20

Algorithms for Computing Limits and Asymptotic Forms

John R. Shackell
University of Kent, Canterbury

[summary by Bruno Salvy]

Due to problems of cancellations, it is well known that computing the limit of a real function is as difficult as computing an asymptotic expansion. However, existing symbolic computation systems only perform some more or less generalised power series manipulation and are unable to compute difficult limits. The problem studied by J. Shackell in this talk is the computation of asymptotic forms for a real function of one variable which is given either explicitly by composition of the usual field operations, exponentials, logarithms (exp-log functions) or by integrals (liouvillian functions). There are two important steps to perform: *i) computing* the asymptotic expansion, *ii) proving* that this computation is valid for a well-defined class of functions.

1 Nested expansions

The following example helps understanding the type of difficulty one encounters when performing asymptotic expansions:

$$f(x) = \exp(x^{-1} + e^{-x\log x}) - \exp(x^{-1}), \quad x \to \infty.$$

The classical way to find the asymptotic expansion of such a function is to expand both terms and then add up the resulting expansions. The first subproblem one has to solve in order to do so is to prove that

$$\forall k > 0, \qquad e^{-x\log x} = o(x^{-k}), \quad x \to \infty.$$

The second problem is that if one only computes a finite number of terms of each expansion, the only result one will get is $f(x) = O(x^{-k})$, for all finite k. A better way to compute the expansion is to first determine that a cancellation will take place, then rewrite f as

$$f(x) = \exp(x^{-1}) \cdot (\exp(x^{-x}) - 1),$$

and from there derive the expected result:

$$f(x) = \exp(x^{-1})[x^{-x} + \frac{x^{-2x}}{2!} + \cdots + O(x^{-kx})], \quad x \to \infty.$$

The difficulties we faced in this computation are those of the general case: comparing orders of growth, deciding whether a cancellation occurs and dealing with it. Except for the problem of deciding whether an elementary constant is zero or not (called the "constant problem"), these are all the difficulties of the general case. Note that the constant problem is central to symbolic computation. It has not been proved to be undecidable, but is certainly difficult since it is connected

to subtle conjectures in number theory, such as Schanuel's conjecture. However, in 1984 Dahn and Göring gave a non-constructive proof that the problem of computing a limit for an exp-log function could be reduced to the problem of comparing constants. Then in 1990, J. Shackell gave an actual algorithm [3] which shows that the procedure followed in the above example can be automated and will solve the general case. Our next section will discuss the proper setting for this and we shall concentrate on an intuitive description of the algorithm in the rest of this section.

The first thing to do is to determine in which asymptotic scale the expansion will take place. A prior step to this is to compute the set of the different orders of growth occurring in the expression of f.

A proper definition of order of growth, both practical and general is difficult to arrive at. The philosophy here is that sums and products are easy; the logarithm has a reducing role, and thus can only matter in the form of an iterated logarithm of the variable; all the problems have their source in the exponential. The proper attitude towards exponential is to avoid expanding it more than necessary, and this led J. Shackell in [2] to the definition of nested form and nested expansion. Using the classical notations $l_k(x)$ for the logarithm iterated k times and likewise $e_k(x)$ for the iterated exponential, a *nested form* is a finite sequence $\{(s_i, \epsilon_i, m_i, d_i, \phi_i), i = 1 \ldots k\}$, where s_i and m_i are positive integers, ϵ_i is ± 1, d_i is a real number, and ϕ_i will be made more precise later. Such a sequence represents

$$e_{s_1}^{\epsilon_1}(l_{m_1}^{d_1}(x)\phi_1(x)),$$

and recursively

$$\phi_{i-1}(x) = e_{s_i}^{\epsilon_i}(l_{m_i}^{d_i}(x)\phi_i(x)),$$

with the additional constraint that ϕ_i is of a smaller order of growth than l_{m_i}, and that ϕ_k tends to a non-zero finite limit, plus some conditions ensuring that this cannot be reduced by simplifying an $\exp(\log(.))$. Another condition is that the ϕ_i's should belong to a Hardy field and we shall come back to this in our next section. Having thus defined a nested form, one defines a *nested expansion* as a sequence of nested forms N_i, such that N_{i+1} is a nested expansion for ϕ_{i,k_i} minus its limit. These nested expansions should be viewed as a very general asymptotic scale.

An important point is that it is possible to compute effectively with these nested expansions. The ordering, the exponential and the logarithm are easy, the integral can be done but there are difficulties with constants of integration, and the product can be done provided one can compute addition, which is now the difficult operation, because of possible cancellations. To compute the asymptotic expansion of a sum, one first builds up the set of growth orders of the subexpressions of the sum, order it, and then starting from the largest order of growth t_1, compute the *shadow* of the expression with respect to it. This shadow is obtained in many cases (integrals are more complicated) by substituting zero for all the occurrences of t_i with $i \geq 1$ in the expression. Now we appeal to an oracle to decide whether the resulting expression is identically zero or not. If it is, we proceed with the next growth order. The first t_i for which the expression is not zero gives the scale in which the expansion must be performed. This argument is turned into an algorithm for all the exp-log functions in [3].

2 Hardy fields

Once a sketch of the algorithm is clear, it is necessary to determine a class of expressions for which it works, and to prove that it actually works. A good framework for doing this is provided by *Hardy fields*.

Let $\mathcal{X}$ be the ring of germs at ∞ of C^∞ functions. (Think of it as the set of possible asymptotic behaviours.) A *Hardy field* is a subring of $\mathcal{X}$ which is a field closed under differentiation.

The main constraint here is that non-zero elements of Hardy fields have to be invertible, and thus cannot have arbitrarily large zeros. Consequently, since their derivative belongs to the field, they have to be ultimately monotonic and tend to a possibly infinite limit. Also, differences of two (germs of) functions of a Hardy field are also in the field and possess a limit, so that this field is ordered. A short introduction to Hardy fields is given in [4]. If f and g are two elements of a Hardy field tending to infinity, they are said to be *comparable* when there exists a positive integer n such that

$$f < g^n \quad \text{and} \quad g < f^n,$$

where the order is that of the field. Extending this by saying that $\pm f$ and f^{-1} are comparable and that two elements tending to a non-zero finite limit are comparable yields a decomposition of the Hardy field into equivalence classes called *comparability classes* and denoted $\gamma(f)$. One should think of these classes as basic functions of an asymptotic scale. Their number minus one is called *the rank* of the field.

As an important special case of Hardy fields, *Rosenlicht fields* have been considered by J. Shackell. These are Hardy fields of finite rank, closed by $f \mapsto f^c$, for all real c. In [4] it is proved that *any element of a Rosenlicht field has a nested expansion.* Besides, an algorithm is given in [4] to compute a nested expansion of solutions of algebraic differential equations that lie in a Hardy field. Basically, from the order of the equation, one deduces all the possible asymptotic forms of its solutions lying in a Hardy field, and then one tries to substitute in the equation and either get a contradiction or an induction to a "simpler" problem.

A complementary algorithm, described in [2] will compute a nested expansion for liouvillian functions. This needs a little more algebra. If $\mathcal{F}$ is a Hardy field, we need a computable set $\mathcal{I}(\mathcal{F}) = \{t_1, \ldots t_n\}$ of representatives of its comparability classes. Recall that such a set is computable for a Hardy field containing the solutions of an algebraic differential equation. Now the algebraic analogue of *shadows* together with its complementary notion of *ghost* is defined by associating to an element $t \in \mathcal{F}$ tending to zero the set

$$\mathcal{I}_t(\mathcal{F}) = \{f \in \mathcal{F} \,;\, \exists \delta > 0, |f| < t^\delta\}.$$

Then a subfield $\mathcal{S}_t$ of $\mathcal{F}$ has the *shadow property with respect to* t if its intersection with $\mathcal{I}_t(\mathcal{F})$ is $\{0\}$ and it is closed under relative differentiation ($a, b \in \mathcal{S} \Rightarrow a'/b' \in \mathcal{S}$). The ith shadow of $f \in \mathcal{F}$, $\eta_i(f)$ should be thought of as the part of f which is in some $\mathcal{S}_{t_i}(\mathcal{F})$, and the ith-ghost is simply $f - \eta_i(f)$. Given a computable Hardy field $\mathcal{F}$, the extension by a function θ is computable provided that

(i) we can compute a $\mathcal{I}(\mathcal{F})$ monomial T such that θ/T has a non-zero finite limit;

(ii) we can compute the ith shadows and ghosts ϕ_i and ψ_i for θ/T;

(iii) $\mathcal{S}_i(\mathcal{F})(\phi_1, \ldots, \phi_i)$ has the shadow property with respect to t_i and $\psi_i \in \mathcal{I}_{t_i}$;

(iv) there is a zero-equivalence algorithm for $\mathcal{F}(\phi_1, \ldots, \phi_n, \theta)$.

We now display three important cases when this is possible (except perhaps *(iv)*):

Exponential extensions If $g \in \mathcal{F}$ then setting $\eta_i(\exp(g)) = \exp(\eta_i(g))$ works.

Integral extensions If $f \in \mathcal{F}$, then by a theorem of Hardy one can find T. The computation of the shadow depends on the relative orders of growth of T and t_i and can be done (see [2]).

Algebraic extensions If P is a polynomial over $\mathcal{F}$ then it is known that the possible comparability classes of its solutions are those of $\mathcal{F}$, so one can just substitute an arbitrary monomial and compute indeterminate coefficients. This is the basis of the algorithm, together with a general form of Sturm's theorem.

References

[1] G. H. Hardy. *Orders of Infinity*, volume 12 of *Cambridge Tracts in Mathematics*. Cambridge University Press, 1910.

[2] J. R. Shackell. Limits of Liouvillian functions. Preprint.

[3] J. R. Shackell. Growth estimates for exp-log functions. *Journal of Symbolic Computation*, 10:611–632, December 1990.

[4] J. R. Shackell. Rosenlicht fields. To appear in *Transactions of the American Mathematical Society*, 1991.