Diss. ETH No. 11432

Dominik Gruntz

On Computing Limits in a Symbolic Manipulation System

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZÜRICH

for the degree of
Doctor of Technical Sciences

presented by
Dominik Wolfgang Gruntz
Dipl. Informatik-Ing. ETH
born May 26, 1964
citizen of Basel-Stadt, BS

accepted on the recommendation of
Prof. Dr. G. H. Gonnet, examiner
Prof. Dr. M. Bronstein, co-examiner
Dr. J. Shackell, co-examiner

1996

To Simone, Stephanie, Patrick and Benedikt

Acknowledgements

I am indebted to Prof. Gaston H. Gonnet for his supervision of this thesis. I have profited substantially from his own work in the area of symbolic asymptotic analysis. His confidence in the ultimate success of our algorithm was very encouraging. Moreover, working in his research group I became acquainted with many experts in the field of symbolic computation from all over the world.

I thank Prof. Manuel Bronstein and Dr. John Shackell for their willingness to be co-examiners as well as for their valuable advice. The critical comments of Prof. Bronstein helped to improve the thesis considerably. With Dr. Shackell I could sort out the final problems I had with my implementation of his algorithm to compute nested forms during my visit at the University of Kent at Canterbury.

I also thank Bruno Salvy who invited me to INRIA where I could present and discuss my work. Furthermore, my thanks go to Jacques Carette and Dave Hare. I had many exchanges of ideas with them over e-mail on how MAPLE's series facility could be improved. Particular thanks go to Mike Monagan who initiated me into the secrets of MAPLE programming. I would also like to thank all the members of the MAPLE development team, who have (consciously or unconsciously) tested my code.

I am grateful to my colleagues Lukas Knecht and Wolfgang Weck for carefully proofreading parts of this thesis as well as for our fruitful and exciting discussions.

Last but not least my warmest thanks go to my wife Simone for her great encouragement and her support through all these years.

Table of Contents

Zusammenfassung

In dieser Dissertation stellen wir einen Algorithmus zum Berechnen von einseitigen Grenzwerten vor. Das Bestimmen von Grenzwerten wird in einem Computeralgebra-System in vielen Algorithmen benötigt, etwa beim Berechnen endlicher Integrale oder aber um qualitative Informationen über eine Funktion zu erhalten.

Der Algorithmus, der vorgestellt wird, ist sehr kompakt, einfach zu verstehen und einfach zu implementieren. Zudem löst er das sogenannte Auslöschungsproblem, unter dem andere, klassische Ansätze leiden. Der Schlüssel liegt darin, dass eine Funktion als ganzes betrachtet in eine Reihe entwikkelt wird, und zwar in jenem Teilausdruck, der alle anderen dominiert. Mit diesem Ansatz unterscheidet sich unser Algorithmus von allen anderen auf Reihenentwicklung basierten Algorithmen, die normalersweise einen rekursiven Ansatz über die Struktur der gegebenen Funktion verwenden. Hier müssen bei allen Approximationen stets exakte Restglieder mitgeführt werden, damit das Problem der gegenseitigen Auslöschung von Termen gelöst werden kann, und dies kann sich beim Berechnen von Grenzwerten unangenehm bemerkbar machen (intermediate expression swell). Unser Ansatz umgeht diese Probleme und eignet sich daher besonders zur Implementation in Computeralgebra-Programmen.

Im ersten Kapitel werden ältere Ansätze diskutiert, welche immer noch die Basis von in aktuellen Computeralgebra-Systemen eingebauten Grenzwertberechnungsalgorithmen bilden. Danach präsentieren wir unseren Algorithmus detailliert anhand von exp-log Funktionen und vergleichen ihn mit anderen aktuellen Algorithmen und Ansätzen zur Berechnung von Grenzwerten von exp-log Funktionen.

In einem weiteren Kapitel zeigen wir, wie der Algorithmus für weitere Funktionen erweitert werden kann. Diese Erweiterungen sind so gestaltet, dass sie einfach in heutigen Computeralgebra-Programmen implementiert werden können. Obwohl wir dabei einen sehr pragmatischen Ansatz verfolgt haben, hat sich herausgestellt, dass sehr viele Funktionen damit behandelt werden können.

Des weiteren stellen wir einen Algorithmus zur Berechnung von (verallgemeinerten, hierarchischen) asymptotischen Reihen vor, der auf unserem Algorithmus zur Berechnung von Grenzwerten aufbaut. Dieser Algorithmus wird kurz diskutiert und an Beispielen demonstriert.

In einem letzten Kapitel gehen wir schliesslich auf spezielle Probleme bei der Implementation in einem Computeralgebra-Programm ein und stellen eine Implementation des Algorithmus in Maple vor. Diese Implementation wird dann anhand von Beispielen mit Grenzwertalgorithmen in anderen Computeralgebra-Systemen verglichen.

Abstract

This thesis presents an algorithm for computing (one-sided) limits within a symbolic manipulation system. Computing limits is an important facility, as limits are used both by other functions such as the definite integrator and to get directly some qualitative information about a given function.

The algorithm we present is very compact, easy to understand and easy to implement. It also overcomes the cancellation problem other algorithms suffer from. These goals were achieved using a uniform method, namely by expanding the *whole* function into a series in terms of its most rapidly varying subexpression instead of a recursive bottom up expansion of the function. In the latter approach exact error terms have to be kept with each approximation in order to resolve the cancellation problem, and this may lead to an intermediate expression swell. Our algorithm avoids this problem and is thus suited to be implemented in a symbolic manipulation system.

After discussing older approaches which are still prevalent in current computer algebra systems we present our algorithm in the context of exp-log functions. The algorithm is then compared with other approaches to compute limits of exp-log functions.

We show then how the algorithm can be extended to larger classes of functions. This extension has been designed in the spirit of symbolic manipulation systems, i.e., we have tried to find an algorithm which can easily be implemented in today's computer algebra systems. Although a very pragmatic approach is used for this extension, it turns out that many functions can be covered.

Furthermore we present an algorithm for computing hierarchical asymptotic series, which is based on our limit computation algorithm. This algorithm is discussed and results are presented.

In a final chapter we focus on some particular problems which appear during an actual implementation in a symbolic manipulation system. We show an implementation of the algorithm in MAPLE and compare it on a set of examples with other implementations of limit algorithms in other symbolic manipulation systems.

1. Introduction

This thesis explores the problem of the automatic computation of a limit within a symbolic manipulation system or, as they are called now, a computer algebra system. The concept of a limit $\lim_{x \to x_0} f(x)$ which describes the behaviour of a function $f(x)$ as x approaches some limiting value x_0 is a classical mathematical problem and is fundamental to mathematical analysis. For example the differentiation rules are derived through a limiting process, and also many other problems require the computation of limits in their solution process. Examples are the computation of definite integrals or the determination of the convergence of a series.

As a consequence, a tool to compute limits automatically is very useful in a computer algebra system and does increase its capability for doing analytical calculus. Indeed, all current general purpose computer algebra systems (Axiom, Derive, Macsyma, Maple, Mathematica, MuPAD, Reduce) offer a facility to compute limits. In particular, the limit computation facility is used in symbolic manipulation systems to evaluate definite integrals [89] and definite sums. It is also used in the computation process to determine discontinuities and singularities of functions and to maximize and minimize functions. There are other applications such as the computation of closed form formulas for formal power series [32] and the derivation of nested forms and nested expansions of functions (see Section 4.1).

A facility to compute limits of a function is also very useful by itself. Hamming [35] said, concerning numerical computations, that "The purpose of computing is insight, not numbers". However, in contemplation of results from a computer algebra system exceeding one page, one may adapt Hamming's statement to "The purpose of symbolic computation is understanding, not formulas". In [86], David Stoutemyer made the point that it may be difficult to interpret complicated expressions, and that one would therefore like a symbolic computation system to be able to give *qualitative* information about functions. As possible qualitative properties Stoutemyer lists among others the determination of zeros, singularities and extrema, specification of bounds, the determination of asymptotic representations as certain variables approach infinity and of course the computation of limits.

Strongly related to the computation of limits is the determination of the asymptotic behaviour of a function through an asymptotic series or a nested form, which both provide more information about a function than simply its limit. Additionally, asymptotic series are a powerful tool to enhance numerical definite integration in a symbolic manipulation environment [26, 28] and to perform average case analysis of algorithms [23].

Today's computer algebra systems are very powerful and can solve rather complicated problems, such as the integration of elementary functions or the factorization of polynomials over algebraic extension fields or Galois fields. These systems however are surprisingly poor if they have to solve the apparently simple problem of computing a limit. Many systems even don't have the expertise of a freshman calculus student! Let us look at three examples.

The first one is passed to Reduce 3.6. The system cannot solve this problem and returns the limit unevaluated, although the limit obviously is zero. We will see in Section 2.3.1.1 why Reduce fails on this problem.

```
1: limit(x^7/exp(x), x, infinity);

            7
           x
limit(--------,x,infinity)
        exp(x)
```

The value of the following limit is 1 which can be obtained easily if the first exponential is expanded. Axiom 2.0 however returns *failed* which means that the limit does not exist ([40, p. 249]).

```
(1) ->limit(exp(x+exp(-x))-exp(x), x=%plusInfinity)

   (1)  "failed"
                                              Type: Union("failed",...)
```

In the third example we try to determine the derivative of $\arccos(x)$ with the help of the definition of a derivative. Maple V Release 3 returns the following result:

```
> limit((arccos(x + h) - arccos(x))/h, h=0, right);
```

$$\operatorname{signum}\left(\frac{1}{2}\pi - \arcsin(x) - \arccos(x)\right)\infty$$

Note that $\arccos(x) = \pi/2 - \arcsin(x)$ and thus the result returned by Maple is $0 \cdot \infty$ which is indefinite. The problem is that Maple does not recognize $\pi/2 - \arcsin(x) - \arccos(x)$ to be zero. We will see in subsequent chapters why the other two systems fail on the first two examples.

The history of automatic algorithms for computing limits goes back to the very earliest computer algebra systems. The first program was presented by Fenichel [21] in 1966. The algorithm was based on a number of heuristics to apply the mathematician's classical "bag of tricks" for computing limits, such as the famous l'Hôpital's rule. The approach was to program a computer to behave as a freshman calculus student. Similar programs have been presented by Iturriaga [39] (1967), Wang [89] (1971), Laurent [46] (1973) and Harrington [37] (1979). Some of these algorithms, or at least their ideas, are still used in some modern computer algebra systems.

The next generation limit computation algorithms were no longer based on heuristics. They used series expansions as the underlying concept. Zippel [95] (1976) built his algorithm on top of Taylor series expansions and Geddes and Gonnet [27] (1988) proposed to use a generalized series model. A slightly extended approach was used by Salvy [71] (1991).

Dahn and Göring [20] showed that the problem of determining the limiting behavior of exp-log functions is Turing reducible to the zero equivalence problem for constants. exp-log functions are those obtained from the constant 1 and the variable x by means of rational operations and the functions $\exp(.)$ and $\log(|.|)$. This proof however is not a constructive one, i.e. does not induce an algorithm to compute limits. The first complete limit computation algorithm for exp-log functions has been presented by Shackell in 1990 [77]. This algorithm assumes the existence of an oracle which decides whether a constant expression is zero or not. This algorithm has been extended to meromorphic [82] and Liouvillian [83] functions. None of today's computer algebra systems, however, use implementations of these algorithms.

The algorithm which is presented in this thesis has the same scope as Shackell's algorithm for exp-log functions, but overcomes some difficulties as the techniques used are different. Our algorithm is tailored for the problem of *computing* limits and more suitable for the implementation in a computer algebra system, whereas Shackell's algorithm solves a more general problem, namely the determination of a nested form of a given function. The two algorithms are compared in detail in Section 4.1. An extension of the algorithm for functions more general than exp-log functions is given in Chapter 5. This extension again is tailored towards a concrete implementation in a computer algebra system. Our implementation is compared with other algorithms on a set of examples in Chapter 8.

To complete this introduction, we present two important issues related to the computation of limits. In Section 1.1 we show that the zero-recognition problem sets limits to the computability of limits, and in Section 1.2, the limitations of numerical approximations of limits are discussed.

1.1 Limits of Computing Limits

The problem of computing the limit of functions belonging to certain classes is known to be unsolvable. This follows from the undecidability results for the problem of zero recognition [59, 15]. Richardson has shown, that for the class $\mathcal{R}$ which is the closure of the rational functions in $\mathbb{Q}(\pi, \ln 2, x)$ under the application of the sine, exponential and absolute value functions, the predicate "$E = 0$" is recursively undecidable for $E \in \mathcal{R}$. Consider now the limit

$$\lim_{\varepsilon \to 0} \frac{\varepsilon}{E + \varepsilon} \qquad \text{with } E \in \mathcal{R}. \tag{1.1}$$

The result is 1 if $E = 0$, otherwise the limit of (1.1) is 0. Since the decision $E = 0$ is not recursively decidable for all $E \in \mathcal{R}$, the limit problem is also not recursively solvable in general. This result seems to be very pessimistic at first view, but it turns out that in practice the zero equivalence problem is a minor issue as it can be "solved" using probabilistic or approximative methods, and for the classes of functions which are most used in a computer algebra system such as exp-log functions or Liouvillian functions it can be reduced to the zero equivalence problem of constant expressions. In the discussion of the algorithm we therefore postulate the existence of an oracle to decide the zero equivalence of functions (or constants, respectively).

1.2 A Numerical View of the Limit Problem

One may think that the problem of determining the limiting behaviour of a function is a rather simple problem, since one only needs to look at the graph of the function to immediately know approximately what the limit is. Looking at the graph of the function is nothing else than evaluating the function at selected points in the neighbourhood of the limiting point. Consider the limit

$$\lim_{x \to 0^+} \frac{1}{x^{\ln \ln \ln \ln 1/x - 1}}. \tag{1.2}$$

We first plug in some values in the right neighbourhood of 0, as far as we can approach 0 with MAPLE numerically.

```
> e := 1/x^(ln(ln(ln(ln(1/x))))-1);
```

$$e := \frac{1}{x^{\ln(\ln(\ln(\ln(1/x))))-1}}$$

```
> evalf(subs(x=0.01, e));
```

$$.0001910870078$$

```
> evalf(subs(x=0.001, e));
```

$$.00005602749469$$

```
> evalf(subs(x=0.0001, e));
```

$$.00001246466308$$

```
> evalf(subs(x=10^(-10), e));
```

$$.2176869412\,10^{-8}$$

```
> evalf(subs(x=10^(-100), e));
```

$$.4870365788\,10^{-47}$$

```
> evalf(subs(x=10^(-1000), e));
```

$$.1569728343\,10^{-283}$$

```
> evalf(subs(x=10^(-10000), e));
```

$$.3421606009\,10^{-1640}$$

```
> evalf(subs(x=10^(-100000), e));
```

$$.1066964575\,10^{-7835}$$

```
> evalf(subs(x=10^(-1000000), e));
Error, object too large
```

```
> plot(e, x=0..0.01);
```

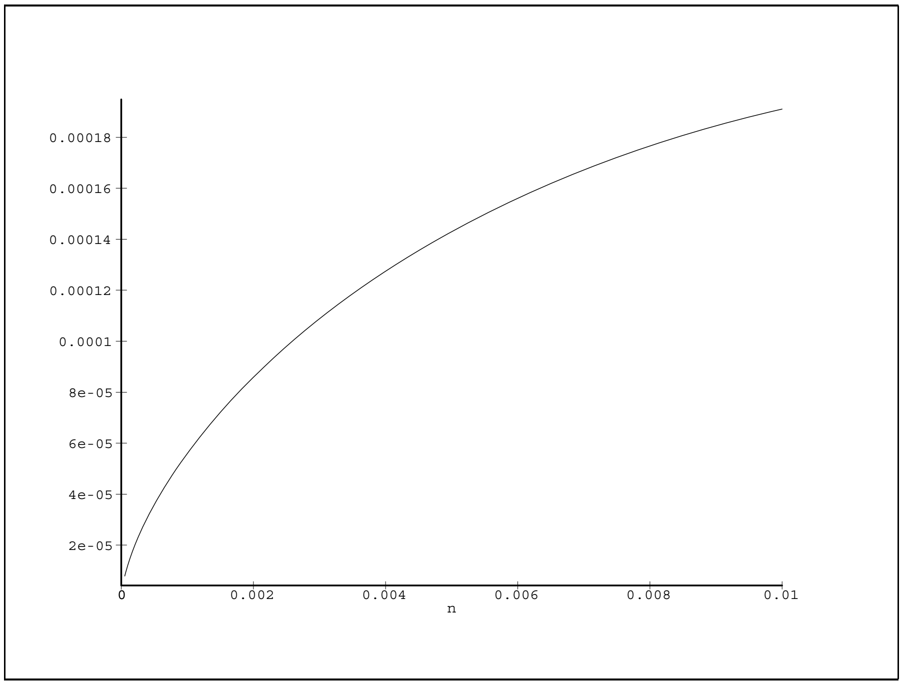

Figure 1.1. Plot of the argument of the limit (1.2) around 0

If we look at the plot in Figure 1.1, we immediately can convince ourselves, that the conclusion drawn from the numerical values is correct: The limit (1.2) really seems to be 0. This method of inspecting the function in the neighbourhood of the limiting point may easily be formalized. Looking at the plot is similar to extrapolating the limiting value from a number of approximations. Provided that the sequence of approximations tends to a limit, we can try to accelerate the convergence by well known sequence transformation techniques. MAPLE offers this technique in its *evalf(Limit())* construct and uses Levin's u-transform [47] to estimate the limit. MATHEMATICA contains the package *NLimit.m* which uses either Wynn's epsilon algorithm [11] or Euler's transformation[1].

```
> evalf(evalf(Limit(e, x=0, right), 40));
```

$$.0002093055787 - .4930030240\, 10^{-50}\, I$$

Our observation appears justified, although the complex part appearing in that result may be surprising. The reason for this is, that for some arguments x, the nested logarithms become negative and complex. In MATHEMATICA the starting point of the sequence can be specified with the *Scale* option. With a suitable starting point we get the result we expected.

```
In[1]:= <<NumericalMath/NLimit.m
In[2]:= NLimit[1/x^(Log[Log[Log[Log[1/x]]]]-1), x -> 0,
                                                 Scale -> 1/100, Terms -> 20]
                       -12
Out[2]= 1.40806 10
```

Why do we need a tool for the exact computation of limits at all? Computing limits seems to be an easy task and algorithms based on numerical approximations seem to have potential! However, the remaining problem is that the above limit is not 0, it is infinity!

```
> limit(e, x=0, right);
```

$$\infty$$

Whenever the infinite is evaluated using finite samplings, it is possible that the results returned are incorrect, and that is exactly what is happening in this example. The iterated logarithms suppress the effect of $1/x$ so much, that for x not too small, the expression is similar to $1/x^{\varepsilon-1} = x^{1-\varepsilon}$ whose limit is zero as x goes to zero. As soon as ε becomes greater than 1, the expression starts growing towards ∞. This happens for $x < e^{-e^{e^{e}}} \approx 0.429 \cdot 10^{-1656520}$ and thus one needs a very high resolution printer and eyes like a hawk to determine the limit by simply looking at the plot.

Another problem with the numerical approach is that it may not be possible to evaluate the function numerically due to overflow, and for the domain in which it can be evaluated, the function may be badly conditioned and the

[1]For a good introduction to convergence acceleration of sequences we refer to the overview article [93].

floating point approximations may suffer from numerical cancellations. For example, consider

$$\lim_{x\to\infty} \left(\operatorname{erf}(x - e^{-e^x}) - \operatorname{erf}(x)\right) e^{e^x} e^{x^2},$$

whose limit is $-2/\sqrt{\pi}$. However, this function cannot be evaluated for $x > 22$ using MAPLE due to limits for the size of the exponent of a floating point number[2], and for $x = 22$ a precision of over 10^9 digits is necessary to overcome the round off errors, which exceeds the maximal possible floating point accuracy in MAPLE. For $x > x_0$ the floating point approximation of $\operatorname{erf}(x-e^{-e^x})-\operatorname{erf}(x)$ is always zero if x_0 is the root of $e^x + \ln x - d \ln 10$ where d is the accuracy of the underlying floating point arithmetic, i.e., $d =$*Digits* in MAPLE.

[2] The break point is different on non 32-bit machines, as the exponent of a floating point number is restricted to one machine word.

2. Computing Limits: An Overview

In this chapter we recall the basic definitions concerning limits which can be found in every introductory calculus text (e.g. [7]) and we show how mathematician's compute limits. We think that the latter is important in order to compare the mathematician's approach with the computational one. The limits we compute by hand in this section will later also serve as test examples for several limit computation algorithms. In the last part of this section we look at some algorithms for computing limits which try to implement this mathematician's "bag of tricks" and highlight the problems they encounter. This will lead us to the motivation for our algorithm.

2.1 General Definitions

Limits are particularly important for the description of the behaviour of a function $f(x)$ at the border of its domain (e.g., at infinity), as well as at isolated singularities (e.g., $\sin(x)/x$ at $x = 0$). We assume that there exist points in the domain of f in every neighbourhood of the limiting point x_0. The question of the behaviour of $f(x)$ when "x tends to x_0" leads to the question: How would one have to define $f(x_0)$ such that $f(x)$ becomes continuous at $x = x_0$? This leads to the following definition for the limit of a function.

Definition 2.1 (Limit of a function, [13, Section 2.1.4]) *If x tends to x_0 ($x \to x_0$), the function $y = f(x)$ has the limit a,*

$$\lim_{x \to x_0} f(x) = a,$$

if there exist points $x \in \operatorname{dom}(f), x \neq x_0$ in every neighbourhood of x_0, and if for every arbitrary small $\varepsilon > 0$ a number $\delta(\varepsilon) > 0$ exists, such that for all $x \in \operatorname{dom}(f)$ with $0 < |x - x_0| < \delta(\varepsilon)$ the inequality $|f(x) - a| < \varepsilon$ holds.

This ε-δ definition goes back to Weierstrass. Note that the definition does not require that the function f is defined at $x = x_0$. Moreover, in the case that f

is defined at x_0, the limit is not required to be $f(x_0)$. Only if f is continuous at $x_0 \in \operatorname{dom}(f)$, then $\lim\limits_{x \to x_0} f(x) = f(x_0)$ which follows from the definition.

If the function $f(x)$ is real valued and grows above every limit as x approaches x_0, we say that the function tends to infinity. More precisely:

Definition 2.2 *If x tends to x_0, the (real valued) function $y = f(x)$ has the non-real limit $+\infty$,*

$$\lim_{x \to x_0} f(x) = +\infty,$$

if for every arbitrary large C a number $\delta(C) > 0$ exists, such that for all $x \in \operatorname{dom}(f)$ with $0 < |x - x_0| < \delta(\varepsilon)$ the inequality $f(x) > C$ holds. Similarly, a function $f(x)$ has the non-real limit $-\infty$ if $\lim_{x \to x_0} -f(x) = +\infty$.

In other words, the graph of the function $f(x)$ has to be above the level of C for every arbitrary large C, as soon as x is close enough to x_0.

Similar definitions can be given if x tends to an non-real boundary point, e.g., to real infinity. The number $\delta(\varepsilon)$ is then replaced by $D(\varepsilon)$ such that the inequalities $|f(x) - a| < \varepsilon$ or $f(x) > C$ respectively hold for all $x > D(\varepsilon)$.

Furthermore, if for the limiting process only function values $f(x)$ for $x > x_0 \in \mathbb{R}$ should be considered, then we write $x \to x_0^+$ and call

$$\lim_{x \to x_0^+} f(x) =: f(x_0^+)$$

the right-hand limit of $f(x)$ at $x = x_0$. The left-hand limit $f(x_0^-)$ is defined similarly. If the left-hand limit and the right-hand limit at $x = x_0$ agree, then this is called the real (bidirectional) limit of $f(x)$ at $x = x_0$. If on the other hand both the right-hand and the left-hand limit exist at $x = x_0$ but are different, then f has a discontinuity at $x = x_0$ and the bidirectional limit does not exist. The equation $f(x_0^+) = f(x_0^-) = f(x_0)$ holds if and only if f is continuous at x_0.

These definitions can all be extended in the obvious way to function defined on $\mathbb{R}^n$, where, now, absolute values are replaced by Euclidean norms.

2.2 Mathematical Approach

Fortunately, for the computation of limits the general Definition 2.1 is used very rarely. Mathematicians use a set of standard limits and a collection of theorems such as l'Hôpital's rule to derive the result. In the following we give an overview over the most important techniques for computing limits.

In most calculus books the description of these techniques is distributed over several chapters, since the preconditions for their proofs are usually presented first. A rather complete presentation (although on a basic freshman level) can be found in [58].

2.2.1 Composition of Limits

The basic approach to compute limits is to reduce them to simpler ones, eventually to so called standard limits whose results are known. The reduction is done recursively, i.e., the arguments of a function are replaced by their limits. The following two lemmas state the conditions under which this basic rule is applicable.

Lemma 2.3 (Composition of Limits) *If the limits*

$$\lim_{x \to x_0} f(x) =: y_0 \quad \text{and} \quad \lim_{y \to y_0} g(y) =: z_0$$

exist, and if either g is continuous at y_0 or f does not take on its limiting value y_0, then

$$\lim_{x \to x_0} g(f(x)) = \lim_{y \to y_0} g(y) = z_0.$$

Since a rational function is always continuous on its domain, the following lemma can be deduced from the last one.

Lemma 2.4 (Algebra of Limits) *Let $R(f_1(x), \ldots, f_n(x))$ be a rational function in the f_i and let for each i the limit*

$$\lim_{x \to x_0} f_i(x) =: y_i, \quad 1 \leq i \leq n$$

exist, and let $R(y_1, \ldots, y_n)$ be defined. Then

$$\lim_{x \to x_0} R(f_1(x), \ldots, f_n(x)) = R(y_1, \ldots, y_n).$$

This lemma tells us that every subexpression of a rational function may be replaced by its limit, if the resulting expression is defined. This defines an algebra of limits which may be used to compute limits: The limit of a sum or a product is the sum or product of the limits of the terms, and the limit of a quotient is the quotient of the limits of the numerator and the denominator. Note that this lemma may be generalized to arbitrary continuous functions R.

However, if $R(y_1, \ldots, y_n)$ is not defined, i.e., if it is an indefinite form such as $\infty - \infty$, $1/0$ or $0/0$, then other rules must be applied. The rest of this section is dedicated to this situation.

2.2.2 Transformations

If the limits y_i of all the arguments of the function R exist but $R(y_1, \ldots, y_n)$ itself is of indefinite form, then one might try to transform R into another form $\tilde{R}$ so that $\tilde{R}(y_1, \ldots, y_m)$ is not indefinite. Possible transformations are the normalization of a rational function into factored normal form, application of the expansion rules for the exponential, logarithm and trigonometric functions, or the inverse operations of these transformations. However, there is no strategy to choose those transformations which will succeed. Mathematical intuition is recommended for a successful application of this method.

Example 2.5 Consider $\lim\limits_{x \to +\infty} \sqrt{\ln(x+1)} - \sqrt{\ln x}$. The limits of both terms of this sum are infinity and using the algebra of limits the form $\infty - \infty$ is obtained which is indefinite. By the following sequence of transformations we obtain a quotient whose form is $0/\infty$ after replacing the numerator and the denominator by their limits. Thus the limit is 0.

$$\begin{aligned}\lim_{x \to +\infty} \sqrt{\ln(x+1)} - \sqrt{\ln x} &= \lim_{x \to +\infty} \frac{\ln(x+1) - \ln x}{\sqrt{\ln(x+1)} + \sqrt{\ln x}} \\ &= \lim_{x \to +\infty} \frac{\ln(1+1/x)}{\sqrt{\ln(x+1)} + \sqrt{\ln x}} = 0.\end{aligned}$$

¶

Other examples requiring clever transformations will suggest themselves to the reader.

2.2.3 Power Series Expansion

If R is a rational function, and if the transformation into factored normal form still leads to the indefinite form $\frac{0}{0}$ or $\frac{\infty}{\infty}$ then an expansion of R into a power series may cancel the zeros or the poles respectively and an inspection of the leading term may reveal the limit.

Example 2.6 For the limit $\lim_{x \to 0} \frac{(1+x)^s - 1}{x}$ the algebra of limits leads to the indefinite form $0/0$, but if we expand the numerator in a power series and divide it through by x, then the result becomes obvious.

$$\lim_{x \to 0} \frac{(1+x)^s - 1}{x} = \lim_{x \to 0} \frac{1 + s\,x + O(x^2) - 1}{x} = \lim_{x \to 0} s + O(x) = s.$$

¶

Example 2.7 Another nice example is $\lim_{x\to 1}(\sqrt[n]{x}-1)/(\sqrt[m]{x}-1)$. Direct substitution leads to the indefinite form $0/0$, but power series expansion succeeds.

$$\lim_{x\to 1}\frac{\sqrt[n]{x}-1}{\sqrt[m]{x}-1}=\lim_{\varepsilon\to 0}\frac{\sqrt[n]{1+\varepsilon}-1}{\sqrt[m]{1+\varepsilon}-1}=\lim_{\varepsilon\to 0}\frac{\frac{1}{n}\varepsilon+O(\varepsilon^2)}{\frac{1}{m}\varepsilon+O(\varepsilon^2)}=\lim_{\varepsilon\to 0}\frac{\frac{1}{n}+O(\varepsilon)}{\frac{1}{m}+O(\varepsilon)}=\frac{m}{n}.$$

¶

Obviously, the power series approach may also be applied to other functions.

2.2.4 L'Hôpital's Rule

L'Hôpital's rule is the most famous rule to resolve indefinite forms. It was discovered by Johann Bernoulli in 1694[1]. It may be applied to limits which lead to an indefinite expression of the form $0/0$ or ∞/∞ when applying the algebra of limits.

Lemma 2.8 (Bernoulli-de l'Hôpital) *Let f and g be two differentiable, real valued functions on $]a, x_0[$ and let*

$$\lim_{x\to x_0{}^-} f(x)=0 \quad \textit{and} \quad \lim_{x\to x_0{}^-} g(x)=0 \quad \textit{or}$$
$$\lim_{x\to x_0{}^-} f(x)=\infty \quad \textit{and} \quad \lim_{x\to x_0{}^-} g(x)=\infty,$$

and $g'(x)\neq 0$ for all x in some interval $]b, x_0[$. Then we have

$$\lim_{x\to x_0{}^-}\frac{f(x)}{g(x)}=\lim_{x\to x_0{}^-}\frac{f'(x)}{g'(x)}$$

provided that the limit on the right hand side exists.

The proof of this lemma is a nice application of the mean value theorem. Note that if $\lim_{x\to x_0} f'(x)/g'(x)$ does not exist, we are not entitled to draw any conclusion about $\lim_{x\to x_0} f(x)/g(x)$. Consider $\lim_{x\to\infty}\frac{x-\sin x}{x+\sin x}$. The limit of both the numerator and the denominator is ∞ and they are both differentiable. If we want to apply l'Hôpital's rule we must first determine the value of

[1]G.F.A. de l'Hôpital (1661-1704) was a French Marquis who was taught in 1692 in the Calculus of Leibniz by Johann Bernoulli (1667-1748), a member of the famous Bernoulli family. They made a contract which obliged Bernoulli to leave his mathematical inventions to de l'Hôpital in exchange for a regular compensation. That is the reason why one of the very important results of Bernoulli (made in the year 1694) was named according to de l'Hôpital, who published the result first in a book in 1696. After the death of de l'Hôpital, Bernoulli complained about de l'Hôpital's "plagiarism". These facts have been resolved by historians based on the exchange of letters between the two [84].

$\lim_{x\to\infty} \frac{1-\cos x}{1+\cos x}$, but this one does not exist, as the denominator has arbitrary large zeros at which the function is not defined. However, the original limit $\lim_{x\to\infty} \frac{x-\sin x}{x+\sin x}$ exists nevertheless and its value is 1.

Example 2.9 Using l'Hôpital's rule we can show that the exponential function grows faster than every integral power. Let $n \in \mathbb{N}$. Then we get

$$\lim_{x\to\infty} \frac{x^n}{e^x} = \lim_{x\to\infty} \frac{nx^{n-1}}{e^x} = \ldots = \lim_{x\to\infty} \frac{n!x}{e^x} = \lim_{x\to\infty} \frac{n!}{e^x} = 0$$

after n applications of l'Hôpital's rule. ¶

Unfortunately, not every indefinite form $0/0$ (or ∞/∞) can be resolved with this rule. It may happen, that $\lim_{x\to x_0} f'(x)/g'(x)$ always leads to the same indefinite form. The simplest example for this is $\lim_{x\to\infty} e^x/e^x$.

A more complicated example is the limit

$$\lim_{x\to 0} \frac{x}{R} \quad \text{where} \quad R = \sqrt{\sqrt{x^4 + 2x^2(r^2+1) + (r^2-1)^2} + x^2 + r^2 - 1}$$

and $r^2 < 1$ (taken from [5]). When l'Hôpital's rule is applied, then the following equality is obtained:

$$\lim_{x\to 0} \frac{x}{R} = \lim_{x\to 0} \frac{1}{\partial R/\partial x} = \lim_{x\to 0} \frac{R}{x} \frac{1}{1 + \frac{x^2+r^2+1}{\sqrt{x^4+2x^2(r^2+1)+(r^2-1)^2}}} = \frac{1-r^2}{2} \lim_{x\to 0} \frac{R}{x}.$$

A further application of l'Hôpital's rule would bring us back to the original function, and a blind application of l'Hôpital's rule would lead to an infinite loop!

Note, that if the limit exists, then the above equation contains the result (up to the sign), namely

$$\lim_{x\to 0} \frac{x}{R} = \pm\sqrt{\frac{1-r^2}{2}}.$$

A numerical test shows that the limit is positive if 0 is approached from right and negative otherwise.

With the help of l'Hôpital's rule also other indefinite forms such as $0\cdot\infty$, $\infty-\infty$, 1^∞, ∞^0 or 0^0 may also be resolved. They only have to be transformed into a function which has the indefinite form $0/0$ or ∞/∞ at the critical point.

If $\lim_{x\to x_0} f(x) = 0$ and $\lim_{x\to x_0} g(x) = \infty$ then $f(x)\,g(x)$ has the indefinite form $0\cdot\infty$ at x_0 and l'Hôpital's rule may be applied either to $f(x)/(1/g(x))$ or to $(1/f(x))/g(x)$. If both $\lim_{x\to x_0} f(x) = \infty$ and $\lim_{x\to x_0} g(x) = \infty$ then $f(x) - g(x)$ has the indefinite form $\infty - \infty$ which may be resolved using the transformation

$$f(x) - g(x) \Longrightarrow \frac{\frac{1}{g(x)} - \frac{1}{f(x)}}{\frac{1}{f(x)\,g(x)}}$$

which leads to the indefinite form $0/0$ at x_0.

Example 2.10 If we apply this rule in the following example with $f(x) = \frac{x}{x-1}$ and $g(x) = \frac{1}{\ln x}$ for $x \to 1$, the indefinite form $\infty - \infty$ is transformed to $0/0$:

$$\lim_{x\to 1}\left(\frac{x}{x-1} - \frac{1}{\ln x}\right) = \lim_{x\to 1}\frac{\ln x - \frac{x-1}{x}}{\frac{x-1}{x}\ln x} = \lim_{x\to 1}\frac{x\ln x - x + 1}{x\ln x - \ln x} = \frac{0}{0}.$$

Two applications of l'Hôpital's rule lead to

$$\lim_{x\to 1}\frac{x\ln x - x + 1}{x\ln x - \ln x} = \lim_{x\to 1}\frac{\ln x}{\ln x + 1 - \frac{1}{x}} = \lim_{x\to 1}\frac{\frac{1}{x}}{\frac{1}{x} + \frac{1}{x^2}} = \frac{1}{2}.$$

¶

The indefinite forms 1^∞, ∞^0 or 0^0 may appear if the operands of a power are replaced by their limits. These indefinite forms can also be resolved with l'Hôpital's rule, if the power $f(x)^{g(x)}$ is converted into the exponential form $\exp(g(x)\,\ln f(x))$. Then the indefinite form of the argument of the exponential becomes $0 \cdot \infty$ which can be resolved using l'Hôpital's rule.

Example 2.11 As an example we compute $\lim\limits_{x\to 0^+} x^x$. The function x^x can be written as $e^{x\,\ln x}$, and for the limit of the argument of the exponential we get

$$\lim_{x\to 0^+} x\,\ln x = \lim_{x\to 0^+}\frac{\ln x}{\frac{1}{x}} = \lim_{x\to 0^+}\frac{\frac{1}{x}}{-\frac{1}{x^2}} = \lim_{x\to 0^+} -x = 0$$

and hence $\lim\limits_{x\to 0^+} x^x = 1$ according to Lemma 2.3. ¶

2.2.5 Squeeze Theorem

The following lemma also has classical applications. Some mathematicians call the method based on this lemma *limit computation using inequalities*.

Lemma 2.12 (Squeeze Theorem) *If for the three functions $f(x), g(x)$ and $h(x)$ the inequality $f(x) \le g(x) \le h(x)$ holds in a neighbourhood of x_0 and if $\lim_{x\to x_0} f(x) = \lim_{x\to x_0} h(x) = a$, then*

$$\lim_{x\to x_0} g(x) = a.$$

Example 2.13 A very classical application of this theorem is the proof that

$$\lim_{x\to 0} \frac{\sin x}{x} = 1. \tag{2.1}$$

It is easy to conclude from a picture that $\frac{\sin x \cos x}{2} \le \frac{x}{2} \le \frac{\sin x}{2\cos x}$. Starting with this inequality we can bound the function $\frac{\sin x}{x}$ between the two functions $\cos x$ and $\frac{1}{\cos x}$ whose limits are both 1.

The limit (2.1) could also be computed using the power series approach or using l'Hôpital's rule, provided that the derivative of $\sin(x)$ is known. This derivative can be obtained independently of the limit (2.1) if Euler's formula for the trigonometric functions $\sin(x)$ is used and if the derivative of the exponential function is known. ¶

2.2.6 Generalized Series Expansion

This approach is a generalization of the algebra of limits in the sense that the arguments of a function are not replaced by their limits but by asymptotically equivalent approximations. It is also a generalization of the power series approach since a more general type of series is used, not necessarily a series in x, but also in other functions such as $e^{-1/x}$. The problem is usually to choose the right scale of expansion.

Example 2.14 As an example of this technique we compute the limit

$$\lim_{x\to+\infty} \frac{\exp(\exp(\psi(\psi(x))))}{x} \tag{2.2}$$

where $\psi(x)$ is the digamma function, defined to be $\Gamma'(x)/\Gamma(x)$. In any mathematical handbook we can find the asymptotic expansion for $\psi(x)$ (e.g. [3, (6.3.18)]) to be

$$\psi(x) = \ln x - \frac{1}{2x} + O\left(\frac{1}{x^2}\right) \tag{2.3}$$

and we see that the limit of $\psi(x)$ itself is ∞. Next we can compute the asymptotic approximations for $\exp(\psi(x))$ and $\exp(\exp(\psi(x)))$ by simple transformations.

$$\begin{aligned}
\exp(\psi(x)) &= x\, e^{-\frac{1}{2x}}\, e^{O(\frac{1}{x^2})} = x - \frac{1}{2} + O\left(\frac{1}{x}\right)\\
\exp(\exp(\psi(x))) &= e^x\, e^{-\frac{1}{2}}\, e^{O(1/x)} = e^x\, e^{-\frac{1}{2}}\,(1 + O\left(\frac{1}{x}\right)).
\end{aligned}$$

Finally, we replace x by the asymptotic form for $\psi(x)$ in the last approximation. Note that the asymptotic form of $\exp(\psi(x))$ has already been derived above.

$$\exp(\exp(\psi(\psi(x)))) = (x - \frac{1}{2} + O\left(\frac{1}{x}\right))\, e^{-\frac{1}{2}}\,(1 + O\left(\frac{1}{\psi(x)}\right))$$

and for the final result we get

$$\exp(\exp(\psi(\psi(x))))/x = e^{-\frac{1}{2}} + O\left(\frac{1}{\ln x}\right)$$

and the result of (2.2) is $e^{-\frac{1}{2}}$. ¶

Our algorithm (and others) also follows the idea of generalized series expansions, but the automatic execution of this approach has its own problems (cf. Section 2.3.3). In Example 5.6 we will see how this particular problem is solved with our algorithm.

2.2.7 Other Tricks

In this part we finally describe some techniques which are difficult to classify. The basic idea is to transform the limit problem (or parts thereof) into a special form which is amenable to an application of the mean value theorem or which defines the derivative of a function at the limit point. If for example a function $f(x)$ is differentiable in x_0, then we know that

$$\lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0} = f'(x_0)$$

and it may happen that this pattern appears in the expression whose limit is to be computed, or that it may be transformed into this form.

Mathematicians use quite a variety of rules and tricks which depend on intuition and experience, and it does not seem to be obvious that the task of computing limits can be implemented in a general algorithm. Laurent [46] stated, in his article concerning the problem of determining the transformations which lead to a form on which general techniques succeed, that *"the problems of this kind are so various, that we might think there is no algorithm or general method at this level. At this point only his flair and his experience will guide the mathematician."* Nonetheless, incorporating the mathematician's classical "bag of tricks" into a computational method for computer algebra systems has been studied. We look at these methods in the next section, and will also illustrate the problems these approaches have.

To support Laurent's statement we close this section with an advanced limit example. This example grew out of the tests we generated for our algorithm, and it turned out that it was easier to solve this problem using the techniques of our algorithm than using a classical approach.

Example 2.15 Consider the limit $\lim_{x\to\infty} e_4(x-1/e^x)/e_4(x)$ where e_n is the n times iterated exponential function. Let us first set up the following two inequalities:

$$\begin{array}{llcll} (a) & e^{-c} & < & 1-\frac{c}{2} & \text{if } 0<c<1 \\ (b) & e^{-c} & < & \frac{1}{c} & \text{if } c>0 \end{array}$$

Using the above two relations, we can define the following ones:

$$\begin{array}{rcl} e_1(x-e^{-x}) & = & e^x e^{-e^{-x}} \overset{(a)}{<} e^x(1-e^{-x}/2) = e^x - 1/2 \\[2ex] e_2(x-e^{-x}) & < & \exp(e^x-1/2) = e_2(x)e^{-1/2} \overset{(a)}{<} e_2(x)(1-1/4) \\[2ex] e_3(x-e^{-x}) & < & \exp(e_2(x)-e_2(x)/4) = e_3(x)\exp(-e_2(x)/4) \\ & & \overset{(b)}{<} e_3(x)4/e_2(x) < e_3(x)/2 \\[2ex] e_4(x-e^{-x}) & < & \exp(e_3(x)-e_3(x)/2) = e_4(x)\exp(-e_3(x)/2) \\ & & \overset{(b)}{<} e_4(x)2/e_3(x) \end{array}$$

and consequently we get the final inequality

$$0 < \frac{e_4(x-1/e^x)}{e_4(x)} < \frac{2}{e_3(x)}$$

and hence the above limit tends to 0.

¶

2.3 Computational Approach

There has been an interesting evolution concerning limit computation programs over the last twenty years. The first algorithms pioneered by Paul Wang [90] use heuristic methods and try to simulate the techniques we have described in the last section. The next generation systems were based only on series expansions, namely power series as in [95] and generalized series expansion as in [27, 71]. The latter approaches suffer from the problem of cancellation, which we will describe in the last section of this chapter. The algorithms proposed by John Shackell [77, 80] and our approach overcome this final difficulty.

It is surprising, though, that many commercially available computer algebra systems still use limit computation algorithms which are based on the heuristic ideas of the very early approaches. To demonstrate the difficulties and problems of these algorithms we thus can still use some of today's computer algebra systems as a reference.

2.3.1 Heuristic Approach

Fenichel [21] has studied automating the computation of limits and provided some basic routines for computing two sided limits in the FAMOUS[2] system. The class of problems solved by his program is restricted to piecewise analytical functions. Indefinite forms are resolved only by using l'Hôpital's rule. The algorithm was specified through a collection of rules. It seems that Fenichel got discouraged in his effort to study the mechanization of limits by the undecidability results of Richardson [59, 60], which prove that there is no decision procedure for some classes of limit problems (see Section 1.1). It turns out however that this result is not a reason to throw in the towel as heuristic methods exist which "solve" the zero equivalence problem reasonably well in practice. We think that it is very challenging to implement an algorithm whose only restriction is the undecidability result of Richardson as all current implementations of limit algorithms have still other deficiencies.

Iturriaga [39] worked on one sided limits in his thesis. In addition to l'Hôpital's rule for resolving indefinite quotients, he uses some asymptotic analysis to resolve indefinite forms of quotients of polynomials. Essentially he replaces those polynomials by their leading terms, a technique which is easy to outfox. This program was written in Formula Algol.

Wang [89] provided in his thesis a limit computation facility called DELIMITER[3], which he needed for the evaluation of some definite integrals. DELIMITER is also a heuristic program written for computing limits of real or complex analytic functions, which uses several approaches. For the special case of rational functions a fast special routine is provided. Composition of limits is used for limits of continuous functions. Complicated expressions are reduced by replacing them with asymptotically equivalent and simpler ones. Other techniques which are used are l'Hôpital's rule and heuristics for the comparison of orders of infinity. L'Hôpital's rule is only applied if the function does not contain exponential functions whose arguments tend to $\pm\infty$. In some cases, power series are also used to obtain the limit. This algorithm is the basis of the current limit implementation in the Macsyma system.

The main emphasis of the DALI[4] program written by Laurent [46] is on transformations (cf. Section 2.2.2) in the case that the function has an indefinite form at the limit point. The methods which are applied after the transformations are truncated Taylor series expansions (with rational coefficients) and simple comparisons of orders of infinity. The latter only distinguishes between the three classes: powers, exponentials of a power and logarithms of a power.

Harrington [37] implemented a symbolic limit evaluation program in Mode-Reduce [38]. Besides l'Hôpital's rule to resolve indefinite forms, he uses the

[2] Fenichel's Algebraic Manipulator for On-line USe
[3] DEfinite LIMIT EvaluatoR
[4] Détermination Automatique des LImites

power series approach and a comparison of orders of infinity. The comparison of orders of infinity is not as complete as in Wang's algorithm, but improved over the DALI approach. The functions are identified and ordered according to the scale

$$\ln\ln x \prec \ln x \prec \frac{1}{x^n} \prec e^{1/x}$$

where $x \to 0$. Additionally, special transformations and simplifications are performed before a particular method is applied.

All these algorithms are based on heuristics and consequently fail on those examples where the heuristic does not succeed. Furthermore, l'Hôpital's rule, which is used by all these programs, has its own difficulties. Finally, some rules borrowed from mathematicians are applied without testing (or without having the ability to test) whether the preconditions are met on which these rules are valid. These problems are discussed next.

2.3.1.1 L'Hôpital's rule may not terminate

One problem with l'Hôpital's rule is, that for some functions it may not terminate. The simplest example is probably $\lim_{x\to+\infty} e^x/e^x$ ([21, p. 52]), and another one is $\lim_{x\to+\infty}(e+1)^{x^2}/e^x$ ([90, p. 462]). Since one cannot decide for a given function whether the application of l'Hôpital's rule will eventually succeed, a heuristic is needed to decide when to stop using the rule. In Macsyma the number of applications is controlled through the variable *LHOSPITALLIM* which is set to four by default. Harrington only performs three applications in his implementation, but then he asks the user whether to continue or not. Mathematica and Reduce apply l'Hôpital's rule a fixed number of times which cannot be changed by the user. The limit $\lim_{x\to+\infty} x^n/e^x = 0$ may be used as a black box test to figure out whether a given algorithm is based on l'Hôpital's rule and how often it is applied at most. In Reduce 3.6 and Mathematica 2.2 this bound is three.

```
1: limit(x^3/exp(x), x, infinity);

0

2: limit(x^4/exp(x), x, infinity);

             4
            x
limit(--------,x,infinity)
        exp(x)

In[1]:= Limit[x^3/Exp[x], x -> Infinity]
Out[1]= 0

In[2]:= Limit[x^4/Exp[x], x -> Infinity]
                  4
                 x
Out[2]= Limit[--, x -> Infinity]
                 x
                E
```

All these heuristics are unsatisfactory because they may stop the computation when further applications of l'Hôpital's rule would succeed. It is also does not help to leave this bound under user control, because the limit may be part of a larger computation, and the user may not realize that the whole computation failed due to a unresolved limit.

2.3.1.2 Two ways to apply l'Hôpital's rule

Another problem with l'Hôpital's rule is that there are two ways it may be applied, corresponding to the two cases in Lemma 2.8. If $f(x)$ and $g(x)$ tend both to 0 (or both to ∞), then we can apply l'Hôpital's rule either to the quotient $f(x)/g(x)$ or to $f(x)^{-1}/g(x)^{-1}$. This distinction is in particular important if $f(x)$ tends to 0 and $g(x)$ to ∞, since then l'Hôpital's rule may be applied either to $f(x)/g(x)^{-1}$ or to $f(x)^{-1}/g(x)$. Pursuing all possibilities in breadth first search manner would lead to an exponential growth of the run time and hence a heuristic is needed to decide which branch should be executed. Note that with wrong branch decision the algorithm may not terminate. Consider the following example where we use the heuristic to take the version of l'Hôpital's rule where both the numerator and the denominator of the quotient tend to zero.

$$\lim_{x\to 0} \ln(x)\, x = \lim_{x\to 0} \frac{x}{\ln^{-1}(x)} \Rightarrow \lim_{x\to 0} \frac{x}{-\ln^{-2}(x)} \Rightarrow \lim_{x\to 0} \frac{x}{2\ln^{-3}(x)} \Rightarrow \cdots$$

In this example, another choice of the branches would succeed, as we have seen in Example 2.11.

2.3.1.3 Growth of expressions

Computing the derivatives when applying l'Hôpital's rule on an expression may increase the size of the problem. It has been shown in [16] that there exist expressions whose representation (in a certain measure) requires $O(n)$ space, but whose k'th order derivative requires $O\left(n\binom{n+k-2}{k-1}\right)$ space. If l'Hôpital's rule is applied too often the expressions become intractable, and this is another reason to bound the number of applications of l'Hôpital's rule. The heuristic of Wang is to stop the process if the size of the function grows for three consecutive applications of l'Hôpital's rule.

2.3.1.4 L'Hôpital's rule may be wrong

Another problem is that l'Hôpital's rule may be wrong [8] if it is not applied properly. The particular problem in the context of a computer algebra system is that it may be difficult to discover that $\lim f'(x)/g'(x)$ does not exist. Let us consider the following problem [85]:

$$\lim_{x\to +\infty} \frac{\frac{1}{2}\sin 2x + x}{e^{\sin x}\,(\cos x\,\sin x + x)}.$$

Both the numerator and the denominator tend to $+\infty$ as $x \to +\infty$, and both are differentiable. Hence l'Hôpital's rule may be applied. We will use Maple to perform the necessary steps. Since $f'(x)$ and $g'(x)$ contain trigonometric functions, the quotient $f'(x)/g'(x)$ is simplified before it is passed to *limit*.

```
> f := sin(2*x)/2+x:
> g := exp(sin(x))*(cos(x)*sin(x)+x):
> limit(f, x=infinity), limit(g, x=infinity);
```

$$\infty, \infty$$

```
> e := diff(f,x)/diff(g,x):
> simplify(normal(e, expanded));
```

$$2\,\frac{\cos(x)e^{-\sin x}}{2\cos(x)+\cos(x)\sin(x)+x}$$

```
> limit(",x=infinity);
```

$$0$$

Since the latter limit exists and is zero, $\lim_{x\to+\infty} f(x)/g(x)$ is also zero according to Lemma 2.8. However, this is not true, since $f(x)/g(x) = e^{-\sin x}$ and its limit is indefinite and bounded between $1/e$ and e. MAPLE obtains this result as it does not use l'Hôpital's rule to compute limits.

```
> limit(f/g, x=infinity);
```

$$e^{-1} \;..\; e$$

What went wrong in the above derivation? The problem is, that in fact $\lim_{x\to+\infty} f'(x)/g'(x)$ does not exist, since $g'(x)$ has zeros in every neighbourhood of $+\infty$, and hence we were not entitled to apply l'Hôpital's rule. The simplification step however normalized the fraction and removed the term $\cos x$ from both the numerator and the denominator. The new denominator is bounded away from 0 and tends to $+\infty$, hence the limit of the normalized quotient exists and is zero. This (automatic or optional) cancellation of removable singularities is a difficult problem in today's computer algebra systems [87]. It usually leads to more concise results, but, as we have just seen, may also be the source of errors. It is not easy to handle this type of problems correctly in general [18].

2.3.1.5 Difficulty detecting continuity

All the heuristic algorithms mentioned in this section apply the rule

$$\lim_{x\to x_0} f(g(x)) = f(\lim_{x\to x_0} g(x)),$$

which follows from Lemma 2.3 on the composition of limits. As we have seen in Section 2.2.1, this rule is only valid if f is continuous at $\lim_{x\to x_0} g(x)$. Unfortunately, it is rather difficult to decide in a computer algebra system whether a function is continuous or not at a given point. Note that the limit facility should not be used on $f(g(x))$ to answer this question in this particular situation. Some systems are simplistic and assume continuity if nothing else is known. As a consequence, wrong results may be returned. We give two examples here. The first one is $\lim_{x \to 0} \lceil x \rceil$, which is undefined. The right-hand limit at 0 is 1 and the left-hand limit is 0. We use the special *Limit.m* package [4] written in Mathematica. It seems that the code does not realize the discontinuity at $x = 0$ and simply replaces x with 0 in $\lceil x \rceil$ which then simplifies to 0.

```
In[1]:= <<Calculus/Limit.m

In[2]:= Limit[Ceiling[x], x -> 0]
Out[2]= 0

In[3]:= Limit[Ceiling[x], x -> 0, Direction -> -1] (* from above *)
Out[3]= 0
```

Another example which is more difficult, and which is done incorrectly by almost all systems, is the limit on branch cuts of the principal value of multi-valued functions such as the square root. Let us consider the limit $\lim_{x \to 0^+} \arctan(2\,i - x)$. The limit of the argument of the arctan lies on the imaginary axis, where arctan is discontinuous. The correct result is

$$\lim_{x \to 0^+} \arctan(2\,i - x) = \arctan(2i) - \pi = i \ \mathrm{arctanh}(2) - \pi = (i \ln 3 - \pi)/2,$$

which can be verified with numerical approximations. Macsyma, Mathematica and Axiom however return the wrong result:

```
(c1) limit(atan(2*(-1)^(1/2)-x), x, 0, plus);
(d1)                                %i atanh(2)

In[1]:= Limit[ArcTan[2*(-1)^(1/2)-x], x -> 0, Direction -> -1]
Out[1]= I ArcTanh[2]

(1) ->limit(atan(2*(-1)^(1/2)-x), x=0, "right")
                 +---+
   (1)  atan(2\|- 1 )
                    Type: Union(OrderedCompletion Expression Integer,...)
```

The problem is that these systems test for continuity using a Taylor series expansion of $\arctan(x)$ at $x = 2i$. All three systems return for this series a result which indicates that $\arctan(x)$ is continuous at $x = 2i$, although numerical approximation of $\arctan(x)$ in the neighbourhood of $x = 2i$ shows that all systems define the line $[i, \infty i)$ as branch cut for $\arctan(x)$. This particular problem is discussed in detail in Section 7.4.

2.3.2 Power Series Approach

Zippel [95] proposed an algorithm which is not based on heuristics, but rather on the concept of univariate power series expansion. In order to compute the limit of $f(x)$ at x_0, the power series of $f(x)$ at x_0 is computed. This power series only exists, if $f(x)$ is continuous at $x = x_0$, hence only two-sided limits are considered with this approach. With a linear or a bilinear transformation x_0 can be transformed to the origin and thus we may assume $x_0 = 0$. If $f(x)$ has the power series expansion

$$f(x) = c_0\, x^{e_0} + c_1\, x^{e_1} + \cdots$$

at $x = 0$, then the limit of $f(x)$ is 0 if $e_0 > 0$, c_0 if $e_0 = 0$ and $\pm\infty$ if $e_0 < 0$. The sign of the latter result depends on the sign of c_0, on e_0 and, in case that e_0 is odd, on the direction 0 is approached.

One major advantage of this approach over the heuristic ones is that it is quite easy to incorporate new functions. As soon as the underlying series model gets updated the limit facility can profit thereof. For the heuristic approach however, all the heuristics concerning for example transformations need to be updated. Further changes are then normally needed at various other places in the system.

The type of series Zippel considered may also contain logarithmic singularities in their coefficients (similar to MAPLE's *series* model), but the problem of higher order essential singularities is ignored. He considers "the complete incorporation of the higher class essential singularities in a power series system to be one of the most difficult of all the problems".

Also, with this restriction it may happen that a function is analytic at the expansion point but contains some subexpressions which have essential singularities there. The straightforward series expansion algorithms hence fail. An example is

$$\lim_{x\to 0} \frac{e^{\csc x}}{e^{\cot x}}. \tag{2.4}$$

Both the numerator and the denominator contain essential singularities at $x = 0$ since both $\csc x$ and $\cot x$ have a pole of order one at the origin.

```
> series(exp(csc(x))/exp(cot(x)), x);
Error, (in series/exp) unable to compute series
```

However, if the function in (2.4) is simplified, i.e. if the two exponentials are combined, then the two poles cancel and a Taylor series expansion can be computed without problems.

```
> combine(exp(csc(x))/exp(cot(x)), exp);
```

$$e^{\csc(x)-\cot(x)}$$

```
> series(", x);
```

$$1 + \frac{1}{2}\,x + \frac{1}{8}\,x^2 + \frac{1}{16}\,x^3 + O\left(x^4\right)$$

The leading term is 1 and thus the result of (2.4) is also 1. Note that again heuristics would be necessary to perform the right transformations, and moreover not all problems can be solved with such an approach.

Zippel proposed another technique to handle essential singularities, namely to allow a power series be multiplied by an essential singularity. For the example (2.4) the series of both the numerator and the denominator become

$$\begin{aligned} e^{\csc x} &= e^{\left(x^{-1}+\frac{1}{6}x+\frac{7}{360}x^3+O(x^5)\right)} = e^{1/x}\left(1+\frac{1}{6}x+\frac{1}{72}x^2+O(x^3)\right) \\ e^{\cot x} &= e^{\left(x^{-1}-\frac{1}{3}x-\frac{1}{45}x^3+O(x^5)\right)} = e^{1/x}\left(1-\frac{1}{3}x+\frac{1}{18}x^2+O(x^3)\right). \end{aligned}$$

When the two series are divided, the essential singularities cancel and we obtain the expected power series.

As Zippel has pointed out, this technique is extremely dangerous if it is not applied properly, since the essential singularity terms dominate the power series, unlike the logarithmic terms. A straight forward computation with essential singularities would yield the "power series"

$$\frac{1}{1+x\,e^{1/x}} = 1 - e^{1/x}\,x + e^{2/x}\,x^2 + O\left(e^{3/x}\,x^3\right)$$

which, although formally valid, is completely foolish.

We mention these ideas since they form the basis of the generalized series approach which is discussed next and which is the basis of our algorithm. Additionally, we want to note that our algorithm manages to perform the necessary transformation automatically on $\lim_{x\to 0} e^{\csc x}/e^{\cot x}$ and finally also computes the power series of $e^{\csc x-\cot x}$ (cf. Example 5.3).

2.3.3 Generalized Series Approach

The algorithm proposed by Geddes and Gonnet [27] is based on the unifying concept of series expansion. In order to overcome the limitations inherent in Zippel's approach, they defined the concept of *hierarchical series*. A hierarchical series of a function $f(x)$ is a sum of terms of the form

$$f(x) = s_1(x)\,h_1(x) + s_2(x)\,h_2(x) + \cdots$$

where each $s_i(x)$ is a generalized power series and each $h_i(x)$ is a canonical representation of an essential singularity. It approximates the function $f(x)$ in a right-neighbourhood of 0. As a consequence only one-sided limits are considered. A generalized power series is a Puiseux series whose coefficients may be (bounded) functions, and the h_i's are strictly ordered according to their growth, i.e.

$$\forall\, k \in \mathbb{N} \quad : \quad \lim_{x \to 0} \frac{|h_{i+1}(x)\, x^k|}{|h_i(x)|} = 0.$$

An extension of this approach was used by Salvy [71] in his thesis. He used asymptotic series expansion of the form

$$f(x) = a_1(x)\, \varphi_{i_1}(x) + a_2(x)\, \varphi_{i_2}(x) + \cdots$$

where the φ_i are functions from an asymptotic scale $S = \{\varphi_i\}_{i \in I}$ and the $a_i(x)$ are bounded functions. S is called an asymptotic scale, if

$$\lim_{x \to x_0} \frac{|\varphi_j(x)|}{|\varphi_i(x)|} = 0 \qquad \text{for} \quad i < j.$$

This latter approach is implemented in the MAPLE package *gdev* [70] (which stands for generalized development).

All limit algorithms in this class are based on the idea of expanding the function into a kind of generalized series and then of examining the leading term. For the series expansion an algorithm similar to Algorithm 2.16 is used, which expands the function inside out (or bottom up if we look at the expression tree representation of the function).

Algorithm 2.16 Bottom Up Recursive Algorithm

```
series(e)
    if   $e = \sum e_i \rightarrow$   RETURN($\sum$ series($e_i$))
    elif $e = \prod e_i \rightarrow$   RETURN($\prod$ series($e_i$))
    ⋮
    elif $e = f(z) \rightarrow$   $x_0$ := limit($z$);
        s := series($z$);
        if $f(x)$ has essential singularity at $x = x_0 \rightarrow$
               RETURN(HandleEssentialSingularity($f, x_0, s$))
        elif $f(x)$ has a pole at $x = x_0 \rightarrow$
               RETURN(HandlePole($f, x_0, s$))
        else   RETURN$\left(\sum_{k=0}^{\infty} f^{[k]}(x_0)\,(s - x_0)^k/k!\right)$
        fi
    ⋮
    fi
```

The problem this algorithm may run into is called the *cancellation problem* [77, p. 617]. If the algorithm is computing the series expansion at a node in the expression tree of the given function it may happen that all terms computed vanish and that the algorithm keeps calling forever for new terms of the series of the arguments. This obviously happens if the sub expression which is expanded is identically zero, as e.g. for $\sin^2 x + \cos^2 x - 1$ or even simpler for $e^x - e^x$. These cases can be caught with a test for zero equivalence. Unfortunately, the cancellation problem may also appear if the function which is expanded is *not* the zero function.

Example 2.17 Consider the following limit problem which we try to solve using a generalized power series approach:

$$\lim_{x \to +\infty} e^x \left(\sin\left(1/x + e^{-x}\right) - \sin\left(1/x\right)\right). \tag{2.5}$$

The arguments of the two sin functions both tend to zero at $x = +\infty$ and thus these subexpressions can directly be expanded into their power series. This leads to the expression

$$e^x \Bigg(\left\{\left(\tfrac{1}{x} - \tfrac{1}{6x^3} + \tfrac{1}{120x^5} + \ldots\right) + \left(1 - \tfrac{1}{2x^2} + \tfrac{1}{24x^4} + \ldots\right) e^{-x} + \ldots\right\} - \left(\tfrac{1}{x} - \tfrac{1}{6x^3} + \tfrac{1}{120x^5} + \ldots\right)\Bigg)$$

which needs to be expanded further. The powers in x dominate the series expansions of the two sin functions. In the difference however, all these terms cancel out. If we compute the difference of the two sin functions in a straightforward manner, we will never get an answer. Let us demonstrate this behaviour with the *gdev* package, which uses the generalized series approach.

```
> gdev(sin(1/x + 1/exp(x)), x=infinity, 4);
```

$$\left(\frac{1}{x}\right) + \left(-\frac{1}{6}\frac{1}{x^3}\right) + \left(-\frac{1}{5040}\frac{1}{x^7}\right) + \left(O\left(\frac{1}{x^9}\right)\right)$$

```
> gdev(sin(1/x), x=infinity, 4);
```

$$\left(\frac{1}{x}\right) + \left(-\frac{1}{6}\frac{1}{x^3}\right) + \left(-\frac{1}{5040}\frac{1}{x^7}\right) + \left(O\left(\frac{1}{x^9}\right)\right)$$

```
> gdev(sin(1/x + 1/exp(x)) - sin(1/x), x=infinity, 4);
```

$$\left(O\left(\frac{1}{x^{15}}\right)\right)$$

The last answer returned by *gdev* is correct, since the asymptotic *power* series of $\sin(1/x + e^{-x}) - \sin(1/x)$ at $x = +\infty$ is indeed $0 + 0\,x^{-1} + 0\,x^{-2} + 0\,x^{-3} + 0\,x^{-4} + \cdots$ although the function is not zero. One can also say that the two series expansions of $\sin(1/x + e^{-x})$ and $\sin(1/x)$ are equal for the first n terms for any finite n. The information that the arguments of two functions were different is lost at this point, i.e. if only the first n terms of the series expansions of the two sin functions are available.

However, *if we know that the dominant terms all cancel out*, then we can cancel them by hand and obtain the expansion

$$e^x\left(\sin(\frac{1}{x}+e^{-x})-\sin(\frac{1}{x})\right)\asymp\left(1-\frac{1}{2x^2}+\frac{1}{24x^4}-\frac{1}{720x^6}+\dots\right)+O(e^{-x})$$

for our function. As a consequence the result of (2.5) is one. However, automatically realizing that all dominant terms cancel out is not a trivial task.

Problem (2.5) could directly be resolved if we expand the first sin function and collect the terms appropriately. We then get the function

$$e^x\,\sin(1/x)\,(\cos(e^{-x})-1)+e^x\,\cos(1/x)\,\sin(e^{-x})$$

whose generalized series expansion can be computed with the help of *gdev*.

```
> gdev(exp(x)*sin(1/x)*(cos(exp(-x))-1), x=infinity);
```

$$\left(-\frac{1}{2}\frac{1}{x\,e^x}\right)+\left(O\left(\frac{1}{x^3\,e^x}\right)\right)$$

```
> gdev(exp(x)*cos(1/x)*sin(exp(-x)), x=infinity);
```

$$(1)+\left(O\left(\frac{1}{x^2}\right)\right)$$

Note that the expression has been broken up in such a way that the two series can be expanded in their appropriate asymptotic scales. However such a solution can only be generalized into a heuristic and is thus disqualified for obvious reasons. Furthermore, not all functions can be expanded like the sin function. The latter approach would therefore fail if we replaced the sin function with the Gamma function in example (2.5). ¶

With this example we have demonstrated that it is not always that easy to find the right entry in the asymptotic scale in which the series expansion has to be performed. The problem is aggravated by the fact that a wrong choice of the expansion of a particular subexpression may only be detected at a higher level in the expression tree, and the series expansions of the subexpressions would thus have to be redone. Furthermore there is the additional problem of detecting cancellation. The *gdev* procedure simply gives up if more than a bounded number of terms cancel.

Shackell [77, 80] proposed an algorithm which resolves this problem. The basic idea of his solution is that each series expansion contains in its last term the whole rest of the series. In other words, a series expansion is no longer an approximation of a function but rather just another representation for it. No information is ever lost and the cancellation problem, once detected, can be resolved. The detection of the cancellation problem is done with the help of zero equivalence tests. For more details we refer to Chapter 4.

Our solution solves this problem by simply steering clear of it. We do not take any precautions for the case that the cancellation problem may show up, as in Shackell's approach, but rather operate in such a way that the cancellation problem never appears. Recall that we ran into the cancellation problem in Example 2.17 since the expansion of the sin functions was performed in terms of x instead of e^{-x}. Roughly speaking, there are always several possible forms in which a function can be expanded. It turns out that these functions can be ordered according to their growth: $f \prec g$ if and only if $\ln|f| = o(\ln|g|)$, e.g. $x \prec e^x$. The cancellation problem only appears if a function is expanded in terms of a function which is too small. In our algorithm we expand the whole function always in terms of the largest possible scale entry. We denote this as the "most rapidly varying" subexpression. If necessary, this process may be applied again to the leading coefficient of this series. However, as a consequence, the cancellation problem can never appear and so our approach overcomes the difficulties inherent in the generalized series approach.

3. Algorithm for Computing Limits of exp-log Functions

In this chapter we will present our algorithm for computing limits. As we outlined at the end of the last section, we also use a series expansion approach, but we first determine the most rapidly varying term, on which we perform the expansion. However, we have not yet specified how this order is defined and which class of function can be treated with this approach. It turns out that Hardy fields are the natural domain in which to work. In the first section we thus recall some facts from the theory of Hardy fields. A more comprehensive overview can be found in the papers of Rosenlicht [67, 68, 69] and in [79]. For the rest of the chapter we will then restrict our view to exp-log functions (defined below). After a discussion of the current state of deciding zero equivalence in this particular function field we will present the details of the algorithm and of the proof of its termination. In Chapter 5 we will discuss how the algorithm can be extended to other function classes.

3.1 Hardy Fields

We shall consider real functions of a real variable x which are defined in a semi-infinite interval $x > x_0 \in \mathbb{R}$. Let $\mathcal{K}$ be the set of all these functions. We can define an equivalence relation on $\mathcal{K}$ as follows: Let f_1 and f_2 be two elements of $\mathcal{K}$ which are defined for $x > x_1$ and $x > x_2$ respectively, then f_1 is said to be equivalent to f_2 if there exist an $x_0 > \max(x_1, x_2)$ so that $f_1(x) = f_2(x)$ for all $x > x_0$. The equivalence classes of $\mathcal{K}$ with respect to this equivalence relation are called the *germs of functions at* $+\infty$. We identify a germ of functions at $+\infty$ with any representative member, and we will thus refer to germs *as* functions. The derivative of a germ f is the equivalence class formed by the derivative of any element of g. It is obvious to see that this definition of the derivative of germs is well-defined, that is, that the derivatives of two functions in the same germ are also in the same germ.

Definition 3.1 (Hardy field [10, p. V.36]) *A Hardy field is a set of germs of real-valued functions on positive half lines in* $\mathbb{R}$ *that is closed under differ-*

entiation and that forms a field under the usual addition and multiplication of germs.

If $\mathcal{H}$ is a Hardy field and $f \in \mathcal{H}^*$, i.e. f is a non-zero element of $\mathcal{H}$, then $\mathcal{H}$ contains an element $1/f$. This implies that $f(x) \neq 0$ for $x \in \mathbb{R}$ sufficiently large. Since $f' \in \mathcal{H}$, f is differentiable for x sufficiently large, therefore continuous, and thus f is either always positive or always negative or zero. The same holds for $f' \in \mathcal{H}$ hence each f is ultimately monotonic. As a consequence, for each $f \in \mathcal{H}$, $\lim_{x\to+\infty} f(x)$ exists and is either a finite constant in $\mathbb{R}$ or $\pm\infty$.

Examples of simple Hardy fields are $\mathbb{Q}$ and $\mathbb{R}$, i.e., function fields where all functions are constants and where the derivatives of all functions are the constant function zero. The following theorems allow us to extend a given Hardy field. The proof of the second theorem is due to M. Singer and can be found in [67].

Theorem 3.2 (Robinson [66]) *Let $\mathcal{H}$ be a Hardy field, $f \in \mathcal{H}[x]$ with $f \neq 0$. Let y be the germ of a continuous, real-valued function on a positive half line such that $f(y) = 0$. Then $\mathcal{H}(y)$ is a Hardy field.*

Theorem 3.3 (Singer) *Let $\mathcal{H}$ be a Hardy field, $f, g \in \mathcal{H}[x]$ with $g \neq 0$. Let y be the germ of a differentiable, real-valued function on a positive half line such that $y' = f(y)/g(y)$. Then $\mathcal{H}(y)$ is a Hardy field.*

As a consequence, $\mathbb{R}(x)$ $(x' = 1)$ is a Hardy field, and exponentials $(y' = y\,e')$ and logarithms $(y' = e'/e)$ of elements can be added to a Hardy field as well. The field which is obtained from $\mathbb{R}(x)$ by closing it under the operations $f \to \exp(f)$ and $f \to \log|f|$ is called $\mathcal{L}$-field (or logarithmico-exponential field or field of exp-log functions for short) and was investigated by Hardy himself [36]. Since $a^b = e^{b \ln a}$, $\mathcal{L}$ also contains real powers of positive elements. Hardy proved that every $\mathcal{L}$ function is ultimately continuous, of constant sign, monotonic, and tends to $\pm\infty$ or to a finite real constant as $x \to +\infty$ [36, p. 18].

3.1.1 Valuation

Let $a, b \in \mathcal{H}^*$, then we write $a \asymp b$ if $a(x)/b(x)$ tends to a non-zero, finite limit. The relation $\asymp$ is an equivalence relation on $\mathcal{H}^*$. Let us denote the equivalence class of $a \in \mathcal{H}^*$ as $\nu(a)$ and the set of all equivalence classes of $\mathcal{H}^*$ as $\Upsilon = \{\nu(a) \mid a \in \mathcal{H}^*\}$. Υ is an Abelian group under the multiplicative operation inherited from $\mathcal{H}^*$. Furthermore, the set Υ is totally ordered due to the relation $\nu(a) > \nu(b)$ if $\lim_{x\to+\infty} a(x)/b(x) = 0$. This observation is summarized in the following theorem:

Theorem 3.4 (Rosenlicht [67, Theorem 4]) *Let $\mathcal{H}$ be a Hardy field. Then there exists a homomorphism (a canonical valuation) ν from $\mathcal{H}^*$ onto an ordered Abelian group with $\nu(1) = 0$, such that*

(1) if $a, b \in \mathcal{H}^*$, then $\nu(ab) = \nu(a) + \nu(b)$;

(2) if $a \in \mathcal{H}^*$, then $\nu(a) \geq 0$ if and only if $\lim_{x\to+\infty} a(x) \in \mathbb{R}$;

(3) if $a, b \in \mathcal{H}^*$ and $a+b \in \mathcal{H}^*$, then $\nu(a+b) \geq \min(\nu(a), \nu(b))$ with equality if $\nu(a) \neq \nu(b)$;

(4) if $a, b \in \mathcal{H}^*$ and $\nu(a), \nu(b) \neq 0$, then $\nu(a) \geq \nu(b)$ if and only if $\nu(a') \geq \nu(b')$;

(5) if $a, b \in \mathcal{H}^*$ and $\nu(a) > \nu(b) \neq 0$, then $\nu(a') > \nu(b')$.

From (1) follows directly that for any $a \in \mathcal{H}^*$, $\nu(a^{-1}) = -\nu(a)$, and together with (2) we see that $\nu(a) < 0$ if and only if $a(x)$ tends to $\pm\infty$, that $\nu(a) > 0$ if and only if $a(x)$ tends to zero, and that $\nu(a) = 0$ if $a(x)$ tends to a non-zero finite limit as $x \to +\infty$. Furthermore, (3) can be extended to the whole $\mathcal{H}$ if we define $\nu(0) = +\infty$. Note that (4) follows directly from l'Hôpital's rule (cf. Lemma 2.8) for $x \to +\infty$.

3.1.2 Comparability Classes and Rank

We can now define the measure of growth we will use in our algorithm. It is the notion of a comparability class as it was introduced in [68]. Nonzero elements α and β of an ordered Abelian group are called *comparable* if there exist positive integers m, n so that $m|\alpha| > \beta$ and $n|\beta| > \alpha$. This relation defines an equivalence relation on the set of non-zero elements of a ordered Abelian group.

Let us now apply this definition on the Abelian group Υ. Two functions in $\mathcal{H}^*$ are comparable if $m|\nu(f)| > |\nu(g)|$ and $n|\nu(g)| > |\nu(f)|$. Translated back to the Hardy field $\mathcal{H}$ we get the definition that two non-zero elements f and g of $\mathcal{H}$ with $f, g \to +\infty$ are called comparable if there exist positive integers m and n so that $\nu(f^m/g) < 0$ and $\nu(g^n/f) < 0$, i.e., $\lim_{x\to+\infty} f(x)^m/g(x) = +\infty$ and $\lim_{x\to+\infty} g(x)^n/f(x) = +\infty$. In other words, f and g have both to be bounded above and below by suitable integral powers of the other. For this translation we used the fact that $\nu(f^m) = m\nu(f)$ and $\nu(g^n) = n\nu(g)$ according to Theorem 3.4 (1). This definition may be extended to the whole $\mathcal{H}^*$ by specifying firstly that $\pm f$ and $\pm 1/f$ are in the same comparability class, and secondly, that all a with $\nu(a) = 0$ form their own comparability class (following the convention of Shackell [79]). The comparability class of f is denoted by $\gamma(f)$. If f and g both tend to $+\infty$, then we call $\gamma(f)$ *greater* *than* $\gamma(g)$ if f is greater than any power of g, i.e. $\nu(f) < p\,\nu(g)$ for all $p \in \mathbb{N}$. We additionally specify that $\gamma(1)$ is the lowest comparability class. Thus γ is defined on the whole $\mathcal{H}^*$ and the comparability classes are totally ordered.

We also use the notation $f \succ g$ for $\gamma(f) > \gamma(g)$ and we will say "f is more rapidly varying than g". Furthermore we write $f \asymp g$ if f and g are in the same comparability class, and $f \preceq g$ if $\gamma(f) \leq \gamma(g)$.

Examples:

$$\begin{array}{rcl}
e^x & \succ & x^m \\
e^{x^2} & \succ & (e^x)^2 \\
e^x & \asymp & e^{x+e^{-x}} \\
e^{e^x} & \succ & e^{x+e^{-e^x}} \\
e^{x\ln x \ln\left(x e^x - x^2\right)} & \prec & \ln\left(x^2 + 2\,e^{e^{3\,x^3\,\ln x}}\right) \\
e^{x\ln x \left(\ln\left(x e^x - x^2\right)\right)^2} & \asymp & \ln\left(x^2 + 2\,e^{e^{3\,x^3\,\ln x}}\right)
\end{array}$$

The number of different comparability classes of $\mathcal{H}^*$ minus one is called the *rank* of $\mathcal{H}$. For example, the Hardy field $\mathbb{R}(x, \ln x, e^x)$ has rank 3. The comparability classes are $\gamma(1)$, $\gamma(x)$, $\gamma(\ln x)$ and $\gamma(e^x)$.

We show next how the valuation and the comparability classes are related to each other. The following lemma is an extended version of Proposition 4 in [68].

Theorem 3.5 (Shackell [79, Lemma 1]) *Let $\mathcal{H}$ be a Hardy field, $a, b \in \mathcal{H}^*$ with $\nu(a), \nu(b) \neq 0$. Then*

(1) $\nu(a'/a) = \nu(b'/b)$ *if and only if* $\gamma(a) = \gamma(b)$;

(2) $\nu(a'/a) > \nu(b'/b)$ *if and only if* $\gamma(a) < \gamma(b)$;

(3) $\nu(a'/a) \geq \nu(b'/b)$ *if and only if* $\gamma(a) \leq \gamma(b)$.

To be precise, (3) appeared as Proposition 1 in [69] and is an obvious consequence of (1) and (2).

The above theorem together with Theorem 3.4 (4) and (5) gives us a nice way to compare the comparability classes of two functions.

Lemma 3.6 *Let $\mathcal{H}$ be a Hardy field with $\mathbb{R} \subset \mathcal{H}$ and $f, g \in \mathcal{H}^*$ with $\nu(f), \nu(g) \neq 0$. Then*

(1) $f \prec g$ *if and only if* $\lim\limits_{x\to+\infty} \frac{\ln|f(x)|}{\ln|g(x)|} = 0$;

(2) $f \preceq g$ *if and only if* $\lim_{x\to+\infty} \frac{\ln|f(x)|}{\ln|g(x)|} \in \mathbb{R}$;

(3) $f \asymp g$ *if and only if* $\lim_{x\to+\infty} \frac{\ln|f(x)|}{\ln|g(x)|} \in \mathbb{R}^*$.

Proof. The result follows directly as a consequence of Theorems 3.5 and 3.4 (4) and (5). For example for (2) we have $f \preceq g$ if and only if $\nu(f'/f) \geq \nu(g'/g)$ if and only if $\nu(\ln|f|) \geq \nu(\ln|g|)$ which holds by definition if $\lim_{x\to+\infty} \ln|f(x)|/\ln|g(x)| \in \mathbb{R}$.

For clarification purposes we state a direct proof for (1). Let us assume that both f and g ultimately tend to $+\infty$. $f \prec g$ holds if and only if $\nu(g) < \nu(f^p)$ for all $p \in \mathbb{N}$. By definition this holds if $\lim_{x\to+\infty} f(x)^p/g(x) = \lim_{x\to+\infty} e^{p\ln(f(x))-\ln(g(x))} = 0$ and this is true if $\ln(g) - p\ln(f) = \ln(g)(1 - p\ln(f)/\ln(g))$ ultimately tends to $+\infty$. As $\ln(g)$ already ultimately tends to $+\infty$ only the sign of $\lim_{x\to+\infty} 1 - p\ln(f(x))/\ln(g(x))$ must be positive, i.e., the relation $\ln(f)/\ln(g) < 1/p$ must ultimatively be satisfied for all $p \in \mathbb{N}$. The latter condition however is only met if $\lim_{x\to+\infty} \ln(f(x))/\ln(g(x)) = 0$. □

Lemma 3.7 *Let $\mathcal{H}$ be a Hardy field, $f, g \in \mathcal{H}^*$ with $\nu(f), \nu(g) \neq 0$, then*

$$\nu(f) = \nu(g) \Rightarrow \gamma(f) = \gamma(g).$$

Proof. From Theorem 3.4 (4) follows that $\nu(f) = \nu(g)$ if and only if $\nu(f') = \nu(g')$. Thus, $\nu(f) = \nu(g)$ implies $\nu(f') - \nu(f) = \nu(g') - \nu(g)$ which is equivalent to $\nu(f'/f) = \nu(g'/g)$ according to Theorem 3.4 (1). The latter equation is true if and only if $\gamma(f) = \gamma(g)$ as stated in Lemma 3.5. □

Lemma 3.8 *Let $\mathcal{H}$ be a Hardy field, $f, g \in \mathcal{H}^*$ with $\nu(f), \nu(g) \neq 0$ and $f'/f + g'/g \in \mathcal{H}^*$. Then*

$$\gamma(f\,g) \leq \max(\gamma(f), \gamma(g))$$

with equality if $\gamma(f) \neq \gamma(g)$.

Proof. Theorem 3.5 implies that $\gamma(f\,g) \leq \max(\gamma(f), \gamma(g))$ if and only if $\nu(\frac{f'g+fg'}{f\,g}) = \nu(f'/f + g'/g) \geq \min(\nu(f'/f), \nu(g'/g))$ and the latter relation holds according to Theorem 3.4 (3), with equality if $\nu(f'/f) \neq \nu(g'/g)$, which is equivalent to $\gamma(f) \neq \gamma(g)$. □

The next lemma is well known from calculus and can be proven using l'Hôpital's rule. We prove it here in the context of Hardy fields using the theorems stated above.

Lemma 3.9 *Let $\mathcal{H}$ be a Hardy field, $f, g \in \mathcal{H}^*$ with $\nu(f) \neq 0$ and $\nu(g) < 0$, then*

(1) $\gamma(\ln|f|) < \gamma(f)$, *and*

(2) $\gamma(e^g) > \gamma(g)$,

where $\ln|f|$ *and* e^g *are in an extension field* $\mathcal{H}_1 \supset \mathcal{H}$.

Proof. (1) According to Theorems 3.5 and 3.4, $\gamma(f) > \gamma(\ln|f|)$ if and only if $\nu(f'/f) < \nu((f'/f)/\ln|f|)) = \nu(f'/f) + \nu(1/\ln|f|)$, thus we have to show that $\nu(1/\ln|f|) > 0$, but this holds since $\nu(f) \neq 0$.

(2) Similarly we have $\gamma(g) < \gamma(e^g)$ if and only if $\nu(g'/g) > \nu(e^g\, g'/e^g) = \nu(g')$, and the latter is true if $\nu(1/g) > 0$, but that is what the precondition $\nu(g) < 0$ asserts. □

In the subsequent sections we will restrict our attention to the field of exp-log functions. An extension of the algorithm to other Hardy fields is discussed in Chapter 5.

3.2 Zero Equivalence

In order to compare two comparability classes we must be able to decide whether $\nu(f) > \nu(g)$. This problem is also called the dominance problem [60]. Dahn & Göring [20] have shown that for exp-log functions this problem is Turing reducible to the problem of deciding zero equivalence for exp-log constants. However, they did not give an algorithm for this reduction. That is what Shackell has done in [77] and what we will do in the next section.

There is a somewhat simpler problem than the dominance problem, namely the problem of deciding whether a given exp-log function is identically zero. This problem is called the identity problem, and it is also Turing reducible to the identity problem of exp-log constants.

Hardy showed in [36], that a non-zero exp-log function has only a finite number of real zeros. Richardson [60] and Macintyre [49] showed how to bound the number of zeros. If the function is zero at more points than the bound allows, then it must be the zero function. However, this algorithm is not a very practical one since firstly the bound may be rather large and secondly the methods involve differentiating the given expression to a high order, which can cause the size of the expression to grow rapidly [16].

Another perhaps more practical approach, which is based on the use of the structure theorems of Risch [65, 12], is to determine all algebraic dependencies between the basic functions making up the given function. The problem is thus reduced to the algebraic case. However, the structure theorem computations are also quite difficult. A similar approach based on differential algebra methods has been presented by Shackell [76, 81].

Thus, for solving either the dominance problem or the identity problem for exp-log functions, zero equivalence must be decidable for exp-log constants. Unfortunately, for the identity problem of exp-log constants itself, no solution is actually known at the present time. The zero equivalence of exp-log constants is decidable if Schanuel's conjecture is true. D. Richardson [62, 63] presented an algorithm which solves this problem and which (eventually) terminates, unless it is working on a counter example to Schanuel's conjecture.

As a consequence, we assume that the constants can somehow be handled. In particular, we postulate the existence of an oracle which can determine the sign of exp-log functions. Note that if the function is non-zero, then its sign can be determined in practice by successive approximations [60], e.g., using interval arithmetic.

3.3 The *MrvLimit* Algorithm

We have the material available to present an outline of our algorithm now. Let $f \in \mathcal{L}$ be an exp-log function whose (one sided) limit is to be computed. We assume that we can always access the functions in $\mathcal{L}$ in the form of an expression tree, i.e., a tree whose leaves are either x or are elements of $\mathbb{Q}$, and whose nodes are labeled with rational operations or with the functions exp or log. Note that expressions and functions are not equivalent, since every function may be represented by many different expression trees, in the context of a computer algebra system however, every function is normally given as one particular expression. By some abuse of notation we speak of an expression when we mean the function it represents.

In order to compute the limit of f we first look at all the subexpressions (subnodes) of the expression tree of f. We then determine those in the greatest comparability class and call them the set Ω of most rapidly varying subexpressions. Let ω be an exp-log function which is in the same comparability class as the elements in Ω and let us assume that $\omega > 0$ and that ω tends to 0. We rewrite all the elements in Ω in terms of ω and other expressions of lower order. The expression f can then be rewritten so that all subexpressions except ω are in a lower comparability class than ω itself.

The rewritten function is then expanded as a series in ω around $\omega = 0^+$. This series expansion has the form

$$c_0\,\omega^{e_0} + c_1\,\omega^{e_1} + \cdots + O\left(\omega^{e_n}\right), \tag{3.1}$$

where for all i we have $e_i \in \mathcal{C} \subseteq \mathbb{R}$ and $e_i < e_{i+1}$, and the most rapidly varying subexpression of every c_i is in a lower comparability class than ω.

Moreover, the leading coefficient c_0 must not be 0. $\mathcal{C}$ is the constant field of the function class under consideration, in our case $\mathcal{C} = \text{const}(\mathcal{L})$ is the set of exp-log constants.

Once we have found the series approximation (3.1), we can use the same arguments as in the power series approach (Section 2.3.2). If the leading exponent $e_0 > 0$ then the limit of f is 0 (remember that $\omega \to 0$). If $e_0 < 0$ the limit is $\pm\infty$ where the sign depends on the sign of c_0. If $e_0 = 0$ finally, then the limit of f is equivalent to the limit of the leading coefficient c_0. In this case the algorithm is applied recursively to c_0 (unlike in the power series approach where the leading coefficient is a constant).

During the process of the series expansion (and not only at the end of it) we must assert, in some particular situations, that the leading coefficient of the series expansion of a subexpression is not zero. Otherwise wrong results may emerge (see Section 7.2 for examples). This is one situation where the postulated oracle for testing zero equivalence is used in our algorithm. Furthermore, the oracle is used to compare elements in $\mathcal{C}$.

Let us recall the particular steps of our algorithm to compute the limit of $f(x)$ as x tends to x_0.

(1) Determine the set Ω of the most rapidly varying subexpressions of $f(x)$ (see Section 3.3.1). Limits may have to be computed recursively in this step (cf. Lemma 3.6).

(2) Choose an expression ω which is positive and tends to zero and which is in the same comparability class as any element of Ω. Such an element always exists. Rewrite the other expressions in Ω as $A(x)\,\omega^c$ where $A(x)$ only contains subexpressions which are in lower comparability classes (Section 3.3.2).

(3) Let $f(\omega)$ be the function which is obtained from $f(x)$ by replacing all elements of Ω by their representation in terms of ω. Consider all expressions independent of ω as constants and compute the leading term of the power series of $f(\omega)$ around $\omega = 0^+$ (see Section 3.3.3).

(4) If the leading exponent $e_0 > 0$ then the limit is 0 and we can stop. If the leading exponent $e_0 < 0$ then the limit is $\pm\infty$. If we only have to solve the dominance problem, we can stop in this situation as well. The sign is defined by the sign of the leading coefficient c_0, which can be computed in a similar manner. If the leading exponent $e_0 = 0$ then the limit is the limit of the leading coefficient c_0. If $c_0 \notin \mathcal{C}$ we must apply the same algorithm recursively on c_0.

In the following sections we describe these steps of the algorithm in more detail. When executing step (2), new expressions may be generated whose comparability class do not appear in the set of the comparability classes of all the subexpressions of $f(x)$. Furthermore, the limit facility is used recursively at several places in the algorithm. In Section 3.4 we will prove that the algorithm does terminate nevertheless.

3.3.1 Computing the Most Rapidly Varying Subexpressions

In this section we show how to determine the set of most rapidly varying subexpressions of a given function $f(x)$. This set is denoted by $mrv(f(x))$. If $f(x)$ does not depend on x at all, then we set $mrv(f(x)) = \{\}$. The relation "to be a subexpression" will be used rather often and thus we define the following notation: If $h(x)$ is a subexpression of $g(x)$ we write $h(x) \lhd g(x)$.

Definition 3.10 (mrv-set)

$$mrv(f(x)) = \begin{cases} \{\} & \textit{if } x \not\lhd \text{ f(x)} \\ \Big\{g(x) \mid g(x) \lhd f(x) \wedge \Big(\nexists\, h(x) \lhd f(x) : h(x) \succ g(x)\Big)\Big\} \end{cases}$$

As the *mrv* set depends on the form of the expression which represents the function $f(x)$, mathematically equivalent expressions may have different *mrv* sets.

Definition 3.10 implies that all the elements in $\Omega = mrv(f(x))$ are in the same comparability class. Let $g_1, g_2 \in \Omega$ be two subexpressions of f. $g_1 \in \Omega$ implies that $\nexists\, h \lhd f$ with $h \succ g_1$, thus $g_2 \preceq g_1$. We can similarly conclude $g_1 \preceq g_2$ if we interchange g_1 and g_2, therefore $g_1(x) \asymp g_2(x)$.

As all elements in a *mrv* set are in the same equivalence class, we allow the notation $mrv(f(x)) \asymp g(x)$ and mean that $g(x)$ is in the same equivalence class as any element of $mrv(f(x))$, provided that the latter is not empty, or that otherwise $g(x) \in \mathcal{C}$. The next observation also follows directly from Definition 3.10.

Fact 3.11 *Let $f(x)$ be a function. Then*

$$\forall\, g(x) \lhd f(x) \quad \Longrightarrow \quad mrv(f(x)) \succeq g(x).$$

Since $f(x) \lhd f(x)$ it follows that $mrv(f(x)) \succeq f(x)$.

In order to determine the set of most rapidly varying subexpressions of $f(x)$ we must look at all subexpressions of $f(x)$ and pick up those in the highest comparability class. As the comparability class of a product cannot be greater than the classes of its factors, it is enough to investigate only the factors of a product. Even if $\gamma(ab) = \gamma(a) = \gamma(b)$ it is enough only to record a and b in the *mrv* set, since if we rewrite both a and b in terms of ω, the product ab gets rewritten as well. For a sum the story is slightly more complicated, but it turns out, that $\gamma(a+b) \leq \max(\gamma(mrv(a)), \gamma(mrv(b)))$ and thus it is also sufficient only to look at the terms of a sum. This leads to Algorithm 3.12 for computing the set of most rapidly varying subexpressions of a given function f.

Algorithm 3.12 Computing the *mrv* set of f

```
mrv($f$ : exp-log function in $x$)
    if   $x \ntrianglelefteq f$          → RETURN( $\{\}$ )
    elif $f = x$           → RETURN( $\{x\}$ )
    elif $f = g \cdot h$       → RETURN( $\max(mrv(g), mrv(h))$ )
    elif $f = g + h$       → RETURN( $\max(mrv(g), mrv(h))$ )
    elif $f = g^c \wedge c \in \mathcal{C}$   → RETURN( $mrv(g)$ )
    elif $f = \ln g$        → RETURN( $mrv(g)$ )
    elif $f = e^g$          →
        if   $\lim_{x\to+\infty} g = \pm\infty$ → RETURN( $\max(\{e^g\}, mrv(g))$ )
        else                     RETURN( $mrv(g)$ )
        fi
    fi
```

The function max() computes the maximum of two sets of expressions which are in the same comparability class, i.e. max() compares (two elements of) its argument sets and returns the set which is in the higher comparability class or the union of both, if they have the same order of variation.

The rule for $mrv(\ln g(x))$ is in accordance with Lemma 3.9. If $\nu(g(x)) \neq 0$ then $\ln g(x) \prec g(x)$ and thus $mrv(\ln g(x)) = mrv(g(x))$. Otherwise, if $\nu(g(x)) = 0$ then the limit of $\ln g(x)$ is finite as well and $\gamma(g(x)) = \gamma(\ln g(x)) = \gamma(1)$ and both $g(x)$ and $\ln g(x)$ will never appear in any *mrv* set. It is hence enough to search for the most rapidly varying subexpression within $g(x)$.

The case where the argument is an exponential $e^{g(x)}$ is the only difficult one. If $\nu(g(x)) \geq 0$ then $\gamma(e^{g(x)}) = \gamma(1)$ and $mrv(e^{g(x)}) = mrv(g(x))$. If however $\nu(g(x)) < 0$, then $\gamma(e^{g(x)}) > \gamma(g(x))$ (cf. Lemma 3.9 (2)) and $mrv(e^{g(x)}) = \max(\{e^{g(x)}\}, mrv(g(x)))$, i.e. it is either $\{e^{g(x)}\}$ or $mrv(g(x))$ or the union of both. The following examples illustrate these three possibilities:

$$\begin{aligned}
mrv\left(e^{x+1/x} \right) &= \{e^{x+1/x}\} \\
mrv\left(e^{x+e^{-e^x}} \right) &= \{e^{-e^x}\} \\
mrv\left(e^{x+e^{-x}} \right) &= \{e^{x+e^{-x}}, e^x\}.
\end{aligned}$$

For the computation of the *mrv* set of $e^{g(x)}$ we have to determine the limiting behaviour of the argument $g(x)$. This is a recursive call to the limit facility. However, the algorithm does not enter into an infinite loop, since $g(x)$ is a *smaller* expression than $e^{g(x)}$, which by itself is a subexpression of the expression whose limit is currently being computed. Thus the size of an expression (e.g., height of the expression tree) is an upper bound for the number of iterations.

Example 3.13 As an example we compute the *mrv* set of $f = e^{x+e^{-x^2}}$. The limit of $x + e^{-x^2}$ is $+\infty$ and we therefore must compare f with an element of $mrv(x + e^{-x^2})$. For the latter we get

$$\begin{aligned}
mrv\left(x + e^{-x^2}\right) &= \max\left(mrv(x), mrv\left(e^{-x^2}\right)\right) \\
&= \max\left(\{x\}, \{e^{-x^2}\}\right) = \left\{e^{-x^2}\right\}
\end{aligned}$$

where max refers to the relation $\prec$. The comparison of f with e^{-x^2} is done according to Lemma 3.6 by computing the limit of the quotient of the logarithms of the two functions,

$$\lim_{x\to+\infty} \frac{\ln e^{x+e^{-x^2}}}{\ln e^{-x^2}} = \lim_{x\to+\infty} \frac{x + e^{-x^2}}{-x^2} = \lim_{x\to+\infty} -\frac{1}{x} - \frac{e^{-x^2}}{x^2} = 0$$

and hence $mrv(f) = \{e^{-x^2}\}$.

Further examples:

$$\begin{aligned}
mrv\left(e^{\frac{1}{x}+e^{-x}}\right) &= \{e^{-x}\} \\
mrv\left(e^{x^2} + x\,e^x + \frac{\ln(x)^x}{x}\right) &= \{e^{x^2}\} \\
mrv\left(e^x\left(e^{\frac{1}{x}+e^{-x}} - e^{\frac{1}{x}}\right)\right) &= \{e^x, e^{-x}\} \\
mrv\left(\ln\left(x^2 + 2\,e^{e^{3\,x^3\,\ln x}}\right)\right) &= \{e^{e^{3\,x^3\,\ln x}}\} \\
mrv\left(\frac{\ln(x-\ln x)}{\ln x}\right) &= \{x\}
\end{aligned}$$

¶

From the structure of Algorithm 3.12 to compute the *mrv*-set we can deduce the following lemma:

Lemma 3.14 *Let $f(x)$ be an exp-log function with $x \lhd f(x)$ and let $\Omega = mrv(f(x))$ be the set of most rapidly varying subexpressions of $f(x)$. Then for every $g(x) \in \Omega$*

(1) $g(x) = x$ *or* $g(x) = e^{h(x)}$ *with* $h(x) \to \pm\infty$;

(2) *the sign of* $g(x)$ *is* 1, *i.e.* $g(x) > 0$;

(3) $\nu(g(x)) \neq 0$;

(4) $$\lim_{x\to+\infty} g(x) = \begin{cases} \infty & \text{if } g(x) = x \text{ or } g(x) = e^{h(x)} \wedge h(x) > 0 \\ 0 & \text{if } g(x) = e^{h(x)} \text{ and } h(x) < 0 \end{cases}$$

(5) $g(x) \succeq x$;

(6) *if* $g(x) = e^{h(x)}$ *then* $mrv(h(x)) \preceq g(x)$;

(7) $g(x) \in mrv(g(x)) \subseteq \Omega$.

Proof. Statement (1) follows directly from the structure of the algorithm. The only results which are returned from the procedure are either the empty set or sets which contain x or $e^{h(x)}$. Statements (2), (3) and (4) are simple consequences thereof. Statement (5) is a consequence of Fact 3.11 as $x \lhd f(x)$ implies $mrv(f(x))$ $(\asymp g(x)) \succeq x$. Statements (6) and (7) are consequences of the Definition 3.10 of the mrv set itself. □

In the outline of the algorithm we said that we choose an expression which is positive and tends to zero and which is in the same comparability class as $\Omega = mrv(f(x))$. We see now that such an element always exists. Let $g(x) \in \Omega$. If $g(x) = x$ then we can set $\omega = x^{-1}$, and if $g(x) = e^{h(x)}$ then we can set $\omega = g(x)$ if $h(x) < 0$ and $\omega = e^{-h(x)} \asymp \Omega$ otherwise. From Lemma 3.14 it follows that for such a choice of ω, $\omega > 0$ and $\omega \to 0$.

To complete the explanation of Algorithm 3.12 it remains to show how to compare two mrv sets, as this is needed to compute the maximum of two sets. Since all elements of a mrv set are in the same equivalence class, it is enough to compare single representatives of each set only. By Lemma 3.14 (3) we can simply apply Lemma 3.6 and compute the limit of the quotient of the logarithms of the two. We will prove in Section 3.4 that this recursive approach must terminate.

From Fact 3.11 it follows that $x \lhd f(x)$ implies $mrv(f(x)) \succeq x$. In other words, $\gamma(x)$ is the smallest comparability class which can be returned when computing $mrv(f(x))$, provided that $x \lhd f(x)$. However, if $x \in mrv(f(x))$ then it may happen that $f(x)$ does *not* have a power series expansion in $\omega = 1/x$ around $\omega = 0^+$. The simplest example for this is $f(x) = \ln x$. This issue could be resolved if we use a more general tool than power series for the series expansions in ω. One possibility would be to use *generalized power series* as defined in [27] in this situation. We will see in Section 3.3.4 how this issue is resolved in our algorithm.

Example 3.15 As we know now how to determine the *mrv* set of a function, we can demonstrate how the algorithm proceeds on a simple example which does only lead to *mrv* sets with one element which do not have to be rewritten and where the power series in ω can be computed. Let us compute $\lim_{x\to+\infty} f(x)$ for

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

In Chapter 8 we will see what some other computer algebra systems return on this problem (see example (8.1)).

The set of most rapidly varying subexpressions of $f(x)$ is $\{e^{-x}\}$ and we replace e^{-x} by ω and get $(e^{1/x-\omega} - e^{1/x})/\omega$. The series thereof around $\omega = 0$ is

$$-e^{1/x} \cdot \omega^0 + \frac{1}{2}\, e^{1/x} \cdot \omega^1 - \frac{1}{6}\, e^{1/x} \cdot \omega^2 + O(\omega^3).$$

The leading term is $-e^{1/x} \cdot \omega^0$ and the leading exponent is 0. Thus,

$$\lim_{x\to+\infty} f(x) = \lim_{x\to+\infty} -e^{1/x},$$

i.e. we have to apply the algorithm recursively to the leading coefficient $-e^{1/x}$. Although the result is obvious now, let us follow the steps of the algorithm. Next we compute $mrv(-e^{1/x}) = mrv(1/x) = \{x\}$ and set $\omega = 1/x$. The series of $-e^{\omega} = -1 - \omega + O(\omega^2)$ and thus $\lim_{x\to+\infty} f(x) = -1$. ¶

3.3.2 Rewriting Functions in the Same Comparability Class

If the set Ω of *most rapidly varying* subexpressions of a given expression u contains more than one element, then we must rewrite all of them in terms of a single one, ω, which is in the same comparability class than Ω. We have just seen that any element in Ω can be designated to be ω or $1/\omega$. Therefore it is enough to show that all elements in Ω can be rewritten in terms of a particular one in Ω. It turns out that this rewriting process is rather trivial in our context.

We show next how to rewrite f in terms of g where f and g are two elements in $\Omega = mrv(u)$. We assume that $x \notin \Omega$ and that both $f = e^s$ and $g = e^t$ are exponentials. The case where $x \in \Omega$ is discussed in Section 3.3.4. From Lemma 3.14 we know, that $f > 0$, $g > 0$, $f \in mrv(f)$ and $g \in mrv(g)$.

According to Lemma 3.6, $\lim_{x\to+\infty} s/t = c \in \mathbb{R}^*$ as $\gamma(f) = \gamma(g)$. As a consequence and since $f, g > 0$ we can rewrite f as $A \cdot g^c$ where

$$A = \frac{f}{g^c} = e^{\ln f - c\,\ln g} = e^{s - c\,t}. \tag{3.2}$$

When computing A we have to compute c as the result of another limit. We will show in Section 3.4 that these recursive limit calls also cannot lead to an infinite recursion.

In the next lemma we show that $A \prec g$. Unfortunately, as we will see in Example 3.17, the condition $A \prec g$ is not strong enough. What we really need is $mrv(A) \prec g$ in order to have the guarantee that $mrv(c_i) \prec \omega$ in the series expansion (3.1) of u in terms of ω.

Lemma 3.16 *Let f, g be two exponentials so that $f \asymp g$, $f, g > 0$ and let $A = e^{\ln f - c\,\ln g}$ where $c = \lim\limits_{x\to+\infty} \frac{\ln f}{\ln g}$. Then $A \prec g$.*

Proof. According to Lemma 3.6 we must show that $\lim\limits_{x\to+\infty} \frac{\ln A}{\ln g} = 0$:

$$\lim_{x\to+\infty} \frac{\ln A}{\ln g} = \lim_{x\to+\infty} \frac{\ln f - c \ln g}{\ln g} = \lim_{x\to+\infty} \frac{\ln f}{\ln g} - c = c - c = 0.$$

□

Example 3.17 Let us compute the limit of $u = 1/e^{-x+e^{-x}} - e^x$ as $x \to +\infty$. The set of most rapidly varying subexpressions of u is

$$\Omega = mrv(u) = mrv(\frac{1}{e^{-x+e^{-x}}} - e^x) = \left\{e^{-x+e^{-x}}, e^x, e^{-x}\right\}.$$

Let us choose $\omega = e^{-x+e^{-x}}$ to be the representative of this equivalence class and let us rewrite e^{-x} in terms of ω. According to the rule (3.2) we get, with $f = e^{-x}$ and $g = e^{-x+e^{-x}}$,

$$A = e^{-x-(-x+e^{-x})} = e^{-e^{-x}}.$$

However, if we rewrite $f = e^{-x}$ in terms of $A\,g^1$, then f, the expression we want to eliminate, is reintroduced as a subexpression of A. It seems that this choice for ω is not very clever. From this observation we will derive a condition on ω so that the rewritten expression disappears completely and does not get reintroduced inadvertently.

Let us try next to rewrite the elements in Ω in terms of $\omega = e^{-x}$. The function e^x can simply be rewritten as $1/\omega$. In order to rewrite $e^{-x+e^{-x}}$ we set $f = e^{-x+e^{-x}}$ and $g = \omega = e^{-x}$ and again apply the rewrite rule with

$$A = e^{\ln f - \ln g} = e^{-x+e^{-x}+x} = e^{e^{-x}}.$$

Note that with this choice of ω, $f \not\triangleleft A$. If we replace in u every instance of f by $A\,\omega^1$, then we get

$$u = \frac{1}{A\,\omega} - \frac{1}{\omega} = \left(\frac{1}{e^{e^{-x}}} - 1\right)\omega^{-1} \tag{3.3}$$

which no longer contains f as a subexpression.

The expression (3.3) can be seen as the series expansion of u in terms of ω. This series, however, is not a proper power series. Although $\gamma(A) = \gamma(1) < \gamma(\omega)$, the leading coefficient $1/A - 1$ is in the same comparability class as ω, which can be shown by computing the limit of the quotient of the logarithms of $1/A - 1$ and ω (recursively using our algorithm with $\omega = e^{-x}$):

$$\begin{aligned}\lim_{x\to+\infty} \frac{\ln|1/A-1|}{\ln|\omega|} = \lim_{x\to+\infty} \frac{\ln(1-e^{-e^{-x}})}{-x} &= \lim_{x\to+\infty} \frac{\ln(e^{-x}\,(1+O(e^{-x})))}{-x} \\ = \lim_{x\to+\infty} \frac{-x+O(e^{-x})}{-x} &= 1 \in \mathbb{R}^*.\end{aligned}$$

As a consequence, the powers of ω don't necessarily dominate the coefficients of the series (3.3). The conclusion, that the limit of u is $-\infty$ as $x \to +\infty$ according to the negative leading exponent and to the negative sign of the leading coefficient is therefore wrong.

The series does not meet the conditions stated in equation (3.1) since

$$mrv(1/A - 1) = mrv(A) = \{e^{-x}\} \asymp \omega.$$

However, if we rewrite A in terms of ω as well, then we finally get the series

$$u = \frac{1}{\omega\,e^{\omega}} - \frac{1}{\omega} = -1 + \frac{1}{2}\,\omega + O(\omega^2)$$

and the correct result, which is -1. ¶

In the above example we met two problems. First, it may happen due to a unlucky choice of ω that the rewritten expressions recurs inside of A. Furthermore, it is not guaranteed that $mrv(A) \prec \omega$ and thus further rewriting may be necessary. The source of both problems is that for two expressions in the mrv-set, one is a subexpression of the other.

In the next lemma we show under which conditions on f and g we can rewrite f in terms of g such that $mrv(A) \prec g$. The problems illustrated above cannot occur if these conditions are met.

Lemma 3.18 *Let u be a function such that $x \notin \Omega = mrv(u)$, let $f = e^s$ and $g = e^t$ be in Ω, and let $A = e^{s-ct}$ with $c = \lim_{x\to+\infty} \frac{s}{t}$. Then*

$$(mrv(f) = \{f\}) \;\wedge\; (mrv(g) = \{g\}) \Rightarrow mrv(A) \prec g. \tag{3.4}$$

Proof. If A is a constant the implication (3.4) obviously holds, so let us assume that A is an exponential. Furthermore, $mrv(f) = \{f\}$ and $mrv(g) = \{g\}$ holds if and only if $mrv(s) \prec f$ and $mrv(t) \prec g$, and thus the left hand side of (3.4) implies $\max(mrv(s), mrv(t)) \prec g$, and we see that

$$mrv(A) \subseteq \{A\} \cup mrv(s - ct).$$

Together with the facts $mrv(s - ct) \preceq \max(mrv(s), mrv(t))$ and $A \prec g$ (Lemma 3.16) we can conclude that $mrv(A) \prec g$. □

Note that the other direction does not hold. A counterexample for this is $f = e^{x+1/x+e^{-x}}$ and $g = e^{x+e^{-x}}$ where $A = e^{1/x}$ and thus $mrv(A) \prec g$ but $mrv(f) = \{f, e^{-x}\}$ and $mrv(g) = \{g, e^{-x}\}$. However, a slightly weaker form for the other direction is proven in the next lemma.

Lemma 3.19 *Let u be a function such that $x \notin \Omega = mrv(u)$ and let $f = e^s$ and $g = e^t$ be in Ω and $A = e^{s-ct}$ with $c = \lim_{s\to+\infty} \frac{s}{t}$. Then*

$$mrv(f) \neq \{f\} \wedge mrv(g) = \{g\} \Rightarrow mrv(A) = mrv(s) \asymp g.$$

Proof. $mrv(f) \neq \{f\}$ implies $mrv(s) \asymp g$ and $mrv(g) = \{g\}$ implies $mrv(t) \prec g$. As a consequence $mrv(s - ct) = mrv(s)$. Equality holds, since the most rapidly varying subexpressions in s cannot cancel with those in t. Together with Lemma 3.16 we get $mrv(A) = mrv(s)$. □

If we rewrite f in terms of g with $mrv(f) \neq \{f\}$ and $mrv(g) = \{g\}$ then $mrv(A) = mrv(s) = mrv(f)\backslash\{f\}$. This leads to the following strategy for rewriting all elements in Ω.

Let ω be an element of the set of all the expressions which do not have a subexpression in Ω, i.e. let $\omega \in \Omega$ with $mrv(\omega) = \{\omega\}$. We can then rewrite all elements in Ω in terms of ω. These elements are eliminated one by one. An element $f \in \Omega$ which contains a subexpression $g \in \Omega$ has to be rewritten before g is rewritten.

Let $u_0 = u$ and $\Omega_i = mrv(u_i)$. Then u_{i+1} is obtained from u_i by rewriting $f_i \in \Omega_i$ in terms of ω, where f_i is chosen such that for all $f \in \Omega_i$ with $f \neq f_i$, f_i is not a subexpression of f. This condition on f_i is equivalent to $|mrv(f_i)| = \max\left\{|mrv(f)| \,\middle|\, f \in \Omega_i\right\}$ which is easily tested in a program.

According to Lemma 3.18 and Lemma 3.19 we have $\Omega_{i+1} = \Omega_i \backslash \{f_i\} \cup \{\omega\}$. Eventually we obtain $\Omega_n = mrv(u_n) = \{\omega\}$ for $n = |\Omega|$ and the rewriting of u is complete.

Up to this point we have discarded the limiting behaviour of ω itself. Thus if it turns out that $\omega = e^h \to +\infty$, then we set $\omega = e^{-h}$ and substitute ω by $1/\omega$ in u_n. The following lemma concludes the recent observations.

Lemma 3.20 *Let $\Omega = mrv(u)$ with $x \notin \Omega$. Then we always can rewrite all elements in Ω in terms of $\omega = e^h$ where*

(1) ω or $1/\omega \in \Omega$ and $\omega \to 0$;

(2) $mrv(\omega) = \{\omega\}$, which implies that $mrv(h) \prec \Omega$.

Example 3.21 In this example we want to compute

$$\lim_{x \to +\infty} \frac{e^h \, e^{-\frac{x}{1+h}} \, e^{e^{-x+h}}}{h^2} - e^x + x, \tag{3.5}$$

where $h = e^{-x/(1+e^{-x})}$. The set of most rapidly varying subexpressions is

$$\Omega = \left\{ e^{-x+h}, e^{-\frac{x}{1+h}}, h, e^x, e^{-x} \right\}.$$

We choose $\omega = e^{-x}$ and then rewrite all the elements in Ω "from left to right", namely

$$\begin{aligned} f_1 = e^{-x+h} &= e^h \, \omega \\ f_2 = e^{-\frac{x}{1+h}} &= e^{x - x/(1+h)} \, \omega \\ f_3 = h &= e^{x - x/(1+e^{-x})} \, \omega \\ f_4 = e^x &= \omega^{-1} \\ f_5 = e^{-x} &= \omega \end{aligned}$$

and the starting expression is transformed into

$$u_5 = \frac{e^{\omega e^{x - x/(1+\omega)}} \, e^{x - x/(1+\omega e^{x-x/(1+\omega)})} \, e^{\omega e^{\omega e^{x - x/(1+\omega)}}}}{\left(e^{x - x/(1+\omega)}\right)^2 \omega} - 1/\omega + x.$$

We now have $mrv(u_5) = \{\omega\}$. The series of u_5 in ω is

$$2 + (3/2\, x^2 + 3)\, \omega + O(\omega^2)$$

and the result of (3.5) is 2. In Chapter 8, example (8.18) we will see how some other computer algebra systems behave on this problem. ¶

It is important to note that the exponential $A = e^{s-c\,t}$ must never be expanded into a product of exponentials, because this would reveal functions of higher classes, for example $e^s = f \asymp g$ and $e^{-c\,t} = g^{-c} \asymp g$. $mrv(e^s\,e^{-c\,t}) \asymp g$ and the important fact that $mrv(A) \prec g$ would be lost. Termination of the whole algorithm would also no longer be guaranteed. Consider, e.g., $f = e^{-x/(1+1/x)}$, which might be rewritten as $f = A\,e^{-x}$ with $A = e^{x-x/(1+1/x)}$. We have $mrv(A) \prec f$, but as soon as A is expanded, we get $A = e^x\,f$ and are back at square one. The argument $s - c\,t$ of the exponential however can be simplified without consequences.

3.3.3 Series Expansion

In this section we will prove that the power series in the most rapidly varying subexpression ω at $\omega = 0^+$ always exists, provided that $x \notin \Omega$. The treatment of the case $x \in \Omega$ is deferred to the next section. Let us assume that ω is an exponential and that we have rewritten the set Ω of most rapidly varying expressions according to Lemma 3.20 in terms of ω.

Theorem 3.22 *Let f be an exp-log function with $mrv(f) = \{\omega\}$, $\omega \to 0$ and $\omega = e^h$ with $mrv(h) \prec \omega$. Then the power series of f in ω at $\omega = 0^+$ exists and has the form $\sum_{i=0}^{\infty} c_i \omega^{e_i}$ with $mrv(c_i) \prec \omega$ and $e_i \in \mathcal{C}$.*

Proof. The proof is performed inductively over the expression tree of f. The conditions are obviously satisfied if f does not depend on ω or if $f = \omega$.

For the induction step we have to show that the sum or the product of two series s_1 and s_2 and the exponential, the logarithm and the inverse of a series s is also a series with coefficients whose mrv set is in a lower comparability class than ω, provided that the same condition holds for the series s, s_1 and s_2.

For the sum and the product of two series it is obvious that the series exists, since the new coefficients are built from the coefficients of the series of the arguments by means of multiplication and addition only. The condition on the coefficients is also preserved since $mrv(ab) \preceq \max(mrv(a), mrv(b))$ and $mrv(a+b) \preceq \max(mrv(a), mrv(b))$.

Next we consider the case of the series of the inverse of g in the case that the the series of g in ω exists. Let the latter one be

$$\mathrm{Series}(g, \omega) = c_0\,\omega^{e_0} + c_1\,\omega^{e_1} + \cdots.$$

The series of the inverse of g is given by

$$\mathrm{Series}(1/g, \omega) = \frac{1}{c_0}\,\omega^{-e_0} \sum_{k=0}^{\infty} (-1)^k \Phi^k$$

where $\Phi = c_1/c_0\, \omega^{e_1-e_0} + c_2/c_0\, \omega^{e_2-e_0} + \cdots$. This series exists, provided that the leading coefficient $c_0 \neq 0$. With the help of the oracle for deciding zero-equivalence this condition has to be asserted. The condition on the coefficients is also preserved since they are also generated from the coefficients c_i and $1/c_0$ by means of multiplication and addition only and since $mrv(1/c_0) = mrv(c_0)$.

Next we discuss the series expansion for e^g where the series of g exists. The leading term of the series of g must satisfy $e_0 \geq 0$. For, if $e_0 < 0$, g would tend to $\pm\infty$ as $\omega \to 0$, i.e. as $x \to +\infty$, and thus $mrv(e^g) = max(\{e^g\}, mrv(g))$. $mrv(g)$ is obviously ω, so let us compare e^g with ω by computing the limit of the quotient of their logarithms.

$$\lim_{x\to+\infty} \frac{g}{\ln\omega} = \lim_{x\to+\infty} \lim_{\omega\to 0+} \frac{c_0}{h}\omega^{e_0} + \frac{c_1}{h}\omega^{e_1} + \cdots = \pm\infty$$

since $mrv(h) \prec \omega$ according to the hypothesis of the theorem and $mrv(c_i) \prec \omega$ according to the induction hypothesis, and we would have $e^g \succ \omega$, which is a contradiction.

If $e_0 > 0$ then the series of e^g is given by

$$\text{Series}(e^g, \omega) = \sum_{k=0}^{\infty} \frac{\text{Series}(g, w)^k}{k!}$$

and the problem is reduced to addition and multiplication of series. For the case $e_0 = 0$ we get

$$\text{Series}(e^g, \omega) = e^{c_0} \sum_{k=0}^{\infty} \frac{\psi^k}{k!}$$

with $\psi = c_1\,\omega^{e_1} + c_2\,\omega^{e_2} + \cdots$. In order to establish the conditions on the coefficients we only have to show that $mrv(e^{c_0}) \prec \omega$. From $\lim_{x\to+\infty} \frac{g}{c_0} = \lim_{x\to+\infty} \lim_{\omega\to 0+} \frac{c_0+\psi}{c_0} = 1$ follows that $\gamma(e^{c_0}) = \gamma(e^g)$. Furthermore we know that $e^g \prec \omega$ as $mrv(f) = \{\omega\}$, i.e., e^g would otherwise have been rewritten as a power of ω. From this and the fact that $mrv(e^{c_0}) \subseteq \{e^{c_0}\} \cup mrv(c_0)$ it follows that $mrv(e^{c_0}) \prec \omega$. Again note, that the condition $mrv(e^{c_0}) \prec \omega$ only holds for the expression e^{c_0}, i.e. this exponential must also not be expanded. Expanding e^{c_0} may produce functions in higher comparability classes.

The last case is the series of $\ln g$.

$$\begin{aligned} \text{Series}(\ln(g), \omega) &= \text{Series}\left(\ln(c_0\,\omega^{e_0} + c_1\,\omega^{e_1} + \cdots), \omega\right) \\ &= \ln c_0 + e_0\, \ln\omega + \sum_{k=1}^{\infty} \frac{(-1)^{k-1}\,\Phi^k}{k} \end{aligned}$$

Due to the precondition, $\ln\omega$ simplifies to h and the above is a power series, provided that $c_0 \neq 0$. This latter condition must be asserted again with the help of the oracle.

Concerning the condition on the coefficients of this series we know from the induction hypothesis that $mrv(h) \prec \omega$ and from the definition of the set of the most rapidly varying subexpressions, it follows that $mrv(\ln c_0) = mrv(c_0) \prec \omega$, which completes the proof. □

3.3.4 Moving up in the Asymptotic Scale

In the previous sections we have always postulated that $x \notin \Omega$, because then all elements in Ω are exponentials (cf. Lemma 3.14) which simplifies both the rewriting and the series expansion step. Although the difficulties with the rewriting step for the case that $x \in \Omega$ could be resolved by setting $x = e^{\ln x}$, the real problem is that we are not able to distinguish the comparability classes below $\gamma(x)$. If we apply our algorithm to a function where the comparability classes of all the subexpressions are lower than $\gamma(x)$, then the leading coefficient of the series in x will always be identical to f. An example is $f = \ln x = \ln(e^{\ln x})$ where we get the series $\ln x\, \omega^0 = f$ if we would expand it in terms of $\omega = e^{-\ln x}$. In the following we show how this problem is treated by our algorithm.

The idea is that we move up one level in the asymptotic scale, i.e., we go from x to e^x. This is based on the following lemma:

Lemma 3.23 *Let $f(x)$ and $g(x)$ be two exp-log functions and let $\lim_{x \to +\infty} f(x) = \alpha$ and $\lim_{x \to +\infty} g(x) = +\infty$ then*

$$\lim_{x \to +\infty} f(g(x)) = \alpha.$$

Proof. The statement follows at once from the theorem on the continuity of composite continuous functions and from the fact that exp-log functions are ultimately continuous for $x \to +\infty$. □

In our case we choose $g(x) = e^x$. If we replace x by e^x in the given expression $f(x)$, the limit remains the same. However, if we want to use our algorithm on the transformed expression, we need to show that the ordering of the comparability classes is not changed.

Lemma 3.24 *Let $f(x)$, $g(x)$ and $v(x)$ be exp-log functions with $\lim_{x \to +\infty} v(x) = +\infty$. Then,*

$$f(x) \prec g(x) \Rightarrow f(v(x)) \prec g(v(x)).$$

Proof. Since $f(x) \prec g(x)$ we know that $\lim_{x \to +\infty} \frac{\ln f(x)}{\ln g(x)} = 0$. The quotient $\frac{\ln f(x)}{\ln g(x)}$ itself is also an exp-log function, which we name $u(x)$. Since both $u(x)$

and $v(x)$ are exp-log functions, with $\lim_{x\to\infty} v(x) = \infty$, we can apply Lemma 3.23 to get

$$\lim_{x\to+\infty} \frac{\ln f(v(x))}{\ln g(v(x))} = \lim_{x\to+\infty} u(v(x)) = \lim_{x\to+\infty} u(x) = 0$$

which is equivalent to $f(v(x)) \prec g(v(x))$. □

If we replace x by e^x the expression may be simplified as $\ln x$ gets transformed to $\ln e^x = x$. It is not necessary to perform this simplification, as it will be performed automatically during the series expansion. We would then have $\omega = e^{-x}$ and the series expansion of $\ln e^x = \ln(1/\omega)$ would become x as well.

We repeat this substitution procedure until we eventually get an expression f_n which contains e^x as a subexpression. Then $e^x \in \Omega_n = mrv(f_n)$ and hence we can apply our algorithm, i.e. we know how to rewrite the elements in Ω_n (Lemma 3.20) and the power series of f_n in $\omega = e^{-x}$ exists (Theorem 3.22), with coefficients in a lower comparability class than e^x.

Example 3.25 Let

$$f = \frac{\ln(\ln x + \ln\ln x) - \ln\ln x}{\ln(\ln x + \ln\ln\ln x)} \ln x$$

and let us compute $\lim_{x\to+\infty} f(x)$. $mrv(f) = \{x\}$ and we move up one level in the scale. We get

$$f_1 = \frac{\ln(x + \ln x) - \ln x}{\ln(x + \ln\ln x)} x.$$

Again $mrv(f_1) = \{x\}$ and thus we move up a further level and obtain

$$f_2 = \frac{\ln(e^x + x) - x}{\ln(e^x + \ln x)} e^x.$$

Now the most rapidly varying subexpression is $mrv(f_2) = \{e^x\}$. We set $\omega = e^{-x}$ and rewrite f_2 as

$$\frac{\ln(\omega^{-1} + x) - x}{\ln(\omega^{-1} + \ln x)} \omega^{-1}$$

whose power series at $\omega = 0^+$ is

$$f_2 = 1 - \frac{x^2 + 2\ln x}{2x}\omega + \frac{2x^4 + 3\ln(x)x^2 + 3\ln(x)^2 x + 6\ln(x)^2}{6x^2}\omega^2 + O(\omega^3).$$

Thus, $\lim_{x\to+\infty} f = 1$. ¶

3.4 Proof of Termination

In every iteration step of the algorithm one comparability class is eliminated. However, due to rewriting and to the series expansions *new* expressions which may form *new* comparability classes may be introduced, and termination is therefore not obvious. Additionally, the limit procedure is used recursively to compare two *mrv* sets and it must be shown, that the algorithm does not enter into an infinite loop.

In the next section we will show that the global iteration will terminate provided that the *mrv* set can always be computed, i.e. provided that the recursive calls will not lead to an infinite recursion. The verification of the validity of this assumption is given in Section 3.4.2.

3.4.1 Global Iteration

In every iteration step of the algorithm, the largest comparability class is eliminated. To prove that this process is monotone, i.e., that it finally leads to an expression whose *mrv* set is empty, we define the size of an expression which bounds the number of comparability classes which may ever evolve during the limit computation process. The size of a function f is an upper bound the number of iterations which the algorithm will perform at most to compute the limit of f as well as for the rank of a Hardy field which contains f.

The size of a function f is defined to be the cardinality of the set Ω which contains all the possible candidates of *mrv* expressions and moreover all the *active* exponentials and logarithms, i.e. exponentials and logarithms whose argument depend on x:

$$\mathrm{Size}(f(x)) = \Big|S(f(x))\Big|.$$

The definition of the set S is similar to the definition of the MAPLE function *indets*. Algorithm 3.26 shows how to compute the set S of a given exp-log function f. This procedure is very similar to Algorithm 3.12 presented on page 39.

The size of an expression is always integral and nonnegative. If it is zero, then the expression does not depend on x and is constant. The size of an expression may therefore be used as a *variant function* to prove termination of the global iteration. In every iteration step ω, one of the most rapidly varying subexpressions, is eliminated and cannot appear as a subexpression of the leading coefficient which is followed up. As ω is an active exponential, the size of the leading coefficient is smaller, provided that the series expansion

Algorithm 3.26 Computing S to determine the size of an expression

```
S($f$ : exp-log function in $x$)
    if   $x \ntrianglelefteq f$        → RETURN( $\{\}$ )
    elif $f = x$              → RETURN( $\{x\}$ )
    elif $f = g \cdot h$          → RETURN( $S(g) \cup S(h)$ )
    elif $f = g + h$          → RETURN( $S(g) \cup S(h)$ )
    elif $f = g^c$            → RETURN( $S(g)$ )
    elif $f = \ln g$          → RETURN( $\{\ln g\} \cup S(g)$ )
    elif $f = e^g$            → RETURN( $\{e^g\} \cup S(g)$ )
    fi
```

step and all the necessary operations described in the last sections do not increase the size of the expression. As a consequence, the following three claims have to be shown:

1. The size of an expression is not increased by the rewriting process.
2. The size of a rewritten function u is not increased by the series expansion, provided that $x \notin mrv(u)$.
3. When moving up an expression u one level in the scale (if $x \in mrv(u)$) and taking the leading coefficient of the series expansion, the size thereof is smaller than the size of u.

These three statements are proven in the following three subsections.

3.4.1.1 Rewriting Process

Whenever a set of *mrv* expressions contains more than one element, all elements can be rewritten in terms of a single one according to Lemma 3.20, provided that all elements in the *mrv* set are exponentials. Let $f = e^s$ and $g = e^t$ be two elements in $\Omega = mrv(u_i)$ with $mrv(g) = \{g\}$ such that for all $m \in S\backslash\{f\}$, $f \ntrianglelefteq m$. Then f is rewritten as $f = A \cdot g^c$ where $A = e^{s-ct}$, $c = \lim_{x\to+\infty} s/t$ and $mrv(A) \subseteq \Omega\backslash\{f\} \cup \{\omega\}$. If every occurrence of f in u_i is replaced by $A \cdot g^c$, then the exponential f disappears and the size is reduced by one. On the other hand, the new expression A which is introduced may increases the size by at most one. As a consequence, the rewriting process does not increase the size of the whole expression and $\mathrm{Size}(u_{i+1}) \leq \mathrm{Size}(u_i)$.

More formally, we can state, that if an expression u_i contains both f and g, then their portion in the size of u_i is

$$\Big|S(f) \cup S(g)\Big| = \Big|\{f,g\} \cup S(s) \cup S(t)\Big| = 1 + \Big|\{g\} \cup S(s) \cup S(t)\Big|,$$

since $f \not\trianglelefteq s$ and $f \not\trianglelefteq t$. The latter follows from the condition $mrv(g) = \{g\}$ which implies $mrv(t) \prec g$. If f is replaced by $A\,g^c$ in u_i then the portion of the transformed f and of g in the size of u_{i+1} becomes

$$\Big|S(A\cdot g^c)\cup S(g)\Big| = \Big|S(A)\cup S(g)\Big| \le \Big|\{A,g\}\cup S(s)\cup S(t)\Big| = 1+\Big|\{g\}\cup S(s)\cup S(t)\Big|,$$

with equality if A is active. The overall size of an expression containing f and g cannot be increased due to rewriting f in terms of g.

If A is not an active exponential, rewriting reduces the size of the expression. For example, when rewriting $f = e^{2x}$ in terms of $g = e^{x-1}$, A becomes $e^{2x-2(x-1)} = e^2$ if the argument is simplified. If the argument is not simplified, A is active and the size of the transformed expression remains the same.

It is very important to rewrite the elements in Ω by an element which is also in Ω. If the elements would be rewritten by a function of the same comparability class which does not appear in Ω, then the size of the expression could increase and termination would no longer be guaranteed in general. However, the size of the expression does not grow if it is rewritten in terms of the inverse of an element in Ω which does not appear in Ω itself. $f = e^h$ can be rewritten in terms of its inverse simply as $f = 1 \cdot (e^{-h})^{-1}$ and this process does not change the size of the expression.

3.4.1.2 Series Expansion

When all the elements in $\Omega = mrv(u)$ have been rewritten in terms of $\omega = e^h \in \Omega$, where ω tends to zero, then the series of u in terms of ω is computed. The leading coefficient is the new expression potentially to be followed up. We show next that the size of the series of u in terms of ω is not greater than that of u. This implies that the size of the leading coefficient is smaller, as ω itself does not appear as a subexpression of it.

The size of a series is defined to be the size of the (possibly infinite) set of terms, i.e.

$$\text{Size}\left(\sum_{i=0}^{\infty} c_i\,\omega^{e_i}\right) = \left|\bigcup_{i=0}^{\infty} S(c_i) \cup S(\omega)\right|.$$

According to this definition the size of the series of a constant is larger than that one of the constant itself. Thus, for our analysis, we consider every constant c as the expression $c\cdot\omega^0$. This approach does not change the overall size of u as ω already appears as a subexpression of u.

We will show that $\text{Size}(\text{Series}(u,\omega)) \le \text{Size}(u)$ by induction over the expression tree of u. To begin, ω and every constant $c\cdot\omega^0$ is by itself a series whose size is equal to the size of the expanded function.

For the arithmetic operations between two functions g_1 and g_2 we have to show, that

$$\begin{aligned}\text{Size}(\text{Series}(g_1,\omega)) \le \text{Size}(g_1) \;\wedge\; \text{Size}(\text{Series}(g_2,\omega)) \le \text{Size}(g_2)\\ \Rightarrow \text{Size}(\text{Series}(g_1 \star g_2,\omega)) \le \text{Size}(g_1 \star g_2)\end{aligned}$$

where $\star$ stands for addition or multiplication. This however follows from our definition of S as the coefficients of a sum or a product of two series are constructed from the old coefficients by means of addition and multiplication only. Note that the size of the series can indeed get smaller if, e.g., terms in a sum cancel out.

Let us assume that the series of the expression g is

$$\text{Series}(g,\omega) = c_0\,\omega^{e_0} + c_1\,\omega^{e_1} + \cdots$$

and that $\text{Size}(\text{Series}(g,\omega)) \le \text{Size}(g)$. To complete the induction step, we need to show that

$$\text{Size}(\text{Series}(f(g),\omega)) \le \text{Size}(f(g))$$

holds for f being the inverse, the exponential and the logarithm. We mark the use of the induction hypothesis with (I).

As we have seen in Section 3.3.3, the series of the inverse of g has the form

$$1/g = \frac{1}{c_0}\,\omega^{-e_0}\sum_{k=0}^{\infty}(-1)^k\,\Phi^k$$

where $\Phi = c_1/c_0\,\omega^{e_1-e_0} + c_2/c_0\,\omega^{e_2-e_0} + \cdots$, and we see that only multiplication and addition of series are used to compute the inverse; hence the size of the resulting series will not be larger than the size of the series of g. More formally, we get

$$\begin{aligned}\text{Size}(\text{Series}(1/g,\omega)) &= \text{Size}\left(\frac{1}{c_0}\,\omega^{-e_0}\sum_{k=0}^{\infty}(-1)^k\,\Phi^k\right)\\ &= \Big|S(c_0)\cup S(\omega)\cup S(\Phi)\Big|\\ &= \left|\bigcup_{i=0}^{\infty} S(c_i)\cup S(\omega)\right| = \text{Size}(\text{Series}(g,\omega))\\ &\overset{(I)}{\le} \text{Size}(g) = \text{Size}(1/g)\end{aligned}$$

which proves this case.

As we have seen in the proof of Theorem 3.22, the series of the argument of an exponential must start with a leading term whose exponent is positive or zero. The series of the exponential can thus be written as

$$\begin{aligned}\exp(g) &= \exp(c_0 + c_1\,\omega^{e_1} + c_2\,\omega^{e_2} + \cdots)\\ &= \exp(c_0)\,\exp(c_1\,\omega^{e_1} + c_2\,\omega^{e_2} + \cdots)\\ &= \exp(c_0)\sum_{k=0}^{\infty}\frac{\psi^k}{k!},\end{aligned}$$

where $\psi = c_1\,\omega^{e_1} + c_2\,\omega^{e_2} + \cdots$ and where c_0 may be 0. If $x \lhd c_0$ then we have

$$\begin{aligned}
\mathrm{Size}(\mathrm{Series}(\exp(g),\omega)) &= \mathrm{Size}\left(\exp(c_0)\sum_{k=0}^{\infty}\frac{\psi^k}{k!}\right)\\
&= \Big|S(\exp(c_0)) \cup S(\psi)\Big|\\
&= 1 + \Big|(S(c_0)\cup S(\psi))\backslash\{\exp(c_0)\}\Big|\\
&\leq 1 + \left|\bigcup_{i=0}^{\infty} S(c_i) \cup S(\omega)\right| = 1 + \mathrm{Size}(\mathrm{Series}(g,\omega))\\
&\overset{(I)}{\leq} 1 + \mathrm{Size}(g) = \mathrm{Size}(\exp(g))
\end{aligned}$$

which completes the proof of this case. Otherwise, if $x \not\lhd\ c_0$ we have

$$\begin{aligned}
\mathrm{Size}(\mathrm{Series}(\exp(g),\omega)) &= \mathrm{Size}\left(\exp(c_0)\sum_{k=0}^{\infty}\frac{\psi^k}{k!}\right)\\
&= \Big|S(\psi)\Big| = \left|\bigcup_{i=1}^{\infty} S(c_i) \cup S(\omega)\right|\\
&= \mathrm{Size}(\mathrm{Series}(g,\omega))\\
&\overset{(I)}{\leq} \mathrm{Size}(g) < \mathrm{Size}(\exp(g)).
\end{aligned}$$

Note that the size of the series is even smaller than the size of the expanded function in this case.

Example 3.27 Consider the function $f = e^{-x+e^{-x}\,e^{-x\,\ln x}}$ whose size is

$$\mathrm{Size}(f) = \Big|\{f, e^{-x}, e^{-x\,\ln x}, \ln x, x\}\Big| = 5.$$

$mrv(f) = \{e^{-x\,\ln x}\}$ and the series of f in $\omega = e^{-x\,\ln x}$ is

$$\mathrm{Series}(f,\omega) = e^{-x} + \left(e^{-x}\right)^2\,\omega + \frac{1}{2}\left(e^{-x}\right)^3\,\omega^2 + \frac{1}{6}\left(e^{-x}\right)^4\,\omega^3 + \cdots$$

and the size of this series is $|\{e^{-x},\omega,\ln x, x\}| = 4$. In this example the size of the series gets reduced although $x \lhd c_0$. The reason is that the new leading coefficient e^{c_0} has already appeared as a subexpression of f. The size of the leading coefficient e^{-x} is two in this case as the expression $\ln x$ gets also eliminated together with the most rapidly varying subexpression ω. ¶

The series of the logarithm of a series has the form

$$\ln(g) = \ln(c_0) + e_0\,\ln\omega + \sum_{k=1}^{\infty}\frac{(-1)^{k-1}\,\Phi^k}{k},$$

where Φ is defined as above. At first sight, two new expressions appear in this series which potentially increase the size, namely $\ln c_0$ and $\ln \omega$. The rest can again be reduced to multiplication and addition of series. Since $\omega = e^h$ is an exponential (whose argument is real), $\ln \omega$ can be replaced by h and so this logarithm disappears. Note that $S(h) \subset S(\omega)$. If c_0 depends on x, we have

$$\begin{aligned}
\mathrm{Size}(\mathrm{Series}(\ln(g),\omega)) &= \mathrm{Size}\left(\ln(c_0) + e_0 \ln\omega + \sum_{k=1}^{\infty} \frac{(-1)^{k-1}\Phi^k}{k}\right)\\
&= \left|S(\ln(c_0)) \cup S(e_0) \cup S(h) \cup S(\Phi)\right|\\
&= 1 + \left|\left(\bigcup_{i=0}^{\infty} S(c_i) \cup S(\omega)\right) \setminus \{\ln(c_0)\}\right|\\
&\leq 1 + \left|\bigcup_{i=0}^{\infty} S(c_i) \cup S(\omega)\right| = 1 + \mathrm{Size}(\mathrm{Series}(g,\omega))\\
&\overset{(I)}{\leq} 1 + \mathrm{Size}(g) = \mathrm{Size}(\ln(g));
\end{aligned}$$

otherwise, the size of $\ln c_0$ is zero and we obtain

$$\begin{aligned}
\mathrm{Size}(\mathrm{Series}(\ln(g),\omega)) &= \left|\bigcup_{i=0}^{\infty} S(c_i) \cup S(\omega) \cup S(\ln\omega)\right|\\
&= \mathrm{Size}(\mathrm{Series}(g,\omega))\\
&\overset{(I)}{\leq} \mathrm{Size}(g) < \mathrm{Size}(\ln g)
\end{aligned}$$

and again the size of an expression gets smaller when expanded as a series in ω.

3.4.1.3 Moving Up in the Scale

If $x \in \Omega = mrv(u)$ then we move up one level in the scale and continue to work with u_1, which is obtained from u by replacing x through e^x. This process *increases* the size of u by at most one and by exactly one if no simplifications are performed, e.g., if $\ln e^x$ is not simplified to x. Let us assume that $\mathrm{Size}(u_1) = \mathrm{Size}(u) + 1$. Then $e^x \in mrv(u_1)$ and we can proceed with the algorithm as usual. We will show that the size of the leading coefficient c_0 of the series of u_1 in terms of $\omega = e^{-x}$ is smaller than that of u.

Let us first assume that u does not contain any active logarithm. Then $mrv(u) = \{x\}$ and the series of u_1 in $\omega = e^{-x}$ can be computed directly, as the arguments of all exponentials which are going to be expanded have to tend to zero and no logarithms ever have to be expanded. The coefficients will be constants whose size is zero and the size of the leading coefficient is thus smaller than that of u.

If u does contain active logarithms, then we can show that the size of the series of u_1 is *smaller* than the size of u_1. As the size of a function does not get increased by series expansion as we have seen in Section 3.4.1.2, it is enough to show that at least at one point during the series expansion the size gets reduced. We claim that at least one active logarithm is eliminated when expanding u_1 in terms of ω, from which the result follows.

Let us look at an active logarithm $\ln g$ in u whose argument does not contain further active logarithms. What can its argument g look like? It cannot contain active logarithms, and it also cannot contain an exponential whose argument goes to $\pm\infty$, because this exponential would have to contain further active logarithms since $x \in mrv(u)$. Consequently, similarly to the first case above, g can be expanded into a series whose coefficients do not depend on x. The size of the leading coefficient of this series of g is thus zero and according to the logarithmic case in Section 3.4.1.2, $\mathrm{Size}(\mathrm{Series}(\ln g, \omega)) < \mathrm{Size}(\ln g)$. Since subsequent series expansions do not increase the size of their argument, $\mathrm{Size}(\mathrm{Series}(u_1, \omega)) < \mathrm{Size}(u_1)$ and as a consequence $\mathrm{Size}(\mathrm{Series}(u_1, \omega)) \leq \mathrm{Size}(u)$. The size of the leading coefficient of the series of u_1 in ω is therefore smaller than that of u, which completes the proof.

3.4.2 Recursive Calls

In the previous section we have seen that the global iteration terminates, provided that the computation of the *mrv* sets can always be performed. The procedure which determines the *mrv* set of a given function u needs to compute limits to compare two subexpressions. Furthermore, once the set of most rapidly varying subexpressions has been determined, its elements must be rewritten. In this rewriting step we have to compute the limit of the quotient of their logarithms. It is the task of this section to show that these recursive calls do not lead to infinite recursions. We will show that the size of the arguments of these subsequent limit calls is *smaller* than the size of the function u whose limit is currently being computed. We use the same definition for the size of a function as in the last section. The chain of recursive limit calls must terminate, when the size of the argument is zero, as then the expression passed to limit is a constant.

From Lemma 3.14 we know that the candidates to be compared are always exponentials or equal to x and that the comparison is done by computing the limit of the quotient of the logarithms of the two candidates. Let $f \lhd u$ and $g \lhd u$ be two subexpressions of u which have to be compared, then the following three cases may be distinguished:

(1) $f = x$ and $g = x$

(2) $f = e^s$ and $g = e^t$

(3) $f = e^s$ and $g = x$.

In the first case nothing needs to be done as f and g are in the same comparability class.

Let us consider the second case where both f and g are exponentials. The logarithms of f and g can be simplified to s and t, respectively, and we get

$$\begin{aligned}\text{Size}\left(\frac{\ln f}{\ln g}\right) &= \text{Size}\left(\frac{s}{t}\right) = \left|S(s) \cup S(t)\right| \\ &< \left|\{e^s, e^t\} \cup S(s) \cup S(t)\right| = \left|S(e^s) \cup S(e^t)\right| \leq \text{Size}(u).\end{aligned}$$

The first inequality holds since $e^s \lhd t$ and $e^t \lhd s$ would imply $e^s \lhd s$ which is a contradiction. Note that whenever an exponential is compared with the most rapidly varying subexpression of its argument, then we have $f \lhd g$ or $g \lhd f$ and thus $e^s \lhd t$ or $e^t \lhd s$. In this case however the above inequality also holds and the size of the quotient of the logarithms of f and g is smaller than the size of u.

Let us now look at the third case, which turns out to be the most difficult one. To compare $f = e^s$ and $g = x$ we must compute $\lim\limits_{x\to+\infty} s/\ln(x)$. The size of $s/\ln(x)$ is clearly smaller than the size of u if $\ln(x) \lhd s$, since then we get

$$\text{Size}\left(\frac{\ln f}{\ln g}\right) = \text{Size}\left(\frac{s}{\ln x}\right) = \text{Size}(s) < \text{Size}(e^s) \leq \text{Size}(u).$$

Otherwise, if $\ln(x) \not\lhd s$, the size of the argument passed to limit is equal to the size of f, which, in turn, may be equal to the size of u, i.e.,

$$\text{Size}\left(\frac{s}{\ln x}\right) = \left|S(s) \cup \{\ln x\}\right| = 1 + \text{Size}(s) = \text{Size}(e^s) \leq \text{Size}(u).$$

Let us investigate what happens when we compute the limit of $s/\ln x$. First $mrv(s/\ln x)$ is determined. $mrv(\ln x) = \{x\}$ and so $mrv(s/\ln x) = mrv(s)$ and thus only $mrv(s)$ needs to be computed. This computation however does not pose further problems, since

$$\text{Size}(s) < \text{Size}(e^s) \leq \text{Size}(u).$$

Let us first assume that $x \notin mrv(s)$. Then $\omega \in mrv(s)$ is an exponential and the series of $s/\ln(x)$ in ω can be computed. According to Section 3.4.1.2 the size of the leading coefficient of this series is smaller than the size of $s/\ln x$ and so also smaller than the size of u. It may thus be processed further without problems if necessary.

If, on the other hand, $x \in mrv(s(x))$, then we must move up one level in the scale and compute the limit of $s(e^x)/x$. We have already shown that the size of the leading coefficient of the series of $s(e^x)/x$ in ω is smaller than that of $s/\ln(x)$, provided that the mrv sets of $s(e^x)/x$ can be computed. This computation might, however, lead to an infinite recursion. As an example consider the computation of $mrv(e^x)$. According to Algorithm 3.12 we first have to compare e^x with $mrv(x) = \{x\}$ which is a comparison of type (3) with $s = x$. Since $\ln x \not\lhd\ s$ we determine $mrv(s) = \{x\}$ and have to move up one level and end up with the function e^x/x. Computing the mrv set thereof requires the determination of $mrv(e^x)$ first and we are in a loop.

However, we do not have to compute the mrv set if we move up one level. We can rather derive it directly from $mrv(t)$ by simply moving it up as well. As $x \in mrv(s)$, the elements in $mrv(s(e^x)/x)$ can be rewritten in terms of $\omega = e^{-x}$. Rewriting of the elements in $mrv(s(e^x)/x)$ requires further limit calls, but now all elements of the mrv set are exponentials. Furthermore, as $\ln(x) \not\lhd\ s$,

$$\mathrm{Size}(s(e^x)/x) = 1 + \mathrm{Size}(s(x)) = \mathrm{Size}\left(\frac{s(x)}{\ln x}\right) \leq \mathrm{Size}(u).$$

As a consequence the size of the arguments for those limit calls in the rewriting step (which are of type (2)) are smaller than $\mathrm{Size}(u)$ and do not lead to further problems. The size of the leading coefficient of the series expansion of $s(e^x)/x$ in terms of ω is smaller than that of $s(e^x)/x$ and hence smaller than the size of u. Therefore, it can also be processed by further iterations if necessary without problems.

3.5 A Complete Example

In the previous sections our examples have demonstrated single aspects of the algorithm, but in this section we want to go through a complete example. We compute the limit of

$$f = \ln\ln\left(xe^{xe^x} + 1\right) - \exp\exp\left(\ln\ln x + \frac{1}{x}\right)$$

for $x \to +\infty$. The example is taken from [64].

In a first step we have to determine the set of most rapidly varying subexpressions of f. For the first term this means computing the mrv set of xe^{xe^x}. $mrv(x) = \{x\}$, and in order to compute $mrv(e^{xe^x})$ we first have to determine the limiting behavior of xe^x. To compute $mrv(xe^x)$ we compare x and e^x, that is we compute the limit of $\ln x/x$. This limit is 0 and so $e^x \succ x$. As a consequence $mrv(e^x) = \{e^x\}$, $mrv(xe^x) = \{e^x\}$, $xe^x \to \infty$ and $mrv(e^{xe^x}) = \{e^{xe^x}\}$. Finally we have to compare x and e^{xe^x}. The quotient of the logarithms of the two is $\ln(x)/(xe^x) = \frac{\ln x}{x}e^{-x}$ and tends to 0. Thus $e^{xe^x} \succ x$ and $mrv\left(\ln\ln\left(xe^{xe^x}+1\right)\right) = \left\{e^{xe^x}\right\}$.

Let us now turn to the second term of f. The argument of the inner exponential of the second term tends to infinity and so we compare $\exp(\ln\ln x + 1/x)$ with $mrv(\ln\ln x + 1/x) = \{x\}$. The mrv set of the quotient of the logarithms of the two is $\{x\}$ and we expand this quotient into a series in terms of $\omega = e^{-x}$ after having moved up one level:

$$\frac{\ln\ln x + 1/x}{\ln x} \rightsquigarrow \frac{\ln x + \omega}{x} = \frac{\ln x}{x} + \frac{1}{x}\,\omega.$$

The leading coefficient of this series tends to zero and thus $x \succ e^{\ln\ln x + 1/x}$ and $mrv(\exp(\ln\ln x + 1/x)) = \{x\}$. For the outer exponential we therefore have to compare $\exp\exp(\ln\ln x + 1/x)$ with x. Again, x is in the mrv set of the quotient of the logarithms and we move up one level and compute the series in terms of $\omega = e^{-x}$:

$$\frac{\exp\left(\ln\ln x + \frac{1}{x}\right)}{\ln x} \rightsquigarrow \frac{\exp(\ln x + \omega)}{x} = 1 + \omega + \frac{1}{2}\,\omega^2 + O(\omega^3),$$

which shows that $\exp\exp(\ln\ln x + 1/x) \asymp x$.

At this point our algorithm has determined that $mrv(f) = e^{xe^x}$ and it has derived the following order on the subexpressions of f:

$$e^{xe^x} \succ e^x \succ \{x, \exp\exp(\ln\ln x + 1/x)\} \succ \exp(\ln\ln x + 1/x). \tag{3.6}$$

Now we can start eliminating comparability classes. The size of f is 9, so at most 9 iterations are necessary. We first expand f in terms of $\omega_1 = e^{-xe^x}$ and get

$$\begin{aligned}&\ln\ln\left(x\omega_1^{-1} + 1\right) - \exp\exp\left(\ln\ln x + 1/x\right) = \\ &\quad\big(\ln(\ln x + xe^x) - \exp\exp\left(\ln\ln x + 1/x\right)\big) + \frac{1}{x(\ln x + xe^x)}\,\omega_1 + O\left({\omega_1}^2\right).\end{aligned}$$

Since the leading exponent is zero we continue with the leading coefficient whose size is 7. (It is left as an exercise to the reader to figure out why the size got reduced by 2.) According to the information available in relation (3.6), the most rapidly varying subexpression of $\ln(\ln x + xe^x) - \exp\exp\left(\ln\ln x + 1/x\right)$ is e^x and we set $\omega_2 = e^{-x}$. For the series we get

$$\begin{aligned}&\ln(\ln x + x\omega_2^{-1}) - \exp\exp\left(\ln\ln x + 1/x\right) \\ &\qquad= \big(\ln x + x - \exp\exp\left(\ln\ln x + 1/x\right)\big) + \frac{\ln x}{x}\,\omega_2 + O\left({\omega_2}^2\right).\end{aligned}$$

The size of this series is 6 and thus the size of the leading coefficient is 5. We could now directly expand the leading coefficient into a series in x to get the result, but let us strictly follow the rules of the algorithm. As the mrv set of the leading coefficient is $\{x, \exp\exp\left(\ln\ln x + 1/x\right)\}$ we move up one level. The leading coefficient becomes

$$x + e^x - \exp\exp\left(\ln x + e^{-x}\right)$$

and the *mrv* set becomes $\{e^x, \exp(\exp(\ln x + e^{-x}))\}$. We set $\omega_3 = e^{-x}$ and rewrite the second exponential in the *mrv* set in terms of ω_3 (the limit of the quotient of the logarithms of the two has already been computed). The series of the transformed expression in ω_3 is

$$x + \omega_3^{-1} - \exp(\exp(\ln x + \omega_3) - x)\,\omega_3^{-1} = -\left(\frac{x}{2} + \frac{x^2}{2}\right)\omega_3 + O({\omega_3}^2).$$

As the leading exponent of the series is positive, the limit of f for $x \to +\infty$ is 0.

With the above approximation we also found an asymptotic approximation for f at $x = \infty$:

$$f \approx -\frac{\ln^2 x}{2x} + O\left(\frac{\ln x}{x}\right).$$

In Chapter 6 we will look at this application of our algorithm in more detail.

4. Related Work

In this chapter we compare our algorithm with two other approaches which have been proposed in the literature. First we will discuss the nested forms and nested expansions which have been proposed by Shackell [78]. A nested form is just another form of writing a function which corresponds to the first term of an asymptotic series. This form allows one to read off the limit easily. The algorithm to convert a function into its nested form is a bottom up algorithm. A given function is converted recursively into a normal form. Since cancellations may occur during this normalization process, all information about the function must always be kept available. If cancellation occurs, then the right scale of expansion is found with the help of the zero equivalence oracle. The oracle is applied to the function which is obtained by setting all subexpressions which are in a larger comparability class than the potential scale of expansion to zero. The normal form is defined such that this substitution is always possible.

Secondly, we briefly discuss the *ghost and shadow* approach also introduced by Shackell in [83]. This algorithm is based on the same ideas as the nested form algorithm. Instead of replacing subexpressions with zero, projection onto a *shadow* field is used. This method is also based on the idea of asymptotic series expansion. A function is expanded into its asymptotic $\mathcal{T}$ expansion where $\mathcal{T}$ contains all the different comparability classes.

4.1 Nested Forms

The nested form of a function is a normal form for elements in a Hardy field which either tend to infinity or to zero. Nested forms have been introduced by Shackell in [78]. When he was implementing an algorithm for computing asymptotic approximations in Miranda [52] using a generalized power series approach he encountered the cancellation problem. His answers were first *estimate forms* [77] which he subsequently replaced by the nested forms. For an overview article on nested forms we refer to [80]. The following definition is a minor variant of Shackell's. The notation l_k is used for the k-th iterated logarithm and e_k for the k-th iterated exponential.

Definition 4.1 (Nested Form) *Let $\mathcal{F}$ be a Hardy field and let ϕ be a positive element in $\mathcal{F}$. A nested form for ϕ is a finite sequence $\{(\varepsilon_i,\, s_i,\, m_i,\, d_i,\, \phi_i),\, i = 1, \ldots, k\}$ of k elements, $k \geq 0$, which has the following properties:*

(a) For each i, $\varepsilon_i \in \{-1, 1\}$; s_i and m_i are non-negative integers, d_i is a non-zero real number and ϕ_i is another element of a Hardy field;

(b) $\phi = \phi_0$ and for $i = 1, \ldots, k$: $\phi_{i-1} = e_{s_i}^{\varepsilon_i}(l_{m_i}(x)^{d_i}\phi_i)$;

(c) $\phi_i \prec l_{m_i}(x)$ for $i = 1, \ldots, k$;

(d) ϕ_k tends to a positive constant, i.e. $\phi_k = c + \xi$ with $c > 0$ and $\xi \to 0$ as $x \to +\infty$;

(e) $\forall i \;:\; 1 \leq i \leq k \;:\; d_i > 0 \vee s_i = 0$;

(f) $\forall i \;:\; 1 \leq i \leq k \;:\; s_i = 0 \Rightarrow \varepsilon_i = 1$;

(g) $d_k \neq 1$ or $s_k = 0$ or $m_k = 0$.

Condition (e) implies that the argument of the exponential always tends to infinity. Condition (g) can be motivated by the fact that in the case that $d_k = 1$ and $s_k > 0$ and $m_k > 0$ the expression

$$\phi_{k-1} = e_{s_k}^{\varepsilon_k}(l_{m_k}(x)\phi_k) = e_{s_k}^{\varepsilon_k}(l_{m_k}(x)(c + \xi))$$

can be converted to

$$e_{s_k-1}^{\varepsilon_k}\left(l_{m_k-1}(x)^c\, e_1(l_{m_k}(x)\,\xi)\right)$$

where of course $\exp(l_{m_k}(x)\,\xi)$ must be written as a nested form. Condition (c) is satisfied for the above nested form as

$$\lim_{x\to+\infty} \frac{\ln(e_1(l_{m_k}(x)\,\xi))}{\ln(l_{m_k-1}(x)^c)} = \lim_{x\to+\infty} \frac{l_{m_k}(x)\,\xi}{c\, l_{m_k}(x)} = \lim_{x\to+\infty} \frac{\xi}{c} = 0.$$

Up to the representation of ϕ_k, the nested form of a function is unique. Once a function f has been converted into nested form, its limit is apparent. If $k = 0$ then the limit is c, and if $s_1 > 0$ and $\varepsilon_1 = 1$ or $s_1 = 0$ and $d_1 > 0$ then the limit is $+\infty$, otherwise the limit is zero.

A *nested expansion* for a function $f \in \mathcal{F}$ is a sequence of nested forms n_j so that n_1 is a nested form for f and if $n_j = \{(\varepsilon_{ji}, s_{ji}, m_{ji}, d_{ji}, \phi_{ji}),\ i = 1, \ldots, k_j\}$, $j \geq 1$, then n_{j+1} is a nested form for $|\phi_{j,k_j} - \lim \phi_{j,k_j}|$. The finite partial expansions $\{n_1, \ldots, n_j\}$ give successively finer estimates of the asymptotic growth of f in the same way as the partial sums of an asymptotic expansion do.

For example, the nested form of the function $f = \ln(\Gamma(\Gamma(x)))/e^x$ is

$$\Big\{(1,1,0,1,\phi_{11}),(1,0,1,1,\phi_{12})\Big\}$$

where $\phi_{12} = 1 - \xi_1$. Rewritten in usual mathematical notation we obtain

$$\frac{\ln(\Gamma(\Gamma(x)))}{e^x} = e_1^1\left(x\, e_0^1(\ln x\,\{1-\xi_1\})\right) = e^{x\;\ln x\,(1-\xi_1)}.$$

The limit of $\ln(\Gamma(\Gamma(x)))/e^x$ is $+\infty$ as $s_1 = 1$ and $\varepsilon_1 = 1$. The next term of the nested expansion of f is $\{(1,0,1,-1,\phi_{21})\}$, i.e. $\xi_1 = \ln^{-1}(x)\phi_{21}$ with $\phi_{21} = 2 - \xi_2$, and so on. We will see this example once more in the next chapter as Example 5.5 where we show, how an extension of our algorithm handles this problem.

In [78] it has been proven that f has a nested expansion if it belongs to a Rosenlicht field $\mathcal{F}$. Additionally, the existence of a nested form implies that the computation of a nested form for $f \in \mathcal{F}$ is Turing reducible to the problem of computing limits in $\mathcal{F}$. This seems somewhat surprising since a nested form contains much more information than a limit.

Lemma 4.2 *Let $\mathcal{F}$ be a Rosenlicht field and let us assume that we have an oracle which determines the limit at $+\infty$ for any element $f \in \mathcal{F}$. The nested form of a function $f \in \mathcal{F}$ can then be computed if it exists by performing arithmetic in $\mathcal{F}$ and consulting the oracle only.*

Proof. If the limit of f is finite and non zero (which can be checked using the oracle), the nested form of f is simply $\lim f + (f - \lim f)$. Otherwise the nested form of f has the form

$$f = \sigma e_s^{\varepsilon}\left(l_m(x)^d\,\phi\right)$$

with $l_m(x) \succ \phi$, $s, m \in \mathbb{N}$, $d \in \mathbb{R}^*$ and $\varepsilon \in \{1,-1\}$ and where σ is the sign of f. Let us further assume that $\nu(f) < 0$ which can be tested using the limit oracle. If $\nu(f) > 0$ then we first determine the nested form of f^{-1} and derive the nested form for f by adjusting ε and d. As σ holds the sign of f we assume further that $f > 0$.

Since $f > 0$ and $\nu(f) < 0$ which implies $\varepsilon = 1$ we can write

$$\begin{aligned} l_s(f) &= l_m(x)^d\,\phi \\ l_{s+1}(f) &= d\,l_{m+1}(x) + \ln\phi \\ \frac{l_{s+1}(f)}{l_{m+1}(x)} &= d + \frac{\ln\phi}{l_{m+1}(x)}. \end{aligned}$$

As $\phi \prec l_m(x)$ it follows that

$$d(s,m) = \lim_{x\to+\infty}\frac{l_{s+1}(f)}{l_{m+1}(x)} = d \in \mathbb{R}^*. \tag{4.1}$$

In other words, we have to determine integer parameters s and m so that (4.1) is satisfied. One could search for such parameters systematically using a Cantor enumeration of $\mathbb{N}^2$, however a more efficient algorithm can be formulated.

From the definition of $d(s,m)$ follows that

$$\begin{array}{rclcl} d(s,m)=0 & \Rightarrow & d(t,m)=0 \ \forall\ t>s & \text{and} & d(s,n)=0 \ \forall\ n<m \\ d(s,m)=\infty & \Rightarrow & d(s,n)=\infty \ \forall\ n>m & \text{and} & d(t,m)=\infty \ \forall\ t<s. \end{array}$$

Thus, if we look for s and m in the range $s \geq s_0$ and $m \geq m_0$ and if $d(s_0,m_0) = 0$ then the search can be restricted to $s \geq s_0$ and $m > m_0$; if $d(s_0,m_0) = \infty$ then $s > s_0$ and $m \geq m_0$ is implied; and if finally $d(s_0,m_0) \in \mathbb{R}^*$ we are done. Algorithm 4.3 performs this search for an expression f which tends to ∞.

Algorithm 4.3 Computing the nested form of $f \in \mathcal{F}$

```
NestedForm($f$) =
    $s := 0$; $lsf := f$; $m := 0$; $lmx := x$;
    $\{lsf = l_s(f)$ and $lmx = l_m(x)\}$
    $d := \lim_{x\to+\infty} \frac{\ln lsf}{\ln lmx}$
    while $d = \infty$ or $d = 0$ do
        if   $d = \infty$  $\rightarrow$  $s := s+1$; $lsf := \ln(lsf)$
        elif $d = 0$       $\rightarrow$  $m := m+1$; $lmx := \ln(lmx)$
        fi
        $d := \lim_{x\to\infty} \frac{\ln lsf}{\ln lmx}$
    od
    $\{d \in \mathbb{R}^*\}$
```

The existence of a nested form for f implies that Algorithm 4.3 terminates. For computing the parameters s and m, $s+m+1$ inquiries of the limit oracle are necessary. To get the nested form of f Algorithm 4.3 has to be applied recursively to $l_s(f)/l_m(x)^d$ until an expression is obtained whose limit is finite and non-zero. □

A direct implementation to compute nested forms for exp-log functions can be derived from the algorithm described in [77] which computes the *estimate form* of a function. An estimate form is also a normal form for functions which is not as powerful as nested forms but similar enough to allow us to easily adapt the algorithm. We have successfully implemented this algorithm in MAPLE. We will first recall the basic ideas and steps of this algorithm and then compare it with our algorithm from the point of view of practicability for computing limits.

4.1.1 Algorithm for Computing Nested Forms

The basic idea of the algorithm presented in [77] is that the expression tree of the function is converted into a nested form from the bottom to the top. At the leaves of the expression tree we have constants which are already in nested form and the unknown x, whose nested form is $\{(1,0,0,1,1)\}$. Furthermore, algorithms to compute the nested form of the exponential, the logarithm and the inverse of a nested form are given as well as algorithms to compute the nested form of the sum and the product of two nested forms.

Expressions which tend to zero play a special role in this algorithm and are called z-sums. For example ξ is a z-sum in $\phi_k = c + \xi$. A z-sum is a sum of z-prods, and a z-prod is a product of z-terms. A z-term is either a nested form which tends to zero, e.g., a nested form with $\varepsilon < 0$, or a z-function. A z-function is a function which tends to zero and which is applied to arguments which also tend to zero. For exp-log functions we have z-functions which encode the shifted exponential function at $x = 0$, the logarithm at $x = 1$ and a shifted inverse at $x = 1$.[1] There are also z-functions which represent the tail of the series expansion of a z-function.

$$\begin{array}{rclrcl} \mathrm{zexp}_0(x) &=& e^x - 1, & \mathrm{zexp}_n(x) &=& \frac{1}{x}\left(\mathrm{zexp}_{n-1}(x) - \frac{x}{n!}\right) \\ \mathrm{zlog}_0(x) &=& \ln(1+x), & \mathrm{zlog}_n(x) &=& \frac{1}{x}\left(\mathrm{zlog}_{n-1}(x) - \frac{x}{n}\right) \\ \mathrm{zinv}_0(x) &=& 1 - \frac{1}{1+x}, & \mathrm{zinv}_n(x) &=& \frac{1}{x}\Big(\mathrm{zinv}_{n-1}(x) - x\Big) \end{array} \tag{4.2}$$

for $n > 0$.

The basic operation which is used in order to perform the arithmetic operations on nested forms is the conversion of a z-sum Z into a nested form. This process is called z-expansion and is done, roughly speaking, by expanding the z-sum into a series and by taking the leading term as a new nested form. The crucial point however is that this expansion has to be performed in the right asymptotic scale due to the cancellation problem. As a consequence the comparability classes of all the nested form-like z-terms[2] which appear in Z are determined and ordered in a first step. If there are several z-terms which are in the same comparability class, then Z has to be rewritten.

Once the comparability classes of all z-terms have been determined and are all distinct, the correct comparability class to perform the expansion is determined. Let us assume that the comparability classes of the z-terms are $\omega_1 \prec \omega_2 \prec \cdots \prec \omega_r$. Z_k is defined to be the expression being obtained from Z by replacing all z-prods which are in a higher comparability class than ω_k by zero, and in particular $Z = Z_r$. Due to the special form of the z-terms, this

[1] Additional z-functions must be added if the field of functions is extended, e.g. a function $\mathrm{zsin}(x) = \sin(x)$ and $\mathrm{zcos}(x) = \cos(x) - 1$ (cf. [75]).

[2] For a precise definition we refer to [80] where the set of nested form-like terms is called the set $V(Z)$.

substitution is always possible. In order to find the right comparability class in which the series expansion has to be performed, we go through the expressions $Z_1, Z_2, \ldots$ and determine whether they are zero or not using the oracle for deciding zero equivalence. If Z_k is zero, then we know that some positive powers of any ω_i with $i > k$ must appear in the asymptotic expansion of Z, as Z itself is not identically zero. Let k be the smallest index such that $Z_k \neq 0$, then the series can be computed in terms of $v = \omega_k$ and no cancellations will appear.

Furthermore, if a z-function $f(x+y)$ is to be expanded in terms of v then it may happen that x only depends on z-terms which are in a smaller comparability class than v and that y contains a positive power of v. In order to be able to apply the expansion rules (4.2) to compute the power series of f, the function must be expanded, i.e. the x and the y term of $f(x+y)$ must be separated. Special expansion rules are given in [77, Lemma 2[3]].

Once this separation has been done, the z-functions which contain a positive power of v in their argument can be expanded using the expansion rules (4.2). The leading term of a z-function with index $n-1$ is obtained if it is expressed in terms of the z-functions with index n. Eventually Z can be written as $Z = v^r\{H + \eta\}$ with $\eta \to 0$. In a similar way, H can be expanded and finally Z will be transformed into $Z = v_1^{r_1} v_2^{r_2} \cdots v_m^{r_m}\{c + \zeta\}$ where c is a nonzero finite constant, ζ is a z-sum and $v_1 \succ v_2 \succ \cdots \succ v_m$. This form for Z can be converted into a nested form by multiplying the nested forms $v_i^{r_i}$ and $c + \zeta$ together.

4.1.2 Comparison

The main difference between the algorithm to compute a nested form and our algorithm is that in Shackell's approach no terms of any series approximation can ever be discarded. The remaining higher order terms of any series approximation always have to be retained. The reason for this is that Shackell's algorithm operates on the expression tree of a given function recursively from the leaves up to the root. In order to be able to resolve cancellation problems at higher levels in the expression tree all the information for every subexpression must be available. No information can ever be neglected in this approach. As a consequence the nested forms tend to become bigger and bigger during the recursive process and this is a significant disadvantage. Our algorithm, however, can discard the tail of every series expansion as it only needs the leading term. The information which is stored in the tail of the series expansion is no longer needed (in order to determine the limit). As a consequence the size of the expression tends to get smaller at every step, a characteristic we could use to prove termination.

[3] There is a typographic error in the expansion rule for $\mathrm{zinv}_n(x+y)$. The term $U_n(x,y)$ must be added and not subtracted.

To illustrate this point let us look at the size of the nested form of

$$e^{e^x}/e^{e^{x-e^{-e^{e^x}}}},$$

which turns out to be $1+\zeta$ where ζ is the following z-term:

$$\begin{aligned}\mathrm{zexp}_0\Bigg(e_3^{-1}\Bigg(x\Bigg\{&1+\zeta_1+\mathrm{zlog}_0\,(\\ &e_1^{-1}(x\{1+\zeta_1\})\cdot\mathrm{zlog}_0\left(-\,\mathrm{zlog}_0(\mathrm{zexp}_1(-e_3^{-1}(x)))\,e_2^{-1}(x\{1+\zeta_1\})\right)\,x^{-1}\\ &)\Bigg\}\Bigg)\Bigg)\end{aligned}$$

with

$$\zeta_1=\frac{\mathrm{zlog}_0(\mathrm{zlog}_0(-x\,e_2^{-1}(x))e_1^{-1}(x))}{x}.$$

Computing the limit of a function using the nested form approach provides a typical example of an algorithm which suffers from the problem of *intermediate expression swell* as the size of the result we finally are interested in, namely the limit of the function, is very small. This problem is one of the most serious problems computer algebra algorithms can encounter and should whenever possible be avoided.

Another problem which aggravates this situation is the problem of simplifying z-sums. Once a z-function has been expanded to order n it is very expensive to figure out whether it can be combined together with other terms into an expansion of smaller order. As the complexity of such a simplification step is exponential to the size of the expression, it cannot be done in practice, and as a consequence the expressions get larger than they have to be. Consider the computation of the nested form for $e^{1/x}-(e^{1/x}-1)/(e^{1/x})$. The algorithm proceeds as follows: First the nested forms of the numerator and the denominator of the quotient are determined, i.e. $e^{1/x}-1=x^{-1}\{1+\mathrm{zexp}_1(1/x)\}$ and $e^{1/x}=1+\mathrm{zexp}_0(1/x)$. The inverse of the latter is $1-\mathrm{zinv}_0(\mathrm{zexp}_0(1/x))$ and the quotient becomes $x^{-1}\{1+\mathrm{zexp}_1(1/x)-\mathrm{zinv}_0(\mathrm{zexp}_0(1/x))-\mathrm{zexp}_1(1/x)\,\mathrm{zinv}_0(\mathrm{zexp}_0(1/x))\}$ which tends to zero. Thus the nested form of $e^{1/x}-(e^{1/x}-1)/(e^{1/x})$ is

$$\begin{aligned}1+\mathrm{zexp}_0\left(\tfrac{1}{x}\right)-\tfrac{1}{x}-\frac{\mathrm{zexp}_1\left(\frac{1}{x}\right)}{x}&+\frac{\mathrm{zinv}_0\left(\mathrm{zexp}_0\left(\frac{1}{x}\right)\right)}{x}\\ &+\frac{\mathrm{zexp}_1\left(\frac{1}{x}\right)\mathrm{zinv}_0\left(\mathrm{zexp}_0\left(\frac{1}{x}\right)\right)}{x}.\end{aligned}\tag{4.3}$$

If the function first is normalized however, then another, more complicated nested form results. We compare the two nested forms with a MAPLE package we implemented to compute nested forms. The implementation follows the description in [77] and was adjusted to nested forms where necessary. *L(n)* denotes the n times iterated logarithm, i.e. $L(0)=x$, and $\mathrm{NF}(0,c,\zeta)$ denotes $c+\zeta$.

```
> e := exp(1/x)-(exp(1/x)-1)/(exp(1/x)):
> JSnestform[Input](e);
```

$$\mathrm{NF}\left(0, 1, -\frac{1}{L(0)} + \frac{\mathrm{zinv}_0\left(\mathrm{zexp}_0\left(\frac{1}{L(0)}\right)\right)}{L(0)} - \frac{\mathrm{zexp}_1\left(\frac{1}{L(0)}\right)}{L(0)} + \frac{\mathrm{zexp}_1\left(\frac{1}{L(0)}\right)\,\mathrm{zinv}_0\left(\mathrm{zexp}_0\left(\frac{1}{L(0)}\right)\right)}{L(0)} + \mathrm{zexp}_0\left(\frac{1}{L(0)}\right)\right)$$

```
> JSnestform[Input](normal(e));
```

$$\mathrm{NF}\left(0, 1, -\mathrm{zinv}_0\left(\mathrm{zexp}_0\left(\frac{1}{L(0)}\right)\right) + \frac{1}{L(0)} + 2\frac{\mathrm{zexp}_1\left(\frac{2}{L(0)}\right)}{L(0)} - \frac{\mathrm{zexp}_1\left(\frac{1}{L(0)}\right)}{L(0)} - \frac{\mathrm{zinv}_0\left(\mathrm{zexp}_0\left(\frac{1}{L(0)}\right)\right)}{L(0)} - 2\frac{\mathrm{zinv}_0\left(\mathrm{zexp}_0\left(\frac{1}{L(0)}\right)\right)\,\mathrm{zexp}_1\left(\frac{2}{L(0)}\right)}{L(0)} + \frac{\mathrm{zinv}_0\left(\mathrm{zexp}_0\left(\frac{1}{L(0)}\right)\right)\,\mathrm{zexp}_1\left(\frac{1}{L(0)}\right)}{L(0)}\right)$$

A sequence of transformations can be applied to equation (4.3) to see that the two representations are indeed equivalent. However, the point is that such a simplification is difficult to perform automatically. Remember, that nested forms are only unique up to the representation of the z-sum ζ in ϕ_k.

As z-sums are difficult to simplify, it may happen that zeros remain unsimplified as huge expressions. If such a zero appears inside another expression, there can be little chance to detect and remove it. The treatment of these hidden zeros implies a lot of unnecessary work. Consider again the above example. The function *SAdd* adds, and the function *SSubtract* subtracts two nested forms. *L(n)* stands for the *n*-times iterated logarithm.

```
> JSnestform[SSubtract](
>    JSnestform[SAdd](JSnestform[Input](1+1/x),""),
>    ");
```

$$\mathrm{NF}\left(0, 1, 2\frac{\mathrm{zinv}_0\left(\mathrm{zexp}_0\left(\frac{1}{L(0)}\right)\right)}{L(0)} + \mathrm{zinv}_0\left(\mathrm{zexp}_0\left(\frac{1}{L(0)}\right)\right) + \mathrm{zexp}_0\left(\frac{1}{L(0)}\right) - \frac{1}{L(0)} - 2\frac{}{L(0)} + 2\frac{\mathrm{zinv}_0\left(\mathrm{zexp}_0\left(\frac{1}{L(0)}\right)\right)\,\mathrm{zexp}_1\left(\frac{2}{L(0)}\right)}{L(0)}\right)$$

```
> normal(JSnestform[Output]("), expanded);
```

$$\frac{x+1}{x}$$

```
> JSnestform[Input](");
```

$$\mathrm{NF}\left(0, 1, \frac{1}{L(0)}\right)$$

We see that the first result contains a large hidden zero, but as long as it does not appear as an isolated term, the zero equivalence oracle cannot be applied. Such hidden zeros slow down the Z-expansion.

The representation of the z-sums also depend on the order the terms of a sum are summed up, and it is not possible to decide easily which order will return the simplest representation. Consider the normalized form of our example function:

$$\frac{\left(e^{1/x}\right)^2 - e^{1/x} + 1}{e^{1/x}}.$$

Once the terms of the sum of the numerator have been converted to nested forms, they can be summed up in three different ways:

```
> JSnestform[SAdd](JSnestform[Input](exp(1/x)^2),
>                  JSnestform[Input](-exp(1/x)+1));
```

$$\mathrm{NF}\left(0, 1, -\frac{1}{L(0)} - \frac{\mathrm{zexp}_1\left(\frac{1}{L(0)}\right)}{L(0)} + \mathrm{zexp}_0\left(\frac{2}{L(0)}\right)\right)$$

```
> JSnestform[SAdd](JSnestform[Input](-exp(1/x)),
>                  JSnestform[Input](exp(1/x)^2+1));
```

$$\mathrm{NF}\left(0, 1, \mathrm{zexp}_0\left(\frac{2}{L(0)}\right) - \mathrm{zexp}_0\left(\frac{1}{L(0)}\right)\right)$$

```
> JSnestform[SAdd](JSnestform[Input](1),
>                  JSnestform[Input](exp(1/x)^2-exp(1/x)));
```

$$\mathrm{NF}\left(0, 1, \frac{1}{L(0)} + 2\frac{\mathrm{zexp}_1\left(\frac{2}{L(0)}\right)}{L(0)} - \frac{\mathrm{zexp}_1\left(\frac{1}{L(0)}\right)}{L(0)}\right)$$

In some cases the z-functions got expanded to a certain degree, in others they did not get expanded. Note also, that for the first and the last ordering a Z-expansion is computed, while for the second order no Z-expansion is necessary. Note, that a normalization of the input does not necessarely resolve the above problem. The terms of the sum could appear as arguments of a z-function f for which no expansion rule is known, e.g. $f(t_1+t_2)+f(t_3)-f(t_1)-f(t_2+t_3)$. As soon as this function is expanded, the hidden zeros may appear.

These examples demonstrate, that different representations of the z-sums may lead to complicated, unsimplified expressions and to hidden zeros. As a consequence the computation of a Z-expansion becomes a very expensive step in this algorithm.

Z-expansion requires that the comparability classes of the argument are determined and ordered. In contrast to our approach however, Z-expansions are always only applied on *subexpressions*. Our algorithm, however, operates on the expression as a whole and always compares the comparability classes of *all* subexpressions. In Shackell's algorithm this situation only occurs if a Z-expansion of the given function must be computed. However, it turns out that due to the intermediate expression swell the algorithm to compute nested forms breaks down even though potentially less comparisons have to be performed. As a consequence, our algorithm usually runs faster than Shackell's. Computing the limit of $e^{1/x} - (e^{1/x} - 1)/(e^{1/x})$ for $x \to +\infty$ our algorithm is more than two times faster, and for computing the limit of $e^{e^x}/e^{e^{x-e^{-e^{e^x}}}}$ as x tends to $+\infty$ our algorithm is more than five times faster than the nested form algorithm. These timings give an indication only, as it is difficult to specify a fair set of examples. Note also that for most examples we tested, the trivial implementation 4.3 for computing a nested form using our limit facility *MrvLimit* runs faster than the direct implementation of the nested form algorithm in Maple.

Another difference of the two approaches is the complexity of the code of our implementations. The source code for Shackell's algorithm is about three times larger than the code we used for our algorithm.

Although the overall approaches of these two algorithms are different, the algorithm to compute nested forms uses some techniques which have their counterparts in our algorithm and vice versa. For example, when expanding a z-term Z in terms of ω we first have to separate the arguments of any z-function $f(x+y)$ in the case that x only depends on z-terms which are in a smaller comparability class than ω. In terms of our notation a similar separation has to be applied in the case that $mrv(y) = \omega$ and $\gamma(x) = \gamma(x+y) \prec \omega$ (note that in our situation x and y do not necessarily tend to zero). We simply expand f into the Taylor series around $\omega = 0$ in this situation and so automatically perform the separation of the arguments.

Another similarity is the determination of the right scale in which the series expansion has to be performed. In Shackell's algorithm this is done by setting to zero all z-prods which are in a greater comparability class than ω_i for a certain i and then by testing whether this expression is zero. The zero equivalence test for Z_k (i.e. for the first function which is not zero) is also performed by our algorithm, but in another context. When computing the power series in terms of the most rapidly varying subexpressions $\omega \succ \omega_k$, the leading exponent is zero and the leading coefficient is non-zero, which is confirmed using the oracle for deciding zero equivalence. However, the leading coefficient of the series in terms of ω_j for $j > k$ will be Z_k, and so the same zero equivalence test is done. The difference between the two approaches concerning the search for the right entry in the asymptotic scale is that we are searching from the most rapidly varying subexpression down to the correct one in our algorithm, whereas Shackell searches from the smallest comparability class up to the correct one.

Probably the most important difference between the two algorithms is that Shackell's algorithm really solves a much more complicated problem than merely computing limits, namely the computation of a nested form which contains information about the asymptotic behaviour of the given function. This normal form allows, for example, the computation of the nested expansion of the compositional inverse of the nested expansion of a given function f [73]. The conclusion of this comparison is that our approach is better suited for computing limits, in particular with a symbolic manipulation system, and that Shackell's algorithm has its own applications.

4.2 Ghost and Shadow Approach

The ghost and shadow approach is an extension of the algorithm to compute nested forms, and it is presented in [83, 82]. The ghost and shadow approach is an extension over nested forms as it runs in any asymptotic field [83]. The most interesting property of asymptotic fields is, that a real Liouvillian extension of an asymptotic field is also contained in an asymptotic field [83, Theorem 5]. This way, limits of Liouvillian functions can be computed.

However, the algorithm is similar to the nested form algorithm. The step of replacing comparability classes by zero during the Z-expansion in the nested form algorithm is replaced by the projection of the function onto a shadow field. The shadow of a function with respect to a given comparability class is obtained if all subexpressions of equal of higher growth are replaced by zero, and the ghost is the difference of the function and its shadow. Cancellation between shadows is also detected using a zero-equivalence algorithm.

Once the proper comparability class has been identified, an asymptotic series expansion is performed in this and lower comparability classes. This step is similar to the expansion in terms of the most rapidly varying subexpression in our algorithm.

The ghost and shadow approach is also a bottom-up recursive algorithm. As a consequence, the tails of all series expansions have to be retained in closed form – a property we identified to be problematic for the nested form approach. As soon as an implementation of the shadow and ghost approach is available, a comparison is possible.

The ghost and shadow approach however is very interesting from a theoretical point of view as it allows to extend the field of functions to Liouvillian functions, i.e., limits can be computed in a field given by a tower of extension of the basic constants by integrals, exponentials and real algebraic functions. It is a topic of further research whether these ideas can be combined with our approach.

What we will present in the the next section only shows how our algorithm can be extended to functions which fit into the model of our algorithm. It turns out, that almost all functions available in a symbolic manipulation system (e.g. trigonometric functions, error functions, gamma functions, etc.) can be covered with our algorithm, and thus, from a practical point of view, our algorithm is very interesting and already got implemented in several computer algebra systems.

5. Extensions

In Chapter 3 we described our algorithm for exp-log functions. In this chapter we investigate how we can extend the algorithm to work on a wider class of functions. Thereby we try to follow the lines of the algorithm for exp-log functions, i.e. we investigate, which of the properties of exp-log functions, which are needed by our algorithm, are also satisfied by other functions, or how other functions might be rewritten such that they can be covered by our algorithm. It turns out, that with this approach, surprisingly many functions can be handled.

Other algorithms have been presented which allow to compute limit for general classes of functions, e.g. for meromorphic functions [82] and for Liouvillian functions [83]. If the latter algorithm works over some function field, this field can be extended by any real Liouvillian extension, i.e. an exponential, an integral or an algebraic extension. These algorithms are very interesting from a theoretical point of view, but they still await their implementation in a computer algebra system.

Let $\mathcal{F}$ denote a field of germs of functions at x_0 in which limits can be computed with our algorithm for $x \to x_0$ and let $f \in \mathcal{F}$. In the first section we show that if g can be expanded into an (asymptotic) power series at the limit of f (which may be infinity), then our algorithm can also compute limits in $\mathcal{F}(g \circ f)$. In the next section we show how to deal with elements $g \circ f$ which have an exponential or a logarithmic singularity at the limit of f. We close this chapter with a definition of the *Mrv*$\mathcal{H}$ field in which we can compute limits with our algorithm and show which requirements of the algorithm prevent us from extending the function class which can be handled further.

5.1 Tractable Functions

Let us assume that $\mathcal{F}$ is a field of germs of functions in x at x_0 in which we can compute limits at $x = x_0$ with our algorithm and let $f \in \mathcal{F}$. In this section we show that our algorithm can be extended to handle the computation of the limit of $g \circ f$ for $x \to x_0$, provided that g is a function which is *tractable* at the limit of f.

Definition 5.1 *A function $f(x)$ is tractable at $x = a$ from the right or from the left if it does not have an essential singularity at a and if it can be expanded into a (one sided, asymptotic) power series at a, i.e. if for $\omega > 0$ the power series*

$$f(x) = \left\{ \begin{array}{l} f(a+\omega) \\ f(a-\omega) \\ f(\ \ 1/\omega) \\ f(-1/\omega) \end{array} \right\} = c_0\,\omega^{e_0} + c_1\,\omega^{e_1} + \cdots \left\{ \begin{array}{l} \text{for a finite } a \text{ from the right} \\ \text{for a finite } a \text{ from the left} \\ \text{for } a = +\infty \\ \text{for } a = -\infty \end{array} \right.$$

exists with $c_i \in \mathbb{C}, e_i \in \mathbb{R}$ and $e_{i+1} > e_i$.

Note that we allow real exponents in the power series expansion, not just integer or rational ones. To say that the power series is computable means that we have an algorithm to effectively compute the coefficients c_i and the exponents e_i. Examples of tractable functions are analytic and meromorphic functions. Note that the derivative of a tractable function is also tractable at $x = a$. Moreover, if g is assumed to be tractable at the limit of f, then the (one sided) limit of $g \circ f$ exists and is either finite or $\pm\infty$. This limit can be computed provided that the leading term of the series of g at the limit of f is known, that the limit of f can be determined and, in case that the latter is finite, that the sign of $f(x) - a$ can be determined, where a is the limit of f.

Let us discuss now how we have to extend our algorithm in order to be able to compute the limit of functions which contain $g(f(x))$ as a subexpression. Since g is assumed to be tractable at the limit of f we can define $mrv(g(f(x))) = mrv(f(x))$. As a consequence, $g(f(x))$ cannot appear in any mrv set and thus we do not have to discuss how to rewrite $g(f(x))$ in terms of another element in the same comparability class. Moreover, the tractability of g at the limit of f also implies that $g(f(x))$ can be expanded into an asymptotic power series at the limit of $f(x)$.

However, since we are eliminating the comparability classes according to their order, it may happen that g has not to be expanded at a (where a is the limit of f) but rather has to be expanded at a function whose limit is a. Such a situation occurs if $mrv(g(f(x))) \succ \Omega$ where

$$\Omega = \left\{ \begin{array}{ll} \gamma(f(x) - a) & \text{if } a \text{ is finite} \\ \gamma(f(x)) & \text{if } a \text{ is } \pm\infty \end{array} \right. .$$

The series expansion of $f(x)$ in terms of $\omega \in mrv(g(f(x)))$ then has the form $c_0 + h(\omega)$ with $h(\omega) = c_1\,\omega^{e_1} + O(\omega^{e_2})$, where $e_1 > 0$ and $\omega \succ mrv(c_0) \succeq \Omega \succ \gamma(1)$ and $c_0 \to a$. Thus, we have to be able to expand $g(c_0 + h(\omega))$ into a power series in terms of ω. The solution is the same as in the exp-log case, namely to expand $g(c_0 + h(\omega))$ into its Taylor series at $\omega = 0$, i.e.

$$\text{Series}(g(c_0 + h(\omega)), \omega) = g(c_0) + g'(c_0)\,h(\omega) + \frac{g''(c_0)}{2}\,h(\omega)^2 + \cdots .$$

This expansion is a proper asymptotic series as $mrv(g^{(n)}(c_0)) = mrv(c_0) \prec \omega$, as the derivatives of g are tractable as well.

What remains to be shown is that the extended algorithm still terminates. Termination as proven in Section 3.4 uses the size of a function as a variant function. Thus, we only have to extend the definition of this size appropriately. We recall that the size is a measure for the number of comparability classes which may emerge out of the expression during the rewriting and the series expansion steps. It is defined as the size of the set S containing all the expressions which potentially may be contained in an mrv set. Since a tractable function never appears in any mrv set, we can still use the same definition for the size of a function and simply define $S(g(f(x))) = S(f(x))$. It is easy to see that termination is still guaranteed then.

Examples of functions which are tractable at a finite argument are all trigonometric functions, $\Gamma(x)$, $\psi(x)$[1], $\zeta(x)$[2], $\operatorname{erf}(x)$, $\operatorname{Si}(x)$[3] and some Bessel functions. $\arctan(x)$ is additionally tractable at $x = \pm\infty$. The functions $\operatorname{dilog}(x)$[4], $\mathrm{W}(x)$[5], $\operatorname{Ci}(x)$[6] are tractable for finite positive arguments, and $\operatorname{Ei}(x)$[7] is tractable for finite real arguments not equal to zero.

Example 5.2 We are able now to show how our algorithm computes the limit

$$\lim_{x\to+\infty} e^x \left(\sin\left(1/x + e^{-x}\right) - \sin\left(1/x\right)\right), \tag{5.1}$$

which we introduced in Example 2.17. The set of most rapidly varying subexpression is $\{e^x, e^{-x}\}$. If we replace e^{-x} by ω we obtain

$$\frac{1}{\omega}\left(\sin(1/x+\omega) - \sin(1/x)\right). \tag{5.2}$$

Now we have a situation where $mrv(\sin(1/x+\omega)) = \omega \succ \gamma(1/x+\omega) = \gamma(x)$ and we have to expand $\sin(1/x+\omega)$ into its Taylor series at $\omega = 0$,

$$\sin(1/x+\omega) = \sin(1/x) + \cos(1/x)\,\omega - \frac{\sin(1/x)}{2}\,\omega^2 + O(\omega^3) \tag{5.3}$$

and the series expansion of (5.2) becomes

$$\cos(1/x)\,\omega^0 - \frac{\sin(1/x)}{2}\,\omega^1 - \frac{\cos(1/x)}{6}\,\omega^2 + \frac{\sin(1/x)}{24}\,\omega^3 + O(\omega^4). \tag{5.4}$$

[1] $\psi(x)$ is the digamma or zero'th polygamma function defined as $\Gamma'(x)/\Gamma(x)$. The n'th derivative of $\psi(x)$ is the n'th polygamma function.

[2] $\zeta(x)$ is the Riemann Zeta function.

[3] $\operatorname{Si}(x)$ is the sine integral defined as $\int_{t=0}^{x} \sin(t)/t\,dt$.

[4] dilog is the dilogarithm function and defined as $\operatorname{dilog}(x) = \int_1^x \ln t/(1-t)\,dt$.

[5] $\mathrm{W}(x)$ is Lambert's W function defined as the principal branch of the solution of $\mathrm{W}(x)e^{\mathrm{W}(x)} = x$. See [19] for a reference.

[6] $\operatorname{Ci}(x)$ is the cosine integral defined as $\gamma + \ln x + \int_{t=0}^{x}(\cos t - 1)/t\,dt$.

[7] $\operatorname{Ei}(x)$ is the exponential integral defined as $\int_{-\infty}^{x} e^t/t\,dt$.

The leading term of (5.4) is $\cos(1/x) \cdot \omega^0$ and so the limit (5.1) is one, which could be derived with a further iteration on $\cos(1/x)$. ¶

If we have extended the function field by a tractable function, then we can always close the new function field by the exponential and the logarithm functions. Our algorithm can compute limits in this closed function field, as the computation of the most rapidly varying subexpression as well as the series expansion is always computable.

As an example we can now explain how our algorithm solves the limit $\lim_{x\to 0} e^{\csc x}/e^{\cot x}$, which we have already met in Section 2.3.2. Both, the numerator and the denominator have an essential singularity, but both, $\csc(x)$ and $\cot(x)$ are tractable at $x = 0$. We have realized that the power series approach can only solve this problem if we simplify the function first using a heuristic, although $e^{\csc x}/e^{\cot x}$ is analytic at $x = 0$. In this particular example the combination of the two exponentials would do the job. Let us look how our algorithm solves this problem.

Example 5.3 Let us compute the limit

$$\lim_{x\to 0} \frac{e^{\csc x}}{e^{\cot x}}. \tag{5.5}$$

Let us first assume that zero is approached from the right. The set of most rapidly varying subexpressions is $\{e^{\csc x}, e^{\cot x}\}$ and we have to rewrite one in terms of the other. Let us first choose $\omega = e^{-\cot x}$. $e^{\csc x}$ can then be written as $e^{\csc x - \cot x}\,\omega^{-1}$ and $e^{\cot x}$ as ω^{-1}. The rewritten function becomes

$$\frac{e^{\csc x - \cot x}\,\omega^{-1}}{\omega^{-1}},$$

and the leading coefficient of the series in ω is $e^{\csc x - \cot x}$. Conversely, if we choose $\omega = e^{-\csc x}$, the rewritten function becomes

$$\frac{\omega^{-1}}{e^{\cot x - \csc x}\omega^{-1}},$$

and the leading coefficient of the series in ω is $1/e^{\cot x - \csc x}$. The same result is obtained if zero is approached from the left.

We see, that our algorithm *automatically* performs the transformation we have proposed in Section 2.3.2. The most rapidly varying subexpression of both $e^{\csc x - \cot x}$ and $1/e^{\cot x - \csc x}$ is x as the limit of $\csc x - \cot x$ is zero. The latter limit is computed recursively by a simple series expansion. Therefore both leading coefficients $e^{\csc x - \cot x}$ and $1/e^{\cot x - \csc x}$ can directly be expanded into a power series around $x = 0$ and the limit (5.5) turns out to be 1. ¶

5.2 Essential Singularities

If the argument of the function f tends to a limit a as $x \to x_0$ at which f is not tractable, then it cannot be expanded into a power series and the algorithm cannot be applied without modification. This happens for instance if the function f has an essential singularity at a. In this section we show how exponential and logarithmic singularities can be handled. The idea is not to change the algorithm but rather to rewrite f into a function where the singularities appear explicitly and which contains only tractable functions besides of exponentials and logarithms.

If f only contains a logarithmic singularity at a, then we are in a special situation. The coefficients of the (asymptotic) power series of $f(a+\omega)$ at $\omega = 0$ are not constants then but rather depend on $\ln(\omega)$. However, whenever $f(a+\omega)$ is expanded in the context of our algorithm in terms of ω, $\omega = e^h$ is an exponential and thus $\ln(\omega)$ gets simplified to h and no longer depends on ω, and furthermore, $mrv(\ln(\omega)) = mrv(h) \prec \omega$ according to Lemma 3.20. We met the same situation already for the series expansion of the logarithm function in Section 3.3.3. We denote a function f to be *semi-tractable* at a, if the coefficients of the asymptotic power series of $f(a+\omega)$ only depend on $\ln(\omega)$. The derivatives of semi-tractable functions are semi-tractable at a as well. Concerning termination we must add that out of a semi-tractable function two comparability classes may evolve, and in order to be able to specify an upper bound for the number of iterations we have to define $S(f(g(x))) = S(g(x)) \cup \{\ln(g(x))\}$ in case that f is semi-tractable. Examples of semi-tractable functions are $\mathrm{Ei}(x)$ at $x = 0$, $\psi(x)$ at $x = +\infty$, $\mathrm{dilog}(x)$ at $x = +\infty$ and at $x = 0^+$, $\mathrm{Ci}(x)$ at $x = 0$ and some Bessel functions at $x = 0^+$.

All other essential singularities cannot directly be handled by our algorithm. However, if we succeed in isolating the essential singularities in a pre-processing step, then the limit becomes solvable. The transformed expression has to be composed of semi-tractable functions only, composed with exponential and logarithm functions. For all functions which have essential singularities, such a transformation has to be provided for each point where the essential singularity occurs. The error function $\mathrm{erf}(x)$ for example, which has an essential singularity at $x = +\infty$, may be transformed to $1 - \mathrm{erf}_s(x)/\exp(x^2)$ where $\mathrm{erf}_s(x) = (1 - \mathrm{erf}(x))\exp(x^2)$ is tractable at $x = +\infty$.

We give below a list of possible transformations for functions which have essential singularities at $+\infty$:

$$\mathrm{erf}(x) \to 1 - e^{-x^2}\,\mathrm{erf}_s(x) \tag{5.6}$$
$$\mathrm{Ei}(x) \to e^{x}\,\mathrm{Ei}_s(x) \tag{5.7}$$
$$\Gamma(x) \to e^{\Gamma_s(x)} \tag{5.8}$$
$$\mathrm{W}(x) \to \mathrm{W}_s(\ln x) \tag{5.9}$$

$$\begin{aligned}
\zeta(x) &\rightarrow \zeta_s(e^x) && (5.10)\\
\mathrm{Ai}(x)^* &\rightarrow \frac{1}{2} e^{-2/3\sqrt{x^3}} \left(\pi\sqrt{x}\right)^{-1/2} \mathrm{Ai}_s(x) && (5.11)\\
\mathrm{Bi}(x)^\dagger &\rightarrow e^{2/3\sqrt{x^3}} \left(\pi\sqrt{x}\right)^{-1/2} \mathrm{Ai}_s(x) && (5.12)\\
\mathrm{li}(x)^\ddagger &\rightarrow x\,\mathrm{Ei}_s(\ln x) && (5.13)
\end{aligned}$$

where erf_s, Ei_s, Γ_s, etc., are (semi-)tractable functions which are defined by those transformation rules. Obviously, several transformations are possible to isolate an essential singularity for a given function at a particular point. For example, $\Gamma(x)$ at $+\infty$ could also be transformed with

$$\Gamma(x) \rightarrow e^{x \ln x}\, e^{-x}\, \hat{\Gamma}_s(x) \qquad (5.14)$$

at $x = +\infty$ where $\hat{\Gamma}_s(x)$ is tractable, but obviously different from $\Gamma_s(x)$.

For $\mathrm{Ei}(x)$ and $\mathrm{erf}(x)$ the following transformations might be used to separate the essential singularity at $x = -\infty$.

$$\begin{aligned}
\mathrm{Ei}(x) &\rightarrow e^x\, \mathrm{Ei}_s(x) && (5.15)\\
\mathrm{erf}(x) &\rightarrow -1 + e^{-x^2}\, \mathrm{erf}_s(-x) && (5.16)
\end{aligned}$$

In order to apply the algorithm to these new tractable functions $f_s(x)$, their asymptotic power series expansion has to be known. For the above examples, these power series at $x = +\infty$ are

$$\begin{aligned}
\mathrm{erf}_s(x) &= \frac{1}{\sqrt{\pi}} \sum_{k=0}^{\infty} \frac{(2k)!\,(-4)^{-k}}{k!} \left(\frac{1}{x}\right)^{2k+1}\\
&= \frac{1}{\sqrt{\pi}}\, x^{-1} - \frac{1}{2\sqrt{\pi}}\, x^{-3} + O(x^{-5}) && (5.17)\\
\mathrm{Ei}_s(x) &= \sum_{k=0}^{\infty} k! \left(\frac{1}{x}\right)^{k+1} = x^{-1} + x^{-2} + 2x^{-3} + O(x^{-4}) && (5.18)\\
\mathrm{Ai}_s(x) &= 1 - \frac{5}{48} x^{-3/2} + \frac{385}{4608}\, x^{-3} - \frac{85085}{663552}\, x^{-9/2} + O(x^{-6}) && (5.19)\\
\zeta_s(x) &= 1 + x^{-\ln 2} + x^{-\ln 3} + x^{-\ln 4} + O(x^{-\ln 5}) && (5.20)\\
\Gamma_s(x) &= (\ln x - 1)\, x + \frac{\ln 2\pi}{2} - \frac{\ln x}{2} + \frac{1}{12}\, x^{-1} - \frac{1}{360}\, x^{-3} + O(x^{-5}) && (5.21)\\
\hat{\Gamma}_s(x) &= \sqrt{2\pi}\, x^{-1/2} + \sqrt{2\pi}\frac{1}{12}\, x^{-3/2} + \sqrt{2\pi}\frac{1}{288}\, x^{-5/2} + O((x^{-7/2}) && (5.22)\\
\mathrm{W}_s(x) &= x - \ln x + \ln x\, x^{-1} + \frac{1}{2} \ln x(\ln x - 2)\, x^{-2} + O(x^{-3}) && (5.23)
\end{aligned}$$

[*] $\mathrm{Ai}(x)$ is the Airy wave function Ai, [3, (10.4.1)].
[†] $\mathrm{Bi}(x)$ is the Airy wave function Bi, [3, (10.4.1)].
[‡] $\mathrm{li}(x)$ is the logarithmic integral, [3, (5.1.3)].

Note that $\Gamma_s(x)$ and $\mathrm{W}_s(x)$ are actually semi-tractable.

For the semi-tractable functions $f_s(x)$ we must also be able to compute the Taylor series of $f_s(g(x))$ in case that $mrv(g(x)) \succ \gamma(g(x))$, and the derivatives are therefore needed.

$$
\begin{aligned}
\mathrm{erf}_s'(x) &= -\frac{2}{\sqrt{\pi}} + 2\,\mathrm{erf}_s(x)\,x && (5.24)\\
\mathrm{Ei}_s'(x) &= \frac{1}{x} - \mathrm{Ei}_s(x) && (5.25)\\
\mathrm{W}_s'(x) &= \frac{\mathrm{W}_s(x)}{1+\mathrm{W}_s(x)} && (5.26)\\
\Gamma_s{}'(x) &= \psi(x) && (5.27)
\end{aligned}
$$

Since $\zeta(x)$, $\Gamma(x)$ and $\psi(x)$ do not satisfy an algebraic differential equation, the derivatives thereof cannot be expressed in terms of these functions themselves. However, the derivatives still exist and are tractable.

We close this section with some examples which demonstrate how our limit algorithm computes limits of expressions which contains functions with essential singularities. The behaviour of other algorithms on these problems is shown in Chapter 8.

Example 5.4 The following example illustrates this pre-processing step for an easy problem. Consider

$$\lim_{x\to+\infty} \mathrm{Ei}(x+e^{-x})\,e^{-x}\,x.$$

The argument of the exponential integral tends to $+\infty$ as $x \to +\infty$ and we transform the function using the rule (5.7):

$$\mathrm{Ei}(x+e^{-x})\,e^{-x}\,x \to e^{x+e^{-x}}\;\mathrm{Ei}_s\left(x+e^{-x}\right)\;e^{-x}\,x =: u.$$

$mrv(u) = \{e^{-x}, e^{x+e^{-x}}\}$ and we set $\omega = e^{-x}$. The second element in $mrv(u)$ becomes $e^{x+e^{-x}} = e^{\omega}\,\omega^{-1}$. After the rewriting step we must expand $e^{\omega}\;\mathrm{Ei}_s(x+\omega)\,x$ into a series at $\omega = 0^+$. Note that we have $mrv(\mathrm{Ei}_s(x+\omega)) \asymp \omega \succ \gamma(x+\omega)$ and thus $\mathrm{Ei}_s(x+\omega)$ has to be expanded into a Taylor series in ω at $\omega = 0$, although x by itself tends to $+\infty$. With (5.25) we get for the series expansion of u

$$\mathrm{Series}(e^{\omega}\;\mathrm{Ei}_s(x+\omega)\,x, \omega) = \mathrm{Ei}_s(x)\,x + \omega + \frac{x-1}{2\,x}\,\omega^2 + O(\omega^3).$$

The leading term is $\mathrm{Ei}_s(x)\,x$ whose most rapidly varying subexpression is x. After moving up one level we have to compute the series expansion of $\mathrm{Ei}_s(1/\omega)/\omega$. We can now use the series expansion (5.18) and get

$$\mathrm{Series}(\mathrm{Ei}_s(1/\omega)/\omega, \omega) = \sum_{k=0}^{\infty} k!\,\omega^k = 1 + \omega + 2\,\omega^2 + O(\omega^3)$$

and hence $\lim\limits_{x\to+\infty} \mathrm{Ei}(x+e^{-x})\,e^{-x}\,x = 1.$ ¶

Example 5.5 In this example we compute the limit

$$\lim_{x\to+\infty} \frac{\ln(\Gamma(\Gamma(x)))}{e^x}.$$

Since the argument of the inner Gamma function tends to infinity we rewrite it to $e^{\Gamma_s(x)}$. The limit thereof is infinity as well (cf. with the asymptotic expansion (5.21)) and our original expression gets transformed to

$$\frac{\ln(\Gamma(\Gamma(x)))}{e^x} \to \frac{\ln e^{\Gamma_s(e^{\Gamma_s(x)})}}{e^x} = \frac{\Gamma_s(e^{\Gamma_s(x)})}{e^x} =: u.$$

For computing the *mrv* set we must compare $e^{\Gamma_s(x)}$ with e^x. The leading term of the asymptotic series of the quotient of the logarithms of the two is $\ln x$ and thus $e^{\Gamma_s(x)} \succ e^x$ and we set $e^{\Gamma_s(x)} = 1/\omega_1$. $\ln(1/\omega_1)$ thus becomes $\Gamma_s(x)$. The series expansion of the rewritten function u in terms of ω_1 is

$$\frac{\Gamma_s(1/\omega_1)}{e^x} = \frac{\ln(1/\omega_1)-1}{e^x}\,\omega_1^{-1} + O(1) = \frac{\Gamma_s(x)-1}{e^x}\,\omega_1^{-1} + O(1).$$

As a consequence $\nu(\ln(\Gamma(\Gamma(x)))/e^x) < 0$. In order to determine the sign of the leading coefficient we have to continue the iteration with the leading coefficient $(\Gamma_s(x)-1)/e^x$. The most rapidly varying subexpression thereof is e^x and the leading coefficient of the series in $\omega_2 = e^{-x}$ is $\Gamma_s(x)-1$. Strictly following the path of the algorithm we have to move up one level since $mrv(\Gamma_s(x)-1) = \{x\}$. The series of $\Gamma_s(e^x)-1$ in $\omega_3 = e^{-x}$ is

$$\Gamma_s(1/\omega_3) - 1 = (\ln(1/\omega_3)-1)\,\omega_3^{-1} + O(1) = (x-1)\,\omega_3^{-1} + O(1)$$

as $\ln(1/\omega_3) = -\ln(\omega_3) = x$. When moving up the leading coefficient $x-1$ once again and expanding it into a series we get a series with leading coefficient one. Therefore $(\Gamma_s(x)-1)/e^x$ is ultimately positive and

$$\lim_{x\to+\infty} \frac{\ln(\Gamma(\Gamma(x)))}{e^x} = +\infty.$$

¶

Example 5.6 In this example we want to solve the dominance problem between x and $\exp(\exp(\exp(\psi(\psi(\psi(x))))))$, i.e. we want to compute the limit

$$\lim_{x\to+\infty} \frac{e^{e^{e^{\psi(\psi(\psi(x)))}}}}{x}.$$

This example is mentioned in [72] as an example which has to be performed in a very general asymptotic scale. Since $\psi(x)$ is semi-tractable at $x = +\infty$, no pre-processing is necessary. The most difficult step in this example is the computation of the *mrv* set of u where $u = \exp(\exp(e^f))/x$ with $f = \psi(\psi(\psi(x)))$. $mrv(f) = \{x\}$ and according to Algorithm 3.12 we have to compare e^f with x to determine $mrv(e^f)$. This is done by computing the limit of $f/\ln(x)$. To determine this limit we must move up one level and expand the series in terms of $\omega = e^{-x}$. The leading term thereof is $\psi(\psi(x))/x \cdot \omega^0$ and we have to move up a further level. The leading term of the series expansion of the latter leading coefficient in terms of $\omega = e^{-x}$ is $\psi(x)\,\omega \to 0$ and as a consequence $e^f \prec x$ and $mrv(e^f) = \{x\}$.

We similarly get $mrv(e^{e^f}) = \{x\}$ and it remains to compare $e^{e^{e^f}}$ with x. This is done by computing the limit of $e^{e^f}/\ln(x)$. The leading coefficient of the series after moving up the first time is $e^{e^{\psi(\psi(x))}}/x$. The *mrv* set thereof however is $\{e^{e^{\psi(\psi(x))}}, x\}$ as the limit of $e^{\psi(\psi(x))}/\ln(x)$ is 1. Thus, we have to move up once again and rewrite the first element of the latter *mrv* set in terms of the second one which leads to the leading coefficient $e^{e^{\psi(x)}-x}$ after the series expansion. The *mrv* set thereof is $\{e^{\psi(x)}, x\}$ as $\psi(x)/\ln(x)$ tends to 1. We move up once more and rewrite the elements in the *mrv* set and get $e^{(e^{\psi(1/\omega)-x}-1)/\omega}$. The series expansion thereof is $e^{-1/2} + e^{-1/2}/24 \cdot \omega + O(\omega^2)$ and thus $e^{e^{e^f}}$ and x are in the same comparability class. As a consequence

$$S = mrv(u) = mrv\left(\frac{e^{e^{e^f}}}{x}\right) = \left\{x, e^{e^{e^f}}\right\}.$$

Since $x \in S$ we have to move up one level and as $|S| = 2$ we must rewrite the second element in terms of e^x. The limit of $e^{e^f}/\ln(x)$ is $e^{-1/2}$ as we have seen above and the rewritten function u in terms of $\omega = e^{-x}$ becomes

$$u_1 = e^{\left(e^{e^{\psi(\psi(\psi(1/\omega)))}}\right) - e^{-1/2}x}\,\omega^{1-e^{-1/2}}.$$

The series of u_1 at $\omega = 0$ is

$$\mathrm{Series}(u_1, \omega) = e^{\left(e^{e^{\psi(\psi(x))}}\right) - e^{-1/2}x}\,\omega^{1-e^{-1/2}} + O\left(\omega^{2-e^{-1/2}}\right)$$

and since $1 - e^{-1/2} > 0$ we have

$$\lim_{x \to +\infty} \frac{e^{e^{e^{\psi(\psi(\psi(x)))}}}}{x} = 0.$$

¶

Example 5.7 This is an example where cancellations would appear in two different comparability classes if a classical algorithm based on generalized series expansions were applied. Moreover, it shows how functions are handled which do not satisfy a differential equation.

$$\lim_{x \to +\infty} \frac{\Gamma\left(x+e^{-x}+1/\Gamma(x)\right)-\Gamma(x+e^{-x})-\psi(x)}{e^{-x}\,(\ln x)^2} \tag{5.28}$$

After the pre-processing step we have to compute the limit of

$$u = \frac{e^{\Gamma_s\left(x+e^{-x}+e^{-\Gamma_s(x)}\right)} - e^{\Gamma_s(x+e^{-x})} - \psi(x)}{e^{-x}\,(\ln x)^2}$$

The size of this function is eight as both Γ_s and ψ are semi-tractable, and

$$\begin{aligned} S(u) \;=\; & \Big\{ e^{\Gamma_s\left(x+e^{-x}+e^{-\Gamma_s(x)}\right)}, e^{\Gamma_s(x+e^{-x})}, e^{-\Gamma_s(x)}, e^{-x}, x, \ln x, \\ & \qquad \ln\left(x+e^{-x}+e^{-\Gamma_s(x)}\right), \ln\left(x+e^{-x}\right) \Big\}. \end{aligned}$$

Thus we expect to get the result after at most eight iterations.

The set of most rapidly varying subexpressions of u is

$$mrv(u) = \{e^{\Gamma_s\left(x+e^{-x}+e^{-\Gamma_s(x)}\right)}, e^{\Gamma_s(x+e^{-x})}, e^{-\Gamma_s(x)}\}$$

and we rewrite all elements in terms of the last one, which tends to zero and which we call ω_1:

$$\begin{aligned} e^{\Gamma_s\left(x+e^{-x}+e^{-\Gamma_s(x)}\right)} &= e^{\Gamma_s\left(x+e^{-x}+e^{-\Gamma_s(x)}\right)-\Gamma_s(x)}\,\omega_1^{-1} \\ e^{\Gamma_s(x+e^{-x})} &= e^{\Gamma_s(x+e^{-x})-\Gamma_s(x)}\,\omega_1^{-1}. \end{aligned}$$

The rewritten expression becomes

$$u_1 = \frac{e^{\Gamma_s\left(x+e^{-x}+\omega_1\right)-\Gamma_s(x)}\,\omega_1^{-1} - e^{\Gamma_s(x+e^{-x})-\Gamma_s(x)}\,\omega_1^{-1} - \psi(x)}{e^{-x}\,(\ln x)^2},$$

and we have a cancellation between $e^{\Gamma_s(x+e^{-x}+\omega_1)-\Gamma_s(x)}$ and $e^{\Gamma_s(x+e^{-x})-\Gamma_s(x)}$. The leading term of the difference of these two functions vanishes. With the derivative (5.27) for $\Gamma_s(x)$ the series expansion of u_1 in ω_1 becomes

$$\begin{aligned} u_1 &= \frac{\left(e^{\Gamma_s(x+e^{-x})-\Gamma_s(x)}\left(1+\Gamma_s{}'(x+e^{-x})\,\omega_1+O(\omega_1^2)\right)-e^{\Gamma_s(x+e^{-x})-\Gamma_s(x)}\right)\omega_1^{-1}-\psi(x)}{e^{-x}\,(\ln x)^2} \\ &= \frac{e^{\Gamma_s(x+e^{-x})-\Gamma_s(x)}\,\Gamma_s{}'(x+e^{-x})-\psi(x)}{e^{-x}\,(\ln x)^2} + O(\omega_1) \\ &= \frac{e^{\Gamma_s(x+e^{-x})-\Gamma_s(x)}\psi(x+e^{-x})-\psi(x)}{e^{-x}\,(\ln x)^2} + O(\omega_1). \end{aligned}$$

The *mrv* set of the leading coefficient of the above series expansion is $\{e^{-x}\}$ and we expand it into a series in $\omega_2 = e^{-x}$. Note that we have cancellations between $\Gamma_s(x+\omega_2)$ and $\Gamma_s(x)$ and additionally between $\psi(x+\omega_2)$ and $\psi(x)$.

$$\begin{aligned}
u_2 &= \frac{e^{\Gamma_s(x+\omega_2)-\Gamma_s(x)}\psi(x+\omega_2)-\psi(x)}{\omega_2\,(\ln x)^2} \\
&= \frac{(1+\Gamma_s{}'(x)\omega_2+O(\omega_2^2))\big(\psi(x)+\psi'(x)\omega_2+O(\omega_2^2)\big)-\psi(x)}{\omega_2\,(\ln x)^2} \\
&= \frac{\psi'(x)+\Gamma_s{}'(x)\psi(x)}{(\ln x)^2}+O(\omega_2) \\
&= \frac{\psi'(x)+\psi(x)^2}{(\ln x)^2}+O(\omega_2).
\end{aligned}$$

The *mrv* set of the latter leading coefficient is $\{x\}$ and we thus move up one level and expand it in $\omega_3 = e^{-x}$.

$$\begin{aligned}
u_3 &= \frac{\psi'(1/\omega_3)+\psi_s(1/\omega_3)^2}{x^2} \\
&= 1+\frac{1-x}{x^2}\omega_3+O(\omega_3^2)
\end{aligned}$$

and the limit of (5.28) is one. We needed only three iteration steps instead of the possible eight. The size of a function generally significantly overestimates the number of comparability classes which may evolve out of it, but does provide an upper bound. ¶

5.3 *MrvH* fields

As a result of the ideas presented in the last two sections we are now able to specify the class of function in which our algorithm can compute limits. We call a function field $\mathcal{F}$ an *MrvH* field if it has all the properties required for our algorithm to compute $\lim\limits_{x\to x_0} f(x)$ for $f \in \mathcal{F}$.

Definition 5.8 *A field $\mathcal{F}$ of germs of functions at x_0 is an MrvH field if*

(1) $\mathcal{F}$ is a Rosenlicht field;

(2) if $f(g(x)) \in \mathcal{F}$ then $g(x) \in \mathcal{F}$ and one of the following holds

- *(a) $f = \exp$, and if $g(x)$ tends to infinity then $g(x) \in \mathbb{R}$,*
- *(b) f is ultimately semi-tractable, i.e. computable semi-tractable, at the limit of $g(x)$, and the coefficients of the series have to be in $\mathcal{F}$,*
- *(c) $f(g(x))$ can be transformed into an expression which only contains functions of types (a) and (b);*

(3) An oracle is available in $\mathcal{F}$ to decide zero-equivalence for any $f \in \mathcal{F}$.

(4) $f \in \mathcal{F}$ can be represented in finite terms as an expression tree.

A *Rosenlicht* field is a Hardy field which also contains any real power of any positive element and which has finite rank [79]. It might be useful to consider real powers of positive elements and not only rational ones. Note that even if $\mathcal{H}$ has finite rank it may happen that the rank of the new field obtained by adding real powers of any element may be infinite (see e.g. [82]). To prevent such a situation we require the Hardy field to contain real powers *and* to be of finite rank.

Our algorithm cannot work in Hardy fields of infinite rank since we must be able to specify the largest comparability class, which is not possible in a Hardy field of infinite rank in general. Furthermore, a Hardy field of infinite rank may contain functions with essential singularities which cannot be transformed into any finite exp-log function [9].

The condition that an *Mrv$\mathcal{H}$* field $\mathcal{F}$ has to be a *Hardy* field implies that the limit for $f \in \mathcal{F}$ exists and furthermore that the comparability classes in $\mathcal{F}$ can be compared.

The condition that an *Mrv$\mathcal{H}$* field $\mathcal{F}$ has to be a *field* implies that the series expansions can always be performed. If the series is computed in terms of ω, then the coefficients, which are elements in $\mathcal{F}$, have to be invertible.

An example of functions which are not contained in a Hardy field and as a consequence cannot be handled with our algorithm are functions which have oscillating essential singularities, such as the trigonometric functions at $x = \pm\infty$ or $\Gamma(x), \psi(x)$ and $\mathrm{Ai}(x)$ at $x = -\infty$. If the field contains such functions it may no longer be possible to compare comparability classes. For example $e^{x \sin x}$ oscillates between the two comparability classes $\gamma(e^x)$ and $\gamma(1)$. The latter problem could be resolved using a variant of interval calculus. The comparability class of a subexpression could be described by an interval of the smallest and the largest comparability class the function may belong to.

More difficult is to decide whether the coefficients of the series expansions are defined in a neighbourhood of infinity if $\mathcal{F}$ is not a field, i.e. to decide whether they do not have arbitrary large zeros. Consider

$$\lim_{x \to +\infty} \frac{1}{\cos x} e^{-x}.$$

If we take e^{-x} as most rapidly varying subexpression then the series expansion in terms of $\omega = e^{-x}$ is $1/\cos(x)\,\omega$. As the leading coefficient however has arbitrary large zeros, this limit is not defined. In practice it is difficult to decide whether the leading coefficient is defined for arbitrary large x. Shackell proposed in [75] to use an interval calculus approach. With this method our algorithm could be extended to work over domains which are not fields as well.

Mrv$\mathcal{H}$ fields are a proper subset of Rosenlicht fields. An example is $\mathcal{G} = \mathbb{R}(x, \Gamma(x + e^{-x} \sin x))$ which is a Hardy field of finite rank but which is not contained in any *Mrv$\mathcal{H}$* field (see [82]). It is not easy to decide automatically that $\mathcal{G}$ is a Hardy field.

Piecewise semi-tractable functions are also contained in an *Mrv$\mathcal{H}$* field as they are ultimately semi tractable. Examples are $|x|$, $\lfloor x \rfloor$, $\lceil x \rceil$, $[x]$, $\max(f_1(x), f_2(x))$, $\min(f_1(x), f_2(x))$ and others. In practice, these functions are also transformed during the pre-processing step. For example for $\max(f_1(x), f_2(x))$ it is decided whether $f_1(x)$ or $f_2(x)$ is ultimately larger and the max function is replaced by the larger one.

In an *Mrv$\mathcal{H}$* field we must also be able to decide zero equivalence. This problem is central to symbolic computation involving transcendental functions [59, 60, 15, 55, 41, 22, 81] and to handle it in practice a number of partial algorithms have been suggested.

One method is based on evaluations of the function f at a number of points. Mathematically, this can be viewed as a homomorphism from the function field containing f onto some field on which the zero-equivalence problem can be solved. There are two obvious choices for the latter field, namely a finite field of integers or the pseudo field of floating point numbers. The former choice has been developed in detail by Martin [51] under the name of *hash coding,* and has been extended in many directions [29, 30, 54, 74]. The mapping onto the field of floating point numbers has been investigated by Oldenhoeft in [57]. If an upper bound of necessary point evaluations is not available, or if this bound is beyond practical borders, the result is a probabilistic algorithm for deciding zero equivalence. Such an algorithm is always right if the answer returned says that the expression is not zero, but may be wrong with a probability $\varepsilon > 0$ in the other case.

Note, that in many cases the zero equivalence problem of functions can be reduced to the zero equivalence problem of constants, provided that the functions satisfy an algebraic differential equation over the constants.

An attractive alternative to solve the zero equivalence problem for constants is to assume *Schanuels conjecture* [6]: If $z_1, \ldots, z_n$ are complex numbers which are linearly independent over the rationals, then $\{z_1, \ldots, z_n, e^{z_1}, \ldots, e^{z_n}\}$ has transcendence rank at least n. It has been shown for larger and larger sets of constants, that the zero equivalence problem is decidable provided that Schanuel's conjecture is true [17, 61, 62, 63].

6. Asymptotic Series

As a byproduct of the limit computation algorithm we obtain an algorithm for computing asymptotic series. We discuss some aspects of this algorithm in this chapter. Note that this algorithm has already been presented in [31]. As it is based on the same ideas as the *MrvLimit* algorithm, we call the algorithm for computing asymptotic series *MrvAsympt*.

Definition 6.1 (Asymptotic Series) *An asymptotic series at infinity of a function f which is defined in a neighbourhood of infinity is a series of the form*

$$f(x) = c_1\,\varphi_1(x) + c_2\,\varphi_2(x) + \cdots + c_k\,\varphi_k(x) + r(x) \tag{6.1}$$

where the $c_i \in \mathbb{R}$ are constants and $\lim_{x\to+\infty} r(x)/\varphi_k(x) = 0$. The φ_i are functions of an asymptotic scale S which is a set of real-valued functions defined in a neighbourhood of infinity which is totally ordered according to the relation $i < j \Rightarrow \varphi_j(x)/\varphi_i(x) \to 0$ as $x \to +\infty$.

Asymptotic series of this form are also said to be of Poincaré type. More general expansions are obtained if the coefficients c_i are allowed to be functions. An example of an asymptotic scale is $\{x^k \mid k \in \mathbb{Z}\}$. From the asymptotic scale S we expect that it is as expressive as possible, i.e. one should immediately see the behavior of the function f at infinity when looking at its asymptotic series. Very expressive elements are nested exponentials, as e.g. $e^{x\ln x}$ or $e^{e^{\ln^3 \ln x}}$. However, not all functions can be expanded into an asymptotic series in such a scale. For example

$$\begin{aligned} f &= \exp\left(\exp\left(e^x/(1-1/x)\right)\right) \qquad (6.2)\\ &= \exp\left(\exp\left(e^x\left(1 + x^{-1} + x^{-2} + \cdots + x^{-k} + \cdots\right)\right)\right)\\ &= \exp\left(e^{e^x}\, e^{e^x/x}\, e^{e^x/x^2} \cdots e^{e^x/x^k} \cdots\right), \end{aligned}$$

can neither be expanded into a sum of simple exp-log functions nor be represented as an infinite product of simple exp-log functions. The only scale in which this function can be expanded easily is a scale which contains f itself, and then the asymptotic series is $1 \cdot f$. Note that these considerations led to the definition of nested forms and nested expansions in [79].

6.1 The *MrvAsympt* Algorithm

Let f be a pre-processed element of an *Mrv$\mathcal{H}$* field. We first determine $\Omega = mrv(f)$. If $x \in \Omega$ then we move up one level by replacing x by e^x and compute the asymptotic series for $f(e^x)$. The asymptotic series for $f(x)$ can then be obtained by moving back, i.e. by replacing x by $\ln(x)$. If $x \notin \Omega$ we choose $\omega_1 = e^h$ so that $\omega_1 \asymp \Omega$, $\omega_1 \to 0$ and ω_1 or $1/\omega_1 \in \Omega$. f is then rewritten and expanded in terms of ω_1. The result is an asymptotic series with coefficients which are in smaller comparability classes than ω_1. Then we can recursively expand these coefficients one by one into their asymptotic series in terms of their most rapidly varying subexpressions. We stop the recursion as soon as the coefficients are constants and stop the iteration as soon as we have computed enough terms for the approximation. A typical situation for the computation of the asymptotic series of a function f is shown in the following diagram.

$$\begin{tikzcd}
f \arrow[d, "mrv(f) = \omega_1"] & \\
c_{11}\,\omega_1^{e_{11}} + c_{12}\,\omega_1^{e_{12}} + c_{13}\,\omega_1^{e_{13}} + O(\omega_1^{e_{14}}) \arrow[d, "mrv(c_{11}) = \omega_2 \prec \omega_1"] & \\
c_{21}\,\omega_2^{e_{21}} + c_{22}\,\omega_2^{e_{22}} + O(\omega_2^{e_{23}}) \arrow[d, "mrv(c_{21}) = \omega_3"] \arrow[dr, "mrv(c_{22}) = \omega_4 \prec \omega_2"] & \\
c_{31}\,\omega_3^{e_{31}} + c_{32}\,\omega_3^{e_{32}} & c_{41}\,\omega_4^{e_{41}} + O(\omega_4^{e_{42}})
\end{tikzcd}$$

Note that in this artificial example the first two terms of the asymptotic series for c_{21} form an exact representation for c_{21}, and therefore the O-term has been omitted. The first four terms of the asymptotic series for f become

$$\left(\left(c_{31}\,\omega_3^{e_{31}} + c_{32}\,\omega_3^{e_{32}}\right)\omega_2^{e_{21}} + \left(c_{41}\,\omega_4^{e_{41}} + O(\omega_4^{e_{42}})\right)\omega_2^{e_{22}}\right)\omega_1^{e_{11}}, \tag{6.3}$$

or in expanded form

$$c_{31}\,\omega_3^{e_{31}}\,\omega_2^{e_{21}}\,\omega_1^{e_{11}} + c_{32}\,\omega_3^{e_{32}}\omega_2^{e_{21}}\,\omega_1^{e_{11}} + c_{41}\,\omega_4^{e_{41}}\omega_2^{e_{22}}\,\omega_1^{e_{11}} + O\left(\omega_4^{e_{42}}\,\omega_2^{e_{22}}\,\omega_1^{e_{11}}\right), \tag{6.4}$$

where $c_{31}, c_{32}, c_{41} \in \mathbb{R}$. In this particular example we had to expand the coefficient c_{22} into a series as well since the series approximation of c_{21} was an exact one. We prefer to present the result in the following examples in factored form (6.3).

In [83], the series (6.4) is called an asymptotic $\mathcal{T}$-expansion with $\mathcal{T} = \{\omega_4, \omega_3, \omega_2, \omega_1\}$ (provided that $\gamma(\omega_4) < \gamma(\omega_3)$, i.e. the elements in $\mathcal{T}$ must be strictly ordered according to their growth).

Example 6.2 As an example let us look again at the function

$$f = \left(\sin\left(1/x + e^{-x}\right) - \sin\left(1/x\right)\right) \tag{6.5}$$

which we have already seen in Example 2.17 and 5.2.

The *mrv* set of f is $\{e^{-x}\}$ and so $\omega_1 = e^{-x}$. The rewritten expression is $\sin(1/x+\omega_1) - \sin(1/x)$ and the series expansion thereof in ω_1 is $\cos(1/x)\,\omega_1 - \sin(1/x)/2\,\omega_1^2 + O(\omega_1^3)$. Next we compute recursively the asymptotic series expansion of the first coefficient $\cos(1/x) = \cos(\omega_2)$ and get $1 - 1/2\,\omega_2^2 + O(\omega_2^4)$. The final asymptotic series thus becomes

$$\left(\sin\left(1/x + e^{-x}\right) - \sin\left(1/x\right)\right) \approx \left(1 - \frac{1}{2x^2} + \frac{1}{24x^4} + O\left(\frac{1}{x^6}\right)\right) e^{-x}. \tag{6.6}$$

¶

If we look at some further examples we realize, that although effective for some functions, the algorithm produces rather useless results for other functions.

$$e^x \left(e^{\frac{1}{x}+e^{-x}} - e^{\frac{1}{x}}\right) \approx 1 + \frac{1}{x} + \frac{1}{2\,x^2} + \frac{1}{6\,x^3} + O\left(\frac{1}{x^4}\right) \tag{6.7}$$

$$\frac{\Gamma\left(x + \frac{1}{\Gamma(x)}\right) - \Gamma\left(x\right)}{\ln x} \approx 1 - \frac{1}{2\,x\,\ln x} - \frac{1}{12\,x^2\,\ln x} + O\left(\frac{1}{x^4\,\ln x}\right) \tag{6.8}$$

$$\frac{e^{\Gamma(x)}}{\Gamma(e^x)} \approx \frac{1}{\Gamma(e^x)}\,e^{\Gamma(x)} \tag{6.9}$$

$$\Gamma(x) \approx \Gamma(x) \tag{6.10}$$

The algorithm constructs the asymptotic scale which it uses on the fly according to the comparability classes which evolve out of the given function. In example (6.10) the most rapidly varying subexpression of $\Gamma(x) = e^{\Gamma_s(x)}$ turns out to be $\Gamma(x)$ itself. If we take $\Gamma(x)$ as the representative of this comparability class and include it into the asymptotic scale, then the asymptotic expansion of $\Gamma(x)$ becomes simply $1 \cdot \Gamma(x)$. Although in example (6.9) the result is also identical to the input, the algorithm has done a little bit more. The most rapidly varying subexpression of $e^{\Gamma(x)}/\Gamma(e^x)$ has been identified to be $e^{\Gamma(x)}$ which tends to infinity. The coefficient of the series expansion in terms of $e^{\Gamma(x)}$ is $1/\Gamma(e^x) = e^{-\Gamma_s(e^x)}$ which is equivalent to its most rapidly varying subexpression. The asymptotic scale in which (6.9) has been expanded is $S = [e^{\Gamma(x)}, \Gamma(e^x)]$. Note that for functions containing essential singularities, the appearance of the scale depends on how the essential singularities are isolated during the pre-processing step.

Although the expansions (6.9) and (6.10) are correct and valid results of our algorithm, we would like the algorithm to use more expressive scales if possible. In other words, the set of most rapidly varying subexpressions Ω should be rewritten in terms of an element which is not necessarily in Ω itself but rather in some normalized form. As all elements in Ω are exponentials when they are rewritten, the following idea to define a normalized representative might be used. From any exponential in Ω we compute the leading term $c_1\,\varphi_1$ of the asymptotic series of its argument and define $\omega = e^{\varphi_1}$ to be the normalized representative of Ω. According to Definition 6.1 this function is in the same comparability class as all elements in Ω. For example, the normalized representative of $\{\Gamma(x)\}$ is $e^{x \ln x}$ as $\Gamma(x) = e^{\Gamma_s(x)}$ and the leading term of the asymptotic series of $\Gamma_s(x)$ is $x \ln x$ (cf. (5.21)). However, as a consequence of this choice for ω, termination is no longer guaranteed, since in Section 3.4.1.1 (proof of termination) we assumed explicitly that ω or $1/\omega \in \Omega$. Consider for example

$$f = e^{e^x/(1-1/x)}. \tag{6.11}$$

The most rapidly varying subexpression of f is f itself and its normalized representation is e^{e^x}. If we rewrite f in terms of e^{e^x} we become

$$f_1 = e^{e^x/(1-1/x)-e^x} \cdot e^{e^x}.$$

The leading coefficient of the series of f_1 in $\omega = e^{-e^x}$ is $c_{11} = e^{e^x/(1-1/x)-e^x}$. $mrv(c_{11}) = \{c_{11}\}$ and the normalized representation thereof is $e^{e^x/x}$. If we rewrite c_{11} we get

$$f_2 = e^{e^x/(1-1/x)-e^x-e^x/x} \cdot e^{e^x/x}.$$

The normalized representatives of the comparability classes of the subsequent leading coefficients are $mrv(c_{21}) \asymp e^{e^x/x^2}, mrv(c_{31}) = e^{e^x/x^3}, \ldots, mrv(c_{k1}) = e^{e^x/x^k}, \ldots$ and the asymptotic series after k steps is

$$c_{k+1,1}\, e^{e^x/x^k} \cdots e^{e^x/x} e^{e^x}$$

and obviously this recursion does not terminate. The reason is that the size of f gets never reduced during the expansion process. The size of f is $\text{Size}(f) = |\{e^{e^x/(1-1/x)}, e^x, x\}| = 3$ but the size of the rewritten function f_1 is $\text{Size}(f_1) = |\{e^{e^x/(1-1/x)-e^x}, e^{e^x}, e^x, x\}| = 4$. As the leading coefficient of the series of f_1 in ω does not depend on ω its size is smaller than the one of f_1 and turns out to be three, i.e. the leading coefficient of the series expansion of f_1 has the same size as f. The leading coefficient of the series of f_2 is also three and so on. As we see, the size does not get reduced and the recursion will never terminate.

In order to guarantee termination of the recursion of the *MrvAsympt* algorithm we test whether the size of an expression grows due to the rewriting in terms of the normalized representative of the mrv set Ω or not. If it grows then we use an element out of Ω to rewrite the elements in Ω.

However, a growth of the size of a function due to the rewriting step does not necessarily imply that the algorithm does not terminate if it is continued. This may be a temporary effect. As a consequence we allow the specification of an upper bound up to which the size may grow due to *all* rewriting steps along the computation of the asymptotic series. If this bound is set to zero then a growth of the size due to the rewriting step is never allowed. If this bound is positive however, then a growth is allowed and the bound is reduced by the difference of the sizes of the original and the rewritten function. The reduced bound is then applied for the asymptotic expansions of the coefficients.

Let us demonstrate the behaviour with the help of the $\Gamma(x)$ function. The growth-bound can be passed as fourth (optional) argument to the *MrvAsympt* function. Other than this change, the interface of *MrvAsympt* is identical to that of MAPLE's *asympt* command. If the growth-bound is set to two, then the normalized representative $e^{x \ln x}$ is used for the comparability class of $\Gamma(x)$. In order to expand the leading coefficient $e^{\ln(\Gamma(x)) - x \ln x}$ as well, the growth-bound has to be set to at least three, and then we get the usual asymptotic expansion of $\Gamma(x)$.

```
> MrvAsympt(GAMMA(x),x,2,0);
```

$$\Gamma(x)$$

```
> MrvAsympt(GAMMA(x),x,2,2);
```

$$e^{\ln \Gamma(x) - x \ln x} \, e^{x \ln x}$$

```
> MrvAsympt(GAMMA(x),x,2,3);
```

$$\left(\sqrt{2\pi}\, x^{-1/2} + \frac{\sqrt{2\pi}}{12}\, x^{-3/2} + O(x^{-5/2}) \right) e^{-x} \, e^{x \ln x}$$

Another nice feature of the growth-bound is that for functions which can be represented as an infinite asymptotic product, the *MrvAsympt* command computes the first few terms of this product if the growth-bound is set and keeps the rest in closed form. For the function (6.11) we get

```
> MrvAsympt(exp(exp(x)/(1-1/x)),x,1,4);
```

$$\left(e^{\frac{e^x}{1-1/x} - e^x - \frac{e^x}{x} - \frac{e^x}{x^2} - \frac{e^x}{x^3}} \right) e^{\frac{e^x}{x^3}} \, e^{\frac{e^x}{x^2}} \, e^{\frac{e^x}{x}} \, e^{e^x}$$

However, if a representation in a normalized scale is neither possible as an asymptotic sum nor as an asymptotic product, the algorithm has to take an expression out of the mrv set as an entry in the asymptotic scale. The growth-bound has no effect in such a situation. We have already seen an example of such a function in (6.2).

```
> MrvAsympt(exp(exp(exp(x)/(1-1/x))),x,1,5);
```

$$e^{e^{\frac{e^x}{1-1/x}}}$$

Example 6.3 In this example we present further results of the *MrvAsympt* algorithm. Note that none of these problems can be solved with MAPLE's *asympt* command, as the algorithm being used there also suffers from the cancellation problem.

Note as well that example (6.13) is the series expansion of (6.9) but with a positive growth-bound. Example (6.16) has already been discussed in Example 5.6. The asymptotic expansions of example (6.17) and (6.18) differ due to the different growth-bound.

```
> Asympt := proc(e) e = MrvAsympt(args) end:
> Asympt((exp(sin(1/x+exp(-exp(x))))-exp(sin(1/x))), x, 4);
```
(6.12)

$$e^{\sin\left(1/x+e^{-e^x}\right)} - e^{\sin(1/x)} = \frac{1+\frac{1}{x}-\frac{1}{2x^3}+O\left(\frac{1}{x^4}\right)}{e^{e^x}}$$

```
> Asympt(exp(GAMMA(x))/GAMMA(exp(x)), x, 2, 2);
```
(6.13)

$$\frac{e^{\Gamma(x)}}{\Gamma(e^x)} = \left(\frac{1}{\sqrt{2\pi}}\,(e^x)^{1/2} - \frac{1}{12\sqrt{2\pi}}\,(e^x)^{-1/2} + O\left((e^x)^{-3/2}\right)\right)\, e^{e^x}\; e^{-x e^x}\; e^{\Gamma(x)}$$

```
> Asympt((3^x+5^x)^(1/x), x, 3);
```
(6.14)

$$(3^x+5^x)^{1/x} = 5 + \tfrac{5}{x}\,\left(e^{-x}\right)^{\ln 5-\ln 3} + \left(-\tfrac{5}{2x}+O\left(\tfrac{1}{x^2}\right)\right)\,\left(e^{-x}\right)^{2\ln 5-2\ln 3}$$

```
> e3 := x -> exp(exp(exp(x))):
> Asympt(e3(x)/e3(x-1/e3(x)), x, 3);
```
(6.15)

$$\frac{e^{e^{e^x}}}{e^{e^{e^{x-1/e^{e^{e^x}}}}}} = 1 + e^x\, e^{e^x}\left(e^{e^{e^x}}\right)^{-1} + \left(\tfrac{1}{2}(e^x)^2\,\left(e^{e^x}\right)^2 + O\left((e^x)^2\, e^{e^x}\right)\right)\left(e^{e^{e^x}}\right)^{-2}$$

```
> Asympt(exp(exp(Psi(Psi(x)))), x, 3);
```
(6.16)

$$e^{e^{\psi(\psi(x))}} = \left(e^{-1/2} + \frac{1}{24}\frac{e^{-1/2}}{\ln x} + \frac{25}{1152}\frac{e^{-1/2}}{\ln^2 x} + O\left(\frac{1}{\ln^3 x}\right)\right)\, x$$

```
> Asympt(GAMMA(x+exp(-x))-GAMMA(x), x, 4);
```
(6.17)

$$\Gamma(x+e^{-x}) - \Gamma(x) = \left(\ln x - \tfrac{1}{2x} - \tfrac{1}{12x^2} + \tfrac{1}{120x^4} + O\left(\tfrac{1}{x^6}\right)\right)\, e^{-x}\, \Gamma(x)$$

```
> Asympt(GAMMA(x+exp(-x))-GAMMA(x), x, 2, 2);
```
(6.18)

$$\Gamma(x+e^{-x}) - \Gamma(x) = \left(\left(\sqrt{2\pi}\ln x\right)\, x^{-1/2} + \left(\tfrac{\sqrt{2\pi}}{12}\ln x + O(1)\right)\, x^{-3/2}\right)\,(e^{-x})^2\, e^{x\ln x}$$

¶

6.2 Hierarchical Series

In this section we describe another form for representing the resulting asymptotic series, which arises if we slightly change our algorithm. After having determined the normalized representative ω of the set of most rapidly varying subexpressions of f, f is rewritten and expanded in terms of ω, for example, according to the diagram on page 90 the series

$$c_{11}\,\omega^{e_{11}} + c_{12}\,\omega^{e_{12}} + c_{13}\,\omega^{e_{13}} + O(\omega^{e_{14}}) \tag{6.19}$$

is computed. However, this is also an asymptotic series, although of a more general type than that defined in Definition 6.1, as the coefficients are not constants. As an ordinary asymptotic series can be obtained from (6.19) by expanding the coefficients into asymptotic series recursively as well, we call such a series a *hierarchical series*. Hierarchical series have several advantages over conventional asymptotic series.

First, the result provides more information. Consider the function $f = \sin(1/x + \exp(-x))$. The ordinary asymptotic series of Poincaré type is

$$f(x) \approx \frac{1}{x} - \frac{1}{6\,x^3} + \frac{1}{120\,x^5} + O\left(\frac{1}{x^7}\right) \tag{6.20}$$

whereas the hierarchical asymptotic series of f is

$$f(x) \approx \sin(1/x) + \cos(1/x)\,e^{-x} - \frac{1}{2}\sin(1/x)\,\left(e^{-x}\right)^2 + O\left(\left(e^{-x}\right)^3\right). \tag{6.21}$$

The first series is nothing else than the asymptotic series of the leading term of the second series. The rest of the information inherent in (6.21) is lost in (6.20).

Another advantage is that it may be possible to compute some levels of a hierarchical series on functions which are difficult to expand in a regular series. This happens, for example, for functions where the form of the regular asymptotic series depends on relations between the parameters which are not specified. An easy example is

$$f = e^{1/x + e^{-x^2}\,(e^{ax} - e^{bx})} - e^{1/x}.$$

where a and b are real constants.The hierarchical series thereof is

$$f \approx e^{1/x}(e^{ax} - e^{bx})\,e^{-x^2} + \frac{e^{1/x}}{2}(e^{ax} - e^{bx})^2\,(e^{-x^2})^2 + O\left((e^{-x^2})^3\right)$$

whereas the conventional asymptotic series depends on the sign of $a - b$:

$$f \approx \begin{cases} \left(1 + \frac{1}{x} + \frac{1}{2x^2} + O(\frac{1}{x^3})\right)\,(e^x)^a\,e^{-x^2} & \text{if } a > b \\ \left(-1 - \frac{1}{x} - \frac{1}{2x^2} + O(\frac{1}{x^3})\right)\,(e^x)^b\,e^{-x^2} & \text{if } b > a. \end{cases}$$

Finally, if a function f can be expanded into an infinite asymptotic product, then the hierarchical series will expand the first factor and encode the product of the rest into the coefficient. Further factors can be obtained by expanding the coefficient into further hierarchical series.

The problem of termination of the algorithm to compute hierarchical series is obviously not an issue, as only the most rapidly varying comparability class is used. In other words, termination is under user control.

7. Implementation

The MAPLE code of an implementation of our algorithm for computing limits of exp-log functions can be found in Appendix A. In this chapter we focus on some particular problems which appear during an actual implementation in a computer algebra system. The emphasis will thereby be on the underlying series facility. We also discuss the problem of zero recognition from an implementation point of view, as well as the problem of branch cuts. Finally we briefly discuss the implementation of the *limit* routine as it is available in the regular MAPLE system.

7.1 Series Computation

We required in Section 3.3 that the underlying series model must be able to represent power series which contain arbitrary real constants as exponents. Puiseux series are not powerful enough. This is not only a theoretical subtlety. For easy limit computations this power is already needed. Consider

$$\lim_{x\to+\infty} \left(3^x + 5^x\right)^{1/x}. \tag{7.1}$$

First this problem is transformed to

$$\lim_{x\to+\infty} \exp\left(\frac{\ln\left(e^{\ln(3)x} + e^{\ln(5)x}\right)}{x}\right). \tag{7.2}$$

The set of most rapidly varying subexpression is $\{e^{\ln(3)x}, e^{\ln(5)x}\}$. If we set $\omega = e^{-\ln(3)x}$ the above expression can be rewritten as

$$\exp\left(\frac{\ln(1/\omega + (1/\omega)^{\ln 5/\ln 3})}{x}\right) \tag{7.3}$$

whose series is

$$5 + \frac{5}{x}\,\omega^{\ln 5/\ln 3 - 1} - \frac{5(x-1)}{2\,x^2}\,\omega^{2\ln 5/\ln 3 - 2} + O\left(\omega^{3\ln 5/\ln 3 - 3}\right) \tag{7.4}$$

which agrees with the result (6.14) we obtained in Example 6.3. The exponents cannot be represented by rationals, hence the above series is not a Puiseux series. By the way, the result of the limit of (7.1) is 5.

7.1.1 Models to Represent General Power Series

Since Maple does not support such a general series facility, we had to build our own as the basis for our limit implementation. These series approximations cannot be stored in a dense representation as the exponents are not enumerable. The series must be represented as an ordered collection of terms where each term consists of a coefficient and of its corresponding exponent. There are two basically different approaches which may be used here.

7.1.1.1 Truncated Power Series

One possibility is to implement algorithms to work with *truncated* power series. Truncated power series are approximations to the actual series as they only contain a finite number of terms. They can be represented, for example, as a finite list of terms. If a truncated series does not contain all nonzero terms of the exact power series, the representation must encode the order of the truncation. This is usually done by adding a term with the order of the truncation, whose coefficient has the form $O(1)$. Almost all existing computer algebra systems provide facilities for the manipulation of truncated power series.

If a series approximation of a function is computed, the user has to specify, in advance, how many terms the result should contain, or alternatively, which truncation order the result should have. However, the series expansion is usually performed recursively bottom up (cf. Algorithm 2.16) and it is difficult, if not impossible, to specify up to which order (or how many terms of) the intermediate approximations have to be computed. Terms may be lost due to cancellations in the addition operation, and the order may be reduced by division or differentiation. A well known example is $\sin(\tan(x)) - \tan(\sin(x))$, which has to be expanded in Maple up to order eight to get one significant term. If one decides after looking at the result that more terms are needed, the whole series computation has to be restarted from the beginning. The search for the leading term of a series is thus usually done by subsequently computing the series up to the order 1, 2, 3, etc., until the result finally contains at least one non zero term.

The advantage of the truncated power series approach is that efficient algorithms exist to perform the series arithmetic. Multiplication of two Taylor series (over certain coefficient fields) of order n for example can be done with $O(n \ln n)$ coefficient multiplications using the fast Fourier transform (FFT). Division can also be performed with the same complexity using Newton's method [48]. Provided that this fast arithmetic is available and that every series operation on series with n terms can be performed with $O(n \log n)$ coefficient operations the sequential search for the leading term of a series uses $O(\sum_{n=1}^{m} n \log n) = O(m^2 \log m)$ coefficient operations if we have to compute m series approximations to get a non trivial term.

The asymptotic behaviour can be improved if we double the approximation order after every failure when looking for the leading coefficient. Then the overall asymptotic cost for computing the leading term is $O(m \ln m)$ with the fast arithmetic. However, if the fast multiplication is not available, then $O(m^2)$ operations are needed to get the leading coefficient.

Another disadvantage of the truncated power series approach, besides the a priori specification of the truncation order, is the complexity of the code for the series operations. If the domain of the exponents is not enumerable, then an efficient coding of the multiplication operation, for example, is rather complicated.

7.1.1.2 Infinite Power Series

The second approach is to use *infinite* power series instead of truncated ones. With this model one creates power series, performs various operations on them and at the end asks for any number of terms of the result. With this approach one overcomes the difficulty that the number of terms which have to be computed is often not known in advance. Terms are only computed if they are needed. Unnecessary work is prevented.

Infinite series are traditionally represented by a rule which allows one to generate the terms as necessary. Operations on these series will generate new series whose rules are constructed out of the old ones. The rule may, for example, be a function $f : \mathbb{N} \to C$ from the natural numbers to the coefficient domain C to generate the n'th coefficient (provided that the set of exponents is enumerable) or the n'th term. Another possibility is to represent an infinite series as an ordered pair consisting of the leading term (coefficient and exponent) and of a rule to generate the rest of the series as another ordered pair. If in either case the rule is represented as an expression, then obviously it must not be evaluated but rather be held in an unevaluated form. This technique of evaluating an expression only when it is needed is called *lazy evaluation*. Its fundamental idea is to construct a value when it is to be used rather than when it is defined.

For both representations it is necessary to have on-line algorithms for manipulating series, i.e. algorithms which produce the terms of the series one by one and in order, and which do not require the number of terms to be specified in advance. Knuth [44, Section 4.7] presents a few on-line algorithms for power series. Various implementations of infinite Taylor series have been described in the literature. The function model is used in [56, 34] and the lazy evaluation model in [14, 91, 2, 50]. Many series operations have extremely simple on-line algorithms if infinite series are implemented using lazy evaluation.

The asymptotic behaviour of the run time of these algorithms is normally worse than that of the explicit approach. Most of the on-line algorithms use $O(n^2)$ coefficient operations to compute terms up to order n. Semi on-line algorithms, which may double the number of terms on demand, have a better run time behaviour. They are normally also based on Newton's algorithm. In our context of computing limits this sub-optimal efficiency is not a problem, since we are only interested in the leading coefficient of a power series and hence this approach is about as efficient as the explicit approach based on the efficient arithmetic.

The infinite power series approach has one major problem: it is not possible to decide in general whether a series is zero or not. If it is zero, then the search for the first nonzero coefficient may not terminate. This operation is needed when computing the inverse or the logarithm of a series. However, it is an error in these two cases if the argument is zero, and an infinite loop may hence be a tolerable outcome. In our context of computing limits this problem is not an issue. We postulated an oracle for deciding zero equivalence of expressions anyway, and hence one can assert at the time a series is generated that it is non-zero and thus contains at least one non-trivial term.

In the remaining part of this section we describe an implementation of infinite general sparse power series in Maple using the lazy evaluation approach.

7.1.2 Lazy Evaluated General Sparse Power Series in Maple

To simulate delayed evaluation in Maple we use the new *uneval* type[1] which may be specified for parameters of procedures. If a formal parameter has this type specification, then the actual argument is not evaluated when the procedure is called. We use this feature for representation purposes only and define a procedure *lazyseries* which returns unevaluated. It is similar to an unevaluated function call, with the only difference that the second argument is not evaluated. A power series is thus represented as an object *lazyseries(head, tail, x)* where *head* is the first term of the series, *tail* the delayed rest of it which, when evaluated, generates an object of the same type, and x is the expansion variable. Terms are represented as lists consisting of the coefficient and of the corresponding exponent. The evaluation of the tail of a power series can be enforced using the *eval* command. Additionally, we define the constructor *MakeSeries* and the selectors *Head*, *Tail*[2] and *Var*. The empty series is represented by the special symbol *NIL*.

```
> lazyseries := proc(Head, Tail:uneval, x:name) 'procname(args)' end:
> MakeSeries := proc(Head, Tail:uneval, x:name) lazyseries(args) end:
> Head := proc(p) op(1,p) end:
> Tail := proc(s) option remember; eval(op(2,s)) end:
> Var  := proc(p) op(3,p) end:
```

If a facility such as *uneval* is not available (as in older versions of Maple), delayed evaluation can be simulated using procedures. The evaluation of an expression *expr* is delayed by the construct *() –> expr* and evaluation can be

[1] Available since Release 4 of Maple V.

[2] The reason for the option remember will be explained later.

enforced by executing this procedure. This approach is used in the MAPLE implementation of infinite streams described in [34].

As an example let us define the power series for e^x. We first define a procedure *expfrom*(n, x) which constructs the power series of e^x starting with the n'th term. The first term of the series of *expfrom*(n, x) is $x^n/n!$ and its tail is the power series of e^x starting with the $n+1$'th term.

```
> expfrom := (n, x) -> MakeSeries([1/n!, n], expfrom(n+1, x), x):
```

We then define the power series for e^x which simply is *expfrom*$(0, x)$. Notice that its evaluated tail is again a power series object starting with the term x and containing another unevaluated tail. The procedure *PrintSeries* prints the series in a conventional notation.

```
> e := expfrom(0,x);
```

$$e := \mathrm{lazyseries}([1, 0], \mathrm{expfrom}(1, x), x)$$

```
> Tail(e);
```

$$\mathrm{lazyseries}([1, 1], \mathrm{expfrom}(2, x), x)$$

```
> PrintSeries(e);
```

$$1 + x + \frac{1}{2}\,x^2 + \frac{1}{6}\,x^3 + \frac{1}{24}\,x^4 + \frac{1}{120}\,x^5 + O\left(x^6\right)$$

A simple and useful construct for the manipulation of power series is the mapping of a function over all terms of a given series. Applications of such a map function are the operations *Scale* and *Shift* which multiply a series by a constant or by a power of the expansion variable respectively[3]. Similarly, integration (*Int*) and differentiation (*Diff*) of power series can be implemented.

```
> Map := proc(f, p)
>    if p = NIL then NIL
>    else MakeSeries(f(Head(p)), Map(f, Tail(p)), Var(p))
>    fi
> end:
> Scale := proc(c, p) local t; Map(unapply([c*t[1],t[2]],t), p) end:
> Shift := proc(n, p) local t; Map(unapply([t[1],n+t[2]],t), p) end:
> PrintSeries(Shift(-1, Scale(120, e)));
```

$$120\,\frac{1}{x} + 120 + 60\,x + 20\,x^2 + 5\,x^3 + x^4 + O\left(x^5\right)$$

Next we define an on-line program to add two power series which may contain arbitrary real exponents. For comparing the exponents we use MAPLE's *signum* function as an oracle. Note that if the two leading exponents of the arguments are equal then the new coefficient which is generated may be zero.

[3]We use the *unapply* construct in the definitions for *Scale* and *Shift* since MAPLE does not support nested lexical scopes. If it did, then e.g. *Scale* could be defined as

```
> Scale := (c, p) -> Map(t -> [c*t[1],t[2]], p):
```

If we enforced a sparse representation, the sum of the two tails would need to be returned in this case. This however is dangerous and may lead to infinite loops. For example, if the exponent of the second term of the series for $s = \cos^2(x) + \sin^2(x)$ were inspected, the addition routine would enter into an infinite loop, since the sum of all corresponding coefficients is zero. As a consequence we allow zero terms to appear in a series, but the exponents must still be strictly increasing. A positive side effect of this approach is that the potentially expensive test for zero equivalence is delayed until it is really needed.

```
> Add := proc(f, g)
>    local hf, hg, s, x;
>    if f = NIL then g
>    elif g = NIL then f
>    else ASSERT(Var(f) = Var(g));
>       hf := Head(f); hg := Head(g); x := Var(f);
>       s := signum(0, hg[2] - hf[2], 0);
>       if s = 1 then MakeSeries(hf, Add(Tail(f), g), x)
>       elif s = -1 then MakeSeries(hg, Add(f, Tail(g)), x)
>       elif s = 0 then
>          MakeSeries([hf[1] + hg[1], hf[2]], Add(Tail(f), Tail(g)), x)
>       else ERROR(`Oracle cannot compute the sign of `, hg[2]-hf[2])
>       fi
>    fi
> end:
```

The second arguments to *MakeSeries* in the procedure *Add* are recursive calls, but since the evaluation thereof is delayed they do not lead to infinite recursions.

The multiplication of two power series can be defined recursively as well. Let $f(x) = f_0\, x^{d_0} + F(x)$ and $g(x) = g_0\, x^{e_0} + G(x)$, then

$$f(x) \cdot g(x) = f_0\, g_0\, x^{d_0+e_0} + f_0\, x^{d_0}\, G(x) + g_0\, x^{e_0}\, F(x) + F(x) \cdot G(x).$$

Only the multiplication between $F(x)$ and $G(x)$ is a recursive call. The other multiplications are shifts and scalings. Also, the addition between the leading term and the rest is not a series addition, but rather a series concatenation.

```
> Mult := proc(f,g)
>    local hf, hg, f0, g0, d0, e0;
>    if f = NIL then NIL
>    elif g = NIL then NIL
>    else ASSERT(Var(f) = Var(g)):
>       hf := Head(f): hg := Head(g):
>       f0 := hf[1]; g0 := hg[1]; d0 := hf[2]; e0 := hg[2];
>       MakeSeries([f0*g0, d0+e0],
>          Add(Add(Scale(f0, Shift(d0, Tail(g))),
>                  Scale(g0, Shift(e0, Tail(f)))),
>              Mult(Tail(f), Tail(g))
>          ),
>          Var(f)
>       )
>    fi
> end:
```

One problem with this definition is the normal order reduction of expressions which is implied by the lazy evaluation. For example, the expressions *Tail(f)* and *Tail(g)* will be executed twice if the tail of a product is evaluated. This may have bad consequences if the evaluation of the tail of f or g is very expensive. In order to prevent multiple evaluations of the tail of a series, we save all ever computed results in the remember table of the procedure *Tail* by adding option remember to it[4]. Access to elements in the remember table can be done in MAPLE in constant time. In order to prevent the remember table from getting filled up, it is cleared whenever a new function is expanded into a series. If we assume that the power series $f(x)$ and $g(x)$ have already been computed up to n terms, i.e. if the $n-1$ tails can be computed in constant time, then the cost for multiplying $f(x)$ and $g(x)$ up to n terms is $O(n^2)$.

7.1.3 Fixed Point Definitions

It is by now a classic exercise ([44, Exercise 4.7.4] and [2, Exercise 3.49]) to compute power series for elementary functions which satisfy a simple differential equation. The on-line program may be derived directly from the corresponding defining integral equation. For example, the exponential function is defined by the equation

$$e^{s(x)} = e^{s(x_0)} + \int_{x_0}^{x} e^{s(t)} \frac{ds(t)}{dt}\, dt. \tag{7.5}$$

The following procedure for computing the exponential $e^{s(x)}$ at $x_0 = 0$ is almost a one to one translation of equation (7.5).

```
> Exp := proc(s) local h, s0;
>    h := Head(s);
>    if h[2] > 0 then s0 := 0 else s0 := h[1] fi;
>    MakeSeries([exp(s0), 0], Int(Mult(Exp(s), Diff(s))), Var(s))
> end:
> PrintSeries(Exp(MakeSeries([1,1], NIL, x)));
```

$$1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + O\left(x^6\right)$$

Notice that in the definition of the tail of e^s the expression e^s is recursively used. To prevent the procedure *Exp* from being called recursively with the same arguments, we could add option remember to it. A better solution would be to substitute the expression *Exp(s)* in the definition by the result itself, i.e. to "hard code" this recursion.

From another point of view one can also say that $e^{s(x)}$ is defined as the fixed point of the equation (7.5). The concept of defining a power series as a fixed point of a mapping from the set of power series onto the set of power series can also be offered as a construct by itself. Obviously, not all mappings will

[4]See the definition of *Tail* on page 100.

generate a power series. Only mappings which do not perform operations on their argument, but rather simply include it in a new structure which is returned as the value of the map, can be used. Moreover, the leading term of the result must not depend on the argument, as otherwise the power series cannot be "bootstrapped". The idea of such a fixed point operator has first been mentioned in [14, 91].

The implementation of a fixed point operator in MAPLE is rather trivial. The map F is simply applied to a local, anonymous power series p. The result is then assigned to p, thus producing the fixed point, or, in other words, automatically hard-coding the recursion.

```
> FixedPoint := proc(F:procedure) local p;
>    p := F(p)
> end:
```

This simple implementation presumes that no procedure working on power series tests whether the argument is a power series object. The implementation would otherwise have to be slightly more complicated. Furthermore, the procedure should assert that the leading term of p does not depend on p itself.

Let us define the power series for e^x and for $\sin(x)$ at $x = 0$ using the fixed point operator defined above. The series of e^x is defined as the series whose zero-order term is $e^0 = 1$ and whose higher-order terms are given (recursively) by the integral of the series for e^x. A similar statement holds for $\sin(x)$.

```
> FixedPoint(e -> MakeSeries([1,0], Int(e), x));
```

$$\mathrm{lazyseries}([1,0],\mathrm{Int}(p),x)$$

```
> PrintSeries(");
```

$$1+x+\frac{1}{2}\,x^2+\frac{1}{6}\,x^3+\frac{1}{24}\,x^4+\frac{1}{120}\,x^5+O\left(x^6\right)$$

```
> FixedPoint( sin -> MakeSeries([1,1], Int(Int(Scale(-1,sin))), x));
```

$$\mathrm{lazyseries}([1,1],\mathrm{Int}(\mathrm{Int}(\mathrm{Scale}(-1,p))),x)$$

```
> PrintSeries(", 5);
```

$$x-\frac{1}{6}\,x^3+\frac{1}{120}\,x^5-\frac{1}{5040}\,x^7+\frac{1}{362880}\,x^9+O\left(x^{11}\right)$$

The p in the unevaluated parts of both results refers to the whole series itself. It is an exported local name and thus unique. If p were not unique, then the fixed point operator would not work as stated above.

The procedure to compute the exponential of a series looks similar. Additionally, if the leading exponent of the argument is negative, then the sign of the leading coefficient has to be computed. If it is zero, the exponential of the tail of the argument is computed, otherwise an error is issued. This is an example where a zero test must be performed during a series expansion.

Not all fixed point definitions need to be based on integration. As an example we present a fixed point definition of power series division which is based on power series shifting, although multiplication and division have nice integral based fixed point definitions, namely

$$\begin{aligned} f(x)\cdot g(x) &= \mathrm{FixedPoint}\left(\mathbf{p}\to f(0)\cdot g(0)+\int \mathbf{p}\cdot\left(\frac{f'(x)}{f(x)}+\frac{g'(x)}{g(x)}\right)\,dx\right) \\ f(x)/g(x) &= \mathrm{FixedPoint}\left(\mathbf{q}\to f(0)/g(0)+\int \mathbf{q}\cdot\left(\frac{f'(x)}{f(x)}-\frac{g'(x)}{g(x)}\right)\,dx\right), \end{aligned}$$

provided that $f(0)\neq 0$ and $g(0)\neq 0$. This condition can easily be established by shifting $f(x)$ and $g(x)$. Notice the nice symmetry between the multiplication and the division! However, the fixed point definition for the division we give next is more efficient. Let $f(x)=f_0\,x^{d_0}+F(x)$ and $g(x)=g_0\,x^{e_0}+G(x)$, then

$$f(x)/g(x)=\mathrm{FixedPoint}\left(\mathbf{q}\to\frac{f_0}{g_0}\,x^{d_0-e_0}+\frac{1}{g_0}\,x^{-e_0}(F(x)-G(x)\cdot\mathbf{q})\right).$$

An implementation of this rule is presented below. Since MAPLE does not support nested lexical scopes, the desired behaviour must be simulated by substituting global names with the actual local values.

```
> Divide := proc(f, g)
>    local hf, hg;
>    if g = NIL then ERROR(`division by zero`)
>    elif f = NIL then NIL
>    else ASSERT(Var(f) = Var(g)):
>       hf := Head(f): hg := Head(g):
>       FixedPoint(subs(['f0'=hf[1],'g0'=hg[1],'d0'=hf[2],'e0'=hg[2],
>                        'F'=f,'G'=g],
>          q -> MakeSeries([f0/g0, d0-e0],
>                  Scale(1/g0,
>                     Shift(-e0,
>                        Add(Tail(F), Scale(-1, Mult(Tail(G), q)))
>                     )
>                  ),
>                  Var(F)
>               )
>       ))
>    fi
> end:
```

As a final example let us compute the power series expansion of (7.3) using the infinite general power series facility we have just presented.

```
> ln(w) := -ln(3)*x:
> Series(exp((ln(1/w+(1/w)^(ln(5)/ln(3))))/x), w);
```

```
lazyseries([5, 0],
    Int(Mult(p, Diff(
        lazyseries([ln(5), 0],
            Scale(1/x, Tail(
                lazyseries([ln(5) x, 0],
                    Int(Mult(Diff(P), Inv(P))),
                    w))),
            w)))),
    w)
```

The small p comes from the fixed point definition of the exponential and stands for the whole series, whereas the capital P comes from the fixed point definition of the logarithm and stands for the series of the argument of the logarithm.

```
> PrintSeries(Map(t -> [normal(t[1]),t[2]], "), 3);
```

$$5 + 5\,\frac{w^{\frac{\ln(5)}{\ln(3)}-1}}{x} - \frac{5}{2}\,\frac{(x-1)\,w^{2\frac{\ln(5)}{\ln(3)}-2}}{x^2} + O\left(w^{3\frac{\ln(5)}{\ln(3)}-3}\right)$$

Notice that the result of the series construction already contains the leading coefficient of the result, which is the only information we need in the context of computing limits. The simplification of the coefficients has only been done to get the same result as in (7.4).

It is a nice feature of this lazy evaluation implementation that although the routines work for power series with arbitrary real exponents, this fact has to be respected by the addition operation only when comparing the exponents (besides of some elementary functions which have to check the sign of the leading exponent of the argument). The code can thus be made very generic and can be parameterized by the coefficient domain of the power series as well as by the set of exponents.

The first power series facility based on a lazy evaluation scheme is described by Norman [56]. It was implemented in Scratchpad, the predecessor of the computer algebra system Axiom. Further descriptions of this implementation and in particular of the fixed point operator can be found in [14, 91]. A Lisp implementation of infinite streams is presented in [2]; this has also been adapted for Mathematica [50]. Delayed evaluation for streams was already introduced into Lisp in 1976 [24]. The Lisp dialect Scheme has *delay* and *force* commands to simulate delayed evaluation, whereby the results of *force* are memorized, i.e. remembered. Delayed evaluation was also inherent in Algol 60's call by name parameter passing mechanism [45]. An example of a language with full lazy evaluation is Miranda [52].

7.2 Zero Recognition

As we have discussed in Section 5.3 we have to postulate an oracle for deciding zero equivalence to solve the limit computation problem. We have also seen that this problem can be solved for some function classes. For others, probabilistic methods may be used. From an implementation point of view, the question appears to be how to incorporate these techniques for deciding zero equivalence into a computer algebra system. The question in particular is where the decision is made about which test is applied for a given function. Many approaches to answer this question have been tried (see also [33]).

Maple, and other computer algebra systems like it, all have a general representation for mathematical formulas called expressions. All these systems provide a built in simplifier which is automatically applied to every expression created. But, since simplification is expensive in general, not all simplifications are done automatically, and this is a potential source for problems. Many erroneous result returned from a computer algebra system are caused by a zero which was not recognized as such. Consider the following limit taken from [5].

$$\lim_{\omega \to 0} \frac{\omega}{\sqrt{1+\omega}\,\sin^2 x + \sqrt{1-\omega}\,\cos^2 x - 1} \tag{7.6}$$

Derive returns the correct result $2/(1-2\,\cos^2 x)$, which can be found after one application of l'Hôpital's rule. Let us look how the computer algebra systems Maple, Mathematica, Macsyma, Axiom and Reduce behave on this example:

Maple V Release 3:

```
> limit(w/(sqrt(1+w)*sin(x)^2+sqrt(1-w)*cos(x)^2-1), w=0);

                                   0
```

Mathematica 2.2:

```
In[1]:= Limit[w/(Sqrt[1+w]Sin[x]^2+Sqrt[1-w]Cos[x]^2-1), w->0]
Out[1]= 0
```

Macsyma 418.1:

```
(c1) limit(w/(sqrt(1+w)*sin(x)^2+sqrt(1-w)*cos(x)^2-1), w, 0);
(d1)                                 0
```

Axiom 2.0:

```
(1) ->limit(w/(sqrt(1+w)*sin(x)^2+sqrt(1-w)*cos(x)^2-1), w=0)

   (1)  0
                 Type: Union(OrderedCompletion Expression Integer,...)
```

Reduce 3.6:

```
1: limit(w/(sqrt(1+w)*sin(x)^2+sqrt(1-w)*cos(x)^2-1), w, 0);

0
```

The problem encountered here is that none of the systems have recognized $\sin^2 x + \cos^2 x - 1$ to be zero. Reduce and probably Macsyma reduce the problem to the expression $\omega/(\sin^2 x + \cos^2 x - 1)$ using Lemma 2.4 and decide (erroneously) that the result is zero; the other systems, which are based on series expansion, compute the strange Taylor series

$$\frac{1}{\sin^2 x + \cos^2 x - 1}\,\omega - \frac{\sin^2 x - \cos^2 x}{2(\sin^2 x + \cos^2 x - 1)^2}\,\omega^2 + O(\omega^3) \tag{7.7}$$

and conclude from the leading term that the limit is zero.

The problem (7.6) appears as a sub-problem when computing the limit

$$\lim_{x\to+\infty} \frac{e^{-x}}{\sqrt{1+e^{-x}}\,\sin^2(1/x) + \sqrt{1-e^{-x}}\,\cos^2(1/x) - 1} \tag{7.8}$$

with our algorithm, but only Mathematica and Maple V Release 3 return (the wrong result) 0 for this example, all other systems either return unevaluated or "fail" or run forever. For illustration purposes we took the simpler problem (7.6). Maple V Release 4 however returns the correct result of limit (7.8). It is the limit of the leading coefficient $2/(1-2\cos^2(1/x))$ which is obtained if the argument of (7.8) is expanded in terms of e^{-x}.

```
> limit(exp(-x)/
>          (sqrt(1+exp(-x))*sin(1/x)^2+sqrt(1-exp(-x))*cos(1/x)^2-1),
>       x=infinity);
```

$$-2$$

One approach to solve this zero recognition problem in general is to make the built in automatic simplifier very powerful, in particular powerful for the recognition of zeros. This solution turns out to be very inefficient. Moreover, the simplifier must always first analyze the input expression to decide which simplification rules have to be applied. It must also know which rules are available, which is a serious problem when we consider extending the system.

Some computer algebra systems offer all kinds of options and flags that allow the user to tell the simplifier how to do things differently. To solve the limit (7.6) in Macsyma, one might automatically convert all trigonometric expression into exponential form and explicitly use the limit algorithm which is based on Taylor series (*tlimit*). Since all the trigonometric functions are converted to exponential form, the hidden zero is detected by the automatic simplifier through an application of the *expand* function.

```
(c2) exponentialize:true$
(c3) tlimit(w/(sqrt(1+w)*sin(x)^2+sqrt(1-w)*cos(x)^2-1), w, 0);
                                       2 %i x
                                    4 %e
(d3)                              - ------------
                                    4 %i x
```

The REDUCE system has a global simplifier which is applied automatically and whose behaviour can be modified by setting flags and by defining new simplification rules.

```
2: for all x let sin(x)^2+cos(x)^2=1;

3: limit(w/(sqrt(1+w)*sin(x)^2+sqrt(1-w)*cos(x)^2-1), w, 0);

       2
---------------
          2
 2*sin(x)  - 1
```

Both solutions are unsatisfactory as there is no protection from getting wrong results. The chance of getting a correct one depends on the user's knowledge about the flags offered by the system and on the user's intuition about which flags are needed to solve a particular problem correctly. If a limit is computed within a larger computation, the user has no chance to figure out where to support the system as he does not see which expression was not recognized to be zero.

Most systems offer very powerful simplification routines in addition to the general purpose simplifier. For example, in MAPLE there is a procedure *radsimp* to simplify expressions containing radicals, and many other special purpose simplification routines. As it is too expensive to apply all available simplification routines on every expression, MAPLE supports a *parameterized* zero equivalence testing approach. Whenever the system has to decide whether an expression is zero, the procedure *Testzero* is called. This is an environment variable which is initially set to *normal(x) = 0* (*normal* is MAPLE's simplifier for rational functions), but which may be overwritten by an arbitrary function which returns a boolean result. (Since MAPLE may remember previously computed results, we must restart a new session to show how it works).

```
> restart;
> Testzero := e -> evalb(normal(convert(e, exp)) = 0):
> limit(w/(sqrt(1+w)*sin(x)^2+sqrt(1-w)*cos(x)^2-1), w=0);
```

$$\frac{2}{\sin(x)^2 - \cos(x)^2}$$

MACSYMA allows one to overwrite the simplifier which is used when computing a Taylor series by assigning the variable *taylor_simplifier*. The problem with these approaches is still that success depends on the power of the simplifier being used. If the simplifier is not able to recognize a zero, wrong answers may be returned, and, furthermore, without any indication that they may be wrong! At least, the MAPLE approach allows one to install a *Testzero* procedure which issues a warning when zero equivalence cannot be decided. The user would then get at least an indication that the result could be wrong. Unfortunately, not all MAPLE library functions make use of the *Testzero* facility.

```
> restart;
> Testzero := proc(e) option remember; local t;
>    t := testeq(e) or (Normalizer(e) = 0);
>    if t = FAIL then t := false;
>       printf(`%a is assumed to be non zero\n`, e)
>    fi;
>    t
> end;
> limit(w/((1+w)*2^(1/4)*(2^(1/2)+2)-(8+6*2^(1/2))^(1/2)), w=0);
2*2^(1/4)+2^(3/4)-(8+6*2^(1/2))^(1/2) is assumed to be non zero
```

$$0$$

As the expression MAPLE has claimed to be nonzero is in fact identically zero, the result of this computation is wrong. The correct result can be found by assigning an improved zero test to *Testzero*.

```
> restart;
> Testzero := e -> evalb(readlib(radnormal)(e)=0):
> limit(w/((1+w)*2^(1/4)*(2^(1/2)+2)-(8+6*2^(1/2))^(1/2)), w=0);
```

$$\frac{1}{2}\,\frac{2^{3/4}}{\sqrt{2}+2}$$

The most convincing solution is called the domain based approach. The idea is that the procedures to compute e.g. power series over a field F are parameterized by a collection of functions implementing the operations which can be performed in every field, and in particular also a (zero) equivalence test. Such a collection is called a *domain*. This idea has been pioneered by the AXIOM [40] system and has been adopted by systems such as Gauss[5] [53] and others [1, 96].

Every domain, e.g. the one for elementary functions, can offer the best test for zero equivalence. If the system is extended with a new domain, the special knowledge about zero recognition enters into the domain definition and no central simplifier needs to be updated.

Although AXIOM seems to use this approach, the result on the limit (7.6) is wrong. The reason is, that the procedure *zero?* defined in the domain *Expression*, to which $\sin^2 x + \cos^2 x - 1$ belongs, is not a strong zero equivalence tester. This means that the answer from *zero?* that a function is non-zero may be wrong. A better test could be obtained if the expression which is tested for zero equivalence is first normalized. For that purpose we have defined our own domain *MyExpression (MYEX)* which overwrites the two methods *zero?* and = from the *Expression* domain. All the other operations are inherited from the domain *Expression*

```
(1) ->w/(sqrt(1+w)*sin(x)^2+sqrt(1-w)*cos(x)^2-1) :: MYEX INT;
                                                Type: MyExpression Integer
```

[5] Gauss is implementd on top of MAPLE as a package called *Domains*.

```
(2) ->limit(%, w=0)
                  2
   (2)  -----------------
              2        2
        sin(x)  - cos(x)
                 Type: Union(OrderedCompletion MyExpression Integer,...)
```

This problem will hopefully be fixed in a future version of Axiom.

7.3 The Properties of ω

Whenever a series is computed in terms of ω, the domain of computation is extended by the symbol ω and the system has to be tought about the special properties of ω. In particular this is the property that $\ln\omega$ is equal to the argument of the exponential ω. If this information is not passed to the system, then hidden zeros may remain unrevealed. Consider

$$\lim_{x\to+\infty} \frac{\ln\left(1-\frac{\ln\left(\frac{e^x}{x}-1\right)+\ln(x)}{x}\right)}{x}. \tag{7.9}$$

The most rapidly varying subexpression is e^x, which is replaced by ω^{-1}. This implies that $\ln\omega = -x$. The series expansion of the argument of (7.9) at $\omega = 0$ is

```
> PrintSeries(Series(ln(1-(ln(1/w/x-1)+ln(x))/x)/x, w), 1);
```

$$\frac{\ln\left(1+\frac{\ln(w)-\ln\left(\frac{1}{x}\right)-\ln(x)}{x}\right)}{x}+O\left(w\right)$$

When we look at the leading coefficient of this series and keep in mind that $\ln\omega = -x$ and that x is real and positive, i.e. that $\ln(1/x) = -\ln(x)$, then we see that the argument of the logarithm is zero, and that this series is not defined. In order to compute the correct result, we have to pass the knowledge about $\ln(\omega)$ into the series computation facility, in particular into the zero recognizing tool.

In Maple, we could redefine the *Testzero* environment variable which is called within the *Series* facility to perform the necessary transformations. The solution we chose for our implementation is to store the values of $\ln(\omega)$ and $\ln(1/\omega)$ into the remember table of *ln* directly.

```
> ln(w) := -x: ln(1/w) := x:
> Testzero := e -> evalb(normal(e,expanded)=0):
> PrintSeries(Series(ln(1-(ln(1/w/x-1)+ln(x))/x)/x, w), 2);
```

$$-1+\frac{1}{2}\,w+O\left(w^2\right)$$

The unknown ω must obviously not be a global name, for the limit code would otherwise not be reentrant! The result of (7.9) turns out to be -1.

7.4 Branch Cuts

If we extend the problem domain from the real axis onto the complex plane, then we must be careful, since many standard functions become multi-valued. Familiar examples are

$$\sqrt{z}, \ln z, \arcsin z, \arccos z, \arctan z, \operatorname{arcsinh} z, \operatorname{arccosh} z, \operatorname{arctanh} z, z^{1/n}.$$

These functions cannot be defined continuously on the complex plane. It is conventional in computer algebra systems to represent these functions as single-valued functions by choosing arbitrary lines of discontinuity, which are also called *branch cuts*. Branch cuts begin and end at branch points. They are singularities around which neither branch of the function can be represented by a Taylor nor by a Laurent series expansion. As a consequence, these single-valued functions are normally not analytic throughout the whole complex plane; rather, they have discontinuities across their branch cuts, but they sill may be tractable. For example, the conventional branch cut for the complex logarithm functions lies along the negative real axis [3, (4.1.1)]. As a consequence, $\lim_{x\to 0+} \ln(-1 + x\,i) = \pi i$ and $\lim_{x\to 0+} \ln(-1 - x\,i) = -\pi i$.

Most computer algebra systems just ignore the existence of branch cuts when computing series expansions and return results which are only valid in certain regions of the complex plane. As an example we show the Taylor series expansion of such a function on its branch cut in Macsyma.

```
(c1) taylor(sqrt(-1+x),x,0,2);
                                               2
                                  %i x    %i x
(d1)/T/                       %i - ---- - ----- + . . .
                                   2        8
```

This result is only valid on one side of the branch, namely if the imaginary part of x ($\Im x$) is greater or equal to zero, but the system gives no indication that the result is only valid under this proviso. As a consequence, a limit computation based on this series facility may be wrong as well.

```
(c2) limit(sqrt(-1-x*%i), x, 0, plus);
(d2)                                  %i
```

Note that almost all computer algebra systems return an incorrect result on the above limit problem. A notable exception is Derive.

We have solved that particular problem by using a sign function for complex expressions within the coefficients of a series. This sign function is named *csgn* in Maple and defined as follows:

$$\operatorname{csgn}(x) = \begin{cases} 1 & \text{if } \Re x > 0 \vee (\Re x = 0 \wedge \Im x > 0) \\ -1 & \text{if } \Re x < 0 \vee (\Re x = 0 \wedge \Im x < 0) \\ 0^{\dagger} & \text{if } x = 0 \end{cases}$$

With the help of that function the series may be formulated depending on the location of the expansion variable relative to the branch cut. Note that only straight branch cuts can be resolved with this technique. The series model of MAPLE allows the coefficients to depend on the expansion variable x, if their growth is less than polynomial in x. This condition is satisfied, since $\operatorname{csgn}(x)$ is piecewise constant. As a side effect, MAPLE[6] also knows that the Taylor series expansion on a branch cut does not exist, since the coefficients of Taylor series must not depend on the expansion variable.

```
> taylor(sqrt(-1+x*I), x = 0);
Error, does not have a taylor expansion, try series()

> series(sqrt(-1+x*I), x = 0, 2);
```

$$e^{-1/2\,I\,\operatorname{csgn}(I\,(-1+I\,x))\,\pi} - \frac{1}{2}\,I\,e^{-1/2\,I\,\operatorname{csgn}(I\,(-1+I\,x))\,\pi} + O\left(x^2\right)$$

Note that even if only straight branch cuts are involved, the series approximation generated by an inaccurate series facility is not only wrong "half of the time", but rather this problem may accumulate as the following example[7] shows.

```
> Ez := (z-I)/((z-I)^2)^(3/2) + (z+I)/((z+I)^2)^(3/2):
> series(Ez, z=0, 2);
```

$$\left(-I\,e^{3/2\,I\,\operatorname{csgn}(I\,(z^2-2Iz-1))\,\pi} + I\,e^{3/2\,I\,\operatorname{csgn}(I\,(z^2+2Iz-1))\,\pi}\right) + \left(-2\,e^{3/2\,I\,\operatorname{csgn}(I\,(z^2-2Iz-1))\,\pi} - 2\,e^{3/2\,I\,\operatorname{csgn}(I\,(z^2+2Iz-1))\,\pi}\right)\,z + O\left(z^2\right)$$

Within this answer four results for different regions are encoded, depending on the choice for z, i.e. the choices for the *csgn* values, namely

$$\begin{array}{rll} -2+O(z^2) & \text{for} & \Re z>0 \wedge (-1<\Im z\le 1) \\ 2+O(z^2) & \text{for} & \Re z<0 \wedge (-1\le \Im z<1) \\ 4i\,z+O(z^2) & \text{for} & (\Re z>0 \wedge \Im z\le -1) \vee (\Re z<0 \wedge \Im z\ge 1) \\ -4i\,z+O(z^2) & \text{for} & (\Re z>0 \wedge \Im z>1) \vee (\Re z<0 \wedge \Im z<-1) \vee \Re z=0 \end{array} \tag{7.10}$$

All systems which return a result for this series expansion[8] return the one which is only valid in the last region specified in (7.10), e.g. in MACSYMA we get

```
(c1) taylor((z-%i)/((z-%i)^2)^(3/2)+(z+%i)/((z+%i)^2)^(3/2),z,0,3);
                                                     3
(d1)/T/                      - 4 %i z + 8 %i z  + . . .
```

[†]The value at $x = 0$ is arbitrary defined to be zero. This definition may be changed using the environment variable *_Envsignum0*.

[6]Since Release 4 of MAPLE V.

[7]The example is taken from [5] where the limit of E_Z for $r = 0$ is discussed.

[8]AXIOM 2.0, MACSYMA 418.1, MAPLE V Release 3 and MATHEMATICA 2.2.

If we additionally considered the region of convergence for this series, then the result would be even worse, because the series only converges for the first two regions in (7.10) and for the last region, if z is on the imaginary axis only.

The algorithm which computes the leading term must be prepared for the situation that the coefficients may contain sign functions. It must resolve them by inspecting the limiting behaviour of their argument. Moreover, the zero equivalence test which is called within series may get a function which contains *csgn* functions, which must be resolved. As an example consider

$$\lim_{x\to+\infty} \frac{1}{x\,\left(1+(1/x-1)^{1/x-1}\right)} \tag{7.11}$$

which can be rewritten in terms of exponentials and logarithms. After moving up one level in the scale, the series of $\omega/(1+\exp(\ln(\omega-1)\,(\omega-1)))$ has to be computed. If we take into account, that the series expansion is done on the branch cut and that the coefficients may depend on the csgn function, then we get the following result with MAPLE's series command

```
> series(w/(1+exp(ln(w-1)*(w-1))),w,2);
```

$$\frac{1}{1+e^{I\ \mathrm{csgn}(I\,(w-1))\,\pi}}\,w+O\left(w^2\right)$$

Note that the denominator of the leading coefficient is zero most of the time, except for $\omega=1$, in which case the value of the leading coefficient depends on the definition of csgn(0). On the other hand, we know that ω tends to zero and so the series is meaningless, since again a zero has not been recognized.

We see two ways to resolve this problem in MAPLE. One possibility would be to extend the assume facility in MAPLE [92] with a new property *ArbitrarySmall* and to extend the whole system to consider this property. The function $\mathrm{csgn}(I\,(w-1))$ should then automatically simplify to -1 if w has this property. Since this would need an update of the whole library, it is a rather unpractical approach.

The solution we have implemented is again to put this extra information about ω into the zero equivalence test by overwriting the environment variable *Testzero*. If the function which is passed to *Testzero* contains sign functions which depend on the expansion variable ω, then these sign functions are replaced by their limits as ω goes to 0^+ before the function is tested for zero equivalence.

```
> restart;
> Testzero := proc(e) local e1;
>    if has(e,w) then e1 := limit(e,w=0,right)
>    else e1 := e
>    fi;
>    evalb(normal(e1) = 0)
> end:
```

For the above expression we now obtain the following series:

```
> series(w/(1+exp(ln(w-1)*(w-1))),w,2);
```

$$\frac{1}{e^{I\ \mathrm{csgn}(I\,(w-1))\,\pi}\,(-I\ \mathrm{csgn}(I\,(w-1))\,\pi+1)}+O(w)$$

If the sign functions in the leading coefficient of this series are replaced by -1 we get the expression $-1/(i\,\pi+1)$ which is the result of (7.11).

To close this section we give some other examples of directional limits on branch cuts which are solved correctly with our series model.

```
> Limit(sqrt(-1+x*I), x=0, left)  = limit(sqrt(-1+x*I), x=0, left),
> Limit(sqrt(-1+x*I), x=0, right) = limit(sqrt(-1+x*I), x=0, right);
```

$$\lim_{x\to 0-}\sqrt{-1+I\,x}=-I,\ \lim_{x\to 0+}\sqrt{-1+I\,x}=I$$

```
> limit(arctan(2*I+x), x=0, left), limit(arctan(2*I+x), x=0, right);
```

$$-\frac{1}{2}\,\pi+\frac{1}{2}\,I\,\ln(3),\frac{1}{2}\,\pi+\frac{1}{2}\,I\,\ln(3)$$

7.5 Computing with Parameters

Another general source of bugs in computer algebra algorithms is when a result depends on certain conditions which must be met by the involved parameters. Some computer algebra systems either return a result which may become wrong on specialization, or refuse to compute anything. We again take an example out of [5].

```
> V := 1/sqrt(r^2+(z-I)^2)+1/sqrt(r^2+(z+I)^2):
> Ez := -diff(V,z);
```

$$Ez := \frac{1}{2}\frac{2z-2I}{(r^2+(z-I)^2)^{3/2}}+\frac{1}{2}\frac{2z-2I}{(r^2+(z+I)^2)^{3/2}}$$

```
> limit(Ez, z=0, right);
```

$$0$$

This result is only correct for $r\geq 1$, not in general. Our implementation of series gives a note to the user whenever it makes assumptions about condition on the parameters which cannot be resolved. This way, the user gets at least a hint that the solution returned is only valid under some provisos, but in general this problem is not easy to solve, and a variety of approaches have been discussed and implemented [18].

```
> MRV[Limit](Ez, z=0, right);
PROVISO: NOT(r^2-1,negative)
```

$$0$$

Once we know this, we might specify additional information about the variables and then either get the correct result or further provisos. Assumptions about unknowns may be specified with the assume facility in MAPLE [92].

```
> assume(r^2 < 1, r, real);
> MRV[Limit](Ez, z=0, right);
```

$$-\frac{2}{(1-r\tilde{}^2)^{3/2}}$$

Such knowledge is for example needed within series, when one has to decide whether the expansion point is on a branch cut or not, and that is also what happened in the above example for the fractional power.

In the environment of computing limits with our approach, we may encounter sometimes a special situation which we will discuss next. Consider the example

$$\lim_{x\to+\infty} \ln\left(\frac{x(x+1)}{\ln\left(e^x+e^{\ln^2 x}\,e^{x^2}\right)}+\frac{1}{\ln x}\right). \tag{7.12}$$

The most rapidly varying subexpression is e^{x^2} and we set $\omega = e^{-x^2}$. The series in ω of the argument of the outer logarithm in (7.12) is

$$\frac{1}{\ln x}+\frac{x(x+1)}{\ln^2 x+x^2}+O(\omega)$$

If the series of the logarithm thereof is computed, we must compute the sign of $\frac{1}{\ln x}+\frac{x(x+1)}{\ln^2 x+x^2}$ in order to decide whether we are on the branch cut or not. The sign of this function can be decided to be 1 if we know that $x > 1$. Unfortunately, such a bound above which the sign can be computed is difficult to specify in advance. But we know that x is the variable which approaches infinity, and hence only the value of the sign for $x \to +\infty$ is relevant. Again, this information about the unknown x must be communicated to the sign test which is called in the series computation facility.

Again two approaches seem to be natural for a MAPLE realization. First, a new property, such as *ArbitraryLarge*, for the assume facility could be defined, but as we have already discussed, such global changes imply that a lot of code must be updated. The easier solution is to define also an environment variable for computing the sign of a function. If the zero equivalence test is already offered as an environment variable, it seems to be natural to also offer an environment variable for computing the sign of an expression, since these two tasks are related anyway. In the code in Appendix A, this environment variable is called *_EnvSIGNUM*.

7.6 The *limit* Routine in the Maple Library

In this last section we would like to add some information about the implementation of our algorithm which is available in the MAPLE distribution version through the *limit* command. The main difference of this implementation with the algorithm presented in the appendix is that MAPLE's *limit* facility is built on top of MAPLE's *series* function. This series model however only supports Puiseux series, i.e. the exponents are restricted to be rationals. Real exponents are converted to a rational approximation. This works fine in most of the cases, but examples where this simplification fails can be constructed. An example is

$$\lim_{x\to+\infty}\left(\mathrm{Beta}(x+e^{-x},x+e^{-x})-\mathrm{Beta}(x,x)\right)\, e^{(1+2\ln 2)x}\,\sqrt{x}=-4\ln 2\,\sqrt{\pi}$$

which returns 0 in MAPLE V. The correct result is computed by our implementation which is based on a general power series facility:

```
> (Beta(x+exp(-x),x+exp(-x))-Beta(x,x))*exp((1+2*ln(2))*x)*sqrt(x):
> MRV[Limit](", x=infinity);
                                   1/2
                             - 4 Pi    ln(2)
```

We decided to built *limit* on top of the existing *series* facility and not to build our own series tool, since then the *limit* tool can profit from extensions made in the series facility. However, we hope that eventually MAPLE itself will offer a more general series facility.

The pre-processing step can be extended in the usual MAPLE manner. If the function f needs to be pre-processed, i.e. if it contains essential singularities in its domain of definition, then the user must supply a procedure *limit/preprocess/f*. This procedure is then called from within *limit* with the actual arguments of f and it has to return the pre-processed function (where all the essential singularities are captured in exp-log functions, and all other functions are tractable, i.e. can be expanded in a power series). The arguments are passed as expressions in the variable *limit/X* and the limit is taken as *limit/X* approaches 0 from the right.

Example 7.1 As an example we extend the system by the function *ERF* which is the error function

$$\mathrm{ERF}(x)=\frac{2}{\pi}\int_0^x e^{-t^2}\,dt.$$

In order to instruct MAPLE how to expand ERF(x) into a series, it is enough to define the derivative of ERF(x). This is done by a definition of procedure *diff/ERF*.

```
> `diff/ERF` := (e,x) -> 2/sqrt(Pi)*exp(-e^2)*diff(e,x):
```

Next we define how to pre-process this new function. If the argument tends to infinity, then the function has an essential singularity which we isolate. The tractable part is defined to be the function *ERFs*.

```
> `limit/preprocess/ERF` := proc(e)
>    local lim;
>    lim := limit(e, `limit/X`=0, right);
>    if lim = infinity then 1 - ERFs(e)/exp(e^2)
>    elif lim = -infinity then -1 + ERFs(-e)/exp(e^2)
>    else ERF(e)
>    fi
> end:
```

Additionally, we must instruct MAPLE about how to expand the tractable part *ERFs*(x) into a series. The series facility of MAPLE is extended through the same mechanism, i.e. also by assigning a procedure to *series/ERFs*. Note that the argument of *ERFs* always tends to $+\infty$. Due to our model it may happen that the series of the argument has the form $c_0 + c_1\,\omega^{e_1} + \cdots$, with the leading coefficient not constant but rather tending to infinity as well, though less rapidly than ω. As a consequence we must distinguish two cases within the series expansion of $\mathrm{ERFs}(x)$.

```
> `series/ERFs` := proc(e, x)
>    local e0, s, k;
>    e0 := series(e,x);
>    if not type(e0, 'series') then RETURN(FAIL) fi;
>    ASSERT(op(2,e0)<=0);
>    if op(2,e0) = 0 then
>       s := series((1-ERF(e0))*exp(e0^2), x);
>       eval(subs(ERF=(v -> 1-ERFs(v)/exp(v^2)), s));
>    else # (5.17)
>       series(1/sqrt(Pi)*
>          sum((-1/4)^k * (2*k)! /k! /e0^(2*k+1), k=0..iquo(Order-1,2))
>          + O(1/e0^Order), x)
>    fi
> end:
```

The extension is now complete and *limit* knows about the new function $\mathrm{ERF}(x)$. The only thing MAPLE does not know yet is the relation to other MAPLE function in order to be able to test for zero equivalence. This can be achieved by redefining *Testzero* accordingly.

```
> Testzero := subs(T=eval(Testzero), proc(e) T(subs(ERF=erf, e)) end:
> limit(exp(x)*exp(x^2)*(ERF(x+1/exp(x))-ERF(x)), x=infinity);
```

$$\frac{2}{\sqrt{\pi}}$$

¶

Example 7.2 The following example is slightly more complicated and shows how additional knowledge can be added to the limit facility through the pre-processing transformation. We teach MAPLE to handle Bessel functions whose *order* tends to infinity. According to [3, (9.3.7)] we have

$$J_\nu(\nu \operatorname{sech} \alpha) \sim \frac{e^{\nu(\tanh \alpha - \alpha)}}{\sqrt{2\pi\nu \tanh \alpha}} \left(1 + \sum_{k=1}^{\infty} \frac{u_k \coth \alpha}{\nu^k}\right)$$

for $\nu \to \infty$ through real positive values. α is assumed to be fixed and positive. The definition of the u_k is given by a recursion in [3, (9.3.10)]. We can pass this information into the limit computation by defining the procedure *preprocess/mrv/BesselJ*. The arguments are given as expressions in the variable *limit/X*, and the limit is taken for *limit/X* towards zero from the right side. We directly access the procedures defined in the limit library code.

```
> alias(PreProcess  = `limit/mrv/PreProcess`):
> alias(MrvLeadTerm = `limit/mrv/MrvLeadTerm`):
> alias(Sign        = `limit/mrv/Sign`):
> `limit/preprocess/BesselJ` := proc(n,e)
>    local lte, ltn, alpha;
>    lte := MrvLeadTerm(PreProcess(e));
>    ltn := MrvLeadTerm(PreProcess(n));
>    if ltn[3] < 0 and lte[3] < 0 and traperror(Sign(ltn[1])) = 1 then
>       # A&S [3]: 9.3.7
>       alpha := arcsech(MrvLimit(PreProcess(e)/PreProcess(n)));
>       alpha := convert(alpha, ln);
>       if signum(0,alpha,0) = 1 then
>          exp(n*(tanh(alpha)-alpha))/sqrt(2*Pi*n*tanh(alpha))*
>              BesselJs(alpha,n)
>       else _Range
>       fi
>    elif lte[3] < 0 then _Range # signal for oscillating functions
>    else BesselJ(n,e)
>    fi;
> end:
```

Together with the definition for *series/BesselJs* and after some simplifications we obtain

$$\lim_{x\to\infty} J_{(e^2+1)x}(2e\,x)e^{2x}\sqrt{x} = \frac{1}{\sqrt{2\pi}\sqrt{e^2-1}}.$$

¶

8. Comparison

In this section we compare different algorithms for computing limits which are offered by the currently available computer algebra systems. Such a list of examples is useful as a benchmark to test further limit algorithms, but it also shows the power and the weaknesses of today's computer algebra systems. This comparison shows in particular that some of today's commercially available computer algebra system still use methods based on the ideas presented in Section 2.3. In particular, we tested the following limit algorithms:

MrvLimit: This is our algorithm as it is described in this thesis. We used the version which is presented in Appendix A. It is built on the lazy evaluated general sparse power series facility described in Section 7.1.2. Another implementation of this algorithm is available through the *limit* command in MAPLE V since Release 3 (cf. Section 7.6).

JSLimit: This algorithm refers to our implementation of Shackell's algorithm to compute nested forms. The limit of a function is read off its nested form. Note that the implementation is restricted to exp-log functions.

glimit: This is a function out of the MAPLE *gdev* package for asymptotic expansions written by B. Salvy [70]. *glimit* is using a generalized series approach. We used the 1994 version which runs in MAPLE V Release 3 and which is available through ftp from *ftp.inria.fr* in the directory */INRIA/Projects/algo/programs/gdev*.

Mathematica: MATHEMATICA offers two limit functions, one in the kernel and one in the package *Calculus/Limit.m*. The kernel limit function is based on heuristics, whereas the limit package uses a generalized series expansion approach. The latter has been written by V. Adamchik [4] and is a development version for the evaluation of limits which may replace the algorithm in the kernel in the future. The limit package is available in every MATHEMATICA installation since Version 2.1 and can be loaded with the command *<<Calculus/Limit.m*. We used the version available in MATHEMATICA 2.2.

Macsyma: MACSYMA also offers two limit functions: *limit* and *tlimit*. *limit* uses a heuristic approach as described in [89, 90] whereas *tlimit* calls *limit* with a flag indicating that Taylor series have to be used as first heuristic whenever possible. We ran the tests with Macsyma Version 418.1. If nothing special is mentioned in the footnotes, the *limit* command was used.

Derive: The limit algorithm in Derive is based on heuristics. We tested Derive Version 2.6.

Axiom: Axiom's limit function is based on a power series approach. We ran the tests with Axiom Release 2.0a.

Reduce: The limit package of Reduce is based on heuristics. For non-critical limits a power series approach is used, and to resolve singularities l'Hôpital's rule is applied. A limited amount of bounded arithmetic is also employed where applicable. This package has been implemented by L. Kameny [42]. We tested the limit facility available in Reduce Version 3.6.

We must admit that the test examples we choose are rather difficult and in particular test the behavior of the algorithms on cancellations and essential singularities. The examples (8.1-8.8) are all examples which lead to the cancellation problem if the general power series algorithm is applied directly. For the examples (8.5-8.8) this can be seen if the quotient of the two exponentials is combined into an exponential of a difference. That is also the reason why *gdev* and Axiom give up on most of these examples.

The example (8.9) is taken from Hardy [36]. He used it to describe the difficulty of finding an asymptotic scale. The only scale in which this function can be expanded is a scale which contains the function itself.

The problem (8.15) has been chosen randomly and does not contain special trapdoors. Its result is obvious. It is surprising however that most of the systems cannot solve this problem. Finally, note that the test (8.18) has been discussed in Example 3.21 and (8.19) in Example 3.25.

The kernel limit function of Mathematica could not solve any of the problems. It returned unevaluated for all examples, most times after an error message indicating that an indeterminate expression of the form $\infty - \infty$ was encountered. We only present the results of the *Calculus/Limit.m* package.

$$\lim_{x \to +\infty} e^x \left(e^{1/x - e^{-x}} - e^{1/x}\right) = -1 \tag{8.1}$$

$$\lim_{x \to +\infty} e^x \left(e^{1/x + e^{-x} + e^{-x^2}} - e^{1/x - e^{-e^x}}\right) = 1 \tag{8.2}$$

$$\lim_{x \to +\infty} e^{e^{x - e^{-x}}/(1-1/x)} - e^{e^x} = +\infty \tag{8.3}$$

$$\lim_{x\to+\infty} e^{e^{(e^x/(1-1/x))}} - e^{e^{(e^x/(1-1/x-\ln(x)^{-\ln(x)}))}} = -\infty \tag{8.4}$$

$$\lim_{x\to+\infty} \frac{e^{e^{e^x+e^{-x}}}}{e^{e^{e^x}}} = +\infty \tag{8.5}$$

$$\lim_{x\to+\infty} \frac{e^{e^{e^x}}}{e^{e^{e^x-e^{-e^x}}}} = +\infty \tag{8.6}$$

$$\lim_{x\to+\infty} \frac{e^{e^{e^x}}}{e^{e^{e^x-e^{-e^{e^x}}}}} = 1 \tag{8.7}$$

$$\lim_{x\to+\infty} \frac{e^{e^x}}{e^{e^{x-e^{-e^{e^x}}}}} = 1 \tag{8.8}$$

$$\lim_{x\to+\infty} (\ln x)^2\, e^{\sqrt{\ln x}\,(\ln\ln x)^2\, e^{\sqrt{\ln\ln x}\,(\ln\ln\ln x)^3}} / \sqrt{x} = 0 \tag{8.9}$$

$$\lim_{x\to+\infty} \frac{x\ \ln x\ \left(\ln\left(x\,e^x - x^2\right)\right)^2}{\ln\ln\left(x^2 + 2\,e^{e^{3x^3\ \ln x}}\right)} = \frac{1}{3} \tag{8.10}$$

$$\lim_{x\to+\infty} \left(e^{x\,e^{-x}/\left(e^{-x}+e^{-\frac{2x^2}{x+1}}\right)} - e^x\right)/x = -\exp(2) \tag{8.11}$$

$$\lim_{x\to+\infty} \left(3^x + 5^x\right)^{1/x} = 5 \tag{8.12}$$

$$\lim_{x\to+\infty} x/\ln\left(x^{\ln x^{\ln 2/\ln x}}\right) = +\infty \tag{8.13}$$

$$\lim_{x\to+\infty} \frac{e^{e^{2\ln(x^5+x)\ \ln\ln x}}}{e^{e^{10\ln x\ \ln\ln x}}} = +\infty \tag{8.14}$$

$$\lim_{x\to+\infty} 4/9\frac{\exp\left(e^{5/2\,x^{-5/7}+21/8\,x^{6/11}+2\,x^{-8}+54/17\,x^{49/45}}\right)^8}{\ln\left(\ln\left(-\ln\left(4/3\,x^{-5/14}\right)\right)\right)^{7/6}} = +\infty \tag{8.15}$$

$$\lim_{x\to+\infty} \frac{\exp(4x\,e^{-x}/(1/e^x + 1/\exp(2x^2/(x+1)))) - e^x}{(e^x)^4} = 1 \tag{8.16}$$

$$\lim_{x\to+\infty} \frac{\exp\left(\frac{x\,e^{-x}}{e^{-x}+e^{-2x^2/(x+1)}}\right)}{e^x} = 1 \tag{8.17}$$

$$\lim_{x\to+\infty} \frac{e^{e^{-x/(1+e^{-x})}}\ e^{-\frac{x}{1+e^{-x/(1+e^{-x})}}}\ e^{e^{-x+e^{-x/(1+e^{-x})}}}}{\left(e^{-x/(1+e^{-x})}\right)^2} - e^x + x = 2 \tag{8.18}$$

$$\lim_{x\to+\infty} \frac{\ln(\ln x + \ln\ln x) - \ln\ln x}{\ln(\ln x + \ln\ln\ln x)}\ \ln x = 1 \tag{8.19}$$

$$\lim_{x\to+\infty} \exp\left(\frac{\ln\ln\left(x + e^{\ln x\ \ln\ln x}\right)}{\ln\ln\ln\left(e^x + x + \ln x\right)}\right) = e \tag{8.20}$$

	MrvLim	JSLim	GLimit	Math	Macs	Derive	Axiom	Reduce
8.1	-1	-1	uneval	-1	-1	-1	failed	uneval
8.2	1	1	uneval	0	1	1	failed	uneval
8.3	∞	∞	uneval	0	$-\infty$	∞	failed	uneval
8.4	$-\infty$	$-\infty$	uneval	()[2]	uneval	()[6]	failed	uneval
8.5	∞	∞	uneval	1	∞	uneval[7]	failed	uneval
8.6	∞	∞	∞	e^{-e}	uneval	()[6]	failed	()[4]
8.7	1	1	1	e^{-e}	uneval	uneval[7]	failed	()[4]
8.8	1	1	1	∞	1	1	failed	uneval
8.9	0	0	0	∞	0[5]	uneval[7]	0	uneval
8.10	1/3	1/3	1/3	()[3]	uneval	()[6]	failed	uneval
8.11	$-e^2$	$-e^2$	0	0	uneval	()[6]	failed	uneval
8.12	5	5	5	e	5	()[6]	failed	1
8.13	∞	∞	∞	()[2]	∞	∞	failed	∞
8.14	∞	∞	uneval	1	1[5]	uneval[7]	1	uneval
8.15	∞	∞	uneval	()[4]	uneval	()[6]	()[4]	uneval
8.16	1	1	1	0	1	()[6]	failed	uneval
8.17	1	1	1	0	1	()[6]	failed	uneval
8.18	2	()[1]	∞	0	uneval	uneval[7]	()[4]	()[4]
8.19	1	1	1	1	uneval	uneval[7]	failed	uneval
8.20	e	e	e	1	uneval	()[6]	failed	uneval

Table 8.1. Comparing different limit algorithms and packages on exp-log functions

[1] The current implementation exhausts all memory on this example as the intermediate results get too large.
[2] Mathematica returns `Indeterminate` which represents a numerical quantity whose magnitude cannot be determined.
[3] After issuing some error messages Mathematica exits with the message `Out of memory`.
[4] Stopped after several hours.
[5] Only when using *tlimit*. Macsyma returns unevaluated when using *limit*.
[6] The problem cannot be solved and a `Memory full` message appears.
[7] The function has been transformed into another, more complicated form.

It is difficult to compare timings as the systems behave so differently, i.e. run for ever or return unevaluated immediately. Maple used less than 20 seconds on a Sun4 for the twenty exp-log examples with both implementation, i.e. with the library version and with the version given in the Appendix.

The results on Example (8.18) are very interesting. They show that a result may not be correct, even if it is returned by three different computer algebra systems!

The second collection of problems contains limits of functions in more difficult function fields. The emphasis is again on the cancellation problem for the first ten examples, on which *glimit* gives up, as expected. Again some of the examples have already been discussed, namely test (8.32) in Example 5.5 and test (8.35) in Example 5.6.

The limit function in the kernel of Mathematica could not solve any of these limits and returned unevaluated for all of them, sometimes after error messages indicating that an indeterminate expression was encountered or an essential singularity was found during the series expansion of the Γ function. We therefore only present the results of the *Calculus/Limit.m* package. This package however seems to have many implementation bugs. We have reported them to Wolfram Research Inc.

In order to compute these limits in Reduce, the *specfunc* package has to be loaded first. Reduce and in particular Derive are rather poor on these special functions in contrast to the exp-log problems. It seems that special functions are not incorporated that well in these systems. This may be an indication that the algorithm which is using heuristics is difficult to extend.

The *MrvLimit* algorithm used here is an extension over the one presented in the Appendix A which preprocesses functions with essential singularities. The *limit* function in Maple V Release 4 returns the same results except for the test (8.25). This problem cannot be solved as the underlying series facility is not powerful enough and thus Maple's *limit* returns unevaluated. The Maple V library version of our algorithm needed 16 seconds to solve the special limit problems and the version based on lazy evaluated series used 35 seconds on a Sun4.

$$\lim_{x\to+\infty} e^x \left(\sin(1/x+e^{-x})-\sin(1/x+e^{-x^2})\right)=1 \tag{8.21}$$

$$\lim_{x\to+\infty} e^{e^x} \left(e^{\sin(1/x+e^{-e^x})}-e^{\sin(1/x)}\right)=1 \tag{8.22}$$

$$\lim_{x\to+\infty} (\operatorname{erf}(x-e^{-e^x})-\operatorname{erf}(x))\, e^{e^x}\, e^{x^2}=-2/\sqrt{\pi} \tag{8.23}$$

$$\lim_{x\to+\infty} (\operatorname{Ei}(x-e^{-e^x})-\operatorname{Ei}(x))\, e^{-x}\, e^{e^x}\, x=-1 \tag{8.24}$$

$$\lim_{x\to+\infty} e^{(\ln 2+1)\,x} \left(\zeta(x+e^{-x})-\zeta(x)\right)=-\ln 2 \tag{8.25}$$

$$\lim_{x\to+\infty} e^x \left(\Gamma(x+e^{-x})-\Gamma(x)\right)=+\infty \tag{8.26}$$

$$\lim_{x\to+\infty} \exp(\Gamma(x-e^{-x})\,\exp(1/x))-\exp(\Gamma(x))=+\infty \tag{8.27}$$

$$\lim_{x\to+\infty} (\Gamma(x+1/\Gamma(x))-\Gamma(x))/\ln(x)=1 \tag{8.28}$$

$$\lim_{x\to+\infty} x\,(\Gamma(x-1/\Gamma(x))-\Gamma(x)+\ln(x))=\frac{1}{2} \tag{8.29}$$

$$\lim_{x\to+\infty} \left(\frac{\Gamma(x+1/\Gamma(x))-\Gamma(x)}{\ln x}-\cos(1/x)\right)\, x\,\ln x=-\frac{1}{2} \tag{8.30}$$

$$\lim_{x\to+\infty} \frac{\Gamma(x+1)}{\sqrt{2\pi}}-e^{-x}\left(x^{x+1/2}+x^{x-1/2}/12\right)=+\infty \tag{8.31}$$

$$\lim_{x\to+\infty} \ln(\Gamma(\Gamma(x)))/e^x=+\infty \tag{8.32}$$

$$\lim_{x\to+\infty} \exp(\exp(\psi(\psi(x))))/x=\exp(-1/2) \tag{8.33}$$

$$\lim_{x\to+\infty} \exp(\exp(\psi(\ln(x))))/x=\exp(-1/2) \tag{8.34}$$

$$\lim_{x\to+\infty} \exp(\exp(\exp(\psi(\psi(\psi(x))))))/x=0 \tag{8.35}$$

$$\lim_{x\to+\infty} J_{2x}(x)\, e^{x\,(2\ln(2+\sqrt{3})-\sqrt{3})}\,\sqrt{x}=\sqrt{\frac{1}{2\sqrt{3}\pi}} \tag{8.36}$$

$$\lim_{x\to+\infty} \frac{\max(x,e^x)}{\ln(\min(e^{-x},e^{-e^x}))}=-1 \tag{8.37}$$

	MrvLim	GLimit	Math	Macs	Derive	Axiom	Reduce
8.21	1	uneval	∞	uneval	$()^8$	failed	uneval
8.22	1	uneval	$-\infty$	uneval	$()^8$	failed	uneval
8.23	$-2/\sqrt{\pi}$	uneval	0	uneval	uneval	$()^9$	uneval
8.24	-1	uneval	$-\infty$	uneval[4]	0	$()^9$	uneval
8.25	$-\ln(2)$	uneval	uneval[1]	uneval	uneval	$()^{11}$	uneval
8.26	∞	uneval	0^2	uneval	uneval	failed	$()^{12}$
8.27	∞	uneval	$()^3$	uneval	uneval	failed	$()^{12}$
8.28	1	uneval	0^2	$()^5$	uneval	failed	$()^{12}$
8.29	$1/2$	uneval	0^2	$()^5$	uneval	failed	$()^{12}$
8.30	$-1/2$	uneval	0^2	$()^5$	uneval	failed	$()^{12}$
8.31	∞	∞	$-\infty$	$-\infty$	$()^8$	failed	$()^{12}$
8.32	∞	uneval	0	∞^6	uneval	failed	$()^{12}$
8.33	$e^{-1/2}$	$\frac{1}{e^{1/2}}$	$1/\sqrt{e}$	uneval	$()^9$	failed	$()^{12}$
8.34	$e^{-1/2}$	$\frac{1}{e^{1/2}}$	$1/\sqrt{e}$	0^7	$()^8$	failed	$()^{12}$
8.35	0	uneval	∞	uneval	$()^9$	failed	$()^{12}$
8.36	$\frac{1}{2}\sqrt{\frac{2}{\pi\sqrt{3}}}$	uneval	uneval	uneval	$?^{10}$	failed	uneval
8.37	-1	uneval	uneval	uneval	uneval	$-\infty$	uneval

Table 8.2. Comparing different limit algorithms and packages on general functions

[1] After some error messages from *Series*.
[2] After some error messages which say that a local variable in the *Calculus/Limit* package does not have appropriate bounds.
[3] After some error messages the empty list {} is returned as result.
[4] Macsyma asks here whether `exp_int(inf)` is positive, negative or zero. Independent of the answer, it returns unevaluated.
[5] The system error `Bind stack overflow`, `Caught fatal error` or `Unrecoverable error` was issued.
[6] Only when using *tlimit*. The *limit* command causes a system error.
[7] Only when using *tlimit*. The *limit* command returns unevaluated.
[8] The problem cannot be solved and a `Memory full` message appears.
[9] Stopped after several hours.
[10] A question mark means that the limit is undefined.
[11] Axiom does not know the Riemann Zeta function.
[12] The system error `Binding stack overflow, restarting...` was issued.

9. Conclusions

We have presented a new algorithm for computing limits of exp-log functions. The algorithm overcomes the cancellation problem other algorithms suffer from. The algorithm is very compact, easy to understand, easy to prove and easy to implement. This goal was achieved using a uniform method, namely to expand the *whole* function into a series in terms of its most rapidly varying subexpression. This way, the size of the problem is reduced in every iteration step and the size of the intermediate expressions are kept under control. In this point our algorithm differs substantially from the bottom-up algorithms which solve the cancellation problem. As a consequence, our algorithm is in particular suited to the implementation in a symbolic manipulation system. For particular limits, our algorithm may even be used as straight forward technique to compute the result using paper and pencil.

We have also shown how the algorithm can be extended to handle special functions. We have used a practical approach which tries to rewrite functions so that all essential singularities are captured by exponentials and logarithms only. This allows us to apply the algorithm for exp-log functions directly, provided that the underlying series facility is powerful enough.

As an underlying tool our algorithm uses a series computation facility which must support arbitrary real exponents. We have shown an implementation of a lazy evaluated model within the MAPLE system. The task of analyzing the comparability classes of a function and the series computation facility are nicely separated with our algorithm. When computing a series we never have to worry about comparability classes as e.g. new ones may never evolve. Whenever a series has to be computed, it is guaranteed, that the function can be expanded into a power series. The only relation between the actual asymptotic scale in which the expansion is performed and the series facility is the value of $\ln \omega$.

As a byproduct of the algorithm for computing limits we have described an algorithm for computing asymptotic expansions. The asymptotic scale which is used is thereby determined on the fly.

The algorithm has already been extended recently by van der Hoeven [88, 64] in order to expand solutions to polynomial equations into asymptotic series as well as to handle differentiation, integration, functional composition and inversion. The basic technique which is used is *multiseries*, which is equivalent to our hierarchical series. Moreover, instead of calculating the most rapidly varying subexpression of the whole function a reduced basis is gradually computed. This way unnecessary order comparisons are avoided. As we have seen, such a reduced basis is also computed implicitly by our algorithm.

Our algorithm is available through MAPLE's *limit* function since MAPLE V Release 3. When we have compared our algorithm with the algorithms available in other commercially available computer algebra systems we have discovered, that almost all of these algorithms still use heuristics to solve the limit problem. As a consequence they badly fail on the examples we tested. We think that we have filled a gap with our algorithm and expect that it will soon be implemented in other systems.

The exp-log version of our algorithm is already available in MuPAD [25] Version 1.2.2. It has been implemented by F. Postel [94] and does replace a previous version which was built on top of a Puiseux series facility, i.e. which was a member of the class of algorithms described in Section 2.3.2. Also for the REDUCE system it was planned to extend the limit computation facility using our approach [43].

A. Maple Code for Computing Limits of exp-log Functions

The following module contains the MAPLE code to compute limits of arbitrary real exp-log functions. It is written for MAPLE V Release 4. The code assumes that there exists a procedure *Series* which computes a series approximation of a given function as well as a procedure *LeadTerm* which determines the leading term of a given series. The latter procedure accepts a second argument which controls the search for the leading term in the case that the leading coefficient is zero. If the leading exponent is larger than the second argument then the search is stopped.

The procedure *MODULE* provides modules in MAPLE and allows information hiding. The objects (procedures and variables including environment variables) which are declared to be local to this module cannot be accessed from outside. Only those objects which are explicitly exported are visible outside the scope of the module. They are stored into a table (package) whose name is the name of the module.

```
##############################################################################################
#
# Limit computing facility for exp-log functions
# Copyright 1994 D. Gruntz, Wissenschaftliches Rechnen, ETH Zurich`;
#
##############################################################################################

MODULE(MRV,
# Exported procedures
   [Limit],
# Local procedures and variables
   [MrvLeadTerm, Mrv, Max, Compare, Rewrite, Sign, MoveUp, MoveDown, Simplify,
    ResolveCsgn, ResolveSignum, PROVISO, AssumePositive, ClearAssumption,
    ClearRememberTable, x, _EnvPreProcessCsgn, _EnvLimitW, _EnvLimitWinv]
):
# Environment Variables:
#   Testzero       : used within series (cf. Section 7.2)
#   _EnvSIGNUM     : used to resolve signum expressions (cf. Section 7.4)

Limit := proc(e::algebraic, limpoint::equation, direction::name)
   local z, z0, r, e0, e1, X;

   z := lhs(limpoint); z0 := rhs(limpoint);
   if   z0 =  infinity then e0 := subs(z= x, e):
   elif z0 = -infinity then e0 := subs(z=-x, e):
```

```
   elif nargs = 3 and direction = left then e0 := subs(z=z0-1/x, e)
   else e0 := subs(z=z0+1/x, e)
   fi;


   _EnvSIGNUM := proc(e) local s;
      s := signum(0, Limit(e, x=infinity), 0):
      if s = 0 then s := 1/signum(0, Limit(1/e, x=infinity), 0) fi;
      s
   end:

   Testzero := subs('TESTZERO' = eval(Testzero), proc(e) local e0, e1, e2, t, X;
      e0 := e:
      e1 := select(has, indets(e0, {specfunc(anything,'csgn'),
                                    specfunc(anything,'signum')}), {x,_EnvLimitW});
      if e1 <> {} then AssumePositive(X);
         for t in e1 do
            if op(0,t) = csgn then
               e2 := ResolveCsgn(subs(_EnvLimitW=_EnvLimitWinv,op(1,t)), X);
               if type(e2, integer) then e0 := subs(t = e2, e0) fi
            elif op(0,t) = signum then
               e0 := subs(t = ResolveSignum(subs(_EnvLimitW=_EnvLimitWinv,op(1,t)), X), e0)
            fi
         od;
         e0 := eval(e0);
         ClearAssumption(X)
      fi;
      TESTZERO(e0)
   end):

   ClearRememberTable(MrvLeadTerm); ClearRememberTable(Mrv);
   e1 := MrvLeadTerm(Simplify(e0));

   if testeq(e1[3]) then r := expand(e1[1])
   elif signum(0, e1[3], 0) =  0 then r := expand(e1[1])
   elif signum(0, e1[3], 0) =  1 then r := 0
   elif signum(0, e1[3], 0) = -1 then r := Sign(e1[1])*infinity
   else ERROR(`cannot determine the sign of `, e1[3])
   fi;

   r
end: # Limit

MrvLeadTerm := proc(e::algebraic)
# returns the leading term c0*w^e0 of the series of e in terms of the most rapidly
# varying subexpression w. The results has the form [c0, w, e0].
   local e0, e1, e2, e3, m0, subspat, Winv, s, t, W, X;
   option remember;

   e0 := e;
   e1 := select(has,
            indets(e0, {specfunc(anything, 'csgn'), specfunc(anything, 'signum')}), x);
   if e1 <> {} then AssumePositive(X);
      for t in e1 do
         if op(0,t) = csgn then e2 := ResolveCsgn(op(1,t), X);
            if type(e2, integer) then e0 := subs(t = e2, e0) fi
         elif op(0,t) = signum then
            e0 := subs(t = ResolveSignum(op(1,t), X), e0)
         fi
      od;
      e0 := eval(e0): ClearAssumption(X)
   fi;

   if not has(e0, x) then RETURN([e0,1,0]) fi;

   if nargs = 2 then m0 := select((t,e) -> has(e, t), args[2], e0);
      if m0 = {} then m0 := Mrv(e0) fi;
   else m0 := Mrv(e0);
   fi;
   if member(x, m0) then RETURN(MoveDown(MrvLeadTerm(MoveUp(e0), MoveUp(m0)))) fi;

   AssumePositive(W); _EnvLimitW := W;

   subspat := Rewrite(m0, W);
   Winv := subs( map( x->(op(2,x)=op(1,x)), subspat ), W );

   e3 := op(Winv);
   if Sign(e3) = 1 then subspat := subs(W=1/W,subspat): Winv := 1/Winv: e3 := -e3 fi;
   ln(1/W) := -e3;  ln(W) := e3; _EnvLimitWinv := Winv;

   e1 := eval(subs(op(subspat), e0));
   if not has(e1, W) then e2 := e1: e3 := 0:
   else e2 := LeadTerm(Series(e1, W), 0); e3 := e2[1][2]; e2 := e2[1][1];
   fi;

   ln(1/W) := 'ln(1/W)':  ln(W) := 'ln(W)':
   e2 := eval(subs(W = Winv, e2));
   s := signum(0, e3, 0);
   if not type(s, integer) then s := testeq(e3);
      if   s = true  then s := 0; e3 := 0
      elif s = false then s := 1
      else ERROR(`could not decide zeroequivalence of `, e3)
      fi
   fi;

   ClearAssumption(W);

   if s <> 0 then RETURN( [e2,Winv,e3] ) fi;
   MrvLeadTerm(e2)
end: # MrvLeadTerm

Mrv := proc(e)
   local c, d, m;
   option remember;

   if not has(e, x) then {}
   elif e = x then {x}
   elif type(e, `*`) then Max( Mrv(op(1,e)), Mrv(subsop(1=1, e)))
   elif type(e, `+`) then Max( Mrv(op(1,e)), Mrv(subsop(1=0, e)))
   elif type(e, `^`) then
      if not has(op(2,e), x) then Mrv(op(1,e)) else Mrv(exp(ln(op(1,e))*op(2,e))) fi;
   elif type(e, 'ln(algebraic)') then Mrv(op(e))
   elif type(e, 'exp(algebraic)') then c := Mrv(op(e)); m := MrvLeadTerm(op(e));
      if m[3] < 0 then d := Compare(e, c[1]);
         if d = `>` then {e} elif d = `<` then c else {e} union c fi
      else c
      fi
   elif type(e, function) and nops(e)=1 then Mrv(op(e))
   elif type(e, function) and nops(e)=2 then Max( Mrv(op(1,e)), Mrv(op(2,e)))
   else ERROR(`unknown expr`,e)
   fi
end: # Mrv

Max := proc(f, g)
   local c;

   if f = {} then g
   elif g = {} then f
   elif f intersect g <> {} then f union g
   elif member(x, f) then g
   elif member(x, g) then f
   else c := Compare(f[1],g[1]);
      if c = `>` then f elif c = `<` then g else f union g fi
   fi
end: # Max
```

```
Compare := proc(f, g)
   local lnf, lng, c, s;

   if type(f, exp(anything)) then lnf := op(f) else lnf := ln(f) fi;
   if type(g, exp(anything)) then lng := op(g) else lng := ln(g) fi;
   c := MrvLeadTerm(lnf/lng); s := signum(0, c[3], 0);
   if s = -1 then `>`
   elif s = 1 then `<`
   elif s = 0 then `=`
   else ERROR(`sign could not be determied`)
   fi
end: # Compare

Rewrite := proc(m::set(exp(algebraic)), W::name)
   local f, g, A, c, m0, subspat:

   if nargs = 3 then g := args[3]
   else g := m[1];
      for f in m do if length(f) < length(g) then g := f fi od
   fi;

   m0 := sort(convert(m, list), (a,b)->evalb(nops(Mrv(a)) >= nops(Mrv(b))));
   subspat := NULL:

   for f in m0 do c := MrvLeadTerm(op(f)/op(g));
      ASSERT(c[3]=0, `Elements must be in the same comparability class`);
      c := c[1]; A := exp(op(f)-c*op(g)); subspat := subspat, f = A*W^c;
   od;
   [subspat]
end: # Rewrite

MoveUp := proc(e)
   eval(subs([ln(x)=x, x=exp(x)], e))
end: # MoveUp

MoveDown := proc(e)
   eval(subs([exp(x) = x, x = ln(x)], e))
end: # MoveDown

Simplify := proc(e)
   if type(e, {`+`,`*`,'function'}) then map(Simplify, e)
   elif type(e, `^`) then
      if has(op(2,e),x) then exp(ln(Simplify(op(1,e)))*Simplify(op(2,e)))
      else Simplify(op(1,e))^op(2,e)
      fi
   else subs(E=exp(1), e)
   fi
end: # Simplify

ResolveCsgn := proc(e, X::name)
   local e0, e1, lt;

   lt := 1;
   e0 := MrvLeadTerm(e); lt := lt * e0[2]^e0[3]:
   e1 := csgn(subs(x=X,e0[1]));
   while not type(e1, 'integer') and has(e1, x) do
      e0 := MrvLeadTerm(e0[1]); lt := lt * e0[2]^e0[3]:
      e1 := csgn(subs(x=X,e0[1]))
   od;
   if Re(subs(x=X, e0[1]))=0 then
      if not assigned(_EnvPreProcessCsgn) then
         if Re(subs(x=X, e)) = 0 then
            e1
         else _EnvPreProcessCsgn := Order;
            e0 := ResolveCsgn(e - e0[1]*lt, X);
            if e0 = UNKNOWN then PROVISO(e,` is pure imaginary`); e1
            else e0
            fi
         fi
      else
         _EnvPreProcessCsgn := _EnvPreProcessCsgn-1;
         if _EnvPreProcessCsgn > 0 then procname(e - e0[1]*lt)
         else UNKNOWN
         fi
      fi;
   else
      if type(e1,'integer') then e1 else FAIL fi
   fi
end: # ResolveCsgn

ResolveSignum := proc(e, X::name)
   local e0, e1;

   e0 := MrvLeadTerm(e);
   e1 := signum(subs(x = X,e0[1]));
   while has(e0[1], x) do
      e0 := MrvLeadTerm(e0[1]);
      e1 := signum(subs(x = X,e0[1]))
   od;
   e1
end: # ResolveSignum

Sign := proc(e)
   local sig;

   if not has(e, x) then sig := signum(0, e, 0);
      if sig = 0 then ERROR(`e must not be zero`)
      elif type(sig, integer) then sig
      else ERROR(`cannot compute the sign of`,e)
      fi
   elif type(e,`*`) then map(Sign, e)
   elif e = x then 1
   elif type(e, 'exp(algebraic)') then 1
   elif type(e,`^`) then
      if op(1,e) = x then 1
      elif Sign(op(1,e)) = 1 then 1
      else Sign(MrvLeadTerm(e)[1])
      fi
   elif type(e, function) or type(e, `+`) then
      Sign(MrvLeadTerm(e)[1])
   else ERROR(`cannot compute the sign of `, e)
   fi
end: # Sign

AssumePositive := proc(e)
   global `property/object`;
   readlib('assume'):
   `property/object`[ e ] := RealRange(Open(0), infinity):
end: # AssumePositive

ClearAssumption := proc(e)
   global `property/object`;
   `property/object`[ e ] := evaln(`property/object`[ e ]):
end: # ClearAssumption

ClearRememberTable := proc(p::procedure)
   assign(p, subsop(4=NULL, op(p)))
end:

PROVISO := proc() lprint(`PROVISO: `,args) end:

AssumePositive(x):

END(MRV):
```

Bibliography

[1] S.K. Abdali, G.W. Cherry and N. Soiffer, *An Object Oriented Approach to Algebra System Design,* Proceedings of the 1986 Symposium on Symbolic and Algebraic Computation SYMSAC'86 (B.W. Char, ed.), pp. 24–30, 1986.

[2] H. Abelson and G. Sussman, *Structure and Interpretation of Computer Programs*, The MIT Press, Cambridge Mass, 1985.

[3] M. Abramowitz and I.A. Segun, *Handbook of Mathematical Functions,* Dover Publications, New York, 1968.

[4] V.S. Adamchik, *Mathematica Package* `Limit.m`, delivered with Mathematica since version 2.0, 1991.

[5] J.M. Aguirregabiria, A. Hernández and M. Rivas, *Are we Careful Enough when Using Computer Algebra?,* Computers in Physics **8** (1), pp. 56–61, 1994.

[6] J. Ax, *Schanuel's conjecture,* Annals of mathematics **93**, Series 2, pp. 252–268, 1971.

[7] C. Blatter, *Ingenieur Analysis*, VDF Verlag der Fachvereine, Zürich, 1989.

[8] R.P. Boas, *Counterexamples to L'Hôpital's rule,* American Mathematical Monthly **93**, pp. 644–645, 1986.

[9] M. Boshernitzan, *An Extension of Hardy's class L of "Orders of Infinity",* J. Analyse Math **39**, pp. 235-255, 1981.

[10] N. Bourbaki, *Éléments de Mathémathique,* Fonctions d'une variable réelle, nouvelle edition, Hermann, Paris, 1976.

[11] C. Brezinski, *Accéleration de la Convergence en Analyse Numérique,* Lecture Notes in Mathematics 584, Springer-Verlag, 1977.

[12] M. Bronstein, *Simplification of Real Elementary Functions,* Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'89 (G.H. Gonnet, ed.), pp. 207–211, 1989.

[13] I.N. Bronštein, K.A. Semendjajew, G. Musiol and H. Mühlig, *Taschenbuch der Mathematik,* 2. Auflage, Verlag Harri Deutsch, Thun, Frankfurt am Main, 1995.

[14] W.H. Burge and S.M. Watt, *Infinite Structures in Scratchpad II,* Proceedings of the European Conference on Computer Algebra EUROCAL'87 (J.H. Davenport, ed.), Lecture Notes in Computer Science **378**, Springer Verlag, pp. 138–148, 1987.

[15] B.F. Caviness, *On Canonical Forms and Simplification,* Journal of the ACM **17** (2), pp. 385–396, 1970.

[16] B.F. Caviness and H.I. Epstein, *A Note on the Complexity of Algebraic Differentiation,* SIGSAM Bulletin **11** (3), pp. 4–6, 1977.

[17] B.F. Caviness and M.J. Prelle, *A Note on Algebraic Independence of Logarithmic and Exponential Constants,* SIGSAM Bulletin **12**, pp. 18–20, 1978.

[18] R.M. Corless and D.J. Jeffrey, *Well ... It isn't Quite That Simple,* SIGSAM Bulletin **26** (3), pp. 2–6, 1992.

[19] R.M. Corless, G.H. Gonnet, D.E.G. Hare, D.J. Jeffrey and D.E. Knuth, *On Lambert's W Function,* Advances in Computational Mathematics, to appear.

[20] B.I. Dahn and P. Göring, *Notes on exponential-logarithmic terms,* Fundamenta Mathematicae **127**, pp. 45–50, 1986.

[21] R.R. Fenichel, *An On-line System for Algebraic Manipulation,* Ph.D. Thesis, Project MAC, MIT, MAC-TR-35, 1966.

[22] J.P. Fitch, *On algebraic simplification,* The Computer Journal **16** (1), pp. 23–27, 1973.

[23] P. Flajolet, B. Salvy and P. Zimmermann, *Automatic average-case analysis of algorithms,* Theoretical Computer Science **79**, pp. 37–109, 1991.

[24] D.P. Friedman and D.S. Wise, *CONS should not evaluate its arguments,* in: Automata Languages and Programming, (S. Michaelson and R. Milner, eds.), pp. 257–284, 1976.

[25] B. Fuchssteiner et al., *MuPAD multi processing algebra data tool,* Benutzerhandbuch MuPAD Version 1.1, Birkhäuser Basel, 1993.

[26] K.O. Geddes, *Numerical integration in a symbolic context,* Proceedings of the 1986 Symposium on Symbolic and Algebraic Computation SYMSAC'86 (B.W. Char, ed.), ACM Press, New York, pp. 186–191, 1986.

[27] K.O. Geddes and G.H. Gonnet, *A New Algorithm for Computing Symbolic Limits using Hierarchical Series*, Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'88 (P. Gianni, ed.), Lecture Notes in Computer Science **358**, Springer Verlag, pp. 490–495, 1988.

[28] K.O. Geddes and G.J. Fee, *Hybrid Symbolic-Numeric Integration in MAPLE,* Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'92 (P.S. Wang, ed.), ACM Press, New York, pp. 36–41, 1992.

[29] G.H. Gonnet, *Determining Equivalence of Expressions in Random Polynomial Time,* Proceedings of the 16th ACM Symposium on the Theory of Computing, pp. 334–341, 1984.

[30] G.H. Gonnet, *New Results for Random Determination of Equivalence of Expressions,* Proceedings of the 1986 Symposium on Symbolic and Algebraic Computation SYMSAC'86 (B.W. Char, ed.), ACM Press, New York, pp. 127–131, 1986.

[31] D. Gruntz, *A New Algorithm for Computing Asymptotic Series,* Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'93 (M. Bronstein, ed.), ACM Press, New York, pp. 239–244, 1993.

[32] D. Gruntz and W. Koepf, *Formal Power Series,* Preprint SC 93-31, ZIB, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1993.

[33] D. Gruntz and M.B. Monagan, *Introduction to Gauss*, Maple Technical Newsletter **9**, pp. 23–35, 1993, or SIGSAM Bulletin **28** (2), pp. 3-19, 1994.

[34] D. Gruntz, *Infinite Structures in Maple,* MapleTech **1** (2), pp. 19–30, 1994.

[35] R.W. Hamming, *Numerical Methods for Scientists and Engineers,* McGraw-Hill Book Co., New York, 1962.

[36] G.H. Hardy, *Orders of Infinity,* Cambridge Tracts in Mathematics and Mathematical Physics, No. 12, 1910.

[37] S.J. Harrington, *A Symbolic Limit Evaluation Program in REDUCE,* SIGSAM Bulletin **13** (1), pp. 27–31, 1979.

[38] A.C. Hearn, *A New REDUCE Model for Algebraic Simplification,* Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation SYMSAC'76 (R.D. Jenks, ed.), ACM Press, New York, pp. 46–52, 1976.

[39] J.R. Iturriaga, *Contributions to Mechanical Mathematics,* Ph.D. Thesis, Carnegie-Mellon Institute of Technology, 1967.

[40] R.D. Jenks and R.S. Sutor, AXIOM *The Scientific Computation System,* Springer-Verlag, 1992.

[41] S.C. Johnson, *On the Problem of Recognizing Zeros,* Journal of the ACM **18** (4), pp. 559–565, 1971.

[42] Stanley L. Kameny, *A REDUCE Limits Package,* Package documentation of the REDUCE installation.

[43] Stanley L. Kameny, private communication, 1994.

[44] D.E. Knuth, *The Art of Computer Programming, Volume 2 / Seminumerical Algorithms* Second Edition, Addison-Wesley, Reeading Massachusetts, 1981.

[45] P.H. Landin, *A correspondence between ALGOL 60 and Church's lambda notation, Part I,* Communications of the ACM **8**, pp. 89–101, 1965.

[46] J. Laurent, *A Program that Computes Limits Using Heuristics to Evaluate the Indeterminate Forms,* Artificial Intelligence **4**, pp. 69–94, 1973.

[47] D. Levin, *Development of Non-Linear Transformations for Improving Convergence of Sequences,* International Journal of Computer Mathematics **3** (4), Section B, pp. 371–388, 1973.

[48] J.D. Lipson, *Newton's Method: A Great Algebraic Algorithm,* Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation SYMSAC'76 (R.D. Jenks, ed.), ACM Press, New York, pp. 260–270, 1976.

[49] A. Macintyre, *The Laws of Exponentiation,* Model Theory and Arithmetic, Proceedings, Paris, 1979/80, (C. Berline, K. McAloon and J.-P. Ressayre, ed.), Lecture Notes in Mathematics **890**, Springer-Verlag, pp. 185–197, 1980.

[50] R.E. Maeder, *Computations with Infinite Structures,* The Mathematica Journal **1** (2), Addison-Wesley, pp. 30–33, 1990.

[51] W.A. Martin, *Determining the Equivalence of Algebraic Expressions by Hash Coding,* Journal of the ACM **18** (4), pp. 549–558, 1971.

[52] D.A. Turner, *An overview of Miranda,* ACM Sigplan Notices **21** (12), pp. 158–166, 1986.

[53] M.B. Monagan, *Signatures + Abstract Types = Computer Algebra – intermediate expression swell,* PhD thesis, University of Waterloo, 1989.

[54] M.B. Monagan and G.H. Gonnet, *Signature Functions for Algebraic Numbers,* Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'94, ACM Press, New York, pp. 291–296, 1994.

[55] J. Moses, *Algebraic Simplification: A Guide for the Perplexed,* Communications of the ACM **14**, pp. 527–537, 1971.

[56] A.C. Norman, *Computing with Formal Power Series,* ACM Transactions on Mathematical Software **1** (4), pp. 346–356, 1975.

[57] A. Oldenhoeft, *Analysis of Constructed Mathematical Responses by Numeric Tests for Equivalence,* Proceedings of the ACM 24th Annual Conference, pp. 117–124, 1969.

[58] J. Pichon, *Calcul des limites,* Ellipses, Paris, 1986.

[59] D. Richardson, *Some Undecidable Problems Involving Elementary Functions of a Real Variable,* The Journal of Symbolic Logic **33** (4), pp. 414–520, 1968.

[60] D. Richardson, *Solution of the Identity Problem for Integral Exponential Functions,* Z. Math. Logik. Grundlagen **15**, pp. 333–340, 1969.

[61] D. Richardson, *The Elementary Constant Problem,* Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'92 (P.S. Wang, ed.), ACM Press, New York, pp. 108–116, 1992.

[62] D. Richardson and J. Fitch, *The Identity Problem for Elementary Functions and Constants,* Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'94, ACM Press, New York, pp. 285–290, 1994.

[63] D. Richardson, *A simplified method of recognizing zero among elementary constants,* Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'95, ACM Press, New York, pp. 104–109, 1995.

[64] D. Richardson, B. Salvy, J. Shackell and J. van der Hoeven, *Asymptotic Expansions of exp-log Functions,* Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'96, ACM Press, New York, 1996.

[65] R.H. Risch, *Algebraic Properties of the Elemntary Functions of Analysis,* American Journal of Mathematics **4** 101, pp. 743–759, 1975.

[66] A. Robinson, *On the real closure of a Hardy field,* Theory of Sets and Topology, (G. Asser, J. Flachsmeyer and W. Rinow, eds.), VEB Deutscher Verlag der Wissenschaften, Berlin, pp. 427–433, 1972.

[67] M. Rosenlicht, *Hardy Fields,* Journal of Mathematical Analysis and Applications **93**, pp. 297–311, 1983.

[68] M. Rosenlicht, *The Rank of a Hardy Field,* Transactions of the American Mathematical Society **280** (2), pp. 659–671, 1983.

[69] M. Rosenlicht, *Growth properties of functions in Hardy fields,* Transactions of the American Mathematical Society **299** (1), pp. 261–272, 1987.

[70] B. Salvy, *Examples of Automatic Asymptotic Expansions,* SIGSAM Bulletin **25** (2), pp. 4–17, 1991.

[71] B. Salvy, *Asymptotique automatique et fonctions génératrices,* Phd. Thesis, 1991.

[72] B. Salvy, *General Asymptotic Scales and Computer Algebra,* in: Asymptotic and Numerical Methods for Partial Differential Equations with Critical Parameters, (H.G. Kaper and M. Garbey, eds.), Kluwer Academic Publishers, pp. 255–266, 1993.

[73] B. Salvy and J. Shackell, *Asymptotic expansions of functional inverses,* Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'92 (P.S. Wang, ed.), ACM Press, New York, pp. 130–137, 1992.

[74] J.T. Schwartz, *Fast Probabilistic Algorithms for Verification of Polynomial Identities,* Journal of the ACM **27** (4), pp. 701–717, 1980.

[75] J. Shackell, *Asymptotic Estimation of Oscillating Functions using an Interval Calculus,* Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'88 (P. Gianni, ed.), Lecture Notes in Computer Science **358**, Springer Verlag, pp. 481–489, 1988.

[76] J. Shackell, *A Differential-Equations Approach to Functional Equivalence,* Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'89 (G.H. Gonnet, ed.), ACM Press, New York, pp. 7–10, 1989.

[77] J. Shackell, *Growth Estimates for Exp-Log Functions,* Journal for Symbolic Computation **10**, pp. 611–632, 1990.

[78] J. Shackell, *Computing Asymptotic Expansions of Hardy Fields,* Technical Report, University of Kent at Canterbury, England, 1991.

[79] J. Shackell, *Rosenlicht Fields,* Transactions of the American Mathematical Society **335** (2), pp. 579–595, 1993.

[80] J. Shackell, *Nested Expansions and Hardy Fields,* Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'93 (M. Bronstein, ed.), ACM Press, pp. 234–238, 1993.

[81] J. Shackell, *Zero-equivalence in function fields defined by algebraic differential equations,* Transactions of the American Mathematical Society **336** (1), pp. 151–172, 1993.

[82] J. Shackell, *Extensions of Asymptotic Fields via Meromorphic Functions,* Journal of the London Mathematical Society **52**, pp. 356–374, 1995.

[83] J. Shackell, *Limits of Liouvillian Functions,* Proceedings of the London Mathematical Society **72**, pp. 124–156, 1996.

[84] O. Spiess, editor, *Der Briefwechsel von Johann Bernoulli,* Band 1, Herausgegeben von der Naturforschenden Gesellschaft in Basel, Birkhäuser, Basel, 1955.

[85] O. Stolz, *Über die Grenzwerthe der Quotienten,* Mathematische Annalen **15**, pp. 556–559, 1879.

[86] D.R. Stoutemyer, *Qualitative Analysis of Mathematical Expressions Using Computer Symbolic Mathematics,* Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation SYMSAC'76 (R.D. Jenks, ed.), ACM Press, New York, pp. 97–104, 1976.

[87] D.R. Stoutemyer, *Crimes and Misdemeanors in the Computer Algebra Trade,* Notices of the American Mathematical Society **38** (7), pp. 778–785, 1991.

[88] J. van der Hoeven, *General algorithms in asymptotics I: Gonnet and Gruntz' algorithm,* Research Report LIX/RR/94/10, Ecole Polytechnique, France, 1994.

[89] P.S. Wang, *Evaluation of Definite Integrals by Symbolic Manipulation,* Ph.D. Thesis, MAC TR-92, October 1971.

[90] P.S. Wang, *Automatic Computation of Limits,* Proceedings of the 2nd Symposium on Symbolic and Algebraic Manipulation SYMSAC'71 (S.R. Petrick, ed.), ACM Press, New York, pp. 458–464, 1971.

[91] S.M. Watt, *A Fixed Point Method for Power Series Computation,* Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'88 (P. Gianni, ed.), Lecture Notes in Computer Science **358**, Springer Verlag, pp. 206–217, 1988.

[92] T. Weibel and G.H. Gonnet, *An assume facility for CAS, with a sample implementation for Maple,* Proceedings of the International Symposium on Design and Implementation of Symbolic Computation Systems DISCO'92 (J. Fitch, ed.), Lecture Notes in Computer Science **721**, Springer-Verlag, pp. 95–103, 1993.

[93] E.J. Weniger, *Nonlinear Sequence Transformations for the acceleration of convergence and the summation of divergent series,* Computer Physics Reports **10**, 189–371, 1989.

[94] P. Zimmermann, *New features in MuPAD 1.2.2,* mathPAD, Vol 5, No 1, 27–38, 1995.

[95] R.E. Zippel, *Univariate Power Series Expansions in Algebraic Manipulation,* Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation SYMSAC'76 (R.D. Jenks, ed.), ACM Press, New York, pp. 198–208, 1976.

[96] R. Zippel, *The Weyl Computer Algebra Substrate,* Proceedings of the International Symposium on Design and Implementation of Symbolic Computation Systems DISCO'93 (A. Miola, ed.), Lecture Notes in Computer Science **722**, Springer-Verlag, pp. 303–318, 1993.

Curriculum Vitae

I was born on May 26, 1964, in Basel, Switzerland. After finishing school at the Mathematisch Naturwissenschaftlichen Gymnasium in Basel with Mature Type C in 1983, I started to study Computer Science at the Swiss Federal Institute of Technology (ETH) Zürich. In 1988 I received my Diploma in Computer Science. I was awarded a silver medal for my Diploma thesis in the field of computer vision. From 1988 to 1990 I was a teaching assistant in the group of Prof. W. Gander and from 1990 to 1994 I have been a research assistant in the symbolic computation group of Prof. G.H. Gonnet at the Institute for Scientific Computing at ETH Zürich.