c + f^\downarrow$, ce qui peut faire exploser le calcul, comme dans l'exemple suivant :

$$\exp(x^{10^{100}} e^{1/x} + 1/x) - \exp(x^{10^{100}} e^{1/x}).$$

En contrepartie, nous obtenons une sorte de "développement sous forme normale" de f, pour lequel nous devons donc parfois payer un prix lourd en temps. Si on veut juste comparer deux fonctions exp-logs, nous allons montrer que l'on peut se passer de calculer systématiquement des décompositions de f. Ceci réduit de façon importante la complexité comme le montre l'exemple précédent. Néanmoins, il n'est pas clair que dans la pratique, il est quand même plus utile de calculer des développements sous forme normale. Leur avantage est que l'on peut facilement effectuer des opérations sur des développements, une fois qu'il sont établis (penser aux opérations élémentaires sauf l'exponentiation). Dans le cas de l'algorithme établi ci-dessous, ceci peut conduire à des calculs supplémentaires.

Supposons comme dans les sections précédentes, que l'on veut développer une *expression* exp-log f. Nous insistons sur le fait que l'on prend une expression f, car deux expressions repésentant la même fonction ne donnent pas nécessairement le même résultat. Comme dans les sections précédentes, nous allons associer un belordre pondéré algorithmique (B_f, μ_f) à f. Les seules différences avec la construction précédente est qu'à une expression de la forme e^f, où f est un infiniment grand (récurrence), nous associons le singleton $\{e^f\}$. Si nous appliquons brutalement l'algorithme de linéarisation à B_f (ceci veut dire que K sera le sous-ensemble de J d'indices k, avec $x_k \asymp x_j$), la seule situation dans laquelle l'algorithme pourrait éventuellement échouer serait si dans l'ensemble K, il y a deux indices k et k', avec $x_k \asymp x_{k'}$, mais $x_k \neq x_{k'}$. Pour éviter cette situation il y a deux approches différentes. Soit on réécrit f avant de commencer l'algorithme de sorte que l'algorithme de linéarisation ne peut pas rencontrer d'obstacle. Soit, on ne réécrit f qu'après avoir rencontré un obstacle et on recommence l'algorithme. Dans ce dernier cas, il est nécessaire de montrer qu'après un nombre fini de réécritures l'algorithme de linéarisation se déroule sans problèmes.

Suivons d'abord la première approche et considérons l'ensemble S de sous-expressions de f de e^g, avec $g \succcurlyeq 1$, réuni avec les $\ln_n x$, où n est inférieure à la profondeur de f (en tant qu'arbre). Par récurrence, on sait ordonner l'ensemble S pour la relation $\preccurlyeq\!\!\!\preccurlyeq$ définie par $h \preccurlyeq\!\!\!\preccurlyeq h' \Leftrightarrow \ln|h| \prec \ln|h'|$. Nous allons repeter le procédé suivant :

Si S contient deux éléments h et h' équivalents pour $\preccurlyeq\!\!\!\preccurlyeq$, tels que le rapport de leurs logarithmes n'est pas constant, alors on en prend deux de maximal pour $\preccurlyeq\!\!\!\preccurlyeq$, vérifiant cette propriété. Considérons le cas, où $h = e^g$ et $h' = e^{g'}$ - les autres cas se traitent de façon semblable. On peut calculer $c \neq 0$, tel que $g \sim cg'$. Ensuite, nous remplaçons systématiquement e^g par $e^{cg'}e^{g-cg'}$ dans f.

Montrons la terminaison de ce procédé. La profondeur de f (en tant qu'arbre) reste invariant lors de la réécriture et le nombre d'éléments de S ne peut donc que décroître. Puis, écrivons $S = \{g_1, \cdots, g_n\}$ et supposons que les g_i soient ordonnés pour $\preccurlyeq\!\!\!\preccurlyeq$. Soit k le plus petit entier tel que $g_k \preccurlyeq\!\!\!\preccurlyeq \cdots \preccurlyeq\!\!\!\preccurlyeq g_n$. Alors k ne peut que décroître. Si k reste constant, alors le nombre de g_i qui sont inférieurs à g_k croit pour chaque réécriture. Le nombre de réécritures est donc fini. Il est facile de vérifier qu'après ces réécritures, l'algorithme de linéarisation ne pourra plus rencontrer d'obstacles.

Passons maintenant à la deuxième approche et procédons d'abord à quelques définitions. A l'expression f nous allons récursivement associer un ensemble fini $G_f = \{e^{g_1}, \cdots, e^{g_p}\}$ de sous-expressions de f dit de *générateurs*, tel que les g_i soient des infiniment grands et tel que le poids de tout singleton h de B_f soit de la forme

$$h = c \ln_{\boldsymbol{n}} x (e^{g_1})^{k_1} \cdots (e^{g_p})^{k_p},$$

avec $c \in \mathbb{R}$ et les k_i dans $\mathbb{Z}$. Cette propriété se vérifiera aisément par récurrence. De même nous allons associer récursivement à toute sous-expression g de f sa *complexité* $\xi_g \in \mathbb{N}$. Nous associons également à f son *invariant* I_f, qui est l'élément de $\mathbb{N}^\circ$, obtenu en multipliant formellement les complexités des éléments de G_f.

Nous posons $G_{c \ln_{\boldsymbol{n}} x} = \emptyset$ et $\xi_{c \ln_{\boldsymbol{n}} x} = 0$. Ensuite $G_{-f} = G_f$ et $\xi_{-f} = \xi_f$. Puis $G_{f_1+f_2} = G_{f_1 f_2} = G_{f_1/f_2} = G_{f_1} \cup G_{f_2}$ et $\xi_{f_1+f_2} = \xi_{f_1 f_2} = \xi_{f_1/f_2} = \max(\xi_{f_1}, \xi_{f_2})$. En ce qui concerne $\ln f$, soit P le terme principal de f. Nous pouvons écrire (en utilisant la récurrence) $P = c \ln_{\boldsymbol{n}} x e^{k_1 g_1} \cdots e^{k_p g_p}$. Alors posons $G_{\ln f} = G_f \cup G_{g_1} \cup \cdots \cup G_{g_p}$ et $\xi_{\ln f} = \xi_f$. Enfin considérons le cas de e^f. Si f est infiniment grand, posons $G_{e^f} = \{e^f\}$ et $\xi_{e^f} = \xi_f + 1$. Sinon, posons $G_{e^f} = G_g$ et $\xi_{e^f} = \xi_f$.

Supposons qu'à un moment donné dans l'algorithme de linéarisation, on se trouve dans la situation, où $x_k \asymp x_{k'}$ et $x_k \neq x_{k'}$, pour certains indices k et k' dans K. Alors le rapport entre x_k et $x_{k'}$ s'écrit sous la forme

$$\frac{x_k}{x_{k'}} = \ln_{\boldsymbol{n}} x e^{k_1 g_1} \cdots e^{k_p g_p},$$

où les k_i sont entiers et où $G_f = \{e^{g_1}, \cdots, e^{g_{p'}}\}$. De plus nous pouvons supposer que $\xi_{g_1} \leq \cdots \leq \xi_{g_p}$. Soit f' l'expression obtenue en effectuant systématiquement les

remplacements suivants dans f :

$$e^{g_i} \mapsto (e^{g_i/k_p})^{k_p} = e^{g_i/k_p} \overset{k_p \text{ fois}}{\cdots} e^{g_i/k_p},$$

pour tout $1 \leq i \leq p-1$ et

$$e^{g_p} \mapsto e^h \ln_{\boldsymbol{n}} x (e^{g_1/k_p})^{k_1} \cdots (e^{g_{p-1}/k_p})^{k_{p-1}},$$

avec

$$h = k_p g_p - (\ln \ln_{\boldsymbol{n}} x + (k_1/k_p) g_1 + \cdots + (k_{p-1}/k_p) g_{p-1}).$$

Remarquons que f' désigne la même fonction exp-log que f. Nous reçommencons l'algorithme avec ce nouveau f'.

En appliquant le nouvel algorithme, nous obtenons donc une suite d'expressions $f, f', f'', \cdots$ représantant toutes la même fonction exp-log. Montrons que cette suite s'arrête. Nous munissons $\mathbb{N}^\diamond$ de l'ordre $\leq_{lex}$, tel que $\boldsymbol{n} <_{lex} \boldsymbol{n}'$ si $n_i < n'_i$, pour un certain i et $n_j = n'_j$, pour tous les $j > i$. Alors $\leq_{lex}$ prolonge l'ordre partiel canonique sur les multiensembles et est donc bien fondé. Montrons maintenant que $I_f <_{lex} I_{f'} <_{lex} \cdots$, d'où la justification de notre algorithme. Il suffit de le vérifier pour la première inégalité. En effet, il n'y a que le remplacement de e^{g_p} qui affecte l'invariant de f. Or ce remplacement a pour effet d'enlever le singleton $\{e^{g_p}\}$ à G_f et de rajouter G_{e^h}. Or la complexité de g_p est maximal parmi les g_i et $h \preceq 1$. Donc les éléments de $G_{e^h} = G_h$ ont tous une complexité moindre ou égale à ξ_{g_p}. Mais $\xi_{e^{g_p}} = \xi_{g_p} + 1$, donc $I_{e^h} <_{lex} I_{e^{g_p}}$ et on a bien $I_f <_{lex} I_{f'}$.

5.4 Variantes de l'algorithme

Il reste quelques remarques à faire à propos de l'algorithme de linéarisation dans le contexte des fonctions exp-logs.

Premièrement, on peut se poser la question si le fait de savoir qu'un certain terme ne sera jamais atteint lors de la linéarisation peut servir à optimiser l'algorithme (penser à $f(x) = e^{1/x} e^{e^{-x}}$). Ceci est effectivement faisable, car si $J = J_1 \amalg J_2$, dans l'algorithme de linéarisation et si on dispose du renseignement que les éléments de J_2 ne seront jamais atteints lors de la linéarisation, alors le développement de $\mu((J))$ sera le même que celui de $\mu((J) - (J_2))$.

Deuxièmement, on peut se demander si le calcul du poids d'un élément de $\mathcal{B}(I)$ pourrait être calculé en n'utilisant que des développements en série de Laurent. Ceci évite de faire des calculs a priori compliqués dans les belordres pondérés algorithmiques. Ceci est à nouveau possible, et nous laissons au lecteur le soin de le vérifier (on utilise la technique "par tranches" expliquée ci-dessous). En particulier, ceci accélère considérablement l'algorithme de calcul d'un développement sous forme normale, car cette optimisation est particulièrement utile pour le calcul des décompositions.

Enfin, il est souvent possible de décomposer un développement "par tranches". Par ceci, nous voulons dire que l'on peut écrire l'ensemble I sous-jacent au belordre pondéré algorithmique comme réunion disjointe $I = \amalg_{i \in I'} I_i$, avec les I_i dans $\mathcal{B}(I)$. De plus nous demandons que I' soit totalement ordonné par la relation $i < j \Leftrightarrow \mu(I_i) \prec\!\!\prec \mu(I_j)$ et que I' soit un belordre pondéré algorithmique tel que le poids de toute partie $X \in \mathcal{B}(I')$ soit égal à $\mu(\amalg_{i \in X} I_x)$. En gros ceci correspond à assimiler les $\mu(I_i)$ à des constantes. On peut décomposer la linéarisation du développement en calculant la linéarisation de chaque terme qui est fourni par la linéarisation par rapport à I'.

Cette situation arrive systématiquement dans le cas des fonctions exp-logs, grace à l'axiome de finitude AFG. En effet, tout développement asymptotique peut être interprété comme une série de Laurent formelle en plusieurs variables, muni d'un ordre total sur les monômes. De tels ordres sont toujours isomorphes à une somme lexicographique de sousgroupes additifs finiment engendrés de $\mathbb{R}$. Supposons qu'on ait réécrit f comme dans la sous-section précédente (par la première approche). Les éléments de S correspondent (à des "duplications" près) aux variables de la série de Laurent formelle. Maintenant, on peut développer f par rapport uniquement aux éléments maximaux pour $\prec\!\!\prec$ de S. Ceci revient à assimiler tout élément du corps engendré par les autres éléments de S à des constantes. Or un tel développement ne nécessite que des opérations dans $K[\{X^\alpha\}_{\alpha \in \mathbb{R}}]$. Ainsi, on obtient un algorithme élégant évitant la théorie de la section 2.

Néanmoins, il est clair que la décomposition d'un développement par tranches n'est qu'une reformulation du problème. On s'attend donc à ce que les deux algorithmes marche un peu près aussi vite. Gonnet et Gruntz ont implanté le dernier et obtenu de bonnes performances. Nous remarquons également que ce traitement plus intuitif n'est plus possible dans le cas, où l'axiome AFG n'est plus vérifié, comme dans les exemples mentionnés dans l'introduction. La technique utilisant les belordres présente l'avantage d'être parfaitement généralisable à ces cas et de plus, en tenant compte de l'implantation de l'algorithme de Gonnet et Gruntz, on peut s'attendre à de bonnes performances.

6 Références

[Bour 51] N. Bourbaki. *Eléments de mathématiques.* Hermann (2-ième éd. 1961), ch V : Fonctions d'une variable réelle.

[CavPr 78] B.F. Caviness, M.J. Prelle. A note on algebraic independance of logarithmic and exponential constants. *SIGSAM Bull. 12 (p 18-20).*

[DaGö 84] B.I. Dahn, P. Göring. Notes on exponential-logarithmic terms. *Fundamenta Math. 127 (1986) (p 45-50).*

[Ec 92] J. Ecalle. *Introduction aux fonctions analysables et preuve constructive de la conjecture de Dulac.* Hermann, collection : Actualités mathématiques.

[FlO 90] P. Flajolet, A.M. Odlyzko. Singularity analysis of genearting functions. *SIAM Journal on discrete mathematics 3(2) (p 216-240).*

[GoGr 92] G.H. Gonnet, D. Gruntz. Limit computation in computer algebra. *Report technique 187 du ETH, Zürich.*

[Har 10] G.H. Hardy. Orders of infinity. *Cambridge Tracts in Mathematics 12.*

[Har 11] G.H. Hardy. Properties of logarithmico-exponential functions. *Proceedings of the London mathematical society 10,2 (p 54-90).*

[Hig 52] G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc. 2 (p 326-336).*

[Krus 60] J.B. Kruskal. Well-quasi-ordering, the tree theorem and Vázsonyi's conjecture. *Trans. Amer. Math. Soc. 95 (p 210-225).*

[Lang 84] S. Lang. *Algebra.* Addison-Wesley, 2-ième éd. (p 390-403).

[P 1893] H. Poincaré. Les Méthodes nouvelles de la mécanique céleste. *Tome 2, ch VIII, no. 118 et 119.*

[Pou *] M. Pouzet. Survol du belordre. *En préparation.*

[Sh 89a] J. Shackell. A differential-equations approach to functional equivalence. *Proc ISSAC 89, Portland, Oregon, A.C.M., New York (p 7-10).*

[Sh 89b] J. Shackell. Zero-equivalence in function fields defined by algebraic differential equations. *Preprint.*

[Sh 90] J. Shackell. Growth estimates for exp-log functions. *Journal of symbolic computation 10 (p 611-632).*

[VdH *a] J. van der Hoeven. Le belordre comme outil en asymptotique. *En préparation.*

[VdH *b] J. van der Hoeven. Sur l'algèbre asymptotique. *En préparation.*

[VdH *c] J. van der Hoeven. Sur un algorithme de comparaison de constantes exp-logs modulo la conjecture de Schanuel. *En préparation.*