General algorithms in asymptotics I
Gonnet and Gruntz' algorithm

by Joris van der Hoeven
LIX
École Polytechnique
91128, Palaiseau
France
Email: vdhoeven@lix.polytechnique.fr

July 1994

Abstract

General algorithms in asymptotics are used to obtain automatically asymptotic information about explicit functions, or solutions to certain types of equations. The simplest non trivial problem is the automatic expansion of exp-log functions. Shackell was the first to give an algorithm and Gonnet and Gruntz were the first to implement one. We will clarify the latter one and put it into a theoretical context. Our study serves as a base for generalisations to more general classes of functions.

Key words: Asymptotic expansion, exp-log function, transseries, algorithm.

1 Introduction

The problem of expanding exp-log functions has been considered by several authors and goes back to Hardy's work on L-functions [Har 11]. However, the computational version of the problem has been settled only recently. Shackell [Sh 90] was the first to give an explicit algorithm and Gonnet and Gruntz [GoGr 92] were the first to give one, which has been implemented in practice. A variant of this latter algorithm was rediscovered independently by the author [VdH 94a].

We remark that all algorithms make the implicit hypothesis that one can decide whether an exp-log constant is zero (see [Sh 90] for a discussion). This problem has been settled by Richardson [Rich 92] modulo Schanuel's conjecture. We also remark that if we can test whether an exp-log constant is zero, we can also determine its sign. Indeed, theoretically it suffices to do floating point arithmetic at a sufficient precision. In [VdH *b] we will give a more efficient algorithm. As counterexamples to Schanuel's conjecture are probably quite rare, this algorithm should be completely satisfactory in practice.

In this paper, we show how to exploit from a computational point of view the theory of transseries (verifying the AFG axiom), invented by Ecalle [Ec 92]. In fact, the germs at infinity of most of the usual regular functions are transseries. In particular this is the case for algebraic exp-log functions. In section 3 we start by introducing multiseries, which are series with finitely generated support in $\mathbb{R}$. We show that the usual operations on powerseries can be generalized to multiseries. Our point with transseries is that they can be interpreted as lexicographical multiseries in several variables. Therefore, it suffices to automate this analogy, which will be done in section 5. We remark that Shackell as well as Gonnet and Gruntz implicitely use this theoretical fact, although they do not fully exploit it.

In fact, the algorithm we obtain by using these techniques is not really new; it is a mere reformulation of the algorithm of Gonnet and Gruntz. However, we hope that our approach will lead to more transparency in the proofs and a better theoretical insight. Indeed, as soon as the reader has familiarized himself with the basic theory of multiseries and transseries, the algorithm, and its correctness proof, are short and easy to understand.

When we compare our approach to Shackell's one, we observe that the theory of transseries replaces more or less the theory of Hardy fields. However, the theory of transseries has the advantage of being purely algebraic, and we will show in [VdH *c] that it can be generalized to include transseries in several variables with oscillating coefficients. Moreover, when dealing with transseries, one does not need to prove any analytic results; for example we do not have to care about the presence or absense of zeros. In fact, the very powerfull summability theory of Ecalle can handle all analytic aspects of transseries. In particular, it is even possible to obtain the original function from its expansion. Therefore, as we are concerned with algorithms, we prefer to bypass all analytic aspects of the problem and to concentrate on the computational relevant, thus algebraic ones.

Furthermore, the obtained algorithm is different from Shackell's algorithm. In fact, Gonnet and Gruntz' approach seems to be slightly more practical. First, because it is the first algorithm that has been implemented. Secondly, because the method used favourizes quickly performing field operations on automatic transseries, while Shackell's method is more oriented towards quick functional composition and inversion. It seems natural to assume that field operations occur more frequently in practice.

Finally, two slight differences between our approach and Gonnet and Gruntz's one should be noticed. First, our way of constructing transseries implies that we can avoid using "upward movements" in the case of exp-log functions. This should slightly accelerate computations in the absence of exponentials. However, the benifit is small, because upward movements can be implemented efficiently and will be needed in [VdH 94b] and [VdH *a] anyhow. Secondly, we don't calculate the "most rapidly varying subexpressions" in order to expand an expression. Instead, we gradually compute a "normal base" (see section 5). Computationally, this avoids performing unnecessary order comparisons. Moreover, this approach simplifies a lot the expansion of more general types of functions, as will be shown again in [VdH

94b] and [VdH *a]. We remark that similar ideas were used by Salvy in [Sal 91].

I would like to thank J.M. Steyaert for many suggestions and the detection of a certain number of errors in previous versions of this article.

2 Conventions

It will be convenient to fix a certain number of conventions for this and subsequent papers.

We first have the problem of naming different types of series: powerseries, Laurent series, Puiseux series, etc. It is natural only to talk about *series*, if their type can clearly be deduced from the context. A *formal series* will always be a series in its most general meaning and is therefore merely a name for the notation $\sum_{i \in I} c_i f_i$. A *convergent series* will be some kind of series which is convergent for some notion of convergence.

Next, a lot of different types of series with algebraic closure properties have to be distinguished. The most general type is probably obtained by taking a partially ordered semigroup M and considering series over some semiring A of the type $f = \sum_{x \in M} a_x x$, where f has well-quasi-ordered support (see [VdH 94a]). The set of these series can be given the usual ring operations. We didn't find a completely convenient name for this kind of series; one could call them *generalized power series*, or simply *generalized series*, or *algebraic series*, which is perhaps more suggestive.

All other types of series with algebraic closure properties, known by the author, can be obtained by taking special M's or by restrictions on the support. The ring of usual *power series* is obtained by taking $M = \mathbb{N}$ and its quotient field by taking $M = \mathbb{Z}$. *Powerseries in several variables* are obtained by taking $M = \mathbb{N}^n$. In section 3 we introduce the so called *multiseries* which are obtained by restricting the support to be finitely generated. The name is justified by the fact that we can consider elements of $K \llbracket x \rrbracket$ as elements of $f \in K((x_1, \cdots, x_n))$. The only difference is that the x_i are comparable from an asymptotic point of view. As an application, the algebraic closure of $K[[x]]$ is given by $K \llbracket x^{\mathbb{Q}} \rrbracket$. We also have *lexicographical multiseries in several variables*, which are elements of $K \llbracket x^{A^n_{lex}} \rrbracket$, where A is a totally ordered abelian group and $A^n_{lex} = A^n$ is given the lexicographical ordering.

Finally we have series which don't have interesting algebraic closure properties. For example we mention the *Laurent series* (which are formal sums $\sum_{n \in \mathbb{Z}} a_i x^n$) and the *Puiseux series* (which are indexed over $\mathbb{Q}$). These series are only interesting, when convergent in some way.

A similar discussion is possible concerning *transseries*, defined by Ecalle [Ec 92]. Here, different fields of transseries can be constructed by considering different types of finiteness axioms. In this paper we assume that the AFG axiom is verified and we will denote the field of transseries obtained by $\mathbb{R} \llbracket\!\llbracket x \rrbracket\!\rrbracket$, preserving the analogy with multiseries. This field is stable under most of the usual operations including differentiation, composition, inversion, etc. The techniques presented in this paper can be used in this context. More general types of transseries can be handled with similar techniques as explained in [VdH 94a].

A second source of possible confusion comes from the concept of effectiveness. In the field of computer algebra, we encounter a lot of algebraic structures in which a certain number of computations can be performed by algorithm. It is convenient to think about these algorithms as part of the structure, which leads to the concept of an *algorithmic algebraic structure*. However this concept has to be handled with care, because one has to specify very precisely which things can be computed as a function of which other things and which notion of computability has to be taken.

We start by remarking that the exact meaning of computable can usually easily be made concrete, for example in the language of Turing machines. However, in this paper we need a more general notion of computability, when we solve the expansion problem for exp-log functions modulo the constants problem. Indeed, often we can compute x under the hypothesis that we can compute y. This notion of *relative computability* is usually formalized by using oracles.

Next, we remark that several variants of algorithmic algebraic structures arise in computer algebra. For example, sometimes we can compute normal forms of elements of the structure and sometimes we can't. However, it is usually sufficient that we can represent the elements of our structure in at least one way and that all relevant computations are compatible with this representation. From now on we will use the prefix *algorithmic*, if this is the case.

In a logical structure, the relevant operations are precisely determined by its constant, relation and function symbols. For example, in an algorithmic ring, the addition, substraction and multiplication can be performed by algorithm. Moreover, we can effectively represent 0 and 1. Hence, $\mathbb{Z}$ is an algorithmic ring. More generally, if the algebraic structure satisfies some axioms, these axioms should be verified in some computational way. For example, in an algorithmic algebraically closed field $\mathfrak{K}$, we should be able to compute the solution set of each polynomial equation with coefficients in $\mathfrak{K}$.

Adopting this spirit of thinking, we get theorems like "if $\mathfrak{K}$ is an algorithmic field, then we can compute its algebraic closure, which is an algorithmic algebraically closed field". Note the precise statement of this theorem: we not only announce that the algebraic closure is algorithmic, but we can compute it, when we are given $\mathfrak{K}$. We also note that instead of giving an algorithm and prove its correctness, we often proceed by announcing a theorem and giving the algorithm. This doesn't prevent us from giving names to algorithms and comparing different algorithms; it encourages thinking about computer algebra in terms of algorithmic algebraical structures.

Some special attention has to be paid to series expansions. It is natural to define a (power-, multi-, trans-)series to be *automatic* if all its terms can be computed in increasing order. However, this definition implies that the set of automatic series is not stable under the usual algebraic operations. For example, how do we know that $f - f = 0$, if f is some automatic series? The trick is to fix an algorithmic set $\mathfrak{A}$ of series, which is algorithmically stable under certain operations. A series f in $\mathfrak{A}$ will now be said to be automatic, if f can be expanded by some algorithm and if all data defining f, such as its coefficients and its support, belong to some algorithmic algebraic structure. We then consider the subset $\mathfrak{A}_a$ of computable automatic series $\mathfrak{A}$, which is often algorithmically stable under the wanted operations. Again, by "computable" we mean that given an element $f \in \mathfrak{A}_a$ we do not only know that f is automatic, but we can compute an algorithm to expand f. We refer to the sections 3 and 4 for more details.

In the rest of this and subsequent papers, we will denote algorithmic algebraic structures by capital fraktur letters $\mathfrak{A}, \mathfrak{B}, \cdots$. However, automatic series and transseries we be denoted in the usual mathematical font. Finally, we remark that we will sometimes replace words like "computable" by synonyms like "effective". Sometimes, we also use words like "compute" in a non algorithmic context. In that case we talk about *abstract algorithms*. Whenever confusion might arise, we insist on the concrete aspect of the computation by saying that it can be done *algorithmically*.

3 Formal and automatic multiseries

Let A be some abelian totally ordered group. We will say that a subset $S \subseteq A$ is *finitely generated*, if there exist $x_1, \cdots, x_n > 0$ in A and $a_1, \cdots, a_n$ in $\mathbb{Z}$, such that each $x \in S$ can be written as $x = b_1 x_1 + \cdots + b_n x_n$, with $a_i \leq b_i \in \mathbb{Z}$, for each $1 \leq i \leq n$. If all elements of S are (strictly) positive (resp. negative), we will say that S is *(strictly) positive (resp. negative)*.

Proposition 1. *Each finite subset of A is finitely generated. If S and S' are finitely generated subsets of A, then so are $S \cup S'$, $S + S'$. Moreover, if S is positive, then $S^\diamond \stackrel{def}{=} \{x_1 + \cdots + x_n | x_1, \cdots, x_n \in S\}$. is finitely generated.*

Proof. It is clear that if S and S' are finitely generated subsets of G, then so are their union and their sum. Suppose now that S is positive, that $x_1, \cdots, x_n$ are generators of S and let $a_1, \cdots, a_n \in \mathbb{Z}$ be such as above. Then the set $\{b_1 x_1 + \cdots + b_n x_n | \forall i \quad \min(1, a_i) \leq b_i \leq 1\}$ is finite, say equal to $\{y_1, \cdots, y_m\}$. It is easy to see that each element of S is a sum of y_i's. But then each element of $S^\diamond$ is a sum of y_i's and hence $S^\diamond$ is finitely generated. ♡

Suppose now that $\mathfrak{A}$ is an algorithmic abelian totally ordered Archimedian group (see section 2 for the definition). A subset S is said to be *automatic*, if it is finitely generated and all its elements can be computed in increasing order by algorithm.

Proposition 2. *Every finite subset of $\mathfrak{A}$ is automatic. If S and S' are automatic sets, then so are $S\cup S'$, $S+S'$ and $S^\diamond$. Finally, if S is automatic and $S'\subseteq S$, such that we can test by algorithm whether $x\in S'$ and $S'\cap]x,\infty[=\phi$, for each $x\in\mathfrak{A}$, then S' is automatic.*

Proof. Of course, every finite subset of $\mathfrak{A}$ is automatic. Suppose that S and S' are automatic sets, listed by $S=\{p_1,p_2,\cdots\}$ and $S'=\{q_1,q_2,\cdots\}$. Then we can list $S\cup S'=\{\min\{p_1,q_1\},\min(\{p_1,p_2,q_1,q_2\}\backslash\{\min\{p_1,q_1\}\}),\cdots\}$ and $S+S'=\{p_1+q_1,\min(\{p_1+q_1,p_1+q_2,p_2+q_1\}\backslash\{p_1+q_1\}),\cdots\}$. So $S\cup S'$ and $S+S'$ are automatic. Similarly, if S has strictly positive support, then we can list $S^\diamond=\{0,p_1,\min(\{p_1,2p_1,p_2\}\backslash\{p_1\}),\min(\{p_1,2p_1,3p_1,p_2,p_1+p_2,p_3\}\backslash\{p_1\}\backslash\{\min\{2p_1,p_2\}\}),\cdots\}$. Hence $S^\diamond$ and $(S\cup\{0\})^\diamond=S^\diamond$ are automatic. Suppose finally that S and S' are such as in the last assertion. If we list $S=\{p_1,p_2,\cdots\}$, then we can effectively extract all elements which belong to S'. We avoid possible infinite loops, by testing whether $S'\cap]p_i,\infty[$ is empty for each i. ♡

Let K be a totally ordered field and A a totally ordered abelian group, then we denote by $K[\![x^A]\!]$ the set of formal series of finitely generated support included in A. Thus, an element $f\in K[\![x^A]\!]$ is a formal sum $f=\sum_{\alpha\in A}f_\alpha x^\alpha$, such that $\mathbf{supp}\,f$ is finitely generated. We say that f is a *multiseries* in x with support included in A. In the case when $A=\mathbb{R}$, we will denote $K[\![x^{\mathbb{R}}]\!]$ by $K[\![x]\!]$.

If f is a multiseries, we define its *dominant exponent* μ_f to be the smallest element in its support, with the convention $\mu_0=\infty$. Furthermore, f_{μ_f} is said to be the *dominant coefficient* of f and $f_{\mu_f}x^{\mu_f}$ its *dominant term*. An *infinitesimal multiseries* is a multiseries f with $\mu_f>0$ (hence f has strictly positive support). The K-algebra of such series is denoted by $K[\![G]\!]^\downarrow$. Similarly $K[\![G]\!]^\uparrow$ is the set of multiseries with strictly negative support. We have $K[\![G]\!]=K[\![G]\!]^\uparrow\bigoplus K\bigoplus K[\![G]\!]^\downarrow$,

Let us introduce some asymptotic relations on $K[\![x^A]\!]$. The equivalence relation $\sim$ is defined by $f\sim g\Leftrightarrow\mu_f=\mu_g\;\wedge\;f_{\mu_f}=g_{\mu_g}$. We define a relation $\prec\!\!\!\prec$ on $K[\![x^A]\!]$ by $f\prec\!\!\!\prec g\Leftrightarrow\mu_f>\mu_g$, which is called an *asymptotic ordering* We also define $\asymp$ by $f\asymp g\Leftrightarrow\mu_f=\mu_g$. Finally, we define two relations $\prec\!\!\!\prec\!\!\!\prec$ and $\asymp\!\!\!\asymp$, used to compare asymptotic scales by $f\prec\!\!\!\prec\!\!\!\prec g\Leftrightarrow\forall N\in\mathbb{N}^*\;\;N|\mu_f|<|\mu_g|$ and $f\asymp\!\!\!\asymp g\Leftrightarrow\exists N,M\in\mathbb{N}^*\;\;N|\mu_g|\leq|\mu_f|\;\wedge\;M|\mu_f|\leq|\mu_g|$.

If $f,g\in K[\![G]\!]$, we define $-f=\sum_{\alpha\in\mathbf{supp}\,f}-f_\alpha x^\alpha$, $f+g=\sum_{\alpha\in\mathbf{supp}\,f\cup\mathbf{supp}\,g}(f_\alpha+g_\alpha)x^\alpha$ and $fg=\sum_{\alpha\in\mathbf{supp}\,f+\mathbf{supp}\,g}(\sum_{\beta+\gamma=\alpha}f_\beta g_\gamma)x^\alpha$. It is easy to see that the sum $\sum_{\beta+\gamma=\alpha}f_\beta g_\gamma$ is finite, for each α. Therefore, $-f,f+g$ and fg are well defined and belong to $K[\![x^A]\!]$. Similarly, let $h\in K[[x]]$ be a usual power series and suppose that f is infinitesimal. Then we define

$$h\circ f=\sum_{\alpha\in(\mathbf{supp}\,f)^\diamond}\left(\sum_{n\in\mathbb{N}}\sum_{\beta_1+\cdots+\beta_n=\alpha}h_nf_{\beta_1}\cdots f_{\beta_n}\right)x^\alpha.$$

Again it is easy to verify that the coefficient of each x^α is a finite sum and thus well defined. Therefore, $h \circ f \in K[\![x^A]\!]$. In particular, if we take $h = -x + x^2 + \cdots$, we can define $1/(1+f)$. More generally, if $f \neq 0$ and if μ_f is the smallest element in $\mathbf{supp}\, f$, then we define $1/f$ by $1/f = (x^{-\mu}/f_{\mu_f})(1/(x^{-\mu} f/f_{\mu_f}))$. We finally define a total ordering on $K[\![x^A]\!]$ by taking the positive power series to be those of which the dominant coefficient is positive.

Proposition 3. *$K[\![x^A]\!]$ is a totally ordered field for the above laws.*

Proof. Straightforward verification. ♡

There are different ways to define multiseries in several variables. The simplest one is by considering $K[\![x_1^A]\!] \cdots [\![x_n^A]\!]$. In that case, the support of $f \in K[\![x_1^A]\!] \cdots [\![x_n^A]\!]$ is finitely generated, but there are no constraints on $\cup_{\alpha \in A} \mathbf{supp}\, f_\alpha$. In particular the definition is not symmetric in the x_i's. We define *lexicographical multiseries in several variables* to be elements of $K[\![x_1^A; \cdots; x_n^A]\!] \stackrel{\text{def}}{=} K[\![x^{A^n_{lex}}]\!]$, where we give $A^n_{lex} = A^n$ the lexicographical ordering. By proposition 2, this is a field and we observe that it is canonically included in $K[\![x_1^A]\!] \cdots [\![x_n^A]\!]$. The inclusion is strict, if $n > 1$. We remark that the definition is asymmetric in the x_i's.

Finally we have the set of *multiseries in several variables* $K[\![x_1^A, \cdots, x_n^A]\!] \stackrel{\text{def}}{=} K[\![x^{A^n}]\!]$, where A^n has the usual product *partial* ordering (it is easy to generalize the definitions to this case). This set forms a ring for the usual operations, but not a field ($x_1 + x_2$ is not invertible) and we won't use it very much for the moment. We remark that $K[\![x_1^A, \cdots, x_n^A]\!]$ is canonically included in $K[\![x_1^A; \cdots; x_n^A]\!]$ and the inclusion is strict if $n > 1$. However, by changing variables $\overline{x}_1 = x_1, \cdots, \overline{x}_n = x_n \cdots x_1^{\nu_{n,1}}$, each element $f \in K[\![x_1^A; \cdots; x_n^A]\!]$ can be seen as an element of $K[\![\overline{x}_1^A, \cdots, \overline{x}_n^A]\!]$.

It will be convenient to fix some notation for the extraction of coefficients. If f is in $K[\![x_1^A]\!] \cdots [\![x_n^A]\!]$, we denote by $[x_n^{\alpha_n}]f = f_{\alpha_n}$ the coefficient in x_n, which is a multiseries in $x_1, \cdots, x_{n-1}$. Inductively, we denote $[x_i^{\alpha_i} \cdots x_n^{\alpha_n}]f = f_{\alpha_n, \cdots, \alpha_i} = (f_{\alpha_n, \cdots, \alpha_{i+1}})_{\alpha_i}$. If f is in $K[\![x_1^A, \cdots, x_n^A]\!]$ one can define $[x_{i_1}^{\alpha_1} \cdots x_{i_k}^{\alpha_k}]f$ for any $1 \leq i_1 < \cdots < i_k \leq n$, by permuting variables.

Let $\mathfrak{C}$ be an algorithmic subfield of $\mathbb{R}$ and K some field containg $\mathbb{R}$. Let $\mathfrak{M}$ be an algorithmic subfield of $K[\![x_1; \cdots; x_n]\!]$. We will suppose that $\mathfrak{C} \subseteq K \cap \mathfrak{M}$ and that for each $\alpha \in \mathfrak{C}$ and $1 \leq i \leq n$ we can compute $x_i^\alpha \in \mathfrak{M}$ by algorithm. We will say that $f \in \mathfrak{M}$ is an *automatic multiseries* (w.r.t. $\mathfrak{M}$ and $\mathfrak{C}$), if its support is automatic, and if for each $\alpha \in \mathbf{supp}\, f$, we can compute $f_\alpha \in \mathfrak{M}$ by algorithm. Moreover, if $n > 1$, we demand all coefficients f_α to be automatic as elements of the algorithmic field $\mathfrak{M} \cap K[\![x_1; \cdots; x_{n-1}]\!]$. We denote by $\mathfrak{M}_a$ the set of computable automatic multiseries f in $\mathfrak{M}$.

Lemma 1. *$\mathfrak{M}_a$ is a field. Furthermore, if $f \in K[[x]]$, $g \in \mathfrak{M}_a^{\downarrow}$, and if $f : \mathfrak{M}^{\downarrow} \to \mathfrak{M}$ is algorithmic, then $f \circ g$ is in $\mathfrak{M}_a$.*

Proof. The proof proceeds by induction over n. The case $n=0$ is trivial. Suppose $n>0$. If $f,g \in \mathfrak{M}_a$, then the coefficients of $-f$, $f+g$, fg and f/g (when defined) can be computed by the formulas which define them, because of the induction hypothesis. As their differences with any finite $(K[\![x_1;\cdots;x_{n-1}]\!]\cap\mathfrak{M})$-linear combination of the x_n^α's are in $\mathfrak{M}$, we can test by algorithm whether their supports are finite or not. Therefore their supports are automatic by proposition 2. The same argument applies to $f\circ g$, when $f \in K[[f]]$, $g \in \mathfrak{M}_a^\downarrow$ and $f \circ g \in \mathfrak{M}$. ♡

4 Formal and automatic transseries

We will give an alternative definition of the field of transseries defined by Ecalle in [Ec 92]. Our definition will be purely algebraic and we proceed by successive closures. Contrary to Ecalle, we will start by closure w.r.t. logarithm and then perform closure w.r.t. exponentiation. This approach has the computational advantage that we can often avoid the use of "upward movements" (see [GoGr 92]). We remark that the construction can easily be extended to the case of transseries over an arbitrary totally ordered field, stable by exponentiation.

The different types of closure we use presuppose that we dispose of a field $K=\mathbb{R}[\![X]\!]$, where X is a totally ordered commutative multiplicative group. This notation is compatible with the former one, if we suppose that X is the group of l-th powers of some imaginary variable, where l runs over some abelian group isomorphic to X. However, contrary to before, we will interpret elements of $\mathbb{R}[\![X]\!]$ as formal sums $f=\sum_{\varphi\in X} f_\varphi \varphi$, so that the support of f is a subset of X. We will also suppose that we partially defined the logarithm mapping. In particular, this induces a partial exponential mapping, by the formula $\exp\ln x = x$. We will gradually enlarge K and define the logarithm for more and more values. We consider the following 4 types of closure:

Type I. (elementary closure): We suppose that $\ln(xy)=\ln x+\ln y$, for all $x,y\in X$ in the domain of ln. Next, each $a\in K$ can be written as $a=(c+\varepsilon)x$, with $c\in\mathbb{R}$, infinitesimal ε and $x \in X$. We suppose that if $\ln a$ is defined, then so are $\ln c$, $\ln x$ and $\ln(1+\varepsilon/c)$ by

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

where l is a usual power series with coefficients in $\mathbb{R}$, given by $l(x)=x-x^2/2+\cdots$. Under these hypothesis, the closure of K is precisely the field $K'=K$ in which $\ln a$ is defined by the preceding formula for each $a=(c+\varepsilon)x$, such that $c>0$ and such that $\ln x$ is defined (with infinitesimal ε, $c\in\mathbb{R}$ and $x\in X$). We remark that the hypotheses we made in the beginning are verified again in K'.

Type II. (closure by logarithm): Suppose that we can freely decompose the group X as a product $X = \{\cdots, 1/x, 1, x, \cdots\}X'$, with $x >_X 1$ et $x < y$, for all strictly positif $y \in X'$. Moreover, suppose that X' is precisely the subset of elements of X for which ln is defined. Let us add freely an element l to X. Thus, we consider $X'' = \{\cdots, 1/l, 1, l, \cdots\}X$. We naturally define an ordering on X'' by demanding that l^n, with $n \geq 1$ is smaller than each element of X, strictly superior to 1. Then K is naturally contained in the field $K' = \mathbb{R}[\![X'']\!]$ and we can extend the logarithm mapping by $\ln(x^n y) \stackrel{\text{def}}{=} nl + \ln y$, for each $x^n y \in X$ and $y \in X'$. We remark that the field K' verifies the initial hypothesis, which allowed to perform closure by logarithm.

Type III. (closure by exponentiation): We suppose that ln is defined for all positive elements in K and that $\ln x \in K^{\uparrow}$, for $x \in X$. Consider the totally ordered additive group $X' = K^{\uparrow}$. We embed $K = \mathbb{R}[\![X]\!]$ in $K' = \mathbb{R}[\![X']\!]$ by mapping $x \in X$ to $\varphi(x) = \ln x$ in X' and extending by linearity. More generally, we can define $\ln x = \varphi(x)$, for each $x \in X'$. We remark that the elementary closure of K' (when defined) again satisfies the initial hypothesis in order to perform closure by exponentiation.

Type IV. (closure by inductive limit): Consider a sequence $K_0 = \mathbb{R}[\![X_0]\!], K_1 = \mathbb{R}[\![X_1]\!], \cdots$ of fields each naturally contained in the next one; this means that X_{n+1} extends X_n and that ln on K_{n+1} extends its definition on K_n. Then $K = C[\![X]\!]$, with $X = X_0 \cup X_1 \cup \cdots$ naturally contains all the K_n's.

We remark that the closures of type II, III and IV preserve the elementary closure conditions. Moreover, if we take $X_0 \approx \mathbb{R}$ as our initial field, and if ln is defined on $\mathbb{R}_+^*$ in the usual way, then the conditions are satisfied.

Let us proceed with our construction. By applying repeatedly closure by logarithm to $K_0 = \mathbb{R}[\![X_0]\!]$, we construct a sequence $K_0 = \mathbb{R}[\![X_0]\!] \subset K_1 = \mathbb{R}[\![X_1]\!] \subset \cdots$ of fields which we subsequently close by inductive limit and elementary closure. Let $\mathbb{R}_0[\![\![x]\!]\!] = \mathbb{R}[\![X]\!]$ be the so obtained field. Then X is in fact the multiplicative group of elements of the form

$$\begin{aligned} &\ln_{\boldsymbol{p}} x = \ln_{0^{p_0}\cdots k^{p_k}} = x^{p_0}\ln^{p_1} x \cdots \ln_k^{p_k} x, \text{ with} \\ &\ln_k x = \ln \overset{k \text{ times}}{\cdots} \ln x. \end{aligned}$$

where $\boldsymbol{p}$ belongs to $\mathbb{N}^{(\mathbb{R})}$, considered as a multiplicative group. Moreover, ln is defined for all strictly positive elements and maps X into $\mathbb{R}[\![\![x]\!]\!]^{\uparrow}$.

Next, we define by induction $\mathbb{R}_{n+1}[\![\![x]\!]\!]$ to be the field obtained by closing $\mathbb{R}_n[\![\![x]\!]\!]$ by exponentiation and then by elementary closure. We denote by $\mathbb{R}[\![\![x]\!]\!]$ the elementary closure of the inductive limit of the sequence $\mathbb{R}_0[\![\![x]\!]\!], \mathbb{R}_1[\![\![x]\!]\!], \cdots$. We call it the field of *transseries with finitely generated support* or just the field of *transseries*, when no confusion may occur.

We will now draw a parallel between transseries and lexicographical multiseries in several variables. Let $g_1 \prec\!\!\!\prec\!\!\!\prec \cdots \prec\!\!\!\prec\!\!\!\prec g_n$ be infinitely big transseries ($g_i \succ\!\!\!\succ 1$). We remark that $f \prec\!\!\!\prec\!\!\!\prec g \Leftrightarrow \ln|f| \prec\!\!\!\prec \ln|g|$ (and similarly $f \asymp\!\!\!\asymp g \Leftrightarrow \ln|f| \asymp \ln|g|$), for f and g in $\mathbb{R}[\![\![x]\!]\!]$. Let $f \in \mathbb{R}[\![g_1^{-1}; \cdots; g_n^{-1}]\!]$ be a multiseries, where we consider the g_i^{-1}'s as formal variables. We claim that f induces a transseries, which will also be denoted by f. We can write $g_i^{-1} = (c_i + \varepsilon_i)\varphi_i$, for each i, where φ_i is the dominant monomial of g_i^{-1} and ε_i its dominant coefficient. Then we formally write

$$f = \sum_{\alpha_1, \cdots, \alpha_n} f_{\alpha_n, \cdots, \alpha_1} \prod_{i=1}^{n} (c_i + \varepsilon_i)^{\alpha_i} \varphi_i^{\alpha_i}.$$

Now $(c_i + \varepsilon_i)^{\alpha_i}$ can be further expanded, for each i:

$$(c_i + \varepsilon_i)^{\alpha_i} = \sum_{k \in \mathbb{N}} \sum_{\psi_1, \cdots, \psi_k \in \mathbf{supp}\, \varepsilon_i} \binom{c_i}{c_i - k} c_i^{\alpha_i - k} \varepsilon_{i,\psi_1} \cdots \varepsilon_{i,\psi_k} \psi_1 \cdots \psi_k.$$

Putting everything together, proposition 1 implies that the support of f, redefined by this formula, is finitely generated and it is easy to verify that its coefficients are all given by finite sums. Therefore, f is a transseries.

Inversely, we would like to show that each transseries can be interpreted as a multiseries. In fact, we have a stronger result, which will also allow us to interpret operations on transseries in terms of operations on multiseries. We only state the structure theorem, in order to give some theoretical insight. A more general theorem will be proved in [VdH *a].

Theorem 1. **(Structure theorem)** *Let $f_1, \cdots, f_k$ be transseries with finitely generated supports. Then there exist infinitely big $g_1 \prec\!\!\!\prec\!\!\!\prec \cdots \prec\!\!\!\prec\!\!\!\prec g_n$, such that $f_i \in \mathbb{R}[\![g_1^{-1}; \cdots; g_n^{-1}]\!]$, for every $1 \leq i \leq k$.* ♡

Let $\mathfrak{C}$ now be an algorithmic subfield of $\mathbb{R}$, algorithmically stable by exponentiation and logarithm (of positive elements, of course). Let $\mathfrak{T}$ be an algorithmic subfield of $\mathbb{R}[\![\![x]\!]\!]$. We suppose that $\mathfrak{C} \subseteq \mathfrak{T}$ and that $\ln_n^{\alpha} x \in \mathfrak{T}$ can be computed by algorithm, for each $n \in \mathbb{N}$ and $\alpha \in \mathfrak{C}$. By induction over $r \in \mathbb{N}$, we define a transseries f to be automatic of order r (w.r.t. $\mathfrak{T}$ and $\mathfrak{C}$), if the following conditions are satisfied:

- **AT1.** There exist $g_1 \prec\!\!\!\prec\!\!\!\prec \cdots \prec\!\!\!\prec\!\!\!\prec g_n$ in $\mathfrak{T}$, such that $f \in \mathfrak{C}[\![g_1^{-1}; \cdots; g_n^{-1}]\!]$,
- **AT2.** For each $1 \leq i \leq n$, we have either $g_i = \ln_{k_i} x$, for some computable k_i, or $g_i = e^{h_i}$, for some computable automatic transseries $h_i \succ\!\!\!\succ 1$ of order $r-1$,
- **AT3.** f is automatic as a multiseries in $g_1^{-1}, \cdots, g_n^{-1}$ and all its coefficients of the form $[g_n^{\alpha_n} \cdots g_i^{\alpha_i}]f$ are in $\mathfrak{T}$.

We allow n to be 0 in the definition and $1 \leq i \leq n+1$ in **AT3**. In particular, $f \in \mathfrak{T}$. We remark that in **AT2** and **AT3** we obviously assume the transseries and multiseries to be automatic w.r.t. $\mathfrak{T}$ and $\mathfrak{C}$, resp. $\mathfrak{T} \cap \mathfrak{C}[[g_1^{-1}; \cdots; g_n^{-1}]]$ and $\mathfrak{C}$. We say that f is automatic, if f is automatic of some order and denote by $\mathfrak{T}_a$ the set of computable automatic transseries.

In order to prove that a transseries is automatic, one has to build a structural tree, whose nodes are labeled by automatic transseries, such that the children of a node labeled by f are labeled by the k_i's and the h_i's from condition **AT2**. Such a tree is called an *automatic expansion* of f. Given an automatic expansion expansion of f, we can reconstruct r and $g_1, \cdots, g_n$, such that the above conditions are satisfied. We say that the expansion is of order r and with respect to $g_1, \cdots, g_n$. If $\mathfrak{T} = \mathfrak{T}_a$, then we say that $\mathfrak{T}$ is an *automatic expansion field*.

We remark that the order of f only has a meaning with respect to some fixed expansion, because expansions aren't unique. We also remark that if f can be expanded w.r.t. $g_1, \cdots, g_n$ and if g is an automatic transseries of the form e^h with $h \succ\!\!\succ 1$ or $\ln_k x$, not equivalent to any of the g_i's, then f can be expanded w.r.t. the g_i's, with g inserted. From now on, when we talk of an automatic transseries, we will often implicitely fix some automatic expansion.

We observe that given an automatic expansion of f, its limiting behaviour is quite obvious. Indeed, $f = 0$ can be tested in $\mathfrak{T}$, and the sign of f can be computed by taking the sign of the dominant coefficient of f (or just the sign of f, if $n = 0$). Similarly, we can compute the limit $\lim f$ of f. If $f \in \mathfrak{C}$, we have $\lim f = f$. Else, let c be the dominant coefficient of f. If $c > 0$, we have $\lim f = 0$. If $c = 0$, we have $\lim f = \lim f_0$. Finally, if $c < 0$, we have $\lim f = (\mathbf{sign}\, f)\, \infty$.

5 Standard expansions of transseries

We would now like to obtain closure results for automatic transseries. Unfortunately, if we have two automatic transseries, the corresponding g_i's need not to be the same. We therefore need some effective version of the structure theorem. Let us make this idea more precise. We say that a set B of transseries of the form e^g (with $g \succ\!\!\succ 1$) or $\ln_k x$ is a *base* w.r.t. a certain expansion of f, if all the g_i's are in B and B is recursively a base for each h_i, whenever $g_i = e^{h_i}$. A base B is said to be *weakly normal*, if $g_1 \asymp\!\!\asymp g_2 \Rightarrow g_1 = g_2$, for $g_1, g_2 \in B$. Now let $f_1, \cdots, f_n$ be transseries, which can all be expanded w.r.t. a certain weakly normal base. Then the usual operations on the f_i's can be performed by considering them as automatic multiseries.

Unfortunately, the concept of weakly normal base is not strong enough for certain future applications (see [VdH 94b] and [VdH *a]). A weakly normal base B is said to be *normal*, if h can be expanded w.r.t. strictly smaller elements of B, whenever $e^h \in B$. Moreover, we demand $\ln_k x$ to be in B, whenever $\ln_{k+1}$ is. Another advantage of using normal bases is that expressions like $e^{x+e^{-x^2}} - e^x$ will be expanded in e^x and not in $e^{x+e^{-x^2}}$.

Different approaches to compute normal bases are possible: Gonnet and Gruntz just compute a maximal element of the base, called a *most rapidly varying subexpression*. Next, they recursively expand all subexpressions w.r.t. this element. Finally, the coefficients are expanded. In fact, they don't really compute a normal base, but it is easy to extract one from their algorithm. A similar approach is to construct the normal base before expanding the coefficients, by using the same rewriting techniques as Gonnet and Gruntz.

We will choose a slightly different appoach, where the weakly normal base is constructed gradually during the expansion of subexpressions. This means that we have a static variable B w.r.t. the expansion algorithm, which contains a normal base. A new element is inserted into B, each time we encounter a new equivalence class for $\asymp$ during the execution. In this case, we will have to check that B remains normal. Automatic expansions w.r.t. B will be refered to as *standard expansions*.

We will give an extendable version of the algorithm; in its present form, it can handle exp-log expressions, but in future papers, we will be able to extend it. More precisely, the algorithm assumes that we dispose of an algorithmic field $\mathfrak{T}$ satisfying the above conditions, and that we access to elements of $\mathfrak{T}$ by means of *expression trees*. These are labeled trees, so that the label of the root determines an algorithm that recursively converts the tree to elements of $\mathfrak{T}$. For example, *exp-log expressions* are trees, whose leafs are labeled by x or elements of $\mathbb{Q}$ and whose nodes are labeled by $+, -, \cdot, /, \ln$ or exp. By abuse of notation, we denote these trees in the same way as the actual elements of $\mathfrak{T}$ they represent. Now the algorithm below shows how to associate expansions to trees whose root is labeled precisely by the previous possibilies; as the algorithm is recursive, other possibilities can be added later (see [VdH 94b] and [VdH *a]).

Algorithm expand(f). The algorithm takes as input an expression f and computes a standard expansion of f w.r.t. $\mathfrak{T}$. It disposes of a static variable B, which is initialized by $\{x\}$.

case $f = x$ or $f \in \mathbb{Q}$: f forms it's own standard expansion.

case $f = f_1 + f_2, f = f_1 - f_2, f = f_1 f_2$ or $f = f_1/f_2$: Compute standard expansions of f_1 and f_2 and apply lemma 1 to compute a standard expansion of f. An error is returned in the case when $f = f_1/f_2$, and $f_2 = 0$ in $\mathfrak{T}$.

case $f = \ln f_1$: Compute a standard expansion of f_1 is in $g_1, \cdots, g_n$. Next compute $c, \alpha_1, \cdots, \alpha_n \in \mathfrak{C}$, such that $f_1 = c g_1^{\alpha_1} \cdots g_n^{\alpha_n}(1 + \varepsilon)$, where $\varepsilon \prec\!\!\prec 1$. By lemma 1, we can compute a standard expansion of f using the formula

$$\ln f_1 = \ln(1+\varepsilon) + \ln(c g_1^{\alpha_1} \cdots g_n^{\alpha_n}) = \ln(1+\varepsilon) + \ln c + \alpha_1 \ln g_1 + \cdots + \alpha_n \ln g_n,$$

because each g_i is either of the form $\ln_k x$, or e^{h_i}, where we can recursively compute a standard expansion of h_i. Possibly, if $g_1 = \ln_k x$ and $\ln_{k+1} x \notin B$, we insert $\ln_{k+1} x$ into B. An error is returned if $c < 0$, or $f_1 = 0$.

case $f = \exp f_1$: Compute a standard expansion of f_1. If f_1 is bounded ($\lim f_1$ is finite), then we can expand f_1 by lemma 1. In the other case, we can test whether there exists a $g = \exp h \in B$, so that $f_1 \asymp h$. If this is the case, we compute $\lambda = \lim f_1/h$ and recursively expand $\exp f_1 = g^{\lambda} \exp(f_1 - \lambda h)$. In the other case, test whether f_1 can be expanded in elements of B strictly smaller than g. If this is the case, insert $\exp|f_1|$ into B, so that $\exp f_1 = \exp|f_1|)^{\mathbf{sign}\, f_1}$ is a standard expansion of $\exp f_1$. If this is not the case, suppose that the expansion of f_1 were in $g_1 \prec\!\!\prec\!\!\prec \cdots \prec\!\!\prec\!\!\prec g_n$, where $g_i \curlyeqprec g$. Then compute φ and ε so that $f_1 = \varphi + \varepsilon$, with $\varphi = (f_1)_{0, \overset{n+1-i\times}{\cdots}, 0}$. Now we expand $f = \exp \varphi \exp \varepsilon$, which can be done by inserting $\exp|\varphi|$ into B and applying lemma 1.

otherwise: Check whether f can be expanded by one of the extensions of expand. If this is not the case an error is returned.

It is easy to verify that B remains normal during the execution. Therefore, the correctness of the algorithm is easy to check. Finally, its termination is clear by structural induction, except when we are repeatedly in the case $f = \ln f_1$, with $f_1 \asymp h$ and $\exp h \in B$. But this can only happen a finite number of consecutive times, because B remains finite and unchanged during such a loop and as $\exp(f_1 - \lambda h) \prec\!\!\prec\!\!\prec \exp(f_1)$, we expand smaller and smaller (for $\prec\!\!\prec\!\!\prec$) exponentials during the loop, each of which is equivalent (for $\asymp\!\!\asymp$) to some element of B.

Remark. If we just want an expansion of f w.r.t. some weakly normal base, we can content ourselves with inserting $\exp|f_1|$ into B, even if f_1 can not be expanded in elements of B strictly smaller than g (in the case where $f = \exp f_1$). We also remark that in the case when no expansion algorithm is found for f, we can return the expression f instead of an error, possibly by expanding its subexpressions.

By an *exp-log function* we mean an element of the smallest field containing $\mathbb{Q}$ and stable by exponentiation and logarithm of positive elements. We will denote by $\mathfrak{T}$ the field of exp-log functions and by $\mathfrak{C}$ its intersection with $\mathbb{R}$. Shackell gave an algorithm which decides about equality of exp-log functions, modulo the constant problem (see [Sh 89a]). The constant problem has been solved by Richardson modulo Schanuel's conjecture in [Rich 92]. This implies that we can compute signs of exp-log constants modulo this conjecture, by evaluating them with a sufficient precision (in fact, a more practical algorithm to determine signs of constants will be given in [VdH *c]). Hence, the hypotheses on $\mathfrak{T}$ and $\mathfrak{C}$ in the previous section are satisfied. Applying the algorithm, we get

Theorem 2. *Schanuel's conjecture implies that the field of exp-log functions is an automatic expansion field.* ♡

6 Bibliography

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

[GoGr 92] G.H. GONNET, D. GRUNTZ. Limit computation in computer algebra. *Technical report 187, ETH. Zürich.*

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

[Rich 92] D. RICHARDSON. The elementary constant problem. *Proc. ISSAC 92 (p 108-116).*

[Sal 91] B. SALVY. Asymptotique automatique et fonctions génératrices. *PhD. thesis, Ecole Polytechnique, France.*

[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 94a] J. VAN DER HOEVEN. Outils effectifs en asymptotique et applications. *Research report LIX/RR/94/09, Ecole Polytechnique, France.*

[VdH 94b] J. VAN DER HOEVEN. General algorithms in asymptotics II, Common operations. *Research report LIX/RR/94/10, Ecole Polytechnique, France.*

[VdH *a] J. VAN DER HOEVEN. General algorithms in asymptotics III, Algebraic differential equations. *In preparation*.*

[VdH *b] J. VAN DER HOEVEN. General algorithms in asymptotics IV, Comparing constants. *In preparation*.*

[VdH *c] J. VAN DER HOEVEN. General algorithms in asymptotics V, Generalized transseries. *In preparation*.*

* Will appear as a research report at the Ecole Polytechnique.