Limit computation in computer algebra

Dominik Gruntz

ETH Zürich

March 29, 1993

[summary by Bruno Salvy]

The automatic computation of limits can be reduced to two main sub-problems. The first one is *asymptotic comparison* where one must decide automatically which one of two functions in a specified class dominates the other one asymptotically. The second one is *asymptotic cancellation* and is usually exemplified by

$$e^x[\exp(1/x+e^{-x})-\exp(1/x)], \quad x\to\infty.$$

In this example, if the sum is expanded in powers of $1/x$, the expansion always yields $O(x^{-k})$, and this is not enough to conclude.

In 1990, J. Shackell [2] found an algorithm that solved both these problems for the case of *exp-log* functions, i.e. functions built by recursive application of exponential, logarithm, algebraic extension and field operations to one variable and the rational numbers. D. Gruntz and G. Gonnet propose a slightly different algorithm for exp-log functions. Extensions to larger classes of functions are also discussed.

1. Shackell's algorithm

An introduction to Shackell's algorithm can be found in the summary of last year's seminar. To simplify, given an exp-log expression, this algorithm rewrites it into its *nested form*, an almost normal form on which the asymptotic behaviour reads off easily. The difficult operation when doing this is addition because of possible cancellations. When faced with a sum, the algorithm first computes a list of the growth orders in the expressions, then orders it, and by substituting the largest ones by zero, tries to determine which one provides the right asymptotic scale, then it rewrites the expression in terms of this one and a recursive application of this gives the nested form. By this method, J. Shackell effectively reduced asymptotic computation on exp-log functions to the equivalence problem for exp-log constants.

2. The Gruntz-Gonnet algorithm

The Gruntz-Gonnet algorithm proceeds as follows. It first computes a list of orders of growth of the expression, then it finds the most rapidly varying term, expands in terms of it, and applies itself recursively on the leading coefficient if necessary. The three main steps may be summarized as follows:

The most rapidly varying term. This is not quite the most rapidly varying term, but the recursive definition is

$$\begin{aligned}
\mathrm{mrv}(f+g) &= \max(\mathrm{mrv}(f), \mathrm{mrv}(g)),\\
\mathrm{mrv}(f\cdot g) &= \max(\mathrm{mrv}(f), \mathrm{mrv}(g)),\\
\mathrm{mrv}(f^c) &= \mathrm{mrv}(f),\\
\mathrm{mrv}(\log f) &= \mathrm{mrv}(f),\\
\mathrm{mrv}(e^f) &= \mathrm{mrv}(f) \quad \text{if } f \text{ does not tend to } \infty,\\
&= \max(e^f, \mathrm{mrv}(f)) \quad \text{otherwise},\\
\mathrm{mrv}(x) &= x.
\end{aligned}$$

To compute this, the comparison of f and g is done by computing the limit of $\log|f|/\log|g|$, except when $f = x$ or $g = x$. In this case, x is first replaced by $\exp(x)$ throughout both expressions.

Rewriting. All the elements of the set returned by mrv have the same order of growth. In this step all the elements are rewritten in terms of one of them, possibly multiplied by functions with a smaller order of growth.

Expansion. Once the "most rapidly varying term" ω has been found, a Puiseux expansion of the expression in the variable ω is computed. The other terms are considered as harmless parameters. This is then repeated recursively on the leading coefficient if the exponent of ω is 0 and thus does not permit to conclude.

It can be shown that the algorithm based on these steps terminates, and reduces the problem of computing limits to the problem of deciding whether an exp-log function is zero or not.

3. Extensions

Some extensions to accommodate more that exp-log functions are possible, but are limited by the inability to decide or compute heuristically whether an expression of the class is zero or not.

4. Comment

Let us give a brief comparison with J. Shackell's algorithm. Shackell's algorithm drawback is that the rewriting into nested forms can lead to very large expressions, while the Gonnet-Gruntz algorithm can content itself with rougher rewritings. On the other hand, in Shackell's algorithm the difficult step of finding the "right" scale is done only when a sum is encountered, and the expansion is performed directly in the right scale instead of going from the finest one to the right one recursively, which is what the Gruntz-Gonnet algorithm does. Neither complexity analysis nor serious testing of these algorithms have been done, but it is certainly not obvious that the new algorithm is more efficient than the older one. In any case, it seems that a faster one could be devised by merging some ideas of both of them.

Bibliography

[1] Gonnet (Gaston H.) and Gruntz (Dominik). – *Limit Computation in Computer Algebra.* – Technical Report n° 187, Zürich, ETH, November 1992.

[2] Shackell (John). – Growth estimates for exp-log functions. *Journal of Symbolic Computation*, vol. 10, December 1990, pp. 611–632.