Verified Real Asymptotics in Isabelle/HOL

Manuel Eberl
Technical University of Munich
Garching b. München
eberlm@in.tum.de

ABSTRACT

Interactive theorem provers (or *proof assistants*) are software with which mathematical definitions and theorems can be formalised. They assist the user in writing formal proofs and check the correctness of these proofs, typically down to the level of basic logical inference steps. This provides a very high degree of assurance that any proof accepted by them is actually sound. Theorem provers contain varying amounts of tools for automation to assist the user, but unlike computer algebra systems, their focus is not on efficient automatic computation.

In this work, we focus on the particular problem of symbolically determining and proving asymptotics of real-valued functions: limits, 'Big-O' statements, and asymptotic expansions. The tool that is presented here uses an approach based on multiseries expansions and can handle functions built from basic arithmetic and standard functions like exp, ln, sin, $|\cdot|$, etc. as well as parameters. The procedure is efficient enough to handle big problems and it is fully automatic in many cases.

CCS CONCEPTS

• **Mathematics of computing → Mathematical software**.

KEYWORDS

asymptotics, interactive theorem proving, proof assistant, Isabelle, symbolic computation, real analysis

ACM Reference Format:
Manuel Eberl. 2019. Verified Real Asymptotics in Isabelle/HOL. In *Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC'19).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.475/123_4

1 INTERACTIVE THEOREM PROVERS

An *interactive theorem prover* (or *proof assistant*) is a piece of software designed to assist in the development of a *formal proof*. All definitions and proof steps have to be made in a formal way and proofs are checked by the computer – typically down to the level of basic logical inference. Consequently, this involves much more work than a paper proof, but also provides much more clarity and a high assurance that the proofs are actually sound.

There are many different theorem provers; some popular ones are Coq, Isabelle, HOL Light, HOL4, Lean, and Mizar. They differ in their underlying logic and in the infrastructure they provide. In this paper, we focus on *Isabelle/HOL*, which is the combination of the generic *Isabelle* theorem prover with *Higher-Order Logic* as the underlying logic.

ISSAC'19, July 2019, Beijing, China
2019. ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

Unlike computer algebra systems, theorem provers focus not on providing a framework for computation designed to automatically return a result quickly, but on providing a consistent logical infrastructure in which mathematical definitions and proofs can be made with high assurance of correctness. Like many other systems, Isabelle has a *kernel*, which is the only part of the system that can produce theorems. This kernel provides only basic logical inference rules (such as *modus ponens* or ∀-introduction/elimination) and a mechanism for non-recursive definitions. Outside the kernel, many additional tools exist, such as *tactics* to automate proofs (e. g. rewriting, first-order logic, linear arithmetic) and more sophisticated definitional mechanisms (e. g. for recursive functions, inductive predicates, and algebraic datatypes). However, all of these additional tools need to go through the kernel whenever they want to prove a theorem or define a function. The intention behind this design is that:

(1) A user should not be able to accidentally introduce inconsistencies by defining e. g. $f(x) = f(x) + 1$.
(2) Bugs in proof automation or definition tools (e. g. termination checkers for recursive definitions) should not compromise the integrity of the entire system.

Formal proofs – even computer-assisted ones – are very verbose, since every step of a proof has to be written down in much more detail than in a pen-and-paper proof. Many steps that are elided in paper proofs need to be written out explicitly. To make theorem provers usable for the formalisation of non-trivial mathematics, it is therefore crucial that they provide good proof automation to reduce the burden on the user.

This work focuses on the particular problem of proving asymptotic properties of concrete real-valued functions such as that in Figure 1. Computer algebra systems like Mathematica and Maple are very good at solving these problems. They employ specialised algorithms to compute an asymptotic expansions for a wide variety of functions efficiently and fully automatically. On the other hand, to our knowledge, no theorem prover has anything comparable to this so far. In this work, we attempt to integrate a procedure very much like the one used by computer algebra systems into Isabelle/HOL with the goal of providing a similarly effective and automatic 'push-button' solution to proving limits and other asymptotic properties.

It is important to emphasise that, naturally, we did not invent any of the underlying methods used in this work (such as multiseries and expansions of exp-log functions [8, 12]). The novel contribution of our work is that we integrate these methods into a theorem prover in a way that is effective and convenient to use, and that we successfully use this tool for non-trivial mathematical developments.

As of Isabelle 2018, the work described here can be found in the `HOL-Real_Asymp` session, which is a part of the Isabelle distribution.

2 MOTIVATION

Proving limits in a theorem prover can be very tedious. Of course, simple limits like those of $\exp(1/x)$ or $1 - \frac{1}{x} + \ln x$ for $x \to \infty$ can be proven in an entirely syntax-directed way, which makes them very easy to automate. Indeed, just passing the right theorems to Isabelle's general-purpose proof automation is enough to prove these two examples:

lemma filterlim $(\lambda x.\ \exp(1/x))$ (nhds 1) at_top
  by (force intro: tendsto_eq_intros real_tendsto_divide_at_top filterlim_ident)

lemma filterlim $(\lambda x.\ 1 - 1/x + \ln x)$ at_top at_top
  by (force intro: tendsto_diff filterlim_tendsto_add_at_top
    real_tendsto_divide_at_top filterlim_ident ln_at_top)

Even for these simple examples, one can see that one has to search for (or remember) many facts and pass them to the proof method, which a user might expect the system to do for them.

To make matters worse, the above problems were actually cherry-picked: If one permutes the summands in the second example a bit, the method does not work anymore, since many of the theorems only exist in one form; e. g. the rule *filterlim_tendsto_add_at_top* states that if $f(x) \longrightarrow c$ and $g(x) \longrightarrow \infty$, then $f(x) + g(x) \longrightarrow \infty$. However, there is no equivalent theorem for the situation of $g(x) + f(x)$ or $f(x) - g(x)$, or when $g(x) \longrightarrow -\infty$. When one has one of these situations, one first has to rearrange terms in order for the theorem to apply, which can become very tedious especially for larger terms. Proving different forms of these theorems for every possible situations would possibly improve this, but creating this would require a lot of effort and duplication.

Moreover, in many interesting examples, the proofs are *not* entirely syntax-directed. Consider the following examples:

$$\lim_{x\to\infty} \frac{x+1}{x-1} \qquad \lim_{x\to\infty} \frac{x}{(\ln x)^c} \qquad \lim_{x\to 0} \frac{\sin(\tan x) - \tan(\sin x)}{x^7}$$

These are easy to solve by hand:

- The first one can be rewritten to $(1 + x^{-1})/(1 - x^{-1})$, for which the syntax-directed method immediately gives us the limit 1.
- The second one can be attacked by applying L'Hospital's rule again and again until we get $C_1 x(\ln x)^{C_2}$ with $C_1 > 0$ and $C_2 \geq 0$, which clearly goes to ∞.
- The third one can be solved using Taylor expansions to find that the limit is $-\frac{1}{30}$.

The drawback of the first two methods is that they typically require a certain amount of creativity and that there is no clear-cut class of problems on which they will work; the third method can become very tedious and error-prone on paper. In Isabelle, *all* of these methods are typically tedious, since 'obvious' steps need to be made explicit:

- Rewriting limits often requires proving side conditions; e. g. non-zeroness in the above example.
- L'Hospital's rule requires proving several other (albeit usually easier) limits and derivatives. Also, there are many different cases and each one requires a different variant of the rule to be used.
- Manual Taylor expansion-based proofs also require proving many derivatives and asymptotic estimates.

As one of the more extreme examples of the limit problems that can arise, consider Figure 1. This problem is part of Leighton's proof of the *Akra–Bazzi* theorem. The first formal proof of this statement in Isabelle [3] was 700 lines long and required considerable effort to write.

The above examples should demonstrate vividly that in order to do non-trivial mathematics involving asymptotic analysis in a theorem prover, better automation is needed. This state of affairs seems particularly bizarre when compared to the capabilities of modern computer algebra systems, which can typically solve problems like the ones above fully automatically within fractions of a second. The goal of this work is to bring Isabelle's capabilities w. r. t. limit computations and asymptotic analysis closer to that of a computer algebra system – while still constructing a machine-checked proof.

3 ASYMPTOTICS IN ISABELLE/HOL

First, we must explain some notation concerning asymptotics in Isabelle/HOL. Asymptotics in Isabelle are centred around *filters* [2]. For our purposes, a filter can be thought of as a kind of local neighbourhood or approach: The neighbourhood of a real number (i. e. all numbers that are sufficiently close to that number) is a filter, and so are ∞ (all sufficiently large numbers) and $-\infty$. There are also filters for $\pm\infty$ (all numbers whose absolute value is sufficiently large) and 0^+ (all sufficiently small positive numbers).

What is convenient about filters for theorem proving is that they compose in very natural ways and they can be used to uniformly express many different concepts related to topology, analysis, measure theory, or asymptotics: properties that hold 'eventually' ('for all values that are sufficiently …'), limits, summable families, pointwise and uniform continuity, derivatives, Landau symbols, etc.

For a detailed introduction to the way filters are used in Isabelle and the precise definition, see the paper by Hölzl *et al.* [6]. For this presentation, the notation listed in Table 1 will suffice. Similarly to the Isabelle notation, we will use $\forall^\infty x.\ P(x)$ from now on to denote that $P(x)$ holds for large enough x.

4 MULTISERIES

For the sake of simplicity, we only ever consider functions at the neighbourhood of ∞ and reduce all other cases to this. Therefore, any 'eventually', limit, or 'Big-O' from now on is to be understood for $x \to \infty$ unless otherwise indicated.

At the core of our method is the concept of a *multiseries*. This is a representation of a Poincaré expansion of a given real-valued function, where each summand in the expansion is a monomial of the form $c b_1(x)^{e_1} \cdot \ldots \cdot b_k(x)^{e_k}$ with $c, e_1, \ldots, e_k \in \mathbb{R}$. The b_i are *basis functions*; they are functions tending to ∞ and they are the same for all summands. The list of the b_i is assumed to be ordered by descending growth in such a way that $\ln b_{i+1}(x) \in o(\ln b_i(x))$ for all i. The reason for this is that then $b_{i+1}(x)^e \in o(b_i(x)^{e'})$ for any $e, e' \in \mathbb{R}$ with $e' > 0$ so that we can easily compare two monomials asymptotically by comparing their exponent vectors lexicographically. We call such a list a *well-formed basis*. A typical basis would be

$$\big(\exp(x \ln x),\ \exp(x),\ x,\ \ln x,\ \ln\ln x\big).$$

$$\lim_{x\to\infty}\left(1-\frac{1}{b\log^{1+\varepsilon}x}\right)^{p}\left(1+\frac{1}{\log^{\varepsilon/2}\left(bx+x\log^{-1-\varepsilon}x\right)}\right)-\left(1+\frac{1}{\log^{\varepsilon/2}x}\right)=0^{+}$$

for real-valued parameters b, p, ε with $b \in (0;1)$ and $\varepsilon > 0$

Figure 1: A limit-problem related to Leighton's proof of the Akra–Bazzi theorem.

Notation	Meaning
nhds c	neighbourhood filter around $c \in \mathbb{R}$
at_top	neighbourhood of ∞ in $\mathbb{R}$
$\forall x$ in F. $P(x)$	$P(x)$ holds *eventually* at F, i. e. for all x that are sufficiently F
filterlim f F G	$f(x) \xrightarrow{x\to G} F$
$f \in O_{[F]}(g)$	$\exists c{>}0.\ \forall x$ in F. $\lvert f(x)\rvert \le c\,\lvert g(x)\rvert$
$f \in o_{[F]}(g)$	$\forall c{>}0.\ \forall x$ in F. $\lvert f(x)\rvert \le c\,\lvert g(x)\rvert$
$f \sim_{[F]} g$	$f - g \in o_{[F]}(g)$

f and g are real-valued functions; F and G are filters.
If the filter argument $_F$ for the Landau symbols is not given, it defaults to *at_top*.

Table 1: Common asymptotic notation in Isabelle.

Abstractly, a multiseries is a formal power series in k variables standing for the basis functions, i. e. a function mapping exponent tuples $(e_1, \ldots, e_k) \in \mathbb{R}^k$ to coefficients in $\mathbb{R}$. We write this suggestively as

$$\sum C_{e_1,\ldots,e_k}\, b_1(x)^{e_1} \ldots b_k(x)^{e_k}\ .$$

The link between these formal objects and a concrete real function $f(x)$ is made by demanding that the multiseries be a Poincaré expansion of $f(x)$, i. e. that approximating $f(x)$ by the finite sum of all the monomials with exponent vectors $\ge_{\text{lex}} \bar{e}$ for some fixed $(\bar{e}_1, \ldots, \bar{e}_k)$ yields an error that is $o(b_1(x)^{\bar{e}_1} \ldots b_k(x)^{\bar{e}_k})$. Given a multiseries expansion of a function, its limit can then be computed by determining the leading monomial, i. e. the smallest exponent vector $(e_1, \ldots, e_k)$ whose coefficient is non-zero.

A computationally convenient view of multiseries is the following: By isolating the first basis element, we can view a multiseries as a (univariate) formal series in $b_1(x)$ whose coefficients are multiseries w. r. t. the $k-1$ basis functions $b_2(x), \ldots, b_k(x)$. This univariate series can be represented as a (possibly infinite) list whose elements are a pair of a coefficient (which is again a multiseries, but in $k-1$ variables) and an exponent (which is just a real number). Iterating this yields a kind of 'nested infinite list' structure with k levels.

The reason for this nested structure is the following: If one were to view multiseries simply as linear sequences of summands, cancellation can lead to an infinite number of zeros in the front and therefore to non-termination when trying to find the leading term. The nested list representation solves this problem because we can discard infinitely many zeros at once: Each element of the outermost list corresponds to a summand $c(x)b_1(x)^e$ with a multiseries expansion for $c(x)$ in terms of $b_2(x), \ldots, b_k(x)$ attached to it. If $c(x)$ is identically zero, we can discard the element and proceed with the next one until we find a term such that $c(x)$ is not identically zero. We then proceed analogously with the multiseries expansion of $c(x)$ etc. We refer to this process as *trimming* the representation of the multiseries.

When the distinction is necessary, we will write multiseries in double square brackets to distinguish them from the functions of which they are an expansion and separate the monomials with a vertical bar, e. g. $[\![x \mid 1]\!]_{(x,\ln)}$ is a multiseries w. r. t. the basis $(x, \ln)$ that corresponds to the formal sum $1 \cdot x^1 \ln(x)^0 + 1 \cdot x^0 \ln(x)^0$. Similarly, the notation $[\![cb(x)^e \mid \mathcal{F}]\!]_{b\#bs}$ indicates that the leading entry in the list has coefficient $[\![c]\!]_{bs}$ and exponent $e \in \mathbb{R}$ and $[\![\mathcal{F}]\!]_{b\#bs}$ is the remainder of the series. Variables denoting multiseries are in upper-case calligraphic font, e. g. $\mathcal{F}$. We also write addition, multiplication, etc. on multiseries as $\oplus$, $\odot$, etc. to distinguish them from the operations on real numbers.

To obtain nice theoretical properties of these multiseries, further restrictions must be placed on the support of this coefficient function (e. g. well-ordered or 'grid-based' [14]), but for our purposes, such assumptions need not be modelled explicitly. Since all our definitions work directly on the nested list representations, our multiseries have well-ordered coefficient support by construction. In fact, all the multiseries that our constructions yield will even be grid-based, but we do not need to talk about these notions in Isabelle/HOL explicitly to show the soundness of the approach; they are only relevant for a discussion of completeness.

Although the actual implementation in Isabelle is different due to lack of dependent types, we will, for the sake of this presentation, pretend that there exists a type of multiseries *mseries* that takes a list of basis function as its type parameter. Morally, we then have

$$[]\ \text{mseries} = \mathbb{R}$$
$$(b\ \#\ bs)\ \text{mseries} = (bs\ \text{mseries} \times \mathbb{R})\ \text{llist}$$

where α *llist* ('lazy list') is the type of possibly infinitely long lists with elements of type α and # denotes prepending a single basis function b to a list of functions bs. The *llist* type is a *codatatype* defined using Isabelle's codatatype package [1]. This allows us to

define 'infinitely recursive' functions on infinite structures in an intuitive way and reason about them. We make extensive use of this to define operations on multiseries, to link these formal series to real functions, and to prove the correctness of the former w. r. t. the latter.

A technical subtlety that is not shown here is that every Multiseries implicitly 'knows' what function it is supposed to be an expansion of. For a Multiseries $\mathcal{F}$, we denote this function as $\mathcal{F}(x)$. For $\mathcal{F}$ to be *well-formed*, it needs to be a Poincaré expansion of $\mathcal{F}(x)$.

5 TRIMMING AND RECOGNISING ZERO

An important auxiliary operation on our multiseries representation is the afore-mentioned *trimming*. We call a multiseries *trimmed* if its leading monomial is non-zero. The process of *trimming* is to discard terms at the beginning of the list recursively until this is the case. The difficulty here is that when considering a leading term of the form $[\![Cb(x)^e]\!]_{b\#bs}$, we need to decide whether or not it is identically 0 and discard it if it is; if we fail to recognise this and try to trim C instead, we may trim infinitely many zeros from it and never terminate.

Note, however, that the leading zeros can already contain some information: if e. g. $f(x) \sim 0 \cdot x^2 + \ldots$, then we can immediately see that $f(x) \in O(x^2)$. It is therefore sometimes sufficient to only partially trim a series. However, if we want more precise asymptotic information (in particular an asymptotic lower bound), we indeed need to make sure that the expansion is fully trimmed first. We will also see that some of the operations we will define on multiseries only work if the series is already trimmed. Trimming will therefore be an essential part in our expansion algorithm.

This leads us to the problem of recognising zeros, which is one of the central problems in automated symbolic asymptotics. Given some expression representing a real constant, how do we determine whether or not it is zero? Depending on the class of expressions considered, this problem ranges from difficult to undecidable [4]. For exp-log constants, it is known to be semi-decidable but the theory behind the decision procedure is fairly complicated and out of the scope of this work. [7]

We therefore chose a very simple approach: The algorithm uses a modular *zeroness oracle*[1] that receives an Isabelle term c of type $\mathbb{R}$ as an input and either fails with an informative error message or returns a theorem stating that $c = 0$, $c \geq 0$, $c > 0$ etc. depending on the exact configuration. A similar *eventual-zeroness oracle* receives a term f of type $\mathbb{R} \to \mathbb{R}$ and either fails silently or returns the theorem that $f(x)$ is eventually zero.

The standard implementation of these oracles uses Isabelle's *simplifier*, which is one of Isabelle's default proof methods. It is a simple directed term rewriting engine with a large setup of rewrite rules and some additional specialised procedures (e. g. for arithmetic). Since most zeroness problems encountered in practice are trivial (e. g. '$1 + 2 \cdot (1 + 0) = 0$'), this works reasonably well. This oracle is even able to handle parameters, but it sometimes reaches its limits when larger arithmetic expressions or functions like $\sqrt{\ }$, exp, and ln are involved. The user can load an optional additional oracle based on interval arithmetic approximation [5] that can handle many of these cases. Improving Isabelle's automation for problems such as these or adding more zeroness oracles is a worthwhile goal, but certainly outside the scope of this work. If the oracle fails, the users receive an error message indicating on which expression it got stuck so that they can provide a manual proof of its sign and re-run the proof method with this new knowledge.

It should be mentioned that the trimming algorithm sketched here only terminates under the assumption that the eventual-zeroness oracle never fails on a provable result and that we do not encounter an all-zero expansion of a non-zero function. The latter would be an instance where we have some function $f(x)$ that goes to 0 faster than can be measured by the basis of its multiseries (e. g. $\exp(-x)$ with the basis (x)). It is the responsibility of the expansion algorithm to ensure that this does not happen. For the basic class of exp-log functions, this is ensured since the expansion algorithm always produces *convergent* expansions. When we add more functions (e. g. Γ), this is no longer the case and the analysis becomes more complicated. Whether or not the algorithm is still complete in such cases is not clear to us and we consider this beyond the scope of this work since, in the context of a proof method in an interactive theorem prover, completeness is desirable but not absolutely necessary.

Neither of these issues make our procedure untrustworthy: non-termination of the trimming will simply lead to non-termination of the entire algorithm. If the algorithm *does* terminate, a full proof will have been produced and has passed through the Isabelle kernel. The afore-mentioned issues can therefore, by design, only compromise the *completeness* of the algorithm, not its *soundness*.

6 OPERATIONS ON MULTISERIES

We will now give examples of how concrete operations on our multiseries expansion can be implemented. The presentation is fairly close to that in Isabelle but with simplified, type-theory inspired syntax. Due to space constraints, we only show a few basic operations and refer to Shackell [12] for the remaining ones.

6.1 Basic Arithmetic

Constant functions, the identity function and powers thereof have obvious multiseries representations. Also, given an abstract multiseries (i. e. a mapping from exponent vectors to coefficients), it is easy to see how they can be negated, added, and multiplied. As an example, negation and addition can be defined like this:

$$(\ominus) :: \forall bs :: \text{basis}.\ bs\ \text{mseries} \to bs\ \text{mseries}$$
$$\ominus_{[]}\,[\![c]\!] = [\![-c]\!]$$
$$\ominus_{b\#bs}\,[\![Cb(x)^e \mid \mathcal{F}]\!] = [\![(\ominus_{bs} C)b(x)^e \mid \ominus_{b\#bs}\,\mathcal{F}]\!]$$

$$(\oplus) :: \forall bs :: \text{basis}.\ bs\ \text{mseries} \to bs\ \text{mseries} \to bs\ \text{mseries}$$
$$[\![c_1]\!] \oplus_{[]} [\![c_2]\!] = [\![c_1 + c_2]\!]$$
$$[\![C_1 b(x)^{e_1} \mid \mathcal{F}]\!] \oplus_{b\#bs} [\![C_2 b(x)^{e_2} \mid \mathcal{G}]\!] =$$
$$\quad \textbf{if } e_1 > e_2 \textbf{ then}$$
$$\quad\quad [\![C_1 b(x)^{e_1} \mid \mathcal{F} \oplus_{b\#bs} [\![C_2 b(x)^{e_2} \mid \mathcal{G}]\!]]\!]$$
$$\quad \textbf{else if } e_1 < e_2 \textbf{ then}$$
$$\quad\quad [\![C_2 b(x)^{e_2} \mid [\![C_1 b(x)^{e_1} \mid \mathcal{F}]\!] \oplus_{b\#bs} \mathcal{G}]\!]$$
$$\quad \textbf{else}$$
$$\quad\quad [\![([\![C_1]\!] \oplus_{bs} [\![C_2]\!])\, b(x)^{e_1} \mid \mathcal{F} \oplus_{b\#bs} \mathcal{G}]\!]$$

[1] Note that the name 'oracle' does not mean that we blindly trust the oracle. It is still required to return an Isabelle *theorem*.

In the addition algorithm, the first equation is the base case for an empty basis; the second equation 'merges' two series by descending exponents. Note that each of the three cases in the second equation contains a corecursive call to the addition function on the same basis and the third case additionally contains a recursive call to the addition function for the truncated basis.

Multiplication can be implemented analogously (using an auxiliary function that performs multiplication with a monomial $Cb(x)^e$), but operations like the division and powers are slightly more complicated: We first need to implement substitution of a multiseries into a asymptotic or convergent power series.

6.2 Substituting into a Power Series

For this, consider some function $h : \mathbb{R} \to \mathbb{R}$ that has an asymptotic power series expansion $c_0 + c_1 x + c_2 x^2 + \ldots$ at $x = 0$. Other points are, of course, also possible (including at ∞), but $x = 0$ will be enough for our purposes. If we have a function $f : \mathbb{R} \to \mathbb{R}$ with some multiseries expansion $\mathcal{F}$ whose leading exponent is negative, then f clearly tends to 0 for $x \to \infty$ and we can substitute the multiseries expansion for f into the power series expansion for h to obtain a multiseries expansion for $h(f(x))$. This can be implemented as follows:

$$\begin{aligned}
&\text{powser} :: \forall bs :: \text{basis}.\ \mathbb{R}\ \text{llist} \to bs\ \text{mseries} \to bs\ \text{mseries}\\
&\text{powser}\ [\,]\ \mathcal{F} = [\![]\!]_{bs}\\
&\text{powser}\ [c \mid cs]\ \mathcal{F} = [\![c \mid \mathcal{F} \odot \text{powser}\ cs\ \mathcal{F}]\!]_{\text{bs}}
\end{aligned}$$

In particular, this allows us to turn a multiseries expansion for f into a multiseries expansion for $h \circ f$ if $f(x) \longrightarrow 0$ and h is analytic at 0. This will be a key ingredient for the remaining operations that we implement.

6.3 Division and Powers

Consider the multiplicative inverse function $x \mapsto \frac{1}{x}$, which we can use to handle division. Given a function f with a multiseries expansion $\mathcal{F} = [\![Cb(x)^e \mid \bar{\mathcal{F}}]\!]_{b\#bs}$, we consider the remainder after dropping the leading term of the expansion, i. e. $\bar{f}(x) := f(x) - C(x)b(x)^e$ with the expansion $\bar{\mathcal{F}}$. We will use $\bar{\mathcal{F}}$ and $\bar{f}(x)$ with that meaning from now on. We can then write:

$$\frac{1}{f(x)} = \frac{1}{C(x)} b(x)^{-e} \frac{1}{1 + \frac{1}{C(x)} b(x)^{-e} \bar{f}(x)}$$

Since $t \mapsto \frac{1}{1+t}$ has the power series expansion $1 + t + t^2 + \ldots$ at $t = 0$, we can then define the multiplicative inverse $\mathcal{I}(\mathcal{F})$:

$$\begin{aligned}
&\mathcal{I} :: \forall bs :: \text{basis}.\ bs\ \text{mseries} \to bs\ \text{mseries}\\
&\mathcal{I}([\![c]\!]_{[]}) = [\![\tfrac{1}{c}]\!]_{[]}\\
&\mathcal{I}([\![Cb(x)^e \mid \bar{\mathcal{F}}]\!]_{b\#bs}) = [\![\mathcal{I}(C)b(x)^{-e}]\!] \odot_{b\#bs}\\
&\qquad \text{powser}\ [1, 1, \ldots]\ ([\![\mathcal{I}(C)b(x)^{-e}]\!] \odot_{b\#bs} \bar{\mathcal{F}})
\end{aligned}$$

Note that $\mathcal{I}$ is only well-defined if the multiseries it is given is trimmed. Also note that the argument of *powser* indeed always has a negative leading exponent here since $lead\_exp(\bar{\mathcal{F}}) < e$.

The same approach can be used to handle the case $f(x)^u$ for any constant $u \in \mathbb{R}$ by writing

$$f(x)^u = c(x)^u b(x)^{ue} (1 + c(x)^{-1} b(x)^{-e} \bar{f}(x))^u$$

and using the power series expansion for $t \mapsto (1 + t)^u$. Here the condition is that the multiseries must be trimmed with positive leading coefficient (so that it is eventually positive).

6.4 Sine and Cosine

The sine and cosine functions can also be treated similarly: Given $f(x) \sim \mathcal{F}$ with $\mathcal{F} = [\![Cb(x)^e \mid \bar{\mathcal{F}}]\!]_{b\#bs}$, we distinguish three cases:

- If $e < 0$, we substitute $\mathcal{F}$ into the power series for sin or cos.
- If $e > 0$, then $f(x)$ tends to infinity and no multiseries expansion for $\sin f(x)$ or $\cos f(x)$ can exist.
- If $e = 0$, we can write

  $$\sin f(x) = \sin C(x) \cdot \cos \bar{f}(x) + \cos C(x) \cdot \sin \bar{f}(x)$$

  and analogously for $\cos f(x)$. Expansions for $\sin C(x)$ and $\cos C(x)$ can be computed by a recursive call, and $\cos \bar{f}(x)$ and $\sin \bar{f}(x)$ are covered by the $e < 0$ case.

6.5 Logarithm

If we apply this same approach to the ln function, we arrive at the expression

$$\ln f(x) = \ln c(x) + e \ln b(x) + \ln\left(1 + c(x)^{-1} b(x)^{-e} \bar{f}(x)\right).$$

The first and last summand can be handled analogously to the previous cases; however, there is a problem in the second summand: If $e \neq 0$, we need a multiseries expansion for $\ln b(x)$ w. r. t. the basis $b \# bs$. In our definition of a well-formed basis, we assumed that for $i < n$, each $\ln b_i(x)$ has a known expansion in terms of $b_{i+1}, \ldots, b_n$ so that we only have a problem if b is the very last element in the basis (i. e. $bs = [\,]$). The solution in this case is to add $b_{n+1}(x) := \ln b(x)$ as a new basis element at the very end of the basis. This ensures that when we now reach the last basis element b_{n+1}, the exponent e is zero and we do not encounter the problem. Note that inserting a new basis element is a *global* operation, which means that we must lift all expansions computed before to the new, larger basis. E. g. when we expand $f(x) + g(x)$ by first expanding $f(x) \sim \mathcal{F}$ and then $g(x) \sim \mathcal{G}$, the latter step may have enlarged the basis, in which case we need to lift $\mathcal{F}$ to the new basis.

6.6 Exponentials

For the exponential function, the situation is much more complicated. If $e < 0$, we can simply use the power series expansion for exp. If $e = 0$, we have $\exp(f(x)) = \exp(c(x)) \exp(\bar{f}(x))$ and have therefore reduced the problem to a recursive call and the '$e < 0$' case. If, on the other hand, $e > 0$, various case distinctions are required to determine whether $\exp(f(x))$ has an expansion w. r. t. the current basis or whether $\exp(f(x))$ or $\exp(-f(x))$ or some variation thereof has to be added as a new basis element – and if yes, where. For the details of this case distinction, we refer again to Shackell [12].

6.7 Other Functions

All functions discussed so far had sufficiently nice properties w. r. t. addition or multiplication of their argument that allowed us to handle the case $f(C(x) + \bar{f}(x))$. Indeed, this is enough to handle the class of all functions built from basic arithmetic, exp, ln as well as sin, cos, and tan at finite points. For functions without these nice

properties, this simple approach does not work, as we will see later in section 9.

7 CONNECTING SERIES TO FUNCTIONS

We have defined a representation for multiseries and implemented a number of operations on them; however, we have so far not formally defined what it means for a multiseries to be an expansion of a particular function. Since we want to prove theorems inside the logic, this connection has to be defined explicitly inside the logic, and the correctness of all the operations we defined must be formally proven w. r. t. it. This is the crucial difference to Computer Algebra Systems and it results in a high level of confidence in the results produced by the expansion algorithm, but also includes a much greater amount of effort.

We will introduce a predicate $\mathrm{wf}_{bs}(\mathcal{F})$ whose meaning is that $\mathcal{F}$ is a well-formed multiseries w. r. t. the basis *bs* and it is a valid expansion of the function $\mathcal{F}(x)$ that it is implicitly connected to. The former includes things like 'the exponents are strictly decreasing' and that the depth of the nested list structure is the same as the length of the basis. To understand what the latter means explicitly, we recall the notion of a Poincaré expansion: If we take a finite initial segment of the (possibly infinite) series, we obtain an approximation to the function such that the error is 'Big-O' of the first omitted term. However, we will use a slightly different formulation of this that is more suitable for our representation of multiseries.

Recall that the type of multiseries is defined by recursion on the associated basis: A multiseries C w. r. t. the empty Basis is simply $[\![c]\!]_{[]}$ for a real constant c. The obvious definition of well-formedness in this case is to require that $C(x) = c$ eventually.

On the other hand, a multiseries w. r. t. a non-empty basis $b \# bs$ is a (possibly infinite) list. As mentioned before, these lists are a *codatatype* and a natural way to define predicates for a codatatype is *coinductively*. For this, we need to express $\mathrm{wf}_{b\#bs}([\![\mathcal{F}]\!])$ in terms of $\mathrm{wf}_{b\#bs}(\bar{\mathcal{F}})$, where $\mathcal{F} = [\![C b(x)^e \mid \bar{\mathcal{F}}]\!]$. To see how to do this, it is instructive to consider the situation for a power series:

$$f(x) \sim cx^e + \mathcal{F} \longleftrightarrow f(x) \in O(x^e) \wedge e > \mathrm{lead\_exp}(\mathcal{F}) \wedge f(x) - cx^e \sim \mathcal{F}$$

Adapting this for multiseries, we find that $\mathrm{wf}_{b\#bs}(\mathcal{F})$ requires:

$$\mathrm{wf}_{bs}(C) \wedge \mathrm{wf}_{b\#bs}(\bar{\mathcal{F}}) \wedge e > \mathrm{lead\_exp}(\bar{\mathcal{F}}) \wedge \forall e' > e.\ \mathcal{F}(x) \in o(b(x)^{e'})$$

Since in our formalisation, multiseries can also have finite length, we also need to consider the case of an empty multiseries. There are two natural possibilities here:

(1) demand that the function $f(x)$ being expanded is *flat* w. r. t. $b(x)$, i. e. $f(x) \in O(b(x)^e)$ for all e
(2) demand that $f(x) = 0$ eventually

These two differ only if $f(x)$ goes to 0 faster than we can measure with our basis (e. g. $\exp(-x)$ with the basis x). Our algorithm never produces such 'expansions', so we chose the stronger option (2).

We then define *wf* to be a coinductive predicate given by these three rules. This means that *wf* holds iff there is some finite or infinite derivation tree using these three rules. This is equivalent to the Poincaré expansion definition (with the caveat about expansions of finite length), but we never show this connection in Isabelle since we do not need it. That this definition of *wf* makes sense can be seen from the following theorem:

Theorem 7.1 (Connection between *wf* and ∼).
If $\mathrm{wf}_{bs}(\mathcal{F})$ for a well-formed basis bs and a trimmed multiseries $\mathcal{F}$ with leading monomial $cb_1(x) \cdot \ldots \cdot b_n(x)$, then

$$\mathcal{F}(x) \sim cb_1(x) \cdot \ldots \cdot b_n(x)\ .$$

It remains to show the correctness of the multiseries operations we defined. As an example, the correctness theorems for addition and the multiplicative inverse have the following form, assuming a well-formed basis *bs*:

$$\mathrm{wf}_{bs}(\mathcal{F}) \wedge \mathrm{wf}_{bs}(\mathcal{G}) \Longrightarrow \mathrm{wf}_{bs}(\mathcal{F} \oplus \mathcal{G})$$

$$\mathrm{wf}_{bs}(\mathcal{F}) \wedge \mathrm{trimmed}(\mathcal{F}) \Longrightarrow \mathrm{wf}_{bs}(\mathcal{I}(\mathcal{F}))$$

Here, *trimmed* is a predicate that states that $\mathcal{F}$ is trimmed. All of these proofs are straightforward inductions over *bs* where the recursive case requires *coinduction* w. r. t. the coinductive predicate wf. As a simple example, let us consider the correctness proof of the negation operation:

Theorem 7.2 (Correctness of series negation).
If bs is a well-formed basis and $\mathrm{wf}_{bs}(\mathcal{F})$ holds, then $\mathrm{wf}_{bs}(\ominus\mathcal{F})$.

Proof. We proceed by induction over the basis. The case for an empty basis is trivial. Let us therefore consider a basis of the shape $b \# bs$. Then applying coinduction w. r. t. the *wf* predicate gives us the following proof obligation:

$$\begin{aligned}
&\forall \mathcal{F}.\ \mathit{wf}_{b\#bs}(\mathcal{F}) \Longrightarrow \\
&\quad \ominus\mathcal{F} = [\,] \wedge \left(\forall_\infty x.\ -\mathcal{F}(x) = 0\right) \vee \\
&\quad \exists C\ e\ \bar{\mathcal{F}}.\ \ominus\mathcal{F} = [\![C b(x)^e \mid \ominus\bar{\mathcal{F}}]\!] \wedge \\
&\qquad (\forall x.\ -\mathcal{F}(x) - C(x)b(x)^e = -\bar{\mathcal{F}}(x)) \wedge \\
&\qquad \mathrm{wf}_{bs}(C) \wedge \mathrm{wf}_{b\#bs}(\bar{\mathcal{F}}) \wedge \\
&\qquad (\forall e' > e.\ -\mathcal{F}(x) \in o(b(x)^{e'}) \wedge \\
&\qquad e > \mathrm{lead_exp}(\ominus\bar{\mathcal{F}})
\end{aligned}$$

By unfolding one step of the corecursive definition of $\ominus$, this simplifies to the following two cases:

Case 1: $\mathcal{F} = [\![]\!]$
Then $\mathit{wf}_{b\#bs}(\mathcal{F})$ implies $\forall_\infty x.\ \mathcal{F}(x) = 0$ by definition and the proof obligation simplifies to $\forall_\infty x.\ -\mathcal{F}(x) = 0$, which is then obviously true.

Case 2: $\mathcal{F} = [\![C b(x)^e \mid \bar{\mathcal{F}}]\!]$ for some $C, e, \bar{\mathcal{F}}$
It is clear that the values C, e, and $\bar{\mathcal{F}}$ in the existential quantifier must be instantiated with $\ominus C$, e, and $\bar{\mathcal{F}}$, respectively. After simplification, all the proof obligations follow trivially from the induction hypothesis and the definitions. □

Series negation is certainly one of the easiest operations, but the above proof still illustrates three things:

- The proof obligations, even for simple operations, can get quite big and confusing.
- The case distinctions that have to be made are obvious from the definition of the operation.
- Once the right case distinctions are made, the proof obligations become much simpler and seem very obvious.

The proofs for more involved operations are larger and have more cases, but otherwise very similar to this one. Ultimately, writing these proofs is relatively easy due to the guidance from Isabelle. It is always obvious what has to be done, even though this is somewhat obscured by the fact that Isabelle presents the proof obligations in the rather unwieldy form of disjunctions of existential quantifiers that we saw above. We also modified the definitions of the operations several times and were able to adjust the correctness proofs with little effort.

8 COMPUTING AND USING EXPANSIONS

Isabelle is written in Standard ML and exposes this interface to users to add new tools on-the-fly. While the notions of multiseries and expansions is fully formalised in the system, the procedure to *compute* them is not; it is merely ML code. This makes the procedure much easier to implement and more flexible since it lives *outside* the system, but a disadvantage is that we cannot reason about it *inside* the system – e. g. we cannot prove that it is correct or terminates. However, due to the architecture of Isabelle, a mistake in our procedure would result in a run-time error or non-termination at worst, but never an incorrect result since all reasoning performed by the procedure still passes through the Isabelle kernel.

The basic procedure is fairly simple:

(1) Convert the expression defining the function $f(x)$ that is to be expanded into an AST.
(2) Find expansions 'bottom up' to produce a theorem of the form $wf_{bs}(\mathcal{F})$ with $\mathcal{F}(x) = f(x)$. Trim *only* when necessary.
(3) Trim the resulting multiseries until the desired result can be read off.

To perform the trimming, the multiseries expressions (which are Isabelle terms) need to be 'evaluated' partially until the leading monomial can be read off. To this end, we wrote a small lazy evaluation framework for Isabelle terms that supports lazy pattern matching on terms and returns a theorem showing that the reduced term is indeed equal (w. r. t. equality in Higher-Order Logic) to the original term.

The entire procedure is then packaged into a *proof tactic* called `real_asymp` that can be applied to problems of the form

- $f(x) \xrightarrow{x \to G} F$
- $f(x) \in L_{[F]}(g(x))$ where L is any of the five Landau symbols
- eventually $f(x) \leq g(x)$ w. r. t. F
- $f(x) \sim_{[F]} g(x)$

where F and G are filters corresponding to

- the full/left/right/pointed neighbourhood of a real number
- ∞, $-\infty$, or $\pm\infty$.

This tactic constitutes the most important part of this work: Isabelle is a document-oriented *theorem prover*, so the typical use case is that a user will already know the limit of the function, write down the corresponding statement, and then prove it with our method. However, for convenience, we also added the diagnostic commands `real_limit` and `real_expansion` that display the limit (resp. an initial fragment of the multiseries expansion) of a given function.

Since some more advanced functions and their asymptotic behaviour are not available in Isabelle's core library but only in the external *Archive of Formal Proofs*, we also provide an interface to register new user-defined functions with the expansion procedure at a later time. This is done to achieve partial support for the Γ and erf functions.

'Oscillating' expressions like $\sin(x)$ for $x \to \infty$, $(-1)^n$, or $\lfloor x \rfloor$ do not have a multiseries expansion, but limited support for them is provided using what we call 'asymptotic interval arithmetic': When encountering such an expression $f(x)$, we attempt to compute bound functions $l(x)$ and $u(x)$ with known multiseries expansions such that $l(x) \leq f(x) \leq u(x)$. For example, for $\sin(x)$ and $(-1)^n$ the bounds would be $[-1; 1]$; for $\lfloor x \rfloor$ they would be x and $x + 1$. This method is not *complete* since it does not handle cancellations of any kind; e. g. the bounds computed for $\sin(x) - \sin(x)$ would be $[-2; 2]$. Nevertheless, this is enough to provide sufficient asymptotic information in many cases, e. g. to show $\sin(x)/x \xrightarrow{x\to\infty} 0$.

9 LIMITATIONS

There are three limitations of the current implementation:

Zero-checking. As mentioned before, recognising whether a given constant (even an exp-log constant) is zero is difficult. The two methods that we implemented so far use Isabelle's simplifier and interval arithmetic to attack this problem. This works well for many interesting examples, and whenever this fails, the user can simply prove the corresponding fact by hand and add it to the hypotheses. Automating this further would be desirable, but would require great improvements to Isabelle's automation for arithmetic reasoning.

Worst-case performance. It is well-known that the algorithm implemented here has very poor worst-case behaviour; e. g. Richardson *et al.* [8] give the example

$$\frac{1}{1-\frac{1}{x}} - \frac{1}{1-\frac{1}{x}} + x^{-n}\,.$$

Here, many initial zeros need to be trimmed before arriving at the x^{-n} term. The algorithm therefore takes at least linear time in n, leading to poor performance for large n. A possible solution for this problem is given by van der Hoeven [13] in the form of *cartesian representations*, but it would be highly non-trivial to integrate this approach with our current work and we have not observed such severe performance problems in our examples.

Non-exp-log functions. The 'oscillating' functions sin, cos, and tan are only fully supported if their argument is bounded. More comprehensive approaches for sin etc. exist [10], but to our knowledge, these are not used in practice.

Functions like arctan, Γ, and erf are only partially supported: $\Gamma(x)$ and $\operatorname{erf}(x)$ are currently only supported for $x \to \infty$ since other cases tend to involve complicated constants that Isabelle's automation cannot handle well.

As a more fundamental problem, for any of these functions, we cannot expand expressions like $f(x + \exp(-x))$ where the larger basis element $\exp(x)$ is not present in the leading term of the argument of f but *is* present in terms of smaller order. Implementing this requires substituting a multiseries with leading exponent 0 into an asymptotic power series. Shackell [12] shows how to do this, but it is not clear to us how to formally prove that this is correct.

10 EVALUATION

We evaluated our procedure on a large number of examples, including the list given by Gruntz [4]. According to Gruntz, this list consists of various problems that were difficult for computer algebra systems in the early 1990s and indeed Maxima still returns an incorrect result for one of them.[2] All of the 20 exp-log examples can be proven fully automatically by our tactic within less than a second (average 0.24 s, maximum 0.65 s). Of the 17 non-exp-log examples, most lie outside the scope of our procedure due to unsupported functions (e. g. Bessel functions) or our incomplete support for Γ and erf. The 5 that are supported, however, all work in $\leq 2.5\,s$. Our list of examples can be found in the file `src/HOL/Real_Asymp/Real_Asymp_Examples.thy` of the Isabelle distribution. A short user manual is also provided.

Notably, our introductory example from Figure 1 takes only 0.5 s despite containing three parameters. We tried this example with two computer algebra systems capable of using the required additional assumptions on the parameters (Maxima and Mathematica); surprisingly, neither of them could solve this problem at all.

In general, CASs like Mathematica and Maple are, of course, much better both in terms of efficiency and number of supported functions; However, we think that this can be excused considering the immense additional difficulty imposed by working inside a proof assistant where we want not only a *result*, but a *proof*.

11 RELATED WORK

The first complete algorithm for the computation of exp-log functions was given by Shackell in 1990 [11]. Gruntz built on this algorithm and implemented it in Maple [4]. Gruntz's algorithm was later also implemented in Mathematica by Richter [9]. As stated before, our work builds directly on the *multiseries* approach presented by Richardson *et al.* [8].

In our prior work on the proof of the Akra–Bazzi theorem [3], we introduced some very simple automation related to asymptotics, e. g. to automatically prove or disprove statements of the form $f(x) \in O(g(x))$ where f and g are products of powers of iterated logarithms, e. g. $x \ln x \in O(x^2 (\ln \ln x)^3)$. Apart from Isabelle/HOL, there are some other systems that have a library around limits and asymptotics (e. g. Coq, HOL Light, Mizar), but to our knowledge, none of them has any automation for proving limits or other asymptotic properties for any non-trivial class of problems.

12 CONCLUSION

We provide the first implementation of automated real asymptotics inside a proof assistant. The procedure provides:

- full support for basic arithmetic, exp, ln, roots, and $|\cdot|$
- full support for sin, cos, and tan at finite points
- 'best effort' support using interval arithmetic for oscillating functions like sin, cos, tan at infinity and $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$
- partial support for arctan, Γ, and erf
- support for parameters

All results produced by the procedure are trustworthy by construction as they pass through the Isabelle kernel, which reduces them down to definitions and basic logical inferences. On most practical examples, the procedure works quickly and fully automatically.

[2] Maxima claims that $\lim_{x\to\infty} \ln\ln(x + \exp(\ln x \ln\ln x))/\ln\ln\ln(\exp(x) + x) = 0$, whereas the correct result is 1. We reported this bug in February 2018 but have not received a response.

13 ACKNOWLEDGEMENTS

We would like to thank Joris van der Hoeven and Bruno Salvy for their help in understanding Multiseries and Maximilian P. L. Haslbeck and Kristina Magnussen for comments on a draft of this paper.

REFERENCES

[1] Julian Biendarra, Jasmin Christian Blanchette, Aymeric Bouzy, Martin Desharnais, Mathias Fleury, Johannes Hölzl, Ondřej Kunčar, Andreas Lochbihler, Fabian Meier, Lorenz Panny, Andrei Popescu, Christian Sternagel, René Thiemann, and Dmitriy Traytel. 2017. Foundational (Co)datatypes and (Co)recursion for Higher-Order Logic. In *Frontiers of Combining Systems*, Clare Dixon and Marcelo Finger (Eds.). Springer International Publishing, Cham, 3–21.

[2] Nicolas Bourbaki. 1971. *Topologie générale*.

[3] Manuel Eberl. 2017. Proving Divide and Conquer Complexities in Isabelle/HOL. *Journal of Automated Reasoning* 58, 4 (01 Apr 2017), 483–508. https://doi.org/10.1007/s10817-016-9378-0

[4] Dominik Gruntz. 1996. *On Computing Limits in a Symbolic Manipulation System*. Ph.D. Dissertation. ETH Zürich.

[5] Johannes Hölzl. 2009. Proving Inequalities over Reals with Computation in Isabelle/HOL. In *Proceedings of the ACM SIGSAM 2009 International Workshop on Programming Languages for Mechanized Mathematics Systems (PLMMS'09)*, Gabriel Dos Reis and Laurent Théry (Eds.). Munich, 38–45.

[6] Johannes Hölzl, Fabian Immler, and Brian Huffman. 2013. Type Classes and Filters for Mathematical Analysis in Isabelle/HOL. In *Proceedings of the 4th International Conference on Interactive Theorem Proving (ITP'13)*. Springer-Verlag, Berlin, Heidelberg, 279–294. https://doi.org/10.1007/978-3-642-39634-2_21

[7] Daniel Richardson. 1995. A Simplified Method of Recognizing Zero Among Elementary Constants. In *Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation (ISSAC '95)*. ACM, New York, NY, USA, 104–109. https://doi.org/10.1145/220346.220360

[8] Daniel Richardson, Bruno Salvy, John Shackell, and Joris Van der Hoeven. 1996. Asymptotic Expansions of exp-log Functions. In *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation (ISSAC '96)*. ACM, New York, NY, USA, 309–313. https://doi.org/10.1145/236869.237089

[9] Udo Richter. 2005. *Automatische Berechnung von Grenzwerten und Implementierung in Mathematica*. diploma thesis. Universität Kassel.

[10] Bruno Salvy and John Shackell. 2010. Measured limits and multiseries. *Journal of the London Mathematical Society* 82, 3 (2010), 747–762. https://doi.org/10.1112/jlms/jdq057

[11] John Shackell. 1990. Growth estimates for exp–log functions. *Journal of Symbolic Computation* 10, 6 (1990), 611 – 632. https://doi.org/10.1016/S0747-7171(08)80161-7

[12] John R. Shackell. 2004. *Symbolic Asymptotics*. Algorithms and Computation in Mathematics, Vol. 12. Springer Berlin–Heidelberg.

[13] Joris van der Hoeven. 1997. *Automatic asymptotics*. Ph.D. Dissertation. École polytechnique, Palaiseau, France.

[14] Joris van der Hoeven. 2006. *Transseries and real differential algebra*. Lecture Notes in Mathematics, Vol. 1888. Springer-Verlag.