1a) **Project Title.** Formalizing Elementary Analysis Rigorously.

1b) **Project Acronym.** F.E.A.R.

1c) **Principal Investigator.** Freek Wiedijk.

2a) **Summary.** Two classes of systems exist for working with mathematical formulas on a computer: 'computer algebra' programs (of which Mathematica is the best known) and 'proof assistants' (programs for the verification of mathematical proofs). The first kind of system (computer algebra) manipulates formulas that do not necessarily have a precise mathematical meaning (those formulas do not have a 'semantics'). The second kind of system (proof assistants) manipulates formulas that *do* have a precise meaning, but those often do not much resemble the formulas that one encounters in mathematical textbooks.

The F.E.A.R. project will design a language of formulas that have a precise semantics, but still resemble the traditional formulas found in textbooks. We will do this by translating a section from a classic handbook of mathematical formulas (the *Handbook of Mathematical Functions* by Abramowitz & Stegun) into a proof assistant. Since the formulas will be entered in a proof assistant, a sound semantics is guaranteed. Also it will be easy to judge whether the translation sufficiently resembles the original formulas from the handbook.

2b) **Abstract for laymen (in Dutch).** Er zijn twee soorten systemen om met wiskundige formules op de computer te werken: de 'computer algebra' programma's (waarvan Mathematica de bekendste is) en de 'bewijsassistenten' (programma's om wiskundige bewijzen mee te verifiëren). Het eerste soort systeem (computer algebra) manipuleert formules die niet altijd een precieze wiskundige betekenis hebben (die formules hebben geen 'semantiek'). Het tweede soort systeem (bewijsassistenten) manipuleert formules die wel een precieze betekenis hebben, maar vaak niet lijken op wat je in wiskundeboeken tegenkomt.

Het F.E.A.R. project zal een vorm ontwerpen voor formules in de computer zo dat die wèl een precieze betekenis hebben, maar tòch lijken op de traditionele formules uit de wiskundeboeken. We zullen dit doen door één van de secties uit een klassiek tabellenboek met wiskundige formules (het telefoonboek-dikke *Handbook of Mathematical Functions* van Abramowitz & Stegun) voor een bewijsassistent te vertalen. Omdat die formules in de bewijsassistent worden ingevoerd is dan een goede semantiek gegarandeerd. Maar ook zal het duidelijk zijn of de vertaling voldoende lijkt op de originele formules uit het tabellenboek.

3) **Classification.** Multidisciplinary: computer science & mathematics.
Discipline: 6.5 Formal Methods.
NOAG-i 2001-2005: Algorithms and Formal Methods (AFM).
MSC2000: 03B35, 13Pxx, 26-xx, 30Dxx, 68T15.

4) **Composition of the Research Team.**

name	*specialty*	*university*
prof. dr. H.P. Barendregt *promotor*	*type theory*	Radboud University Nijmegen
prof. dr. M.J. Beeson	*computer algebra* *theorem proving*	San José State University
dr. W. Bosma	*computer algebra*	Radboud University Nijmegen
dr. J.H. Geuvers	*type theory* *theorem proving*	Radboud University Nijmegen
dr. F. Wiedijk	*theorem proving*	Radboud University Nijmegen
PhD student		Radboud University Nijmegen

5) **Research Schools.** Dutch Research School in Logic (OZSL), Institute for Programming research and Algorithmics (IPA), Mathematical Research Institute (MRI).

6) **Description of the Proposed Research.** We will first present the background of this research. For a presentation of the specifics of the project, skip to *Core of the Project* on page 5. When in the text below we refer to *Abramowitz & Stegun*, we mean the well-known *Handbook of Mathematical Functions* [1] by Milton Abramowitz and Irene Stegun.

The research of the F.E.A.R. project is related to two kinds of system: *proof assistants* (theorem proving systems) and *computer algebra systems.*

Proof assistants are systems for the verification of mathematical proofs with the computer. Major proof assistants are PVS, ACL2, Theorema, Ωmega, IMPS, HOL, Isabelle, Coq, NuPRL and Mizar [26]. Proof assistants have two main application areas:

- The main application is 'formal methods': checking proofs that occur in the verification of the correctness of computer systems. This kind of verification is done when correctness is of the utmost importance, like in medical systems, systems embedded in spacecraft, systems for public transport (e.g., software controlling driverless trains), and systems that will be used in huge numbers (e.g., subsystems of commercial microprocessor chips, or programs running on smart cards).
- A secondary, more distant, application of these systems is to support mathematics. Currently proof assistants are not powerful enough to be useful in mathematical research, but in the future these systems might change the way we do mathematical proofs. Mathematics might then be routinely proof checked. In that case when a mathematical paper is submitted to a journal, referees will not need to be concerned with its correctness (because the computer will already have checked this), but just with the question of whether it is new and interesting. This – potentially very important, but currently rather utopian – application of proof assistants is nicely described in the so-called QED manifesto [7].

The main weakness of current proof assistants is that mathematically they are not powerful enough. They are very good at keeping track of all of the details of a mathematical proof: one routinely checks proofs using these systems that have hundreds, or even thousands, of cases. However, they are currently not very good at taking by themselves steps that a human mathematician considers to be trivial. One of the most important class of steps that are not supported well are problems that a high school student can solve without difficulty. We call these *high school problems.* Examples of this kind of problem (where $\vdash$ denotes logical consequence) are:

$$\begin{aligned} n \geq 2\ ,\quad x = \frac{1}{n+1} &\ \vdash\ \frac{x}{1-x} < 1 \\ x = i/n\ ,\quad n = m+1 &\ \vdash\ n! \cdot x = i \cdot m! \\ \frac{k}{n} \geq 0 &\ \vdash\ \left|\frac{n-k}{n} - 1\right| = \frac{k}{n} \end{aligned}$$

When one encounters problems like this in a proof assistant, one would like not to spend any time on it, but just to say to the system 'you know how to do this'. Currently this is not possible. We consider this to be the main weakness of current proof assistants.

Computer algebra systems are systems for the calculation of answers to algebraic problems with the computer. Major computer algebra systems are Matlab, Mathematica, Maple, Magma, GAP, Axiom and MathXpert. Computer algebra systems have two main application areas:

- Solving algebraic problems, like the calculation of the size of an algebraic structure, or an explicit enumeration of instances of such a structure. The algebraic objects that these calculations deal with are generally finite and explicitly given.
- Solving problems in elementary analysis ('calculus'). These include simplification of expressions, calculation of differentials and integrals, and solution of ordinary and differential equations. The name 'computer algebra' is misleading for this application, as these problems are not of an algebraic nature. However, these problems are often solved via a translation into an algebraic problem.

The main weakness of current computer algebra systems is that for these ‘calculus’ problems they often give incorrect answers. For instance the latest version of Maple evaluates the following expressions (due to Jacques Carette of McMaster University):

```
> value(eval(Sum(z^n/n!,n=0..infinity),z=0));
```

$$0$$

should be 1 (Maple simplified 0^n to 0)

```
> residue(1/(csgn(z)*z),z=0);
```

$$\frac{1}{\mathrm{csgn}(z)}$$

should not have z free (z was bound)

```
> eval(int(1/(x-a),x=0..1),a=1/2);
```

$$\pi i$$

should not be complex (real integral)

```
> Int(sin(x),x=0..Pi);
```

$$\int_0^\pi \sin(x)\,dx$$

```
> student[changevar](sin(x)=t,%,t);
```

$$\int_0^0 -\frac{t}{(1-t^2)^{\frac{1}{2}}}\,dt$$

these two integrals should be equal (change of variables) but they are not; the first is 2 but the second is 0

The reason for this behavior is, that it is much more difficult to always give a fully mathematically correct answer than a somewhat plausible one. In the tension between the ability to transform expressions for the user as much as possible, and the correctness of what the computer algebra system is doing, the makers of these systems take an intermediate position. The most blatant errors that users complain about are removed, but more subtle ‘errors’ are ignored. The idea is that the user should pay attention and judge for himself whether the program behaves reasonably.

Related to this problem of incorrect answers from computer algebra system is the lack of a formal semantics for the expressions of those systems. For instance in formulas like:

$$\ln(\infty) = \infty$$
$$\int \frac{1}{z}\,dz = \ln(z)$$

(both Mathematica and Maple calculate like this) it is not fully clear what the semantics of the symbols ∞ and $\int$ are (in the second case the question is: where is the constant of integration?) In particular those symbols are not *defined* in terms of some foundational system. For this reason it is sometimes not even possible to say that the answer of a computer algebra system is incorrect, because that might depend on the intended semantics of the symbols. As an example, it is not a priori clear whether

$$\frac{x+x}{x} = 2$$

is correct – because one is calculating in a field $K(x)$ – or incorrect – because for $x = 0$ the left hand side is $0/0$, which probably is not 2 in any way of dealing with $0/0$, and so the (partial) function $\lambda x.\frac{x+x}{x}$ differs from the (total) function $\lambda x.2$.

Summarizing: computer algebra systems structurally give incorrect answers, and the users of these programs do not even have the possibility to criticize these ‘errors’ because the expressions manipulated by these programs do not have an explicit semantics. We consider this to be the main weakness of current computer algebra systems.

Related to the lack of semantics for ‘analysis’ expressions in a computer algebra system is the meaning of expressions in languages that have been designed for the interchange of mathematical expressions between different systems. The two major languages of this kind are MathML [11] and OpenMath [9].

In practice these languages do not always conserve the meaning of expressions. For example, imagine encoding a formula in MathML to be able to move it between various proof assistants (like Coq, HOL or PVS). Then the following table shows that in that case the same MathML expression might get completely different meanings [28]:

	$1/0 = 0$
PVS	not a correct formula
IMPS	negation provable
Coq	not provable, negation not provable
HOL/Mizar	provable

Apparently if one uses MathML or OpenMath to communicate even a simple equation like '$1/x = x$' between systems, the meaning changes!

The aim of the F.E.A.R. project is to design a small expression language like MathML or OpenMath, but one that has a very explicit and well-defined semantics.

This project simultaneously addresses the two problems that we noted:

- Proof assistants are not powerful enough, as they cannot do 'high school problems' by themselves.
- Computer algebra systems are not fully correct; worse, they even lack a semantics that makes discussion of correctness meaningful.

Our approach to solve these problems will be to design an expression language for the kind of expressions that one encounters in a computer algebra system, but *with an explicit semantics.* This semantics will not just be given in a traditional mathematical style. To make sure that it is solid we will define it inside one of the major proof assistants. Proof assistants are built on a small logical foundation, which has a simple notion of 'model'. So formalizing in a proof assistant leads directly to a 'semantics' inside that same model.

This way we make the style of doing calculus in a computer algebra system – which can do 'high school mathematics' by itself – available to proof assistants. And, by providing a proper semantics, we get the possibility of having a computer algebra system that is fully correct. (Our project will lead to more complicated terms than one finds in Maple, with more subtle distinctions and embeddings and liftings and so on. Therefore, it will not give a semantics for Maple expressions the way they are today. But the idea is that one could, in the future, implement a correct Maple-like program based on this research. The project is necessary for that, but not sufficient. Without a proper focus on semantics we will never get correct computer algebra.)

It is relatively easy to render the meaning of formulas from elementary analysis in a proof assistant. However, to design an expression language in such a way that it also is close to the formulas that one encounters in normal mathematics textbooks is more difficult.

If one looks at the statements from elementary analysis that are present in the libraries of the current proof assistants, then they are sometimes remarkably different in style from the way that one would write them in normal mathematical texts. The reason for this is two-fold:

- *The proof assistants do not have the features that one needs to render mathematics reasonably.* For instance, in the HOL system all functions are total. Therefore, the limit operation when modeled as a function would always have a value, even if the limit does not exists. For this reason, the limit operator in the standard HOL library is modeled as a relation, instead of as a function. But that is not the way that limits are written customarily, and not as practical at that.
- *In mathematics one often uses imprecise notations.* Examples of this are calculations with 'infinity', indefinite integration where one omits 'constants of integration', 'multi-valued functions' in the complex plane where one leaves unspecified what branch one is on, etc. In informal practice this works perfectly, but in a situation where precision counts this is problematical.

Core of the Project. *We will formalize a section from* Abramowitz & Stegun *in a modern proof assistant. The expressions in the formalization will be made to resemble the original expressions as found in* Abramowitz & Stegun *as closely as possible. The formalization will be complete, except for the proofs. In particular, all the symbols that are used will be fully defined in terms of the foundations of the proof assistant. Although we will not encode all necessary proofs in the formalization, we will show in the traditional mathematical style that the formalization might be completed that way.*

Two choices will be made during the first phase of the project:

- *Which proof assistant?* The most natural choice is to take Coq [25]. In Nijmegen a large Coq library has been created [13] and there is a large amount of Coq expertise available there. Coq is also one of the most expressive systems for formalization of mathematics in the world today.
  Other possible choices would be HOL [15, 16, 23], Mizar [22] or PVS [24]. The HOL and Mizar systems have impressive libraries of mathematics that we could base our work on. PVS is the most popular proof assistant in computer science, and it supports partial functions in a way that the other three systems cannot.
- *Which section of Abramowitz & Stegun?* The most natural choice is to take the first relevant section: Section 4.1, which gives a list of formulas about the natural logarithm. It is reproduced as an appendix to this proposal on pages 11–13. It contains 57 formula entries which correspond to 74 inferences.
  Section 4.1 contains interesting challenges for a semantical treatment of the formulas. It distinguishes between ‘$\ln(z)$’ (the main branch of the complex natural logarithm) and ‘$\mathrm{Ln}(z)$’ (the ‘multi-valued function’). It also contains ‘fuzzy’ mathematical statements like $\ln(0) = -\infty$ (equation 4.1.13). An important kind of expression that Section 4.1 does not contain is order symbols (‘big-O notation’).
  The semantics for formulas like those in Section 4.1 ideally will give a meaning to a formula that contains infinity, constants of integration, multi-valued functions and order symbols, all at the same time.

In [4] we defined a rigorous semantics for formulas in elementary analysis that contain the symbol ∞. We did this by interpreting expressions that contain this symbol as *filters* over the underlying set of numbers. Part of the current project can be seen as an extension of this work.

If one follows the approach from [4] then one gets a proliferation of types, with coercion functions between them. For instance, the number 0 can then both be interpreted as a number, or as the filter corresponding to that number. When one extends this approach to cover multi-valued functions, it also can be interpreted as a set of numbers (a singleton set). The challenge of the F.E.A.R. project is how to define all these types – numbers, filters, sets – such that they fit together in a way that leads to practical formulas.

Writing the formulas from *Abramowitz & Stegun* using a signature similar to that of MathML is straight-forward. However, it is then clearly impossible to give a semantics to the symbols from that signature in such a way that the formulas become meaningful. For instance formulas 4.1.1 and 4.1.4 from *Abramowitz & Stegun* are $\ln z = \int_1^z \frac{dt}{t}$ and $\mathrm{Ln}\, z = \int_1^z \frac{dt}{t}$. Clearly the symbol $\int$ means something different in those two formulas.

The approach that we will follow in the project is to take a MathML-like encoding of the formulas, and then systematically *disambiguate* the symbols. For every symbol we will have various versions with various types (on numbers, filters, sets, etc.) Also we will have to insert coercion functions in the expressions to have them well-typed.

Finally, to get from a ‘naive’ MathML-like encoding of the formulas to a semantically meaningful disambiguation will be non-trivial. Therefore we will build automated support for this task. In this implementation the formulas will be encoded using the OpenMath/OMDoc standard [9, 17, 18].

This project will involve work on three different levels:

- First of all it will give a mathematical definition of a signature for the formulas of elementary analysis, and a formal definition of an expression language using that signature. This will then be written in the form of a normal mathematical article.
- After that it will create a formalization of these definitions. This will be work with a proof assistant. As working in a proof assistant tends to be time consuming, we expect this to be the main part of the project.
- Finally there will be an implementation component. A converter will be written between formulas in the proof assistant and a – system independent – OpenMath/OMDoc encoding of the same expressions. Also software will be written to help transform a naive rendering of the formulas into a semantically meaningful encoding.

Related Work.

Relating proof assistants to computer algebra systems is the subject of various projects, like the CALCULEMUS project [10], the FoC project [6], and the MathScheme project [14]. (The Foundations group in Nijmegen is a participant of the CALCULEMUS network.) These projects focus on implementation of frameworks that combine those two kinds of system. They do not primarily focus on the semantical questions that are the subject of the F.E.A.R. project.

There are projects that aim to create databases that are electronic versions of *Abramowitz & Stegun*, like the NIST's Digital Library of Mathematical Functions project [19, 21] and INRIA's Encyclopedia of Special Functions project [20] (which is part of its Algorithms Project). We could have based our project on formulas from those databases instead of on *Abramowitz & Stegun*. However, the formulas from those sources do not relate as much to semantics of computer algebra as the 'informal' formulas from *Abramowitz & Stegun*.

Prof. Davenport has written papers [8, 12] that are closely related to the topic of this proposal. At various CALCULEMUS meetings he has given talks in which he stated that the notion of *branch cut* from complex function theory has never been given a formal definition. He claimed to have searched many textbooks, and that the notion of branch cut is always just presented by example instead of being formally defined. It will be part of our project to provide Prof. Davenport with the formal definition he is looking for.

In the Foundations group in Nijmegen we have much expertise with formalization of mathematics. We have built a large library – called the C-CoRN library [13] – of digital mathematics in the proof assistant Coq. The Foundations group participates in various European projects like the CALCULEMUS project, the TYPES working group, the MoWGLI project and the MKM network. The project proposed here will fit seamlessly into the research of the Foundations group.

Some Questions & Answers.

Is this project about mathematics or about computer science?

About both!

From the point of view of mathematics, the project is conceptual: it focuses on definitions. However, it will need a deep understanding of what mathematical meaning is. Also the project will generate mathematical problems that, even if not 'deep', still will need mathematical proof. (It should be pointed out that Prof. Davenport has repeatedly claimed at the CALCULEMUS meetings that the mathematical problems inherent in the subject of this proposal are *not* trivial.)

In computer science, proof assistants are an important tool in the field of *formal methods*. A main goal of the project is to improve proof assistants. The project will heavily use proof assistants, and define a formal language to be used in proof assistants. For this formal language a translation system will be implemented to convert this language between encodings inside proof assistants and a system independent OpenMath encoding.

Is the formalization of a few pages of mathematics – and without full proofs at that – not too small an endeavor for a PhD project that will take four years?

Formalizing the *content* of those pages – without proofs – is not too hard. The difficulty is finding a form for those formulas that combines a solid semantics with a similarity to the informal formulas of *Abramowitz & Stegun* (which are similar to the one that one finds in computer algebra systems). From preliminary experiments with development of such an encoding, this appears to be far from trivial.

Aiming for large-scale formalization of mathematics is rather utopian: why should we support a project that reaches for such a far-away goal?

The goal of the project is to improve proof assistants and computer algebra systems. Using proof assistants for checking mathematics is only *one* of its applications. The use of proof assistants for the verification of critical computer systems is an important field of research, and not utopian at all.

Abramowitz & Stegun is not real mathematics, but 'mathematics for engineers': surely it is not important to mimic this style of doing calculus?

The way *Abramowitz & Stegun* write their formulas is close to the way that computer algebra systems represent those same formulas. The basic techniques of computer algebra systems work best when the formulas are closed algebraic equations or inequalities, instead of more complex logical formulas. If one would represent the mathematics in the traditional way to make it rigorous, then the formulas would not have this simple shape anymore.

Another reason for following the 'informal' way of doing calculus from *Abramowitz & Stegun* is that it is closer to the way a mathematician will reason on a piece of paper. When presenting his mathematics formally he might not use this style, but when doing the calculations himself he certainly will. This is an important reason for following the style of *Abramowitz & Stegun*: we want to improve the proof assistants in such a way that they become more attractive to mathematicians [27].

Elementary analysis is basic mathematics: surely for a competent mathematician it is a triviality to represent this kind of formulas?

The goal of having formal formulas that are close to the informal style of doing calculus is more a conceptual problem than a mathematical problem. To be able to do this project one not only needs mathematical competence, but also an understanding of formalization. Of course if a competent mathematician has this understanding he can work on this project. But we do not expect it to be trivial: if it was trivial, the computer algebra systems would already use an expression language like the one that this project is aiming for.

Why won't the project aim at formalizing the proofs too?

Although the formulas from *Abramowitz & Stegun* are elementary mathematics, formalizing their proofs would take a large amount of work. For the main goals of the project that work is irrelevant. It will just take effort away from the more important parts of the project. (And note that we *will* formalize some of the proofs, we just do not want to take the time to formalize *all* proofs.)

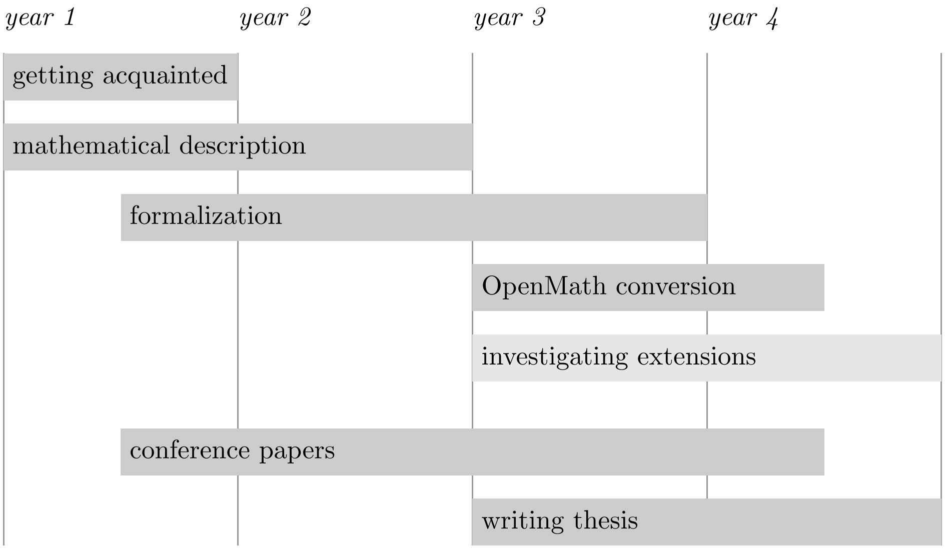
NIST is currently creating a database called the Digital Library of Mathematical Functions. That is a formal version of Abramowitz & Stegun*, isn't it? So hasn't your project already been done by them?*

The database that is being created by NIST will serve a similar purpose as *Abramowitz & Stegun*, but it is *not* a direct translation. Therefore it will not need to have counterparts to the informal formulas that one finds in *Abramowitz & Stegun* and in computer algebra (like $\ln(\infty) = \infty$). Also, the people that create this database do not worry about formal semantics – it is on the level of computer algebra in this respect – and so will put formulas like $\int \frac{1}{z}\,dz = \ln(z)$ in it without seeing a problem. For these reasons, NIST's project does not face the problems of proof assistants and computer algebra systems like we will do.

You claim that the current formulas in proof assistants often look different from their informal counterparts: but why do you think you can do better as you will be using those same proof assistants as well?

The reason that those formulas look different from the formulas of informal calculus is that it is *hard* to make them look similar. The people who did those formalizations were only interested in the *content* of the mathematics, and not in the shape of the formulas. We will spend a whole PhD project on this problem. That has not been done before.

7) **Work Programme.**



year 1 Getting acquainted with the relevant proof assistants. Getting acquainted with the relevant mathematics.
year 1–2 Abstract mathematical description of signature & semantics.
year 2–3 Formalization of signature/definitions in the primary proof assistant.
year 3 Definition of OpenMath CD for the signature. Implementation of conversion from OMDoc to input format of the primary proof assistant.
year 3–4 Investigation of the effort needed to complete the formalization with proofs. Investigation of the effort needed to treat other sections in *Abramowitz & Stegun* similarly. Investigation of the effort needed for formalization in other proof assistants.

year 1–3 Papers in conferences or workshops.
year 3–4 Writing the thesis.

The PhD student in this project will be part of the Institute for Programming research and Algorithmics (IPA). In addition to educational activities organized by IPA, the student will follow local courses at the University of Nijmegen, and participate in one or two summer schools. The primary supervisor will be Wiedijk, with Barendregt as the second supervisor and promotor. The research group has a good track record in supervising PhD students and hosts a good mix of PhD students and more experienced postdoc researchers and staff.

8) **Expected Use of Instrumentation.** Symbolic manipulation programs like proof assistants and computer algebra systems are notoriously resource hungry. For this reason we will dedicate a special computer to the project. On this system the major proof assistants and computer algebra systems will be installed.

hardware	dual processor system	€ 5,000
	maintenance contract	€ 360
software	license, Magma version 2.11	€ 950
	license, Maple version 9.5	€ 400
	license, Mathematica version 5.1	€ 1,460
	license, MathXpert version 3	€ 80
total		€ 8,250

9) Literature.

1. M. Abramowitz and I.A. Stegun, editors. *Handbook of Mathematical Functions With Formulas, Graphs, and Mathematical Tables*, volume 55 of *National Bureau of Standards Applied Mathematics Series.* United States Department of Commerce, Washington, D.C., June 1964. 10th Printing, December 1972, with corrections. `<http://www.math.sfu.ca/~cbm/aands/>`.
2. Henk Barendregt and Herman Geuvers. Proof-assistants Using Dependent Type Systems. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning.* Elsevier Science Publishers B.V., 2001. `<ftp://ftp.cs.ru.nl/pub/CompMath.Found/HBKassistants.ps>`.
3. M. Beeson. Using Nonstandard Analysis to Ensure the Correctness of Symbolic Computations. *International Journal of Foundations of Computer Science*, 6(3):299–338, 1995. `<http://www.mathcs.sjsu.edu/faculty/beeson/Papers/nsappt.ps.gz>`.
4. M. Beeson and F. Wiedijk. The Meaning of Infinity in Calculus and Computer Algebra Systems. In Jacques Calmet, Belaid Benhamou, Olga Caprotti, Laurent Henocque, and Volker Sorge, editors, *Artificial Intelligence, Automated Reasoning, and Symbolic Computatiation, Proceedings of the Joint International Conferences, AISC 2002 and Calculemus 2002, Marseille*, volume 2385 of *LNAI*, pages 246–258, 2002. `<http://www.cs.ru.nl/~freek/pubs/limits.ps.gz>`.
5. Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *Journal of Symbolic Computation*, 24:235–265, 1997.
6. S. Boulmé, T. Hardin, D. Hirschkoff, V. Ménissier-Morain, and R. Rioboo. On the way to certify Computer Algebra Systems. In *Proceedings of the Calculemus workshop of FLOC'99 (Federated Logic Conference, Trento, Italie)*, volume 23 of *ENTCS.* Elsevier, 1999. `<http://www1.elsevier.com/gej-ng/31/29/23/92/36/32/tcs23.3.006.ps>`.
7. R. Boyer et al. The QED Manifesto. In A. Bundy, editor, *Automated Deduction – CADE 12*, volume 814 of *LNAI*, pages 238–251. Springer-Verlag, 1994. `<http://www.cs.ru.nl/~freek/qed/qed.ps.gz>`.
8. R.J. Bradford, R.M. Corless, J.H. Davenport, D.J. Jeffrey, and S.M. Watt. Reasoning about the elementary functions of complex analysis. *Annals Maths Artificial Intelligence*, 36:303–318, 2002. `<http://www.apmaths.uwo.ca/~djeffrey/Offprints/AMAI.ps>`.
9. S. Buswell, O. Caprotti, D.P. Carlisle, M.C. Dewar, M. Gaëtano, and M. Kohlhase. The OpenMath Standard, v. 2.0, 2002. `<http://www.openmath.org/cocoon/openmath/standard/>`.
10. The CALCULEMUS Initiative. `<http://www.calculemus.net/>`.
11. David Carlisle, Patrick Ion, Robert Miner, and Nico Poppelier. Mathematical Markup Language (MathML) Version 2.0 (Second Edition), 2003. `<http://www.w3.org/TR/MathML2/>`.
12. R.M. Corless, J.H. Davenport, D.J. Jeffrey, and S.M. Watt. According to Abramowitz and Stegun. *SIGSAM Bulletin*, 34:58–65, 2000. `<http://www.apmaths.uwo.ca/~djeffrey/Offprints/couth.pdf>`.
13. L. Cruz-Filipe, H. Geuvers, and F. Wiedijk. C-CoRN: the Constructive Coq Repository at Nijmegen. To appear in LNCS 3119, Proceedings of MKM 2004, Białowieza, Poland. `<http://www.cs.ru.nl/~lcf/pubs/paper5.ps>`, 2004.
14. W.M. Farmer and M. v. Mohrenschildt. An overview of a Formal Framework for Managing Mathematics. *Mathematical Knowledge Management, special issue of Annals of Mathematics and Artificial Intelligence*, 38:165–191, 2003. `<http://imps.mcmaster.ca/doc/ffmm-overview.ps>`.
15. M.J.C. Gordon and T.F. Melham, editors. *Introduction to HOL.* Cambridge University Press, Cambridge, 1993.
16. John Harrison. *The HOL Light manual (1.1)*, 2000. `<http://www.cl.cam.ac.uk/users/jrh/hol-light/manual-1.1.ps.gz>`.
17. M. Kohlhase. OMDoc: Towards an Internet Standard for the Administration, Distribution and Teaching of mathematical Knowledge. In Eugenio Roanes Lozano, editor, *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2000*, number 1930 in LNAI. Springer-Verlag, 2000. `<http://www.faculty.iu-bremen.de/mkohlhase/papers/aisc.ps>`.
18. Michael Kohlhase. OMDoc, An Open Markup Format for Mathematical Documents (Version 1.2). Technical report, School of Engineering and Sciences, International University Bremen, 2004. `<http://www.mathweb.org/omdoc/omdoc1.2.ps>`.
19. D.W. Lozier, B.R. Miller, and B.V. Saunders. Design of a Digital Mathematical Library for Science, Technology and Education. In *Proceedings of the IEEE Forum on Research and Technology Advances in Digital Libraries; IEEE ADL '99, Baltimore, Maryland*, 1999. `<http://dlmf.nist.gov/about/publications/nistir6297.ps.gz>`.

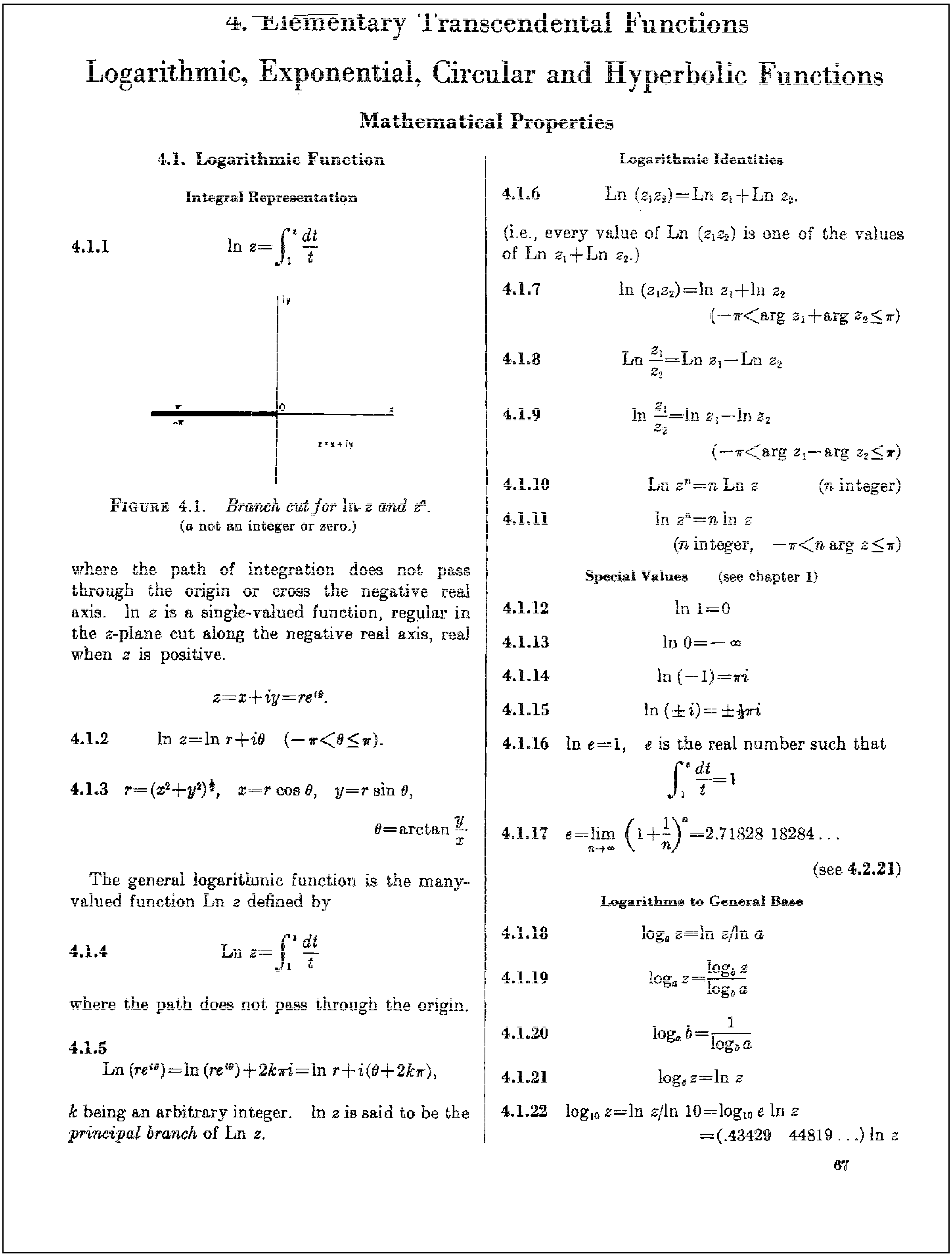
20. Ludovic Meunier and Bruno Salvy. ESF: An automatically generated encyclopedia of special functions. In J.R. Sendra, editor, *Symbolic and Algebraic Computation, Proceedings of ISSAC'03*, pages 199–205. ACM Press, 2003. `<http://algo.inria.fr/meunier/articles/MeSa03.ps>`.
21. B.R. Miller and A. Youssef. Technical Aspects of the Digital Library of Mathematical Functions. *Annals of Mathematics and Artificial Intelligence – Special Issue on Mathematical Knowledge Management*, 38:121–136, 2003. `<http://dlmf.nist.gov/about/publications/MKM-Miller-Youssef.ps.gz>`.
22. M. Muzalewski. *An Outline of PC Mizar.* Fondation Philippe le Hodey, Brussels, 1993. `<http://www.cs.ru.nl/~freek/mizar/mizarmanual.ps.gz>`.
23. T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002. `<http://www.cl.cam.ac.uk/Research/HVG/Isabelle/dist/Isabelle2004/doc/tutorial.pdf>`.
24. S. Owre, J. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *LNAI*, pages 748–752, Berlin, Heidelberg, New York, 1992. Springer-Verlag. `<http://www.csl.sri.com/papers/c/a/cade92-pvs/cade92-pvs.ps.gz>`.
25. The Coq Development Team. *The Coq Proof Assistant Reference Manual*, 2004. `<ftp://ftp.inria.fr/INRIA/coq/current/doc/Reference-Manual.ps.gz>`.
26. F. Wiedijk. Comparing mathematical provers. In Andrea Asperti, Bruno Buchberger, and James Davenport, editors, *Mathematical Knowledge Management, Proceedings of MKM 2003*, volume 2594 of *LNCS*, pages 188–202. Springer, 2003. `<http://www.cs.ru.nl/~freek/comparison/diffs.ps.gz>`.
27. F. Wiedijk. Formal Proof Sketches. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *Types for Proofs and Programs: Third International Workshop, TYPES 2003, Torino, Italy, April 30 – May 4, 2003, Revised Selected Papers*, volume 3085 of *LNCS*, pages 378–393, 2004. `<http://www.cs.ru.nl/~freek/pubs/sketches2.ps.gz>`.
28. F. Wiedijk and J. Zwanenburg. First Order Logic with Domain Conditions. In David Basin and Burkhart Wolff, editors, *Theorem Proving in Higher Order Logics, Proceedings of TPHOLs 2003*, volume 2758 of *LNCS*, pages 221–237. Springer, 2003. `<http://www.cs.ru.nl/~freek/pubs/partial.ps.gz>`.

Key Publications of the Research Team: [2] [3] [4] [5] [13]

10) Requested Budget (in Dutch).

OiO	*aanstelling*	1 fte × 157.683 =	€ 157.683
	benchfee	1 fte × 4.538 =	€ 4.538
	additioneel reisbudget		—
	apparatuur (incl. BTW)		€ 8.250
overig			—
investeringen			—
totaal			€ 170.471

The pages from Abramowitz & Stegun containing Section 4.1



4. Elementary Transcendental Functions

Logarithmic, Exponential, Circular and Hyperbolic Functions

Mathematical Properties

4.1. Logarithmic Function

Integral Representation

4.1.1 $$\ln z=\int_1^z \frac{dt}{t}$$

iy
π
−π
0
x
z=x+iy

FIGURE 4.1. *Branch cut for* ln *z and* z^a.
(a not an integer or zero.)

where the path of integration does not pass through the origin or cross the negative real axis. ln z is a single-valued function, regular in the z-plane cut along the negative real axis, real when z is positive.

$$z=x+iy=re^{i\theta}.$$

4.1.2 $\ln z=\ln r+i\theta \quad (-\pi<\theta\leq\pi).$

4.1.3 $r=(x^2+y^2)^{\frac{1}{2}}, \quad x=r\cos\theta, \quad y=r\sin\theta,$

$$\theta=\arctan\frac{y}{x}.$$

The general logarithmic function is the many-valued function Ln z defined by

4.1.4 $$\mathrm{Ln}\ z=\int_1^z \frac{dt}{t}$$

where the path does not pass through the origin.

4.1.5
$$\mathrm{Ln}\,(re^{i\theta})=\ln\,(re^{i\theta})+2k\pi i=\ln r+i(\theta+2k\pi),$$

k being an arbitrary integer. ln z is said to be the *principal branch* of Ln z.

Logarithmic Identities

4.1.6 $\mathrm{Ln}\,(z_1z_2)=\mathrm{Ln}\,z_1+\mathrm{Ln}\,z_2.$

(i.e., every value of Ln (z_1z_2) is one of the values of Ln z_1+Ln z_2.)

4.1.7 $\ln\,(z_1z_2)=\ln z_1+\ln z_2$
$(-\pi<\arg z_1+\arg z_2\leq\pi)$

4.1.8 $\mathrm{Ln}\,\frac{z_1}{z_2}=\mathrm{Ln}\,z_1-\mathrm{Ln}\,z_2$

4.1.9 $\ln\frac{z_1}{z_2}=\ln z_1-\ln z_2$
$(-\pi<\arg z_1-\arg z_2\leq\pi)$

4.1.10 $\mathrm{Ln}\,z^n=n\,\mathrm{Ln}\,z \qquad (n \text{ integer})$

4.1.11 $\ln z^n=n\ln z$
$(n \text{ integer}, \quad -\pi<n\arg z\leq\pi)$

Special Values (see chapter 1)

4.1.12 $\ln 1=0$

4.1.13 $\ln 0=-\infty$

4.1.14 $\ln\,(-1)=\pi i$

4.1.15 $\ln\,(\pm i)=\pm\frac{1}{2}\pi i$

4.1.16 $\ln e=1$, e is the real number such that
$$\int_1^e\frac{dt}{t}=1$$

4.1.17 $e=\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n=2.71828\ 18284\ldots$
(see **4.2.21**)

Logarithms to General Base

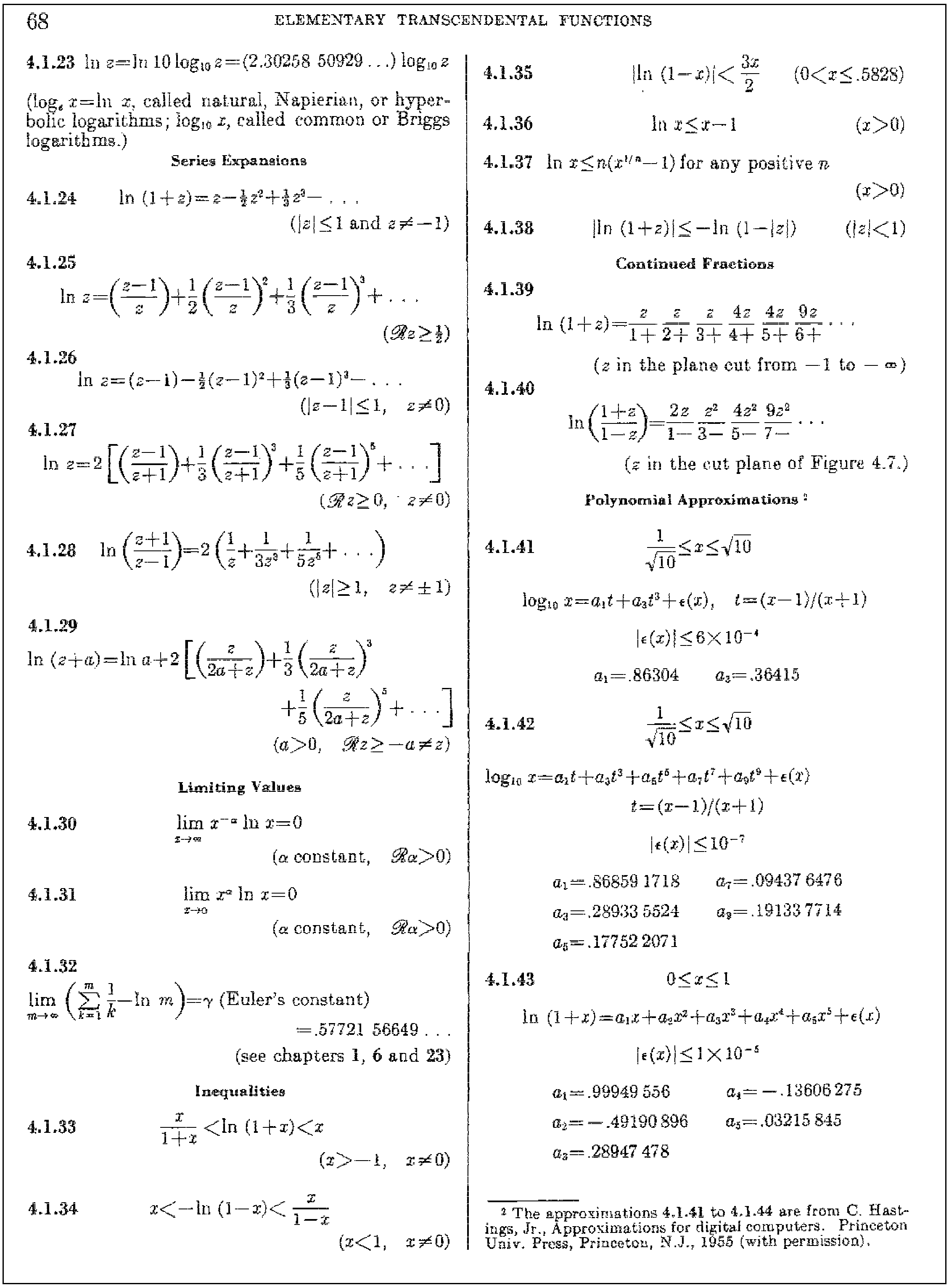
4.1.18 $\log_a z=\ln z/\ln a$

4.1.19 $\log_a z=\frac{\log_b z}{\log_b a}$

4.1.20 $\log_a b=\frac{1}{\log_b a}$

4.1.21 $\log_e z=\ln z$

4.1.22 $\log_{10} z=\ln z/\ln 10=\log_{10} e\ln z$
$=(.43429\quad 44819\ldots)\ln z$



68 ELEMENTARY TRANSCENDENTAL FUNCTIONS

4.1.23 $\ln z = \ln 10 \log_{10} z = (2.30258\ 50929\ldots) \log_{10} z$

($\log_e x = \ln x$, called natural, Napierian, or hyperbolic logarithms; $\log_{10} x$, called common or Briggs logarithms.)

Series Expansions

4.1.24 $$\ln(1+z) = z - \tfrac{1}{2}z^2 + \tfrac{1}{3}z^3 - \ldots$$ $(|z| \leq 1 \text{ and } z \neq -1)$

4.1.25
$$\ln z = \left(\frac{z-1}{z}\right) + \frac{1}{2}\left(\frac{z-1}{z}\right)^2 + \frac{1}{3}\left(\frac{z-1}{z}\right)^3 + \ldots$$ $(\mathscr{R}z \geq \tfrac{1}{2})$

4.1.26
$$\ln z = (z-1) - \tfrac{1}{2}(z-1)^2 + \tfrac{1}{3}(z-1)^3 - \ldots$$ $(|z-1| \leq 1, \quad z \neq 0)$

4.1.27
$$\ln z = 2\left[\left(\frac{z-1}{z+1}\right) + \frac{1}{3}\left(\frac{z-1}{z+1}\right)^3 + \frac{1}{5}\left(\frac{z-1}{z+1}\right)^5 + \ldots\right]$$ $(\mathscr{R}z \geq 0, \quad z \neq 0)$

4.1.28 $$\ln\left(\frac{z+1}{z-1}\right) = 2\left(\frac{1}{z} + \frac{1}{3z^3} + \frac{1}{5z^5} + \ldots\right)$$ $(|z| \geq 1, \quad z \neq \pm 1)$

4.1.29
$$\ln(z+a) = \ln a + 2\left[\left(\frac{z}{2a+z}\right) + \frac{1}{3}\left(\frac{z}{2a+z}\right)^3 + \frac{1}{5}\left(\frac{z}{2a+z}\right)^5 + \ldots\right]$$ $(a > 0, \quad \mathscr{R}z \geq -a \neq z)$

Limiting Values

4.1.30 $$\lim_{x\to\infty} x^{-\alpha} \ln x = 0$$ $(\alpha \text{ constant}, \quad \mathscr{R}\alpha > 0)$

4.1.31 $$\lim_{x\to 0} x^{\alpha} \ln x = 0$$ $(\alpha \text{ constant}, \quad \mathscr{R}\alpha > 0)$

4.1.32
$$\lim_{m\to\infty}\left(\sum_{k=1}^{m}\frac{1}{k} - \ln m\right) = \gamma \text{ (Euler's constant)}$$
$= .57721\ 56649\ldots$

(see chapters **1**, **6** and **23**)

Inequalities

4.1.33 $$\frac{x}{1+x} < \ln(1+x) < x$$ $(x > -1, \quad x \neq 0)$

4.1.34 $$x < -\ln(1-x) < \frac{x}{1-x}$$ $(x < 1, \quad x \neq 0)$

4.1.35 $$|\ln(1-x)| < \frac{3x}{2}$$ $(0 < x \leq .5828)$

4.1.36 $$\ln x \leq x - 1$$ $(x > 0)$

4.1.37 $\ln x \leq n(x^{1/n} - 1)$ for any positive n $(x > 0)$

4.1.38 $$|\ln(1+z)| \leq -\ln(1-|z|)$$ $(|z| < 1)$

Continued Fractions

4.1.39
$$\ln(1+z) = \frac{z}{1+}\,\frac{z}{2+}\,\frac{z}{3+}\,\frac{4z}{4+}\,\frac{4z}{5+}\,\frac{9z}{6+}\cdots$$ (z in the plane cut from -1 to $-\infty$)

4.1.40
$$\ln\left(\frac{1+z}{1-z}\right) = \frac{2z}{1-}\,\frac{z^2}{3-}\,\frac{4z^2}{5-}\,\frac{9z^2}{7-}\cdots$$ (z in the cut plane of Figure 4.7.)

Polynomial Approximations [2]

4.1.41 $$\frac{1}{\sqrt{10}} \leq x \leq \sqrt{10}$$

$$\log_{10} x = a_1 t + a_3 t^3 + \epsilon(x), \quad t = (x-1)/(x+1)$$

$$|\epsilon(x)| \leq 6 \times 10^{-4}$$

$a_1 = .86304 \qquad a_3 = .36415$

4.1.42 $$\frac{1}{\sqrt{10}} \leq x \leq \sqrt{10}$$

$$\log_{10} x = a_1 t + a_3 t^3 + a_5 t^5 + a_7 t^7 + a_9 t^9 + \epsilon(x)$$
$$t = (x-1)/(x+1)$$

$$|\epsilon(x)| \leq 10^{-7}$$

$a_1 = .86859\ 1718 \qquad a_7 = .09437\ 6476$

$a_3 = .28933\ 5524 \qquad a_9 = .19133\ 7714$

$a_5 = .17752\ 2071$

4.1.43 $$0 \leq x \leq 1$$

$$\ln(1+x) = a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + \epsilon(x)$$

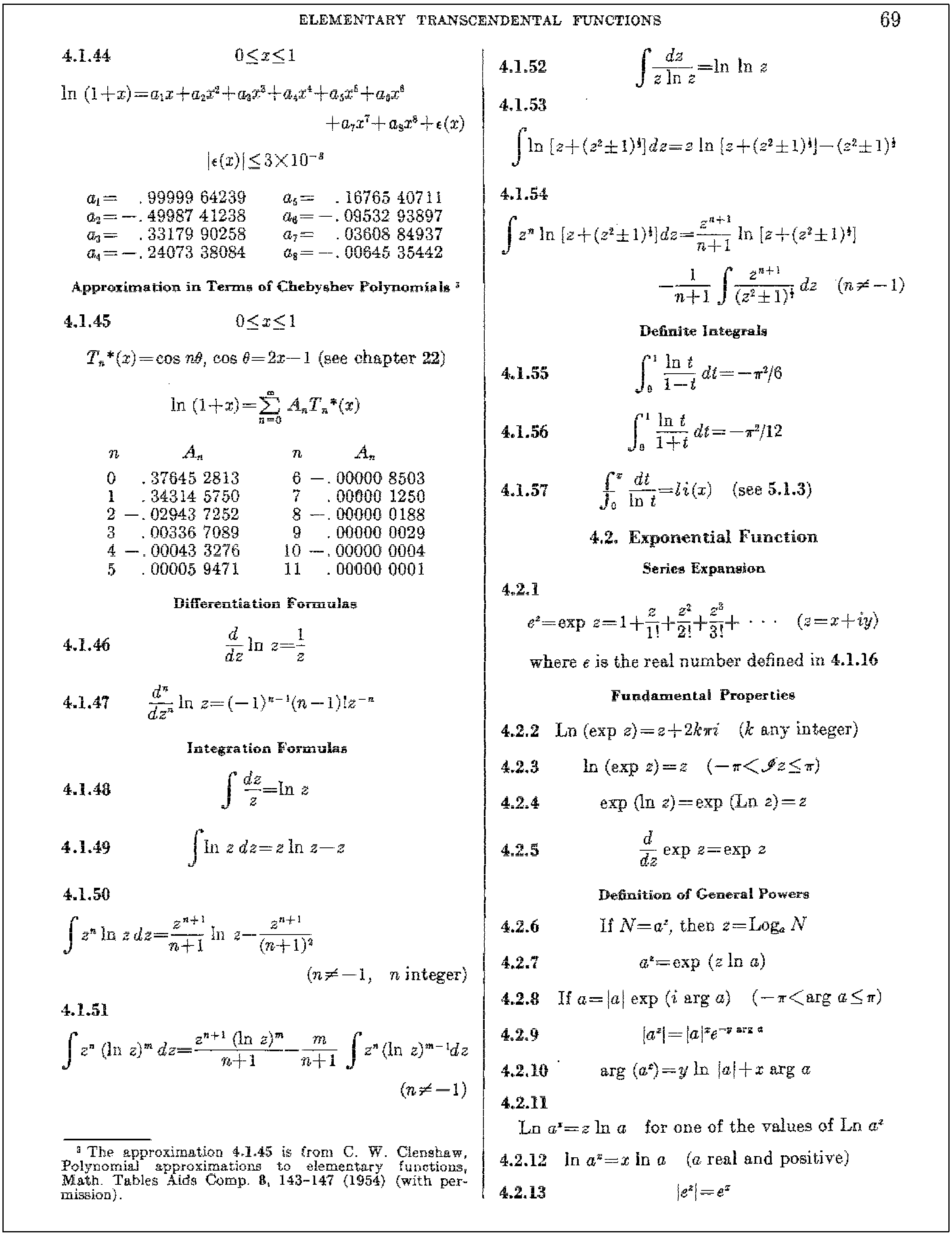
$$|\epsilon(x)| \leq 1 \times 10^{-5}$$

$a_1 = .99949\ 556 \qquad a_4 = -.13606\ 275$

$a_2 = -.49190\ 896 \qquad a_5 = .03215\ 845$

$a_3 = .28947\ 478$

[2] The approximations **4.1.41** to **4.1.44** are from C. Hastings, Jr., Approximations for digital computers. Princeton Univ. Press, Princeton, N.J., 1955 (with permission).



4.1.44 $0 \le x \le 1$

$$\ln(1+x) = a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7 + a_8x^8 + \epsilon(x)$$

$$|\epsilon(x)| \le 3 \times 10^{-8}$$

$a_1 =$	.99999 64239	$a_5 =$	.16765 40711
$a_2 =$	−.49987 41238	$a_6 =$	−.09532 93897
$a_3 =$	.33179 90258	$a_7 =$	.03608 84937
$a_4 =$	−.24073 38084	$a_8 =$	−.00645 35442

Approximation in Terms of Chebyshev Polynomials[3]

4.1.45 $0 \le x \le 1$

$T_n^*(x) = \cos n\theta,\ \cos\theta = 2x-1$ (see chapter 22)

$$\ln(1+x) = \sum_{n=0}^{\infty} A_n T_n^*(x)$$

n	A_n	n	A_n
0	.37645 2813	6	−.00000 8503
1	.34314 5750	7	.00000 1250
2	−.02943 7252	8	−.00000 0188
3	.00336 7089	9	.00000 0029
4	−.00043 3276	10	−.00000 0004
5	.00005 9471	11	.00000 0001

Differentiation Formulas

4.1.46 $$\frac{d}{dz}\ln z = \frac{1}{z}$$

4.1.47 $$\frac{d^n}{dz^n}\ln z = (-1)^{n-1}(n-1)!z^{-n}$$

Integration Formulas

4.1.48 $$\int \frac{dz}{z} = \ln z$$

4.1.49 $$\int \ln z\, dz = z\ln z - z$$

4.1.50

$$\int z^n \ln z\, dz = \frac{z^{n+1}}{n+1}\ln z - \frac{z^{n+1}}{(n+1)^2} \qquad (n \ne -1,\quad n \text{ integer})$$

4.1.51

$$\int z^n (\ln z)^m\, dz = \frac{z^{n+1}(\ln z)^m}{n+1} - \frac{m}{n+1}\int z^n (\ln z)^{m-1} dz \qquad (n \ne -1)$$

4.1.52 $$\int \frac{dz}{z\ln z} = \ln\ln z$$

4.1.53

$$\int \ln[z+(z^2\pm1)^{\frac{1}{2}}]dz = z\ln[z+(z^2\pm1)^{\frac{1}{2}}] - (z^2\pm1)^{\frac{1}{2}}$$

4.1.54

$$\int z^n \ln[z+(z^2\pm1)^{\frac{1}{2}}]dz = \frac{z^{n+1}}{n+1}\ln[z+(z^2\pm1)^{\frac{1}{2}}] - \frac{1}{n+1}\int \frac{z^{n+1}}{(z^2\pm1)^{\frac{1}{2}}}\,dz \qquad (n \ne -1)$$

Definite Integrals

4.1.55 $$\int_0^1 \frac{\ln t}{1-t}\,dt = -\pi^2/6$$

4.1.56 $$\int_0^1 \frac{\ln t}{1+t}\,dt = -\pi^2/12$$

4.1.57 $$\int_0^x \frac{dt}{\ln t} = li(x) \qquad \text{(see 5.1.3)}$$

4.2. Exponential Function

Series Expansion

4.2.1

$$e^z = \exp z = 1 + \frac{z}{1!} + \frac{z^2}{2!} + \frac{z^3}{3!} + \cdots \qquad (z = x+iy)$$

where e is the real number defined in **4.1.16**

Fundamental Properties

4.2.2 $\operatorname{Ln}(\exp z) = z + 2k\pi i$ (k any integer)

4.2.3 $\ln(\exp z) = z \quad (-\pi < \mathscr{I} z \le \pi)$

4.2.4 $\exp(\ln z) = \exp(\operatorname{Ln} z) = z$

4.2.5 $$\frac{d}{dz}\exp z = \exp z$$

Definition of General Powers

4.2.6 If $N = a^z$, then $z = \operatorname{Log}_a N$

4.2.7 $a^z = \exp(z\ln a)$

4.2.8 If $a = |a|\exp(i \arg a) \quad (-\pi < \arg a \le \pi)$

4.2.9 $|a^z| = |a|^x e^{-y \arg a}$

4.2.10 $\arg(a^z) = y\ln|a| + x\arg a$

4.2.11

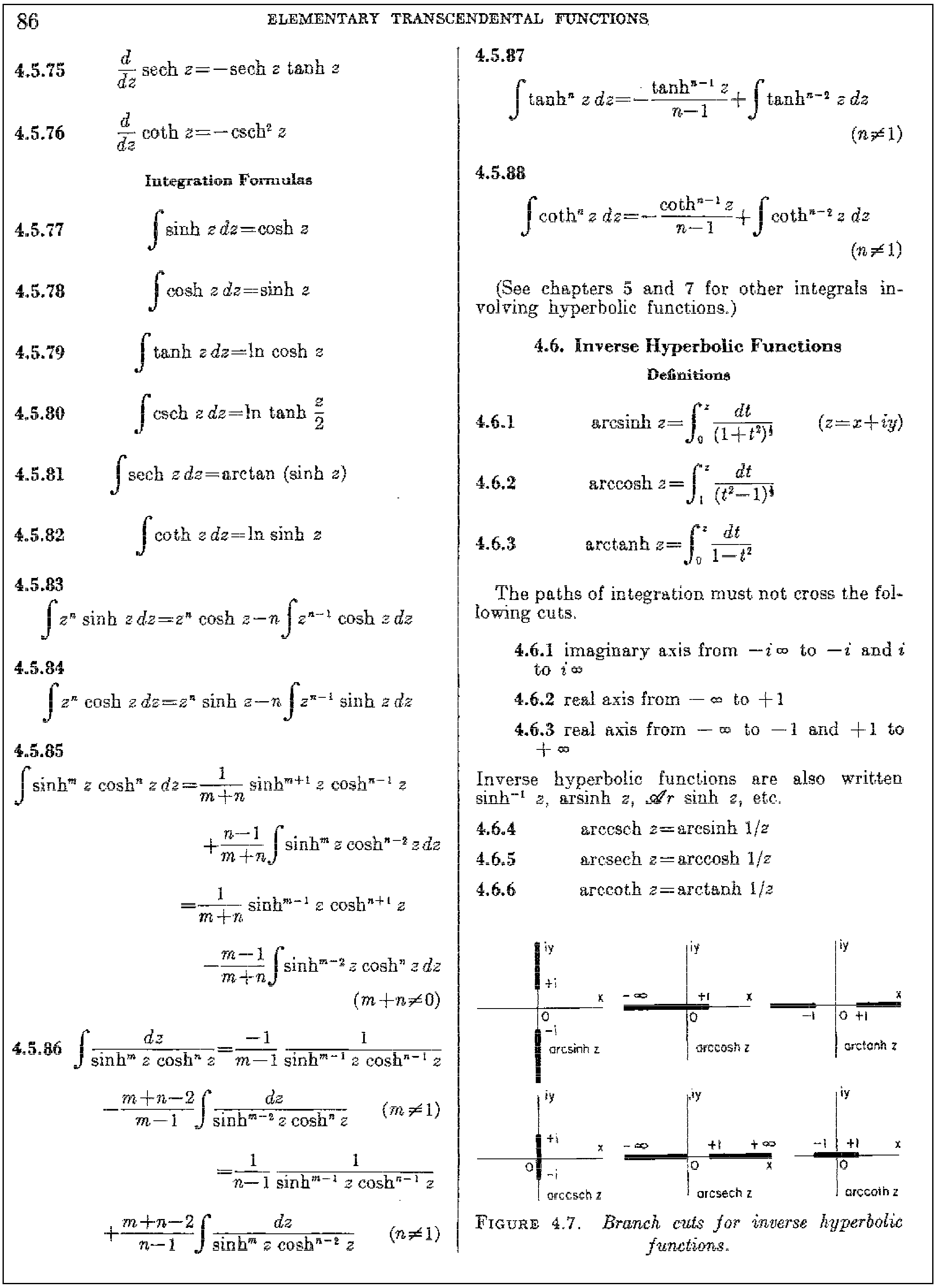
$\operatorname{Ln} a^z = z\ln a$ for one of the values of $\operatorname{Ln} a^z$

4.2.12 $\ln a^x = x\ln a$ (a real and positive)

4.2.13 $|e^z| = e^x$

[3] The approximation 4.1.45 is from C. W. Clenshaw, Polynomial approximations to elementary functions, Math. Tables Aids Comp. 8, 143–147 (1954) (with permission).

(Page 86, as Formula 4.1.40 refers to Figure 4.7)



4.5.75 $$\frac{d}{dz}\operatorname{sech} z=-\operatorname{sech} z \tanh z$$

4.5.76 $$\frac{d}{dz}\coth z=-\operatorname{csch}^2 z$$

Integration Formulas

4.5.77 $$\int \sinh z\,dz=\cosh z$$

4.5.78 $$\int \cosh z\,dz=\sinh z$$

4.5.79 $$\int \tanh z\,dz=\ln\cosh z$$

4.5.80 $$\int \operatorname{csch} z\,dz=\ln\tanh\frac{z}{2}$$

4.5.81 $$\int \operatorname{sech} z\,dz=\arctan(\sinh z)$$

4.5.82 $$\int \coth z\,dz=\ln\sinh z$$

4.5.83
$$\int z^n \sinh z\,dz=z^n\cosh z-n\int z^{n-1}\cosh z\,dz$$

4.5.84
$$\int z^n \cosh z\,dz=z^n\sinh z-n\int z^{n-1}\sinh z\,dz$$

4.5.85
$$\begin{aligned}\int \sinh^m z\cosh^n z\,dz&=\frac{1}{m+n}\sinh^{m+1} z\cosh^{n-1} z\\&\quad+\frac{n-1}{m+n}\int\sinh^m z\cosh^{n-2} z\,dz\\&=\frac{1}{m+n}\sinh^{m-1} z\cosh^{n+1} z\\&\quad-\frac{m-1}{m+n}\int\sinh^{m-2} z\cosh^n z\,dz\\&\qquad(m+n\neq0)\end{aligned}$$

4.5.86
$$\begin{aligned}\int\frac{dz}{\sinh^m z\cosh^n z}&=\frac{-1}{m-1}\,\frac{1}{\sinh^{m-1} z\cosh^{n-1} z}\\&\quad-\frac{m+n-2}{m-1}\int\frac{dz}{\sinh^{m-2} z\cosh^n z}\qquad(m\neq1)\\&=\frac{1}{n-1}\,\frac{1}{\sinh^{m-1} z\cosh^{n-1} z}\\&\quad+\frac{m+n-2}{n-1}\int\frac{dz}{\sinh^m z\cosh^{n-2} z}\qquad(n\neq1)\end{aligned}$$

4.5.87
$$\int\tanh^n z\,dz=-\frac{\tanh^{n-1} z}{n-1}+\int\tanh^{n-2} z\,dz\qquad(n\neq1)$$

4.5.88
$$\int\coth^n z\,dz=-\frac{\coth^{n-1} z}{n-1}+\int\coth^{n-2} z\,dz\qquad(n\neq1)$$

(See chapters 5 and 7 for other integrals involving hyperbolic functions.)

4.6. Inverse Hyperbolic Functions

Definitions

4.6.1 $$\operatorname{arcsinh} z=\int_0^z\frac{dt}{(1+t^2)^{\frac{1}{2}}}\qquad(z=x+iy)$$

4.6.2 $$\operatorname{arccosh} z=\int_1^z\frac{dt}{(t^2-1)^{\frac{1}{2}}}$$

4.6.3 $$\operatorname{arctanh} z=\int_0^z\frac{dt}{1-t^2}$$

The paths of integration must not cross the following cuts.

4.6.1 imaginary axis from $-i\infty$ to $-i$ and i to $i\infty$

4.6.2 real axis from $-\infty$ to $+1$

4.6.3 real axis from $-\infty$ to -1 and $+1$ to $+\infty$

Inverse hyperbolic functions are also written $\sinh^{-1} z$, arsinh z, $\mathscr{A}r$ sinh z, etc.

4.6.4 $\operatorname{arccsch} z=\operatorname{arcsinh} 1/z$

4.6.5 $\operatorname{arcsech} z=\operatorname{arccosh} 1/z$

4.6.6 $\operatorname{arccoth} z=\operatorname{arctanh} 1/z$

FIGURE 4.7. *Branch cuts for inverse hyperbolic functions.*