Using Nonstandard Analysis to Ensure the Correctness of Symbolic Computations

Michael Beeson
Department of Mathematics and Computer Science
San Jose State University
San Jose, California 95192, USA
email: beeson@mathcs.sjsu.edu

July 19, 1995

Abstract

Nonstandard analysis has been put to use in a theorem-prover, where it assists in the analysis of formulae involving limits. The theorem-prover in question is used in the computer program *Mathpert* to ensure the correctness of calculations in calculus. Although non-standard analysis is widely viewed as non-constructive, it can alternately be viewed as a method of reducing logical manipulation (of formulae with quantifiers) to computation (with rewrite rules). We give a logical theory of nonstandard analysis which is implemented in *Mathpert*. We describe a procedure for "elimination of infinitesimals" (also implemented in *Mathpert*) and prove its correctness.

1 Introduction

The general context of this research is the use of logic to ensure the correctness of computational steps. Many computational steps have side conditions that must be satisfied when they are applied. Current computational systems generally ignore these side conditions, with the consequence that their answers are unreliable. For example, in Macsyma one can enter $a = 0$, then divide both sides of the equation by a with the command $\%/a$. Macsyma will reply $1 = 0$, because it has mindlessly applied the rules $a/a = 1$ and $0/a = 0$. More examples can be found in Beeson[5], Harrison and Théry[17], and a larger number of examples can be found in Stoutemeyer.[26] Contrary to the first impressions of some observers, these problems cannot be corrected by simply adding some simple checks to algebraic computation systems.

A computation which appears to be a sequence of mathematical formulae, one per line, each one obtained by transforming the previous one according

to a computation rule, really has a more intricate logical structure. Each line depends on a list of *assumptions*, which are usually not written. Some of the computation rules generate more assumptions, and some use the assumptions to verify their side conditions. As the computation proceeds, more assumptions are generated; but also the assumptions themselves may simplify and even disappear if they simplify to `true`. It can also happen that the simplification of assumptions reveals a contradiction among the assumptions, so that what has been derived is a *non sequitur*.

The side conditions of computational rules are sometimes simple, such as denominators being nonzero, or things inside square roots being nonnegative. This soon leads to questions of the domain of expressions: it is clear that an adequate computational system must be able to calculate the domain of every expression.

The specific impetus for this work arose in the development of *Mathpert*, which is a computerized environment for learning algebra, trigonometry, and calculus. For an introductory description of this system, see [6]. *Mathpert* was designed to support a logical apparatus sufficient to ensure correctness of the answers it delivers. Algebra and trigonometry, if we omit indexed sums, can be supported more or less on the basis so far described. Specifically, *Mathpert* uses an *infer-refute-assume* algorithm[5, 7]. When a rule of computation requires a side condition, *Mathpert* first attempts to infer that condition (from the current assumptions and known, simple, axioms); if that fails it attempts to refute the condition. If the refutation succeeds it refuses to apply the operation. (This stops the MACSYMA example above.) If neither inference nor refutation succeeds, it assumes the condition. In this way the list of assumptions can grow.

Calculus introduces an additional logical problem, not required in elementary algebra and trigonometry, whose solution with the aid of nonstandard analysis is the subject of this paper. The additional problem is the treatment of bound variables. Normally in logic we think of variables bound by quantifiers. In calculus, however, variables are bound by the limit operator, the definite integral operator, and indexed sums.[1] Consider, for example, the problem of determining the domain of

$$\sum_{k=1}^{n} x^k.$$

This domain must come out to be `true`–the function is defined for all x. We must first of all inform the system that index variables are always integers. That being done, the main problem stands out: x^k is defined only for $x \neq 0 \vee k > 0$. *Mathpert* handles this problem as follows: When doing computation, the expression tree is traversed depth-first. Some computation rules are applied

[1] In general, bound variables can be reduced to functional operations and lambda abstraction, as is well-understood by experts. The additional problem is not merely the existence of additional binding operators, but the rules for mixing these operators with symbolic computation correctly.

on the way down (that is, before their arguments are simplified) and some on the way up (after their arguments are simplified). When it passes into an indexed sum, *Mathpert* generates the *temporary* assumption that the index variable is between the lower and upper limits. In the example, that would be the assumption $1 \leq k \wedge k \leq n$. The assumption $1 \leq n$ would also be generated (if it could not be inferred or refuted), but that is not a temporary assumption and not the point of interest here. The temporary assumption is available for use while computation is proceeding inside the indexed sum. In particular, it is available during the analysis of the domain of x^k, which therefore comes out everywhere defined. When computation exits the indexed sum after traversal of the entire expression tree, the temporary assumption is "discharged".

Logically speaking: each line of computation can be viewed as a "sequent" $\Gamma \Rightarrow A$, where A is the visible line and Γ is the list of current assumptions. The list Γ can grow from line to line as computation rules generate assumptions; it can also shrink if assumptions simplify. But during computation, while we are fixed at a single line, it can also grow and shrink as computation goes deeper into and then out of variable-binding functors on the right side. This dynamic quality is essential for the proper handling of bound variables.

2 The Problem, its Solution, and Examples

2.1 The Problem

The scheme described above seems adequate for the treatment of definite integrals and indexed sums. But calculus requires the treatment of limit terms as well, and it is far from obvious what temporary assumptions must be made when entering a limit term. Consider *Example 1*:

$$\lim_{x \to 0} \left(\frac{\sin x^2}{x} + \frac{1}{x-1} \right)$$

In order to work correctly with the "limitand" (the formula inside the limit), we must be able to infer that $x \neq 0$ and that $x \neq 1$. Of course, we cannot generate any assumption involving x that will appear outside the scope of the limit operator.

Here is *Example 2*. In calculating the derivative of $\sqrt{x}$ from the limit definition of derivative, we reach a step in which we want to rewrite $(\sqrt{x+h})^2$ as $x+h$. (Here h is the bound variable.) The rule $(\sqrt{y})^2 = y$ is only valid when $0 \leq y$, and we cannot generate the assumption $0 \leq x+h$, because it involves the bound variable. The correct assumption to generate is $0 < x$. How can this desired result be reached, by a general-purpose algorithm?

The problem is this: we must ensure that when computation is performed inside a limit term, the logical inferences performed in verifying side conditions

are correct, and any assumptions generated by operations performed inside the limit term do not involve the bound limit variable.

2.2 A solution using nonstandard analysis

A workable solution to this problem was described six years ago.[5] It amounts to introducing second-order logic into the computation system, for example a second-order functor `true_in_nbhd`, which takes a proposition and point as argument and is true if the proposition is true in a neighborhood of the point. Although this solution worked, it was complicated.

This paper describes another, more elegant solution, using nonstandard anlysis. In essence, it is a way to replace second-order inferences by first-order computation: calls to the limit-calculator. This saved several thousand lines of complicated code in *Mathpert*.[2]

The idea is to treat limit terms in exactly the same way as definite integrals and indexed sums are treated: to make a temporary (first-order) assumption when entering the term and discharge it on exit, and let computation and inference both proceed normally while inside the term. The assumption is obvious once nonstandard analysis is considered: when entering $\lim_{x \to a} f(x)$, make the temporary assumption that $x - a$ is infinitesimal, and $x \neq a$. For one-sided limits from the right, assume $a < x$ instead of $x \neq a$, and for limits from the left, assume $x < a$.

When computation completes its traversal of the limitand, any assumptions which have been generated involving the limit variable, and not already eliminated by simplification, must be eliminated: we cannot generate an assumption involving a bound variable outside its scope. If this can be done by making additional assumptions, it is done. If it can't, then the computation or inference we were attempting has failed.

We supply the system with both axioms and computational rules for dealing with infinitesimals and infinite nonstandard numbers. The fundamental concepts in our theory are $x \cong y$, which means that $x - y$ is infinitesimal (or zero), and $[\![x]\!]$, the standard part of x. Since we work in the logic of partial terms, **LPT**, there is no problem about $[\![x]\!]$ being undefined for infinite numbers. The axioms are stated using the predicate $\cong$ and the function symbol $[\![\cdot]\!]$. The computation rules are directed towards the goal of being able to compute $[\![t]\!]$ for as many terms t as possible. Here "compute" means to eliminate nonstandard variables.

For example, reconsider the computation of the derivative of $\sqrt{x}$, where we

[2]The effect of this change (made in June, 1992) was invisible to the user. Logic is in the background in *Mathpert*. Students of calculus are not taught to worry about logical niceties. But since *Mathpert*, unlike its answer-only cousins, produces only correct results, it must tend to logical matters behind the scenes. The change to nonstandard analysis was thus doubly invisible: logic is visible only if you explicitly choose **View Assumptions**, and besides, nonstandard analysis produces the same results as second-order inference.

had to rewrite $(\sqrt{x+h})^2$ as $x+h$. This now takes place under the assumption that h is infinitesimal (and not zero). When *Mathpert* tries to verify $0 \leq x+h$, it will compute the standard part of $x+h$ to be x, and then simplify $0 \leq x+h$ to $0 < x \vee (0 \leq x \wedge 0 \leq h)$. The system then simplifies this to $0 < x$. If we had been inside a one-sided limit from the right, it would have simplified to $0 \leq x$.

2.3 Examples

One referee asked for more examples correctly worked by *Mathpert*. The point to be emphasized here is that *Mathpert* will correctly work *every* example of a limit problem found in calculus textbooks. At least, it has worked hundreds of such examples correctly. The relevant point is not that it arrives at the correct answer for the limit (which it does), but that it (1) analyzes the domain of the expression involving the limit correctly, (2) makes no unnecessary assumptions while calculating the limit, (3) makes all necessary assumptions to ensure correctness, and (4) makes no incorrect assumptions, such as assumptions involving the limit variable made outside the scope of the variable. These claims can be verified by choosing **Assumptions** from the **View** menu of *Mathpert* in order to inspect the assumption list as a limit computation progresses. Here are some specific examples.

$$\lim_{x \to 1} 1/x \qquad \textit{Example 3}$$

Mathpert correctly shows an empty assumption list after analyzing the domain of this expression.

$$\lim_{x \to 1} \tan x \qquad \textit{Example 4}$$

Again *Mathpert* shows an empty assumption list, even though the expression for the domain of $\tan x$ is $x \neq (2n+1)\pi/2$, which involves a new existentially quantified variable n of type `int`, generated by the domain analyzer.

$$\lim_{x \to 2} \frac{x^2-1}{\sqrt{x-1}} \qquad \textit{Example 5}$$

First, no assumption is generated by the requirement that things inside square roots be nonnegative. Second, when the expression $\sqrt{x-1}\sqrt{x-1}$ is simplified to $x-1$, the side condition $x-1 \geq 0$ is properly inferred, because this takes place inside a limit as x approaches 2. The side condition reduces to $\alpha > 1$, where α is a nonstandard variable whose distance to 2 is infinitesimal.

$$\lim_{x \to 1^+} \frac{x^2-1}{\sqrt{x-1}} \qquad \textit{Example 6}$$

Now the limit point is at the zero of the denominator, but because it is a one-sided limit, everything goes as with the previous example.

$$\lim_{x \to 1} \frac{x^2-1}{\sqrt{x-1}} \qquad \textit{Example 7}$$

This is the same as the previous example except it is a two-sided limit. The domain analyzer can determine the expression is undefined, since the domain of $\sqrt{\alpha - 1}$ for a nonstandard α whose distance to 1 is infinitesimal will reduce to the proposition $\alpha \geq 1$, which will reduce to to `false` using the algorithm of this paper.

Mathpert includes many mathematical operations with side conditions, and one can give arbitrarily complicated examples, but these mathematically simple examples should suffice to illustrate the logical points involved.

2.4 Correctness of the Algorithm

The above examples illustrate the use of nonstandard analysis in assisting with inferences and reductions made inside the scope of a limit. The paper will give an algorithm making this method systematic. In particular, the algorithm must eliminate a nonstandard variable introduced in this way. Certain steps of the algorithm replace a formula such as $\alpha < 1$ by a formula (for example `false`) which is not logically equivalent. Therefore a correctness proof is called for. Theorem 3 of this paper provides such a correctness proof. The proof is based on "interval semantics": a formula $\phi(\alpha)$ involving the nonstandard variable α is interpreted to mean that $\phi(x)$ is true for all x in a suitable (punctured) neighborhood of the limit point. (The neighborhood will be one-sided for a one-sided limit.)

3 Logic with types and partial terms

The main purpose of this paper is to describe an algorithm for "infinitesimal elimination" and prove its correctness. However, it is not so easy even to *state* the correctness theorem, as that necessarily involves a formal logical theory for calculus. Since the algorithm involves nonstandard analysis, we also need a formal theory for nonstandard analysis. In this section we formulate a suitable first-order theory of standard calculus. This requires some reformulations of traditional first-order logic, to deal with partial terms such as $\sqrt{x}$, and to deal with the type-embedding problem. Readers who do not wish to check the details of the correctness proof may be able to skip this section.

To formulate a theory of real numbers with square root in ordinary predicate logic, we would have to allow $\sqrt{(-1)}$ as a term. It is unnatural to work with $\sqrt{x}$ when x is negative (when we do not have complex numbers in mind). It is better to use a Logic of Partial Terms (**LPT**) which takes "t is defined" as a primitive concept. A suitable logic **LPT** has been formulated[3]. One of its rules for formula formation is that if t is a term, then $t \downarrow$ is a formula, which means t is defined.

The type-embedding problem already arises when we want to discuss both integers and real numbers, and have 3 be both an integer and a real number.

To understand the problem clearly, consider that we will want two types, `int` (for integers) and `real` (for real numbers). In a notation to be fully explained below, we write $\forall x.\texttt{int}$ to quantify over integers, and $x : \texttt{real}$ for a formula expressing that x is a real number. We can then state the commutativity of adding integers and reals, and the commutativity of adding reals and reals, and we should be able to derive the former from the latter.

Specifically, we should be able to derive $\forall n.\texttt{int}\ \forall x.\texttt{real}\ (n + x = x + n)$ from $\forall x.\texttt{real}\ \forall y.\texttt{real}\ (x+y = y+x)$, making use of the type-embedding axiom $\forall n : \texttt{int}\ (n : \texttt{real})$.

There are two traditional methods to solve the type-embedding problem: The first is to use two unary predicates $N(x)$ and $R(x)$, with the type-embedding axioms $N(x) \rightarrow R(x)$. This has the disadvantage that all quantifiers must be explicitly relativized to these unary predicates. The second method is to use a two-sorted predicate calculus, with one sort for integers and one for reals. This has the disadvantage that there must be an explicit type-embedding function that converts the integer 3 to the real number 3. When writing papers, as opposed to computer programs, one can make some conventions that slur the distinctions between these two approaches, but to be precise one needs an apparatus that combines the best of these two approaches. In this approach a variable is labelled with its type (as in two-sorted predicate calculus), but predicate and function symbols are not required to take exactly one type as argument. Thus the formula $x < y$ is well-formed regardless of whether x and y have type `real` or `integer`. I do not know of any mention of such a version of predicate calculus in the literature, but it is a straightforward matter to reduce it to known versions, and we will use it. To illustrate the legal syntax: if we have a type `int`, then $\forall x.\texttt{int}\ \forall y.\texttt{int}(x + y = y + x)$ would be a legal formula. We use the notation $x.\texttt{int}$ to indicate type information attached to a variable. On the other hand $x : \texttt{int}$ will be written for a formula. Officially this should be $x.\texttt{int} : \texttt{int}$, but we will never write it out like this. In an implementation, some bits of (or accessible from) the object representing x will be set aside for type information, and the notation $x.\texttt{int}$ is meant to denote that those bits contain the code for the type `int`. The use of two different symbols $x.\texttt{int}$ (which is a variable) and $x : \texttt{int}$ (which is a formula) helps in writing a precise grammar below.

This shows the type information explicitly. When writing on paper, we show the type labels on variables only when they are bound, as in $\forall n.\texttt{int}\ \forall x.\texttt{real}(x + n = n + x)$, but the type labels are neverthless officially present with each occurrence of the variable. We also allow 'typings' to be used as atomic formulae. Thus $\forall n.\texttt{int}(n : \texttt{real})$ is a legal formula, which can be used to express a type-embedding.

We set out here the syntax and rules of first-order logic with partial terms and typed variables. This version of first-order logic gives a good solution of the type-embedding problem and of the undefined-terms problem.

```
variable::=  identifier.type
term::= variable | constant | functor(termlist)
termlist::= term | term,termlist
atomic_formula::=   term ↓ |
                    true | false |
                    term :  type |
                    term = term |
                    term infix_relation term |
                    pfunctor(termlist)
formula::=  atomic_formula |
            formula ∧ formula |
            formula ∨ formula |
            formula → formula |
            ¬ formula |
            ∀ variable (formula)
            ∃ variable (formula)
formula_list::=  formula | formula,formulalist | <empty>
sequent::=  formulalist ⇒ formula
classical_sequent::= formulalist ⇒ formulalist
```

Note that these rules make $x.A$ a variable, not x standing alone. However, in writing on paper, we suppress the typing information and just write x, except where quantifiers bind variables. We formulate the logic using Gentzen sequents. The rules for propositional calculus are the usual Gentzen rules (see e.g. system G1 in Kleene[19]).

To specify any particular system in this logic, we must add grammar rules for `functor`, `pfunctor`, `constant`, and `type`. This is the same as in the usual presentations of first-order logic, where a particular first-order theory must be specified by its function symbols and constants. Here we must also give types and *pfunctors*, that is, symbols used for atomic propositions. We also adopt the usual convention regarding arities of functors: each functor comes with a specified arity, so that not just any `termlist` can be put into just any functor. The grammar above does not enforce such a restriction, but all readers will know how to make it do so, at some cost in readability. To fix the ideas, one may take the rules

```
type ::- real | int
infix_relation ::- < | ≤ | ≠ | =
functor ::- ln, sqrt, ...
```

and have no rule at all for `pfunctor`.

The Gentzen rules for variables and quantifiers require modification, both because of the use of **LPT** and because of the use of typed variables. In these rules, Γ and Δ are lists of formulae, A is a type, and t is a term. Substitution of a term for a variable is indicated by a slash. Here are the rules:

$$\frac{\Gamma, b : A \Rightarrow \phi[b.A/x]}{\Gamma \Rightarrow \forall x.A(\phi)}$$

$$\frac{\Gamma, b : A, \phi[b.A/x] \Rightarrow \psi}{\Gamma, \exists x.A(\phi) \Rightarrow \psi}$$

$$\frac{\Gamma, t : A, \phi[t/x.A] \Rightarrow \psi}{\Gamma, \forall x.A(\phi) \Rightarrow \psi}$$

$$\frac{\Gamma \Rightarrow \phi[t/x.A] \qquad \Delta \Rightarrow t : A}{\Gamma, \Delta \Rightarrow \exists x.A(\phi)}$$

The first two of these rules are subject to the usual restriction on variables, that b is not free in the conclusion of the rule, that is, the part below the line. These rules are the intuitionistic rules; the classical rules are obtained just by allowing more formulae on the right of $\Rightarrow$. All theories used in this paper will be classical.

The use of constant propositions `true` and `false` requires the axioms

$$\Gamma \Rightarrow \texttt{true}$$

$$\texttt{false}, \Gamma \Rightarrow \phi$$

and a reformulation of the rule for introducing negation:

$$\frac{\Gamma, \phi \Rightarrow \texttt{false}}{\Gamma \Rightarrow \neg\phi}$$

With this rule, we can make sure the succedent (right hand side of $\Rightarrow$) is never empty.

The equality axioms in **LPT** take the form

$$x = x \wedge (x = y \rightharpoonup y = x)$$

$$(t \downarrow \vee s \downarrow \rightharpoonup t = s) \wedge \phi(t) \rightharpoonup \phi(s)$$

The relationship between the two symbols we have used for typing is given in the following "typing axioms":

$$\forall x.A(x : A)$$

$$x.B : A \rightharpoonup \exists z.A(x = z)$$

To illustrate the need for the second typing axiom, consider trying to prove $\forall x.A(x : \texttt{int} \rightharpoonup x : \texttt{real})$, when given only the axiom $\forall x.\texttt{int}\ (x : \texttt{real})$. We have to argue as follows: Let $x.A$ be given. Suppose $x : \texttt{int}$. Then by the

second typing axiom, x is equal to some integer z, and by the type-embedding of integers into reals, z is also a real; hence by equality axioms $x : \texttt{real}$.

There are some rules required by the logic of partial terms. The first one says that there are no undefined objects. Undefinedness is a property that terms can possess, not a property that objects being spoken about can possess. All objects exist.

$$\Gamma \Rightarrow x.A \downarrow$$

The version of **LPT** already in the literature[3] requires *strictness*: for each n-ary function symbol f and each sequence of terms t_i, and each $i \leq n$, we have

$$f(t_1, \ldots, t_n) \downarrow\!\!\rightarrow t_i \downarrow$$

We shall keep this condition in the present formulation, though one would like to weaken it, because it was essential to the completeness proof for **LPT**. In particular, it implies that the meaning of $t : A$ is "t is defined and of type A". Strictness also applies to relation symbols and equality:

$$x = y \rightarrow x \downarrow \wedge y \downarrow$$

and

$$R(x, y) \rightarrow x \downarrow \wedge y \downarrow$$

for each atomic relation symbol R (infix or not).

Note that in the typed version of **LPT** presented here, the definedness symbol does not enter into the quantifier rules.[3] The untyped version[3] has the rule

$$\frac{\Gamma \Rightarrow \phi[t/x] \qquad \Delta \Rightarrow t \downarrow}{\Gamma, \Delta \Rightarrow \exists x(\phi)}$$

in place of the rule used here:

$$\frac{\Gamma \Rightarrow \phi[t/x : A] \qquad \Delta \Rightarrow t : A}{\Gamma, \Delta \Rightarrow \exists x : A(\phi)}$$

You might wonder how the definedness symbol will ever enter into proofs then. There are at least two ways: from the existence axiom, and from non-logical axioms. For example, you might have the nonlogical axiom

$$\sqrt{x} \downarrow\!\!\rightarrow (\sqrt{x})^2 = x$$

[3]Since we did not specify the grammar rules for `type`, there may really be several "typed versions" of our theory. If we further require that there are only constant types, as is the case in *Mathpert*, then the consistency of the typed version follows from that of the untyped version in the same way as for ordinary logic, by interpreting the types as predicates. A completeness theorem would require additional work, but completeness is not used in this paper.

with the aid of which you could deduce

$$x : \texttt{real} \ \wedge x < 0 \Rightarrow \neg\sqrt{x}\downarrow$$

(Of course the proof will have to use some other non-logical axioms enabling us to prove that squares are never negative.)

As an example to clarify the rules, and demonstrate how type embedding is handled, consider the commutativity of addition in the form

$$\forall x.\texttt{real} \ \forall y.\texttt{int}(x + y = y + x)$$

This is derivable from the type embedding $\forall x.\texttt{int}(x : \texttt{real})$ and the commutativity law for reals:

$$\forall x.\texttt{real} \ \forall y.\texttt{real}(x + y = y + x)$$

by a straightforward formalization of the informal argument: Let $x.\texttt{real}$ and $y.\texttt{int}$ be given; then by the type embedding we have $x : \texttt{real}$ and $y : \texttt{real}$, hence $x + y = y + x$. Note that the second typing axiom was not used.

Remark: One referee asked about the semantics of this system, for example, what is the truth value of $\sqrt{-1} \neq 0$? The answer is, that this formula has no truth value because of the undefined constituent formula. A definition of the semantics would be superfluous, because of the reduction of **LPT** to ordinary logic in [4]. For a discussion of interpretations of **LPT** and a completeness theorem for this logic see the exercises on page 99 of Beeson[3], or see Beeson[4].

4 An axiomatization of standard analysis

In this section we first give a "standard" theory adequate for formalizing calculus. We will keep things simple by using just two basic types: integer (abbreviated `int`) and `real`. Here `integer` means negative or nonnegative integer. The basic relation symbols are $=$, $<$, $\leq$, and $\neq$. The basic functions are $x + y$, $x \cdot y$, x^y, x/y, $-x$. We also fix once and for all a set F of unary functions to be represented, for example, square root and natural log. *Mathpert* actually includes many more functions. It is important, however, that the functions defined by terms in this language form a set closed under differentiation.[4]

[4]Hence in *Mathpert*, when we wanted to include the gamma function, we also had to include the polygamma function, which is needed for the derivatives of the gamma function. The derivative of $|x|$ is $|x|/x$; we don't mean to imply that all the functions are differentiable everywhere. Specifically, we assume that for each unary function symbol f in the theory there is a term which can be proved to denote the derivative of f except at isolated points of the domain of f.

4.1 Nonlogical axioms

The nonlogical axioms of the theory of standard calculus are as follows:

(1) The schema of mathematical induction
(2) The axioms of ordered exponential fields
(3) Dedekind's schema
(4) Axioms about the specific functions in F

Dedekind's schema expresses, for each term $t(x)$ of the theory, that if $t(x)$ is defined and uniformly continuous on the interval $a \leq x \leq b$ and if $a < b \wedge t(a) < 0 \wedge t(b) > 0$, then $\exists x : \mathtt{real}(t(x) = 0 \wedge a \leq x \wedge x \leq b)$. The usual ϵ-δ definition of uniform continuity is to be used. The use of uniform instead of pointwise continuity is of no significance.

Under (2) we have included not only the ordered field axioms, but also axioms about exponentiation, including both the usual algebraic axioms and axioms relating exponentiation and order.

Under (4) we include the following axioms for square roots and logarithms

$$x \geq 0 \rightarrow (\sqrt{x})^2 = x)$$

$$x < 0 \rightarrow \neg\sqrt{x}\downarrow$$

$$x > 0 \rightarrow e^{\ln x} = x$$

$$x \leq 0 \rightarrow \neg \ln x \downarrow$$

4.2 Puiseux series

A Puiseux series is like a power series, except the exponents are multiples of some fixed fraction, rather than integers. Also, the series is allowed to begin with a finite number of terms with negative exponents. Puiseux series are defined and discussed, for example, on page 98 of Siegel[25], but the preceding brief description should be adequate to understand this paper. It is necessary that the elementary properties of Puiseux series be provable in T; for example, that the limit as x approaches zero of a Puiseux series in x is equal to the limit of its leading term. For this it suffices that T prove the fundamental properties about which rational powers of x dominate which other rational powers for small x. These results can be derived from the axioms of exponential ordered fields. The nonlogical axioms for each function included in the theory should be sufficient to justify the calculation of Puiseux series for that function where such a series exists.

4.3 Some extensions of the basic theory

This section describes three interesting extensions of the theory, which for simplicity have been omitted from the basic syntax, but to which the results of the paper can be extended.

Traditional first-order logic does not allow the formation of terms with bound variables, such as $\lim_{x \to c} f(x)$ or definite integrals. It is not difficult to formulate the syntax required for allowing such terms, and even, using the logic of partial terms, to allow such terms to be undefined sometimes. (Since freshmen are routinely required to use this syntax, it had better not be too difficult to formulate.)

The system implemented in *Mathpert* has, in addition to the types `int`, `real`, and `complex`, the type `notnumber`, whose members are denoted by constants ∞, $-\infty$, `complexinfinity`, `bounded_oscillations`, and `unbounded_oscillations`, which can be used as values of limit terms. These terms are not treated as undefined, but as defined values which are not real (or complex) numbers. They are vital for the proper treatment of limits at infinity and infinite limits. The additional type `notnumber` will not be dealt with in this paper, but it presents no difficulties; the treatment of nonstandard analysis below extends to a theory with this type.

The simple type theory given here is adequate for freshman calculus. However, the logical framework used extends easily to the type theories promulgated in a series of papers over the last couple of decades by S. Feferman. A recent formulation can be found in Feferman[16]. The results given in this paper will extend to this setting; and the algorithms discussed below may have even more interesting applications at higher types. In particular Feferman's theories allow an interesting mixture of typed and untyped variables.

5 An axiomatization of nonstandard analysis

Given a typed language such as has been described, we add a new type corresponding to each old type. In our case, we add the types of *extended integer* and *extended real*, which we abbreviate as `eint` and `ereal`. Intuitively, the nonnegative extended integers are a nonstandard model of (at least) Peano Arithmetic, and the extended reals are (at least) a non-Archimedean real-closed field including the nonstandard integers. We refer to `int` and `real` as the "standard" types and `eint` and `ereal` as the "extended" types.

The two pieces of syntax that enable us specifically to deal with nonstandard concepts are as follows: a binary relation $x \cong y$, whose intended meaning is "$x-y$ is infinitesimal (or zero)", and a unary function symbol $[\![x]\!]$ for the standard part of x.

The *standard fragment* of our theory consists of the formulae and terms built up without mentioning the *extended types* `ereal` and `eint`, and without mentioning $\cong$ and $[\![\cdot]\!]$. In particular, a *standard formula* is one containing no variables (bound or free) of type `eint` or `ereal`, and not containing $\cong$ or $[\![\cdot]\!]$.

We shall list the nonlogical axioms of our theory precisely. The first group of nonlogical axioms are the type-embedding axioms. They say that an `int` is both a `real` and an `eint`, and everything is an `ereal`.

$$x : \mathtt{int} \ \rightarrow x : \mathtt{real}$$

$$x : \mathtt{int} \ \rightarrow x : \mathtt{eint}$$

$$x : \mathtt{ereal}$$

The next axioms concern the congruence relation:

$$x \cong y \rightarrow y \cong x$$

$$x \cong y \wedge y \cong z \rightarrow x \cong z$$

$$x \cong x$$

The next axioms concern the standard part. They say that the standard part is a partial function mapping extended reals to reals, extended integers to integers, which is the identity on the standard reals and integers, and respects the congruence relation. Moreover, two numbers are congruent if they have the same standard part. (Intuitively, the standard part is a partial function because it is not required to be defined on "infinite" extended reals, but only on those reals which are infinitesimally close to a standard real.)

$$x \cong y \wedge [\![x]\!] \downarrow \rightarrow [\![x]\!] = [\![y]\!]$$

$$[\![x]\!] = [\![y]\!] \rightarrow x \cong y$$

$$x : \mathtt{ereal} \wedge [\![x]\!] \downarrow \rightarrow [\![x]\!] : \mathtt{real}$$

$$x : \mathtt{eint} \ \wedge [\![x]\!] \downarrow \rightarrow [\![x]\!] : \mathtt{int}$$

$$[\![x]\!] \downarrow \rightarrow x \cong [\![x]\!]$$

$$x : \mathtt{int} \ \rightarrow [\![x]\!] = x$$

$$x : \mathtt{real} \ \rightarrow [\![x]\!] = x$$

These axioms do not prevent congruence from being "too small". The following axiom says that the positive numbers congruent to zero are exactly those less then the reciprocals of all the standard integers. This is called the *axiom of infinitesimals*.

$$x \cong y \leftrightarrow \forall n.\mathtt{int} \ \ (n > 0 \rightarrow |x - y| < 1/n)$$

Note that the version of the logic of partial terms we use requires all predicates to be "strict". In particular $x = y \rightarrow x \downarrow$, so the previous axioms imply $[\![x]\!]$ is defined whenever the type of x is a standard type.

The last axiom is needed to make sure the extended types really are different from the standard types:

$$\exists x.\mathtt{eint} \ \forall n.\mathtt{int} \ \ (x > n)$$

Exercise: Prove $[\![x + y]\!] = [\![x]\!] + [\![y]\!]$.

5.1 Transference

The *transfer* of a standard formula A is another formula A^*, obtained by changing the type of bound variables from `int` to `eint` and from `real` to `ereal`. (If there were more types in our theory, they would come in pairs, standard and extended, and the transfer would replace each standard type by its extended counterpart.)

Given a theory T in the standard fragment of our language, we form the *nonstandard version of* T, which we call T^*, by adding to T all the formulae $A^* \leftrightarrow A$ for A a standard formula of T. Any free variables in A just play the role of fixed (standard) parameters.

5.2 Nonstandard Treatment of Limits

Let T be an ordinary mathematical theory formulated in the standard fragment of the language discussed above. We suppose that T contains the schema of mathematical induction, and at least the axioms of the theory of ordered fields (in which every positive element has a square root). This is enough to define absolute values, and hence to state the usual ϵ-δ definition of limit, and prove the basic theorems about that concept.

If T is formulated so as to explicitly include limit terms, we suppose that T also includes a defining axiom schema making the value of limit terms equal to what you get from the ϵ-δ definition. Specifically, for each term t of T, we have

$$\lim_{x \to c} t = y \leftrightarrow \forall \epsilon.\texttt{real}\ \exists \delta.\texttt{real}\ \forall x.\texttt{real}\ \ (0 < |x - c| < \delta \rightarrow |t - y| < \epsilon)$$

In this formula the variable y has type `real` , not `ereal` . Since we have not formulated T with explicit limit terms, the meaning of this formula is that the left-hand side is an abbreviation (at the meta-level) for the right-hand side.

We will now give the nonstandard characterization of limits implemented in *Mathpert* its precise logical statement. This is, of course, not new: it is just the usual nonstandard characterization of limits.[23] The new points about it made here are (1) this characterization can be formalized in the theory T^*, and (2) T^* can be, and has been, computationally implemented.

Remark: It should be noted that our theory is much weaker than the usual theories of nonstandard analysis, in that there are no variables for real-valued functions, or even sequences of reals. All functions are given by explicit terms of the theory. This is not important to the methods illustrated here, which would apply to any natural mathematical theory. It does mean that there is something to be checked, as the usual proof involves using the transference principle in the form that says sequences defined on the standard integers extend to sequences defined on the extended integers.

Theorem. *If T contains the axioms of ordered fields and the schema of mathematical induction, then $\lim_{x \to c} f(x) = y$ is equivalent in T^* to $\forall \alpha.\texttt{ereal}(\alpha \cong c \wedge \alpha \neq c \rightarrow f(\alpha) \cong y)$.*

Proof: Suppose $\lim_{x \to c} f(x) = y$. Then let $\alpha \cong c$, but $\alpha \neq c$. We must show $f(\alpha) \cong y$. We have by the definition of limit

$$\forall \epsilon.\texttt{real}\ \exists \delta.\texttt{real}\ \forall x.\texttt{real}\ \ (0 < |x - c| < \delta \rightarrow |f(x) - y| < \epsilon)$$

Let n be a standard integer, and take $\epsilon = 1/n$. We thus have a standard real δ such that

$$\forall x.\texttt{real}\ \ (0 < |x - c| < \delta \rightarrow |f(x) - y| < 1/n).$$

Note that transference is applicable since y is standard. By transference, we have

$$\forall x.\texttt{ereal}\ \ (0 < |x - c| < \delta \rightarrow |f(x) - y| < \epsilon)$$

Take $x = \alpha$. We have $0 < |\alpha - c|$. Hence

$$|\alpha - c| < \delta \rightarrow |f(\alpha) - y| < 1/n.$$

But since $\alpha \cong c$, we have $|\alpha - c| < \delta$ by the axiom of infinitesimals. Therefore $|f(\alpha) - y| < 1/n$. Hence, by the axiom of infinitesimals, $f(\alpha) \cong y$.

Now for the other direction. We proceed by contradiction.[5] Suppose $x \cong c \rightarrow f(x) \cong y$, and suppose that $f(x)$ does not approach y as $x \rightarrow c$. Then

$$\exists \epsilon.\texttt{real}\ \ (\epsilon > 0 \wedge \forall k.\texttt{int}\ \exists x.\texttt{real}\ \ (0 < |x - c| < 1/k \wedge |f(x) - y| > \epsilon)$$

Fix such a (standard) ϵ. Then applying transference,

$$\forall k.\texttt{eint}\ \exists x.\texttt{ereal}\ \ (|x - c| < 1/k \wedge |f(x) - y| > \epsilon)$$

Choose an infinite integer k, and let x be such that $|x - c| < 1/k \wedge |f(x) - y| > \epsilon$. Then by the axiom of infinitesimals, $x \cong c$, so we have $f(x) \cong y$, which contradicts $|f(x) - y| > \epsilon$.

5.3 Converting nonstandard proofs to standard proofs

What we need to know is that if a standard theorem is proved using nonstandard methods it is true. In order to quote the well-known fact that every theorem with a nonstandard proof also has a standard proof, we need to connect our theory T^* with some theory in the literature, for which that conservation result has been proved.[6] There are several axiomatizations of nonstandard analysis in the literature. Robinson himself began with a type-theoretic version and later gave a set-theoretic version[24]. Kreisel[20] gave one as a preliminary to his theorem. More recently, E. Nelson developed his theory **IST**.[21, 22] The most

[5] Classical logic seems necessary here, because the congruence formulation of limits does not contain the computational information about how to compute δ as a function of ϵ.

[6] Such a theorem was first proved by Kreisel.[20] The original proof is highly model-theoretic in nature, and gives no hint of how to transform a nonstandard proof explicitly into a "corresponding" standard proof.

beautiful proof of such a conservation result is found in Nelson's axiomatization, so we will refer to that version.

Nelson's theory is formed from a theory of standard analysis by adding a new unary predicate *standard* (and three axioms, which we do not discuss here). The main difference between T^* and **IST** is not in the nonstandard part at all, but in the fact that **IST** is based on set theory, while T^* is a typed theory. It is well known, however, how to interpret types in set theory, so that each formula A of T has an interpretation in Zermelo set theory Z. Our first task is to extend that interpretation to an interpretation of T^* in **IST**.

First, I will show how to define the fundamental concepts $x \cong y$ and $y = [\![x]\!]$ of T^* in terms of **IST**. Nelson defines "infinitesimal" to mean positive but less than all standard reals, and then he defines $x \cong y$ if $x - y$ is infinitesimal. One can then define the standard part of x as that (necessarily unique) standard number y such that $x \cong y$, if such a y exists. The fundamental types `ereal`and `real`, `eint` and `int`, also have natural interpretations in **IST**, enabling us to translate atomic formulae of the form $t : A$. Types are simply interpreted as sets. The easiest way to handle the definedness symbol is to choose some otherwise irrelevant set **junk** and make it the value of all undefined terms. Of course, **junk** cannot be a (standard or nonstandard) real number. Then the interpretation of $x \downarrow$ is just $x \neq$ **junk**. These rules determine a natural translation of T^* into **IST**.

Nelson proved the conservation result that if A is an internal formula provable in **IST**, then A has a proof in the standard set theory on which **IST** is based.[7] Applying Nelson's theorem, we obtain the desired result:

Theorem. *Let A be a standard theorem of T^*. Then the translation of A into* **IST** *has a standard proof in classical Zermelo set theory.*

Proof: Let A be a standard theorem of T^*. Then the translation of A into **IST** has a proof in **IST**. By Nelson's theorem, it has a proof in Zermelo set theory.

6 An infinitesimal-elimination algorithm

This section contains a precise definition of the infinitesimal-elimination algorithm used by *Mathpert*. The purpose of the algorithm is to eliminate a nonstandard variable which was introduced when computation entered a limit term, so to each nonstandard variable α.`ereal` there corresponds an assumption $\alpha \cong x$, where x is a standard variable. If the limit is one-sided, there is also an assumption $x < \alpha$ (for limits from the right) or $\alpha < x$ (for limits from the left); otherwise there is an assumption $x \neq \alpha$. Thus each nonstandard variable has

[7] Moreover, the proof is fully constructive, unlike the original proof of Kreisel's theorem. Nelson gives an algorithm for converting nonstandard proofs into standard proofs. The algorithm is reminiscent of the Gödel *Dialectica* interpretation.

a "domain inequality" associated to it. As a matter of notation, we will use α for a (the) nonstandard variable associated to a limit as x approaches c, and we will use h for the corresponding infinitesimal quantity $\alpha - c$. Strictly speaking "infinitesimal elimination" is only accurate when $c = 0$; generally we are eliminating a nonstandard variable α.

The algorithm to be defined here eliminates only *one* infinitesimal. Formulas involving nested (iterated) limits can therefore not be treated by this method. Whether this limitation can be removed by more careful bookkeeping, we do not know. The usefulness of the algorithm depends on computational code for computing limits, which in turn relies on Puiseux series computations, and the computation of nested limits is notoriously more difficult than simple limits. Probably the difficulty of computation is the main obstacle to extending the method, rather than any inherent logical difficulties.

6.1 Introduction to the Algorithm

The method can be partly described as a small collection of rewrite rules, for reducing logical formulae to simpler logical formulae. One hopes that variables of type `ereal` will be eliminated by the rewriting, so that a logical formula generated inside the scope of a bound limit variable (and hence involving an infinitesimal) will be reduced to a standard formula not involving the bound variable. The rules involve the decomposition of an extended real into its "standard part" and its "nonstandard part", analogous to the decomposition of a complex number into real and imaginary part. We have already introduced the notation $[\![x]\!]$ for the standard part; let us define $\mathbf{NS}(x) = x - [\![x]\!]$. Here are the two most important rules. (These are logical equivalences, but they will be used as rewrite rules from left to right.)

$$x < y \rightarrow [\![x]\!] < [\![y]\!] \vee ([\![x]\!] = [\![y]\!] \wedge \mathbf{NS}(x) < \mathbf{NS}(y)$$
$$x \leq y \rightarrow [\![x]\!] < [\![y]\!] \vee ([\![x]\!] = [\![y]\!] \wedge \mathbf{NS}(x) \leq \mathbf{NS}(y)$$

These rules do not yet constitute the definition of an algorithm. For one thing, we haven't said how to apply the rules, or how to compute the standard and nonstandard parts involved. But more importantly, these rules alone won't do the job. Consider the example of computing the derivative of $\sqrt{x}$. Under the limit, we have to simplify $\sqrt{x+h}\downarrow$. This comes to $0 \leq x+h$. The standard part of $x+h$ is x and the nonstandard part is h, so the rules above simplify this to $0 < x \vee (0 = x \wedge 0 \leq h)$. What *Mathpert* does when simplifying $0 \leq h$ is to search for the domain inequality in the list of assumptions (that is, in the antecedent of the current sequent). If this inequality is $h \neq 0$, showing that the limit it came from was a two-sided limit, then *Mathpert* simplifies $0 \leq h$ to `false`. This procedure cannot be described as a rewrite rule, even a rewrite rule with side condition. It does have the desired result, at least in this example, because then $0 < x \vee (x = 0 \wedge 0 \leq h)$ simplifies to $0 < x$. The nonstandard variable h has been eliminated correctly. Note that if we had been calculating the one-sided

limit from the right, $0 \leq h$ would have simplified to `true` by the ordinary rules used by *Mathpert*, using the domain inequality $0 < h$, so we would have come out with $0 < x \vee 0 = x$, which simplifies to $0 \leq x$.

The third rule used in the infinitesimal elimination algorithm is this: If the domain inequality for the nonstandard variable α is $\alpha \neq c$, then $\alpha < c$, $\alpha \leq c$, $\alpha > c$, and $\alpha \geq c$ all rewrite to `false`. This rule may appear problematic, since it certainly shows that the "rewrite" in question does not preserve a direct logical equivalence. However, in the informal "interval semantics" motivating the algorithm, the formula $\phi(\alpha)$ means that ϕ is true in a neighborhood of the limit point c, and interpreted this way, it makes sense to rewrite $\alpha > c$ as `false`, etc.

The standard part of x, denoted $[\![x]\!]$ above, is abstractly defined, so the algorithm for infinitesimal elimination can't be considered defined by the above three rules until we give methods for computing the standard and nonstandard part. A computational procedure **stdpart** (implemented in *Mathpert*) attempts to compute $[\![t]\!]$ for an explicitly given term t. This function computes the standard part of a variable x of type `ereal` or `eint` by looking in the list of current assumptions for an assumption $x \cong t$, and returning **stdpart**(t). The standard part of an expression not involving any variables of type `ereal` or `eint` is just the input expression. On compound expressions containing nonstandard variables, **stdpart** uses leading terms of Puiseux series in a way described in more detail at the end of this section. In applying the two rewrite rules listed above, the function **stdpart**(x) is used to implement $[\![x]\!]$.

We assume that **stdpart** has been correctly programmed, in the sense that it satisfies the following properties:

(1) If **stdpart** returns successfully an answer q on input t then q is a standard term and T^* proves $\Gamma, I \Rightarrow q \cong t$, where Γ is the list of assumptions current when **stdpart** is called and I is the list of assumptions about the nonstandard variable(s) involved in t.

(2) **stdpart** will always terminate, either in success or failure. We emphasize that **stdpart** will no doubt fail to compute a correct answer on a sufficiently complicated input, but it will not go into infinite regress. It will succeed or give up.

(3) If **stdpart** terminates successfully on input t, then T proves

$$\lim_{x \to c} t[x/\alpha] = \mathbf{stdpart}(t)$$

Similarly, there is a computational procedure **nonstdpart** which attempts to compute (an approximation to) the non-standard part of a term. Like **stdpart**, it will always terminate, but may fail to compute the answer. When it does terminate successfully, we have **nonstdpart**$(t) \cong \mathbf{NS}(t)$, provably in T^*. Of course this property would be satisfied by taking **nonstdpart**$(t) = 0$, but such a weak algorithm for **nonstdpart**(t) would not be interesting. The essential

property of **nonstdpart**(t) is that, when it terminates successfully, it correctly determines the sign of $t - \mathbf{stdpart(t)}$ in a neighborhood of the limit point.[8]

The algorithm employed also makes use of *simplification*. Simplification is applied both to mathematical expressions and to logical formulae. You may think of simplification as something like normalizing a term using rewrite rules, but some of the rules used are more general than rewrite rules. **infer** operates by simplifying its argument, hopefully to `true`, while **refute** tries to simplify to `false`.

The exact nature of **infer** and **refute** are not germane: we need just a few properties of them. Namely,

(1) We assume that **infer** and **refute** always terminate (whether successful or not), and that if **infer**(ϕ) succeeds, then ϕ is provable. More precisely, T^* proves $\Gamma, \Delta \Rightarrow \phi$, where Γ is the list of assumptions current when **infer** is called, and Δ is the list (often empty, but not always) of new assumptions generated during the execution of **infer**. As a matter of notation, we will always use Δ for a list of assumptions generated during computation.

(2) We also assume that if t simplifies to q (for terms t and q) then T^* proves $\Gamma \Rightarrow t = q$. Similarly, if ϕ and ψ are propositions, and ϕ simplifies to ψ, then T^* proves $\Gamma \Rightarrow \phi \leftrightarrow \psi$. In practice this hypothesis is satisfied, because the rules of simplification come directly from the mathematical axioms of T, together with simple rules of propositional calculus.

(3) We assume that **infer** is able to prove true inequalities of the form $ah^u > 0$ under assumptions of one of the forms $h > 0$, $h \neq 0$, $h < 0$, where u is a rational number.

We thus have, apparently, several algorithms to consider: the infinitesimal-elimination algorithm **val**; the simplification algorithm; **infer**, and **refute**. In reality these algorithms are not all separate: the infinitesimal-elimination algorithm and the ordinary mathematical and logical simplifications (which satisfy (1) above) are both incorporated into a single algorithm **val**, and **infer** and **refute** both work by calling **val**. **infer**(ϕ) succeeds if **val**(ϕ) = `true`, and **refute**(ϕ) succeeds if **val**(ϕ) = `false`. In addition there are the algorithms **stdpart** and **nonstdpart**, which involve a mutual recursion with **val**.

To avoid confusion: "simplification" is not defined by a mutual recursion with **val**. The word "simplification" in this paper refers to logical and mathematical steps which have nothing to do with infinitesimal elimination, and which preserve ordinary mathematical and logical equivalence. All these steps are mathematically simple and represent equivalences that can be stated and proved in T^*. However, it is allowed that there be simplification steps involving the nonstandard variable. For example, if $c = 0$ so that the nonstandard variable α represents an infinitesimal, we might allow simplifying $\sin\alpha$ to α. No

[8]This does not imply that only functions which do not oscillate near a limit point can be correctly handled. Some such functions can be handled correctly, but not by the part of the algorithm calling **nonstdpart**.

assumption about such simplifications is made for purposes of our correctness proof except the assumption of correctness, property (2) above. In particular the correctness proof does not require a commitment to whether such simplifications are actually used or not. It should be noted that the simplification methods may use the current assumptions Γ, for example by simplifying ϕ to `true` if ϕ occurs in the list Γ. For complete precision we should say "t simplifies to q with current assumptions Γ" instead of just "t simplifies to q", but Γ will always be clear from the context.

The correctness proof for our infinitesimal-elimination algorithm does require more detailed information about how **val** works on inequalities $ah^u > 0$. Specifically, (3) is fine when a and u are specific numbers, but in general it should be strengthened to the following:

(4) **val** will terminate on $ah^u > 0$ when one of the assumptions $h < 0$, $0 < h$, or $h \neq 0$ is in (or immediate from[9]) Γ, with the (correct) result `true` or `false`, possibly generating additional assumptions in the process, if additional assumptions are required to decide the signs of a and u and whether u is odd or not (see below). Here a and u are arbitrary standard expressions.[10]

Although (4) is listed as an assumption here, I will explain how the actual algorithms employed by *Mathpert* are constructed to satisfy this assumption. These details are irrelevant to the infinitesimal elimination algorithm except that they justify (4), which is required for the correctness proof.

The details in question concern the situation in which the exponent u and/or the coefficient a are not specific rational numbers, but expressions containing other variables. Observe that two things are relevant to determining the sign of ah^u under one of the assumptions $h > 0$, $h < 0$, or $h \neq 0$. Namely: Is u an odd fractional exponent (that is, an odd integer or a quotient with even denominator and odd numerator)? What is the sign of a? Observe further that if u is odd and the assumption is $h \neq 0$, then the sign of a is irrelevant, because ah^u will take both signs, while if u is even, then the sign of a determines the sign of ah^u. Therefore *Mathpert* first applies the **infer**, **refute**, **assume** method to the question whether u is odd. If this can neither be inferred nor refuted, it is assumed, and no assumption need be made about the sign of a. If it can be refuted, however, then the **infer**, **refute**, **assume** method is applied to $a > 0$. In any event, then, **val** will terminate on $ah^u > 0$, establishing (4).

[9] The provision about "immediate from" is necessary because if the limit point is not zero, then the domain inequality which is actually in Γ has the form, for example, $\alpha \neq c$ for a nonstandard variable α, and $h = \alpha - c$; so $h \neq 0$ is not literally in Γ, but we still need **val** to terminate.

[10] If the expression a is sufficiently complicated that **infer** and **refute** cannot determine its sign, but its sign is nevertheless actually determined by the assumptions in Δ, an inconsistent assumption may be added, in which case the conclusion of the correctness theorem will be vacuously true.

6.2 Precise Definition of the Algorithm

The algorithm in question, here denoted **val**, takes three inputs: A list Γ of (standard) assumptions, two assumptions I about the nonstandard variable α (namely, $\alpha \cong c$ and a domain inequality), and a logical formula ϕ (which may contain α, but no other nonstandard variables or symbols) to be simplified. The procedure **val** will always terminate, either with success or failure. In case it succeeds, it produces two outputs: the simplified form $\mathbf{val}(\phi)$ of ϕ, which for full precision must be written as $\mathbf{val}(\Gamma, I; \phi)$, and a (possibly empty) list of new assumptions Δ. To be precise, we would have to write something like $\mathbf{val}(\Gamma, I; \phi)$.`prop` for the simplified proposition, and $\mathbf{val}(\Gamma, I; \phi)$.`assumption` for Δ, to indicate that the return value is a pair consisting of a proposition and a list of assumptions. This would be hopelessly unreadable, so instead we write $\mathbf{val}(\phi)$ for the simplified proposition and Δ for the list of generated assumptions.

Moreover, we have to distinguish two kinds of success: when only rules that preserve logical equivalence have been used, we say **val** succeeds "positively". Otherwise success is "negative". The exact definition of the algorithm must indicate at each return whether the return is with failure, with positive success, or negative success. The words "positive" and "negative" have no intrinsic signifance here; we might as well have used "red" and "blue".

The definition of the algorithm **val** follows:

Step 1. When **val** operates on any input formula ϕ, it may first perform some ordinary logical simplifications, or mathematical simplifications on the terms in the formula. The only relevant property of these simplifications is that they preserve logical equivalence under the assumptions Γ, Δ, where Δ is a (possibly empty) list of new assumptions generated by the simplifications performed. If some simplication is performed, with result ψ different from ϕ, return $\mathbf{val}(\psi)$. If no simplifications can be applied, and ϕ contains only standard variables, then **val** terminates with positive success, returning ϕ unchanged. If no simplifications can be applied and ϕ contains the nonstandard variable, go to step 2 if ϕ is a disjunction or a conjunction. If it is an inequality or equality in which both sides contain α, go to step 3. If it is an equality or inequality in which exactly one side contains α, but that side is not exactly α, go to step 4. If one side is exactly α and the other side is c, go to step 5. If one side is α and the other is not c, go to step 6.

Remark: On input containing only standard variables, **val** does nothing but ordinary simplification. We assume that **val** terminates on input containing only standard variables. (This would be guaranteed, for example, if all the simplifications shortened the formula. In practice some simplifications lengthen the formula, but proving the termination of ordinary simplification procedures is not what this paper is about.) The simplifications to be applied include the law rewriting $A \rightarrow B$ as $B \vee \neg A$, and the laws pushing negation inwards, the law that "flattens" disjunctions whose arguments include a disjunction (we allow disjunctions of many arguments; for example $A \vee (B \vee C)$ flattens to $A \vee B \vee C$,

which means $\vee(A, B, C)$), the law that flattens conjunctions, and the laws that rewrite negations of inequalities as other inequalities.[11] Our base system has no atomic formulae other than inequalities and equalities; in general (if the system is enlarged) other atomic formulae might appear negated, but **val** will not perform anything but pure mathematical simplification on such formulae. In subsequent steps of the algorithm, it may therefore be assumed that implication and negation do not occur.

Step 2. On disjunctions and conjunctions, if **val** does not perform some immediate simplifications, it is called on each argument of the input in turn:

$$\mathbf{val}(P_1 \wedge \ldots \wedge P_n) \leftrightarrow \mathbf{val}(P_1) \wedge \ldots \wedge \mathbf{val}(P_n)$$

$$\mathbf{val}(P_1 \vee \ldots \vee P_n) \leftrightarrow \mathbf{val}(P_1) \vee \ldots \vee \mathbf{val}(P_n)$$

These formulae are written with $\leftrightarrow$ instead of $=$, because the actual definition of the algorithm calls for computing the right-hand side, and then performing some simplifications, such as dropping duplicate conjuncts or disjuncts and combining certain inequalities. The exact nature of these simplifications is not important to the correctness proof as long as they preserve (provable) equivalence. If any of the recursive calls $\mathbf{val}(P_k)$ fails, then return with failure. In the case of conjunction, if all the recursive calls return with positive success, then return return with positive success, else return with negative success.

Disjunction is treated differently. If all the recursive calls return with positive success, return with positive success. If exactly one returns with negative success, return with negative success. If one or more fails, return unsuccessfully. If more than one returns with negative success, and one of those returning with negative success is a conjunction, then use the distributive law on the original input to move the disjunction in, and call **val** recursively on the result. This is referred to below as the "forced distribution" clause.

Since disjunctions have been flattened and negations have been pushed in, the only other possibility is two or more negative successes on atomic formulae. In this case, we return unsuccessfully. This is referred to below as the "forced failure clause". (The purpose of this clause should be made clear in Remark 1 following Theorem 3 below.)

The recursive calls to **val** on the right can generate assumptions Δ. The question arises of how these are handled. The simplest thing is just to take their union and return that as the Δ from the call on the left. In actuality, though, the calls on the right are not made in parallel, but sequentially, and the assumptions Δ generated by the first call will be part of the Γ for the second call, and so on. In other words, only one list of current assumptions is maintained, and any new assumptions generated are added to that list.

[11] These laws include rewriting $\neg(a < b)$ as $b \leq a$ and several similar laws. We treat $a \neq b$ as an inequality symbol, not a negation.

Step 3. If the input ϕ is an inequality $r < s$, and both sides contain α, return **val**$(0 < s - r)$.[12]

The part of **val** in which nonstandard analysis is used comes into play only when **val** is called on an equality or inequality containing a nonstandard variable. Recall that we have assumed there is only one nonstandard variable, call it α, in the input. Inequalities could be of the form $r < s$, $r \leq s$, or $r \neq s$, and input $r = s$ is also allowed. For simplicity we consider here only $r < s$; other forms are treated similarly.

Step 4. If the input is $r < s$ and exactly one side contains α, make various purely mathematical simplifications (about which we need to know only that they preserve mathematical equivalence), and then solve the inequality for the nonstandard variable α if possible. (For example, $2\alpha < 1$ will become $\alpha < 1/2$.)[13] If the result of these computations is ψ, return **val**(ψ). If the inequality cannot be solved for α, go to step 6.

Step 5. If the input is an inequality containing a single nonstandard variable α, and the domain inequality for α is $\alpha \neq c$ (indicating that α arose from a two-sided limit as α approaches c), and the input inequality has one of the forms $\alpha < c$, $\alpha > c$, $\alpha \geq c$, $\alpha \leq c$, $\alpha = c$, return `false`. If the input is $\alpha \neq c$, return `true`.[14] In these cases, **val** returns with negative success. (All negative-success returns originate here, and are passed upwards by returns from recursive calls.) If neither side is c, go to step 6.

Step 6. (We get here either from step 5 with an inequality solved for α whose other side is not c, or from step 4 with an inequality that could not be solved for α.) If the input ϕ is $r < s$ with r not zero, set ϕ to the simplified form of $0 < s - r$. Once the input has the form $0 < s$, compute $S =$ **stdpart**(s). We have assumed that this computation will always terminate, either in success or failure. If it fails, then **val**$(0 < s)$ terminates unsuccessfully, returning $0 < s$. This is the only step in the algorithm (besides the forced failure clause in step 2) that can cause an unsuccessful return of **val**. In this case, the nonstandard variable will not have been eliminated.

If S is successfully computed, we try to determine its sign. We assume that the computation of the standard part has already simplified the result so that there is no point in trying to infer $S = 0$; if that would succeed, S is already zero. In case S is zero, we do not proceed immediately to compute the Puiseux series using **nonstdpart**, because in some cases of interest, the result would

[12] As a practical matter, simplification does better at proving quantities are positive than at proving one quantity exceeds another, because it is easier to find cancellations and certain other simplifications that way.

[13] This step will make further assumptions (add to the list Δ) if need be. For example, $q\alpha < 1$ where q is a variable may reduce to $\alpha < 1/q$ generating the assumption $0 < q$, if this can be neither inferred nor refuted.

[14] In the case of a one-sided limit from the right, we would return `true` on $c < \alpha$, $c \leq \alpha$, $\alpha = c$, and `false` on $\alpha < c$ and $\alpha \leq c$.

be failure. We first call **infer** $0 < s$.[15] If this succeeds, we return `true`, with positive success. If not, go to step 9.

If S is not zero, call **infer** on the inequality $0 < S$. If it succeeds, then return `true`, with positive success.[16] If not, go to step 7.

Step 7. Call **infer**$(S < 0)$. If this succeeds, then return `false`, with positive success; else go to step 8.

Step 8. Call **infer** $0 \neq S$. (It might be possible to infer this without being able to determine the sign of S, for example if $0 \neq S$ is a current assumption.) In that case we add $0 < S$ to the list Δ of new assumptions and return `true`. If the call to **infer** does not succeed, return

$$0 < S \vee (S = 0 \wedge \textbf{val}(0 < s - S).$$

If the recursive call to **val** fails, the return is unsuccessful; otherwise the success is positive or negative, the same as is returned from the recursive call. [1]

Step 9. Call **nonstdpart** on s. This results in attempting to compute the leading term of the Puiseux series for s. If this computation does not succeed, **val**$(0 < s)$ terminates with failure. Now suppose that it succeeds. The series in question is, by definition of the algorithm **nonstdpart**, computed in powers of $h = \alpha - c$, where c is the standard part of α.[2] It is important to note that computation of the series must include simplifying the coefficients, because there can be cancellations. Consider, for example, $h - \sin h$, in which the linear terms cancel out, and the actual leading term is $h^3/6$. The leading term can have a symbolic power; the exponent does not have to be a specific rational number. Since simplification is not (and cannot be) perfect, however, the leading term as calculated might still really be zero. We therefore call it the apparent leading term, as the true leading term would be farther out in the series.[3]

[15] This may succeed, even in case s has no Puiseux series. For example, *Mathpert* can infer that $e^{-1/x}$ is positive.

[16] This part of the algorithm enables us, for example, to simplify $h < 1$ to `true` inside a limit as $h \to 0$. Thus we will correctly generate no assumption when cancelling $h - 1$ inside such a limit.

[1] In the recursive call to **val**, the standard part of $s - S$ will come out zero in step 6, and we will go to step 9. It may help the reader to consider the example of Section 6.1, in which we have the input $0 \leq x + h$. In this case the standard part is x, and both the attempt to infer $0 < x$ and the attempt to infer $x < 0$ will fail, so we will return $0 < x \vee (x = 0 \wedge \textbf{val}(0 < h)$. This will turn out to be $0 < x \vee (x = 0 \wedge$ `false`$)$, which simplifies to $0 < x$.

[2] It is necessary **nonstdpart** must "know" about which point to compute the Puiseux series. There seems to be no way to ensure this in general except to make the restriction (which we have made) that there is only one nonstandard variable, arising from a limit term and hence uniquely associated to a point c about which we are to compute series.

[3] An example without extra variables in which the apparent leading term is actually zero would be complicated, because `simplify` will correctly handle simple examples like $h - \sin h$, but such examples surely do exist. The situation of an apparent leading term whose sign cannot be determined will of course occur if extra variables are present whose signs are not determined by the current assumptions, for example $0 < ah$ where a is a variable and no assumption is made about the sign of a.

Let us call the apparent leading term of the Puiseux series ah^u. Then return $\mathbf{val}(0 < ah^u)$.

That completes the definition of **val**. Note that the steps in the computation by **val** are of two kinds: (i) ordinary mathematical and logical reductions that preserve logical equivalence, and (ii) infinitesimal-elimination steps. The latter have been specified with complete precision. The former have not been fully specified; instead specific assumptions have been listed which they must satisfy in order to make the correctness proof work. This means that more simplifications of kind (i) can be added to the algorithm without having to rework the correctness proof.

In particular, we have discussed a theory with nonlogical axioms only for natural log and square root functions, thereby avoiding the need to discuss the formalization of other functions. Our theorem has been formulated in such a way that it immediately extends to a theory including sine, as soon as the diligent reader defines that function by suitable non-logical axioms in T (for example $y'' = -y$ and $y(0) = 0$ and $y'(0) = 1$) and proves the Taylor series expansion of the sine function at an arbitrary center on the basis of those axioms. To cover the theory actually used in *Mathpert*, this would have to be done for the trigonometric, inverse trigonometric, hyperbolic, inverse hyperbolic, Bessel and other special functions.

7 Examples

In this section we trace the algorithm on two illustrative examples. It is not logically necessary to read this section to follow the rest of the paper. We continue the numbering of examples from section 2.

Example 8: Calculate the derivative of $\sqrt{f(x)}$, using the limit definition of derivative, where f is a polynomial. We will have to compute $\mathbf{val}(0 \leq f(x+h))$ for $h \cong 0$. Step 1 will simplify f, if it is not already simplified, and pass the inequality to Step 4. Here some mathematical simplifications will be made; in practice *Mathpert* incorporates nontrivial algorithms for polynomial inequalities, but they are irrelevant here. Let us assume that the resulting inequality can still not be solved exactly, e.g. if f is irreducible with several real roots. For illustrative purposes let us neglect the fact that only rather complicated polynomial inequalities would really remain unsolved, so that we can think about simple examples like $f(x) = x^2 - 1$ or $x^3 + x$. The inequality will then be passed to Step 6, which requires computing the standard part of $f(x+h)$. The standard part of $f(x+h)$ is $f(x)$. If **infer** can show that $0 < f(x)$, that will end the computation; for example if $f(x) = x^2 + 1$. Assuming the efforts in Steps 7 and 8 to determine the sign of $f(x)$ also fail (as they will in practice for a polynomial that has passed Step 4), we come to Step 9, in which **nonstdpart** will be called. It will compute the apparent leading term of the Puiseux series in h of $f(x+h)$. The first candidate for that will be $hf'(x)$. If **infer** can show

that $0 < f'(x)$, as in the example $x^3 + x$, **val** will return `false` on $0 < hf'(x)$, and the top-level call to **val** will compute

$$0 < f(x) \vee (f(x) = 0 \wedge \mathbf{val}(0 \leq hf'(x))$$

which in the example $x^3 + x$ would simplify (using $x^3 + x = x(x^2 + 1)$) to

$$0 < x \vee \mathbf{val}(0 < h)$$

and finally to $0 < x$. In the example $f(x) = x^2 - 1$, **infer** will not be able to determine the sign of $f'(x)$. The computation will come to

$$0 < x^2 - 1 \vee \mathbf{val}(0 < 2hx)$$

and **val** will have to solve the inequality $0 < 2hx$ for h. To do this without making assumptions about the sign of x, *Mathpert* returns $(0 < x \wedge 0 < h) \vee (x < 0 \wedge h < 0)$. When **val** is called recursively on this input, the computation hits the forced distribution clause in step 2. After distribution we get

$$(0 < x \vee x < 0) \wedge (0 < h \vee x < 0) \wedge (0 < x \wedge h < 0) \wedge (0 < h \vee h < 0)$$

The last conjunct simplifies to `true`, in view of $h \neq 0$, in *Mathpert*, although this would not be guaranteed by the general assumptions we have made about simplification. With or without this simplification, **val** applied to the displayed formula returns `false` with negative success. Therefore the original call to **val** returns $0 < x^2 - 1$. This is the correct condition for the differentiability of $\sqrt{x^2 - 1}$.

Example 9: Consider the one-sided limit

$$\lim_{x \to 3+} \sqrt{x^2 - 9}$$

In analyzing the domain of this expression, *Mathpert* will generate the expression $\alpha^2 - 9 \geq 0$, where α is a nonstandard variable subject to the assumptions $\alpha \cong 3$ and $\alpha > 3$. At Step 1, the inequality $\alpha^2 - 9 \geq 0$ will simplify to the disjunction $3 \leq \alpha \vee \alpha \leq -3$. This will reach Step 2, which will call **val** separately on the two inequalities $3 \leq \alpha$ and $\alpha \leq -3$. Let us consider $\alpha \leq -3$ first. This will be sent to Step 6, where it will be put in the form $0 \leq -3 - \alpha$. Since the standard part of α is 3, the standard part of the right side is -6, and so we try to infer $0 < -6$, which fails, and we go to Step 7. In Step 7, we try to infer $-6 < 0$, which succeeds, so **val** returns `false` with positive success.

Now consider the other inequality, $3 \leq \alpha$. This reaches Step 6 and is put in the form $0 \leq \alpha - 3$. Since the standard part of α is 3, we get zero for the standard part, so the attempted inferences in Steps 6, 7, and 8 all fail, and we arrive at Step 9. This calls **nonstdpart** on $\alpha - 3$. This sets $h = \alpha - 3$ and asks for the leading term of the Pusieux series in h of $\alpha - 3$. That leading

term is of course just h itself; that is, $a = u = 1$. Now $\mathbf{val}(h > 0)$ is called. The computation of $\mathbf{val}(h > 0)$ reaches Step 5, and returns `true` with negative success, since the domain inequality is $3 < \alpha$. If this had been a two-sided limit, or a one-sided limit from the left, the return value would have been `false` with negative success.

Returning now to the original call to **val** on the disjunction, we have for the results on the disjuncts, `false` with positive success and `true` with negative success. The rules in Step 2 cause us to return `true` with negative success as the final answer. The meaning of this result is that the limitand is defined in the vicinity of the limit point. If the limit had been two-sided, the return value would have been `false`, reflecting the fact that the limitand is defined only on one side of the limit point.

8 A correctness proof

Our aim here is to formulate and prove a theorem to the effect that the procedure described above is correct. This theorem can be summarized as follows: Although **val** does not preserve strict logical equivalence, it preserves equivalence in "neighborhood semantics". Specifically, a formula involving α is true in "neighborhood semantics" if it is true for all real x in a neighborhood of the limit point c. The neighborhood in question must be a punctured neighborhood and must be one-sided if the original limit was one-sided.

To formulate such a theorem precisely, let $\mathtt{nbhd}(\phi, c)$ be the (standard) formula expressing that ϕ is true in some punctured neighborhood of c as a function of h, namely (in the case of a two-sided limit)

$$\exists a.\mathtt{real}\ \exists b.\mathtt{real}(a < c \wedge c < b \wedge \forall x.\mathtt{real}(a < x \wedge x < b \wedge x \neq c \rightarrow \phi[x/\alpha]))$$

We will not treat the case of one-sided limits explicitly, as the details are quite similar to the treatment of two-sided limits. Since c will be clear from the context we will often write simply $\mathtt{nbhd}(\phi)$ instead of $\mathtt{nbhd}(\phi, c)$.

This semantics depends on each nonstandard variable α being clearly associated with a limit point c and either a one-sided limit or a two-sided limit. For this reason, our elimination procedure assumes that there is only one nonstandard variable and it is so associated. This will be adequate for symbolic computation not involving nested limit terms.

Theorem. *(Elimination of an infinitesimal) Let Γ be a list of standard assumptions, and let I be the list of two assumptions, $\alpha : \mathtt{ereal} \cong c : \mathtt{real}$, and a "domain inequality", either $\alpha < c$, $\alpha > c$, or $\alpha \neq c$. Let ϕ be a quantifier-free formula, standard except that it may contain α. Then $\mathbf{val}(\phi) = \mathbf{val}(\Gamma, I; \phi)$ will terminate. If $\mathbf{val}(\phi)$ terminates with success, let Δ be the list of any additional assumptions generated during the computation of $\mathbf{val}(\phi)$. Then:*

(i) Δ and $\mathbf{val}(\phi)$ are both standard.

(ii) T^* proves $\Delta, \Gamma \Rightarrow \texttt{nbhd}(\phi) \leftrightarrow \textbf{val}(\phi)$ where nbhd(given precisely above) expresses that ϕ is true in a punctured neighborhood of c.

(iii) $\Delta, \Gamma \Rightarrow \texttt{nbhd}(\phi) \leftrightarrow \textbf{val}(\phi)$ is provable in classical set theory, and hence true.

Remark 1: Consider the example input $\alpha \leq c \vee c \leq \alpha$. If step 1 of the algorithm fails to simplify this input to `true`, so that it would be passed to step 2, then **val** will be forced to return unsuccessfully, because it will succeed negatively on each (atomic) disjunct. Since **val** returns `false` on each disjunction, without the "forced failure" clause we would get the incorrect answer `false` on this input; the answer is incorrect because the input is a theorem. Of course, if the logical simplifier in step 1 of the algorithm does anything at all, it is likely to catch this example before it is forced to fail, but there will certainly be more complex examples which escape the simplifier and produce disjuncts of this kind.

Remark 2: The theorem does not place a restriction on the syntactical form of ϕ. However, the theorem only states what happens if $\textbf{val}(\phi)$ can be successfully computed. The "forced failure" clause in Step 1 of the algorithm thus has a similar effect to a restriction on the syntax of ϕ. The following section of the paper will show that (under additional assumptions) the forced failure clause is never used; this is similar to removing a syntactical restriction. This formulation is better because it clearly separates the questions of the correctness of **val** and the applicability of **val**. For example, the forced failure clause isn't used on the example $\alpha \leq c \vee c \leq \alpha$ because it gets simplified to `true` in Step 1 of the algorithm.

Remark 3: Consider the example arising from the limit computation of the derivative of $\sqrt{x}$. When this was informally considered (Section 6.1), the analysis of $0 \leq x + h$ led to the formula $0 < x \vee (0 = x \wedge 0 \leq h)$. Although this is a disjunction, only one disjunct contains the nonstandard variable h, so it meets the restriction. As it turns out, though, when the algorithm is traced on this example, this disjunction is never passed as input to **val** anyway. See the footnotes in the proof.

Proof: The first part of the theorem concerns the termination of **val**. This is proved by a straightforward induction on the computation of **val**, using the assumptions that **infer**, **refute**, **stdpart**, **nonstdpart**, and **simplify** always terminate. The only clause that can cause failure of **val** is the failure to compute a Puiseux series. We now turn to the proof of (i), (ii), and (iii) under the assumption that $\textbf{val}(\phi)$ returns successfully.

According to Nelson's theorem, (ii) implies (iii).[4] We turn to the verification of (i) and (ii). There are three separate cases to the proof, according to the sense

[4]We would like to strengthen (iii) to provability in T. That can probably be done, if T includes the type constructor $A \to B$, by adapting the proof of Nelson's theorem, but if T does not contain the constructor $A \to B$, then the possibility of claiming provability in T in (3) is an open question.

of the domain inequality in I (that is, whether it came from a left limit, right limit, or two-sided limit). We deal explicitly only with the two-sided case; the two one-sided cases are similar.

Before giving the proof of (ii) in detail, we point to a key step. By standard techniques of nonstandard analysis, $\mathtt{nbhd}(\phi)$ is equivalent in T^* to

$$\forall\alpha.\mathtt{ereal}\ \ (\alpha \cong c \wedge \alpha \neq c \rightarrow \phi).$$

Therefore the following is equivalent to (ii) in T^*:

$$I, \Delta, \Gamma \Rightarrow \forall\alpha.\mathtt{ereal}(\alpha \cong c \wedge \alpha \neq c \rightarrow \phi) \leftrightarrow \mathbf{val}(\phi)$$

The heart of the proof is the following simple observation: in case the last step is the problematic reduction of $c < \alpha$ to `false`, it works, because $\forall\alpha.\mathtt{ereal}(\alpha \cong c \wedge \alpha \neq c \leftrightarrow c < \alpha)$ is equivalent to `false` in T^*.

In order to handle the maximum variety of input formulae, we have to keep track of whether **val** returns with positive or negative success. The important point about positive success is this: we will prove that if $\mathbf{val}(\phi)$ returns with positive success, then

$$\forall\alpha.\mathtt{ereal}\ \ (\alpha \cong c \wedge \alpha \neq c \rightarrow (\mathbf{val}(\phi) \leftrightarrow \phi)) \qquad (iv)$$

For example, $\mathbf{val}(\alpha < c + 1)$ is going to be `true`, with positive success, and for any $\alpha \cong c$, we will have $\alpha < c + 1$.

We will prove (ii) by induction on the number of steps of kinds (1) to (9) in the computation of $\mathbf{val}(\phi)$, simultaneously proving (iv) in case $\mathbf{val}(\phi)$ returns with positive success. Note that (iv) implies (ii). The basis case (immediate return) corresponds to step 1 in the definition of **val**. In that case no nonstandard variables are involved, and we have assumed that **val** terminates and preserves logical equivalence in the sense that $\Gamma, \Delta \Rightarrow \phi \leftrightarrow \mathbf{val}(\phi)$ is provable in T. But since no nonstandard variables are present, $\mathtt{nbhd}(\phi) \leftrightarrow \phi$ is a logical triviality, hence provable in T. Then (ii) follows immediately.

There are many cases in the induction step, corresponding to steps 2-9 of the definition of **val**. We take these cases one by one. In each case we repeat the relevant clause of the definition, so that the reader will not need two copies of the paper to follow the proof.

Case (2). First consider conjunctions. For notational simplicity we consider a conjunction of two arguments. The computation rule implies (using the fact that simplication preserves provable equivalence)

$$\mathbf{val}(A \wedge B) \leftrightarrow \mathbf{val}(A) \wedge \mathbf{val}(B).$$

By induction hypothesis we have

$$\mathbf{val}(A) \leftrightarrow \forall\alpha(\alpha \cong c \wedge \alpha \neq c \rightarrow A)$$

$$\mathbf{val}(B) \leftrightarrow \forall\alpha(\alpha \cong c \wedge \alpha \neq c \rightarrow B)$$

Hence

$$\mathbf{val}(A \wedge B) \leftrightarrow \forall\alpha(\alpha \cong c \wedge \alpha \neq c \rightarrow A \wedge B)$$

in view of the logical identity

$$\forall\alpha P \wedge \forall\alpha Q \leftrightarrow \forall\alpha(P \wedge Q).$$

If $\mathbf{val}(A \wedge B)$ returns with positive success, that is because both $\mathbf{val}(A)$ and $\mathbf{val}(B)$ do so, in which case we establish (iv) easily: for $\alpha \cong c \wedge \alpha \neq c$, we have

$$\begin{array}{rcl} \mathbf{val}(A \wedge B) & \leftrightarrow & \mathbf{val}(A) \wedge \mathbf{val}(B) \\ & \leftrightarrow & A \wedge B \end{array}$$

That completes the argument for conjunction.

Now consider disjunctions. Since the identity

$$\forall\alpha P \vee \forall\alpha Q \leftrightarrow \forall\alpha(P \vee Q).$$

is in general not valid, we cannot carry out a similar proof for arbitrary disjunctions. However, in the definition of **val** we imposed the restriction that at most one of the recursive calls returns with negative success. (Otherwise the forced distribution clause calls **val** recursively on a logically equivalent formula; in that case there is nothing to prove.) Therefore (since for simplicity we are showing the proof only for two disjuncts) we can assume that $\mathbf{val}(A)$ returns with positive success, so that by (iv) we have

$$\forall\alpha(\alpha \cong c \wedge \alpha \neq c \rightarrow (\mathbf{val}(A) \leftrightarrow A))$$

Then to complete the proof we need only the valid identity

$$P \vee \forall\alpha Q \leftrightarrow \forall\alpha(P \vee Q)$$

in which P does not contain α. (Here P is $\mathbf{val}(A)$.) To make the argument clear let us write $\forall\alpha..A$ to abbreviate $\forall\alpha.\texttt{ereal}\ (\alpha \cong c \wedge \alpha \neq c \rightarrow A)$. Then we have

$$\begin{array}{rcl} \mathbf{val}(A \vee B) & \leftrightarrow & \mathbf{val}(A) \vee \mathbf{val}(B) \\ & \leftrightarrow & \forall\alpha..A \vee \forall\alpha..B \\ & \leftrightarrow & \forall\alpha..\mathbf{val}(A) \vee \forall\alpha..B \\ & \leftrightarrow & \mathbf{val}(A) \vee \forall\alpha..B \\ & \leftrightarrow & \forall\alpha..(\mathbf{val}(A) \vee B) \\ & \leftrightarrow & \forall\alpha(A \vee B) \end{array}$$

If $\mathbf{val}(A \vee B)$ returns with positive success, that is because both $\mathbf{val}(A)$ and $\mathbf{val}(B)$ do so, in which case we establish (iv) easily: for $\alpha \cong c \wedge \alpha \neq c$, we have

$$\begin{aligned} \mathbf{val}(A \vee B) &\leftrightarrow \mathbf{val}(A) \vee \mathbf{val}(B) \\ &\leftrightarrow A \vee B \end{aligned}$$

Case (3). If the input ϕ is an inequality $r < s$, and r is not zero, and both sides contain α, return $\mathbf{val}(0 < s - r)$.

Argue as follows in T^*:

$$\begin{aligned} \Gamma \Rightarrow \mathbf{val}(r < s) &\leftrightarrow \mathbf{val}(0 < s - r)) \\ &\leftrightarrow \mathtt{nbhd}(0 < s - r) \\ &\leftrightarrow \mathtt{nbhd}(r < s) \end{aligned}$$

(by definition of **val**). The latter is equivalent (by induction hypothesis provably in T) to $\mathtt{nbhd}(0 < s - r)$. Since $\mathbf{val}(r < s) \leftrightarrow \mathbf{val}(0 < s - r))$, we have $\mathbf{val}(r < s) \leftrightarrow \mathtt{nbhd}(0 < s - r)$. Evidently $\mathtt{nbhd}(0 < s - r) \leftrightarrow \mathtt{nbhd}(r < s)$, so we have $\mathbf{val}(r < s) \leftrightarrow \mathtt{nbhd}(r < s)$ as desired.

If the return is with positive success, then (iv) is immediate:

$$\begin{aligned} \Gamma \Rightarrow \mathbf{val}(r < s) &\leftrightarrow \mathbf{val}(0 < s - r) \\ &\leftrightarrow 0 < s - r \\ &\leftrightarrow r < s \end{aligned}$$

Case (4). If the input is $r < s$ and exactly one side contains α, perform some mathematical simplifications that preserve logical equivalence and then solve the inequality for the nonstandard variable α if possible. (For example, $2\alpha < 1$ will become $\alpha < 1/2$.) If the result of these computations is ψ, return $\mathbf{val}(\psi)$.

Solving an inequality can generate an assumption, so the Δ returned from $\mathbf{val}(r < s)$ can be nonempty. However, a solved inequality is equivalent to the original inequality under the generated assumptions (if any). That is, T^* proves

$$\Gamma, \Delta \Rightarrow \psi \leftrightarrow r < s.$$

By induction hypothesis, T^* proves

$$\Gamma \Rightarrow \mathbf{val}(\psi) \leftrightarrow \mathtt{nbhd}(\psi).$$

Therefore T^* proves

$$\Gamma, \Delta \Rightarrow \mathbf{val}(\psi) \leftrightarrow \mathtt{nbhd}(r < s)$$

But since $\mathbf{val}(r < s) = \mathbf{val}(\psi)$, T^* proves

$$\Gamma, \Delta \Rightarrow \mathbf{val}(r < s) \leftrightarrow \mathtt{nbhd}(r < s)$$

which is (ii) in this case.

If the return is with positive success, then we prove (iv) as follows: in T^* we have by induction hypothesis

$$\Gamma, \Delta \Rightarrow \psi \leftrightarrow \mathbf{val}(\psi)$$

Hence

$$\Gamma, \Delta \Rightarrow \mathbf{val}(0 < r) \leftrightarrow \psi$$

Hence

$$\Gamma, \Delta \Rightarrow \mathbf{val}(0 < r) \leftrightarrow 0 < r$$

Case (5). If the input inequality contains a single nonstandard variable α, and the domain inequality for α is $\alpha \neq c$ (indicating that α arose from a two-sided limit as α approaches c), and the input inequality has one of the forms $\alpha < c$, $\alpha > c$, $\alpha \geq c$, $\alpha \leq c$, return `false`. If the input is $\alpha \neq c$, return `true`. Since the return in this case is always with negative success, we do not have to establish (iv).

Let us take the input forms one at a time. Suppose ϕ is $\alpha < c$. Then $\mathtt{nbhd}(\phi)$ says that for all x in some punctured two-sided neighborhood of c, $x < c$, which is refutable since each such neighborhood includes some points larger than c. Hence $\mathtt{nbhd}(\phi) \leftrightarrow \mathbf{val}(\phi)$, since both are equivalent to `false`. Similarly if ϕ is $c < \alpha$, since each punctured two-sided neighborhood of c contains some points less than c. The non-strict inequalities are treated the same way. Finally, consider the input $\alpha \neq c$. For this ϕ, $\mathtt{nbhd}(\phi)$ says that for all x in a punctured neighborhood of c, $x \neq c$. Since it is a *punctured* neighborhood, this is true. Hence $\mathbf{val}(\phi, c) \leftrightarrow \phi$ as desired, since in this case $\mathbf{val}(\phi, c)$ is `true`.

Case (6). If the input ϕ is $r < s$ with r not zero, set ϕ to the simplified form of $0 < s - r$. If the input has the form $0 < s$, compute $S = \mathbf{stdpart}(s)$.

The replacement of $r < s$ by an equivalent form preserves provable equivalence, so we may assume we are already in the case of input $0 < s$. In this part of the proof we are assuming that **val** does not fail, so we can assume that S is successfully computed. If S is zero we go to step 9, so under this case we may assume $S \neq 0$. The definition of **val** then says to call **infer** on the inequality $0 < S$. If it succeeds, return `true`, with positive success. If it does not succeed, we are in case (7), so under this case, we may assume **infer** returns `true` on $0 < S$.

Since the success is positive, we are required to establish not only (ii), but (iv). Since (iv) implies (ii), it will suffice to prove (iv). We must show in T^* that

$$\Gamma, \Delta \Rightarrow \alpha \cong c \wedge \alpha \neq c \rightarrow (\mathbf{val}(0 < s) \leftrightarrow 0 < s)$$

Since **val**$(0 < s)$ is `true`, this boils down to showing

$$\Gamma, \Delta \Rightarrow \alpha \cong c \wedge \alpha \neq c \rightarrow 0 < s$$

Using the nonstandard definition of limit (Theorem 1), it would suffice to prove

$$\Gamma, \Delta \Rightarrow \lim_{x \to c} s[x/\alpha] > 0$$

We have assumed that the algorithm **stdpart** has the property that T^* proves

$$\lim_{x \to c} s[x/\alpha] = \textbf{stdpart}(s)$$

so it will suffice to show that T^* proves

$$\Gamma, \Delta \Rightarrow 0 < \textbf{stdpart}(s)$$

That is, since $S = \textbf{stdpart}(S)$,

$$\Gamma, \Delta \Rightarrow 0 < S.$$

We know that **infer** returns `true` on $0 < S$. We have assumed (property (1) of **infer**) that **infer** is sound, so T^* proves $\Gamma, \Delta \Rightarrow 0 < S$, as desired.

Case (7). When the input has the form $0 < s$, and **stdpart**$(s) = S$, call **infer**$(S < 0)$. If this succeeds, then return `false`, with positive success. Else go to step 8.

We may suppose then that **infer**$(S < 0)$ succeeds. We have to prove not only (ii), but (iv), because the success is positive. We are required then to show that T^* proves

$$\Gamma, \Delta \Rightarrow \alpha \cong c \wedge \alpha \neq c \rightarrow \neg(0 < s).$$

As in the previous case, it would suffice to know T^* proves

$$\Gamma, \Delta \Rightarrow \lim_{x \to c} s[x/\alpha] < 0$$

But we know

$$\Gamma, \Delta \Rightarrow \lim_{x \to c} s[x/\alpha] = S$$

and T^* proves $\Gamma, \Delta \Rightarrow S < 0$ because **infer** succeeds on $S < 0$. The desired conclusion follows immediately.

Case (8). If the input is $0 < s$ and **stdpart**$(s) = S$, call **infer** $0 \neq S$. (It might be possible to infer this without being able to determine the sign of S, for example if $0 \neq S$ is a current assumption.) In that case we add $0 < S$ to the list Δ of new assumptions and return `true`, with positive success. If the call to **infer** does not succeed, return

$$0 < S \vee (S = 0 \wedge \textbf{val}(0 < s - S).$$

If the recursive call to **val** fails, the return is unsuccessful; otherwise the success is positive or negative, the same as is returned from the recursive call.

First consider the case in which **infer**$(0 \neq S)$ succeeds. To establish (iv), we have to prove

$$\Gamma, 0 < S \Rightarrow 0 < S \leftrightarrow \mathtt{true}$$

which is a triviality. (The interesting thing here is that hopefully this will not create contradictory assumptions in the antecedent, but since in general that cannot be entirely avoided, there is nothing to prove about it. Note that this case comes up only when there are variables or complicated expressions whose order relation to c cannot be determined by the system.)

Now suppose that the call to **infer**$(0 \neq S)$ does not succeed. Suppose first that the recursive call to **val** returns with positive success. Then we have to establish (iv). By induction hypothesis, we have

$$\Gamma \Rightarrow \mathbf{val}(0 < s - S) \leftrightarrow 0 < s - S$$

Hence

$$\begin{aligned} \Gamma \Rightarrow \mathbf{val}(0 < s) &\leftrightarrow S = 0 \wedge \mathbf{val}(0 < s - S) \\ &\leftrightarrow S = 0 \wedge 0 < s - S \\ &\leftrightarrow 0 < s \end{aligned}$$

which proves (iv).

Now we prove (ii). Suppose **infer** $0 \neq S$ succeeds. We have to show that T^* proves

$$\Gamma \Rightarrow 0 < S \leftrightarrow \mathtt{nbhd}(0 < s).$$

As in case (6), we have T^* proves

$$\Gamma \Rightarrow \lim_{x \to c} s[x/\alpha] = S$$

and hence, under the assumption $0 < S$, we have $\mathtt{nbhd}(s < c)$ as desired.

If the attempt to infer $0 \neq S$ does not succeed, we are to return

$$0 < S \vee (S = 0 \wedge \mathbf{val}(0 < s - S)).$$

By property (3) of **stdpart**

$$S = \lim_{x \to c} s[x/\alpha]$$

from which it follows that

$$\mathtt{nbhd}(0 < s) \leftrightarrow 0 < S \vee (S = 0 \wedge \mathtt{nbhd}(0 < s))$$

or equivalently

$$\mathtt{nbhd}(0 < s) \leftrightarrow 0 < S \vee (S = 0 \wedge \mathtt{nbhd}(0 < s - S))$$

But by induction hypothesis, we have

$$\Gamma \Rightarrow \texttt{nbhd}(0 < s - S) \leftrightarrow \mathbf{val}(0 < s - S).$$

Therefore

$$\Gamma \Rightarrow \texttt{nbhd}(0 < s) \leftrightarrow 0 < S \vee (S = 0 \wedge \mathbf{val}(0 < s - S).$$

In view of the definition of **val** in this case, that is nothing but

$$\Gamma \Rightarrow \texttt{nbhd}(0 < s) \leftrightarrow \mathbf{val}(0 < s),$$

which was what we had to prove.

Case (9). The input $0 < s$ has $\mathbf{stdpart}(s) = 0$. Call **nonstdpart** on s. This results in attempting to compute the leading term of the Puiseux series for s. Since we have assumed $\mathbf{val}(0 < s)$ returns successfully, we know that this computation succeeds. The series in question is computed in powers of $h = \alpha - c$, where c is the standard part of α. Let us call the apparent leading term of the Puiseux series ah^u. Then $\mathbf{val}(0 < ah^u)$ will be called. If this is a two-sided limit, a domain inequality equivalent to $h \neq 0$ is in the assumption list Γ. Therefore, by assumption (4) in Section 6.1, the return value from $\mathbf{val}(0 < ah^u)$ will be `true` or `false` (possibly this computation adds an assumption about the sign of a to the previously-present assumptions, but since this is the recursive call, by induction hypothesis that assumption is already in the list Γ).

Consider first the case in which the return value is `true`. By induction hypothesis we have

$$\Gamma \Rightarrow \texttt{nbhd}(0 < ah^u)$$

Since T proves the elementary properties of Puiseux series, we have

$$\Gamma \Rightarrow \texttt{nbhd}(0 < s)$$

which establishes (ii).

In case the success was positive, we have by induction hypothesis

$$\Gamma \Rightarrow h \cong 0 \wedge h \neq 0 \rightarrow (\texttt{true} \leftrightarrow 0 < ah^u)$$

and hence

$$\Gamma \Rightarrow (h \cong 0 \wedge h \neq 0 \rightarrow 0 < ah^u)$$

Again using the fact that T proves the elementary properties of Puiseux series, we have

$$\Gamma \Rightarrow h \cong 0 \wedge h \neq 0 \rightarrow 0 < s$$

establishing (iv) in case $\mathbf{val}(0 < ah^u) = \texttt{true}$.

Now consider the case $\mathbf{val}(0 < ah^u) = \texttt{false}$. By induction hypothesis we have T^* proves

$$\Gamma \Rightarrow \texttt{false} \leftrightarrow \texttt{nbhd}(0 < ah^u)$$

from which, by the elementary properties of Puiseux series, we have

$$\Gamma \Rightarrow \texttt{false} \leftrightarrow \texttt{nbhd}(0 < s)$$

establishing (ii) for this case.

If the success is positive, we have

$$\Gamma \Rightarrow \texttt{false} \leftrightarrow 0 < ah^u$$

whence we obtain

$$\Gamma \Rightarrow \texttt{false} \leftrightarrow \texttt{nbhd}(0 < s)$$

establishing (iv) for this case.

9 When will val succeed?

The above theorem contains as a hypothesis the assumption that $\mathbf{val}(\phi)$ terminates with success. This is the result of practical interest: it guarantees that we get no wrong answers from inferences made by *Mathpert* during limit calculations. However, it is also of interest to determine when the algorithm does succeed, or when it does succeed and does not generate a contradictory set of assumptions. The interest of this question is both theoretical and practical. The practical application is this: if the algorithm succeeds, but generates contradictory assumptions, software using this algorithm to guard against incorrect steps may still take an incorrect step. At least in that situation we will know it, provided that we recognize the assumptions as contradictory; but of course it would be better to prove that under certain conditions this does not happen. That is what we shall do in this section.

Originally I had hoped that "all expressions in the formula have Puiseux series that the system can compute" should suffice for success, and the determinability of the signs of the leading coefficients in the Puiseux series should be sufficient for success without contradictory assumptions.[5] This turns out to be true, but only if one makes stronger assumptions about the simplification algorithm than are necessary for the correctness proof. There are two places in the algorithm for **val** that can produce failure: the failure to compute a Puiseux series, and the forced failure clause in step 2. The question we take up in this section is whether the forced-failure clause has any real effect: is there an input ϕ on which **val** will actually use the forced-failure clause? The answer is: Yes, if simplification is weak; No, if simplification is strong enough.

[5] This sufficient condition might be made even stronger, as for example **val** will work on $0 < e^{-1/h}$ and even on $0 < he^{-1/h}$

Recall that our elimination algorithm contains some steps of "simplification" which have not been completely specified; the correctness proof depends only on the assumption that these steps preserve logical equivalence, and hence applies to various versions of the algorithm, with "weak" or "strong" simplification steps allowed. We shall show that the answer to the question is not the same for every version of the algorithm covered by the correctness theorem.

One obvious condition which will guarantee avoidance of the forced-failure clause is that the input ϕ should be a conjunction of equalities and inequalities. Indeed, a disjunction of such things can be allowed, provided at most one of the disjuncts contains the nonstandard variable α. This would not allow $\alpha \leq c \vee c \leq \alpha$; but this formula and many others will be allowed if we allow ϕ to be such that *after simplification* it has the form just stated. For example, $\alpha \leq c \vee c \leq \alpha$ will simplify to `true` in step 1. (This is not guaranteed by any assumptions we have stated about the simplification algorithm, but it would be true of any reasonable simplifier, and certainly is true of *Mathpert*.) Another example is $\alpha < c - 1 \vee c + 1 < \alpha$. This has α in two disjuncts, and will not simplify (or at least might not simplify, if simplification is weak). But when **val** is called on the disjuncts, both return `true` with positive success, so indeed **val** turns out to return `true` with positive success on this formula too. This is the reason for distinguishing two kinds of success.

The example at the end of Section 6 is an indication that without more assumptions about simplification, the forced failure clause can be encountered in natural problems. There we needed to know that simplification could reduce $c < \alpha \vee \alpha < c$ to `true` in the presence of the assumption $c \neq \alpha$, which is not guaranteed by the hypotheses of the correctness theorem. However, even more assumptions are needed, as the following discussion will show. To construct an example that would encounter the forced-failure clause, we want a disjunction $A \vee B$ such that **val** succeeds, but with negative success, on both A and B. For simplicity let us take $c = 0$, and try to find A in the form $0 < u$ and B in the form $0 < v$. We must make sure that $0 < u \vee 0 < v$ does not simplify as in the example $0 < \alpha \vee 0 < -\alpha$, which simplifies to `true` when $0 \neq \alpha$ is in Γ. We try to construct u and v so that $0 < u \vee 0 < v$ is provable in T^*, but the system can't simplify it to `true`, and indeed **val** reduces each disjunct to `false`. But **val** will fail unless it can compute the Puiseux series of u and v, so they can't be arbitrarily complicated. You might try $0 < \alpha^{1/3} \vee 0 < -\alpha^{1/5}$, but *Mathpert* will simplify this to $0 < \alpha \vee 0 < -\alpha$ and then to `true`. In general, the construction of a counterexample along these lines would depend on the Puiseux series computations being more capable than the simplification rules. For example, one could try

$$0 < \sin \alpha \vee 0 < -\arctan \alpha$$

If the system were able to compute the Puiseux series, but not to simplify the inequalities $0 < \sin \alpha$ and $0 < -\arctan \alpha$, we would get a counterexample.

Nothing in the assumptions used in the proof of the infinitesimal elimination theorem would prevent this being the case. This shows that if simplification is too weak to replace infinitesimal terms with the leading terms of their Puiseux series, the forced failure clause can be encountered.

Now I will show that if simplification is strong, the forced failure clause is never used. Suppose that simplification is allowed to compute Puiseux series in $h = \alpha - c$ and reduce inequalities by replacing each side with the leading term of its Puiseux series. If the signs of the coeffients can be determined, each inequality (or equality) containing α on which **val** returns successfully will simplify to an inequality $\alpha < c$, $c < \alpha$, or to `true` or `false`, or to a formula not containing α, under the assumptions $c \cong \alpha, \alpha \neq c$. This set of four inequalities is closed under conjunction and disjunction, so any disjunction containing more than one inequality involving α can be simplified to one involving only one such inequality.

If the signs of the coefficients cannot be determined, as in the example at the end of Section 6, we will get a disjunction of conjunctions of one or the forms $P \wedge c < \alpha$, or $P \wedge \alpha < c$, or just P, where P does not contain α. Either the equality-solving code itself or the forced distribution clause in step 2 will cause the distributive law to be used on such a disjunction. The result is a conjunction of disjunctions. If these disjunctions contain more than one inequality involving α, they can be combined into one as above. Hence the form eventually passed to **val** will be a conjunction of disjunctions of the forms $P \wedge \alpha < c$, $P \wedge c < \alpha$, or P, where P does not contain α. Since these disjunctions have only one disjunct involving α, the computation of **val** cannot encounter the forced failure clause.

10 Related Work

This is not the first use of nonstandard analysis in automatic deduction; Ballantyne and Bledsoe have experimented with it[1, 2]. From the viewpoint of automated deduction, I think that the method has much more potential than has been exploited so far in *Mathpert* to support education in calculus. The University of Texas group under Bledsoe's direction[9, 10] rightfully considered it an important achievement when their prover could automatically find a proof that the sum of two continuous functions is continuous (using the standard epsilon-delta definition of continuity). In nonstandard analysis, this theorem becomes a triviality.[6] Let h be an infinitesimal, and compute:

$$(f+g)(x+h) = f(x+h) + g(x+h) \cong f(x) + g(x) = (f+g)(x)$$

The essential reason why nonstandard analysis was helpful in *Mathpert* and in the above example, is that two alternating quantifiers are reduced to a

[6] This observation is in no way meant to detract from the achievement of the UT prover in finding a standard proof. This same research group pioneered the use of non-standard analysis in automated deduction.

quantifier-free formulation. Once we have a quantifier-free formulation, proof search can be reduced to rewrite-rule style computation, with tremendous gains in efficiency. This is true for computerized inference, and probably for human inference as well.

The idea that side conditions needed for operations can be added to the assumption list has also been used in Wu's method for proofs in geometry, as implemented by Chou[13].

Other systems allowing the use of partial terms have been implemented, for example IMPS[14, 15]. The integral has been treated as a variable-binding operator by Keisler[18].

The problem of combining a theorem-prover with a computer-algebra system has been addressed experimentally by Harrison and Théry[17] (who linked HOL and Maple), and Clarke and Zhao[12], who wrote a theorem prover *Analytica* in the *Mathematica* language. The former paper points out that it is safe to use the result of a symbolic integration performed by a possibly unreliable computer algebra system, if you can check the result by differentiation within the prover. The second is a direct attempt to keep track of conditions of operations. At the end of the paper, the authors call for building an integrated system for computer algebra and logic. This is what has been done in *Mathpert*, although on the logical side a full-fledged theorem prover has not been provided; only enough to support the correctness of symbolic computation.

Acknowledgements

The final form of this paper was greatly influenced by the careful reading, helpful criticisms, and suggestions made by several anonymous referees and by Carolyn Talcott, for which I am very grateful.

References

[1] M. Ballantyne and W. W. Bledsoe, 1977 JACM paper, Automatic Proofs in Analysis Using Non-Standard Techniques, *JACM* **24**(3) (1977) 353-371.

[2] M. Ballantyne, The Metatheorist: Automatic Proofs of Theorems in Analysis Using Non-Standard Techniques, Part II, in: R. S. Boyer (ed.), *Automated Reasoning*, Kluwer (1991).

[3] M. Beeson, *Foundations of Constructive Mathematics*, Springer-Verlag (1985).

[4] M. Beeson, Proving programs and programming proofs, in: Barcan, Marcus, Dorn, and Weingartner (eds.), *Logic, Methodology, and Philosophy of Science VII*, proceedings of the International Congress, Salzburg, 1983, pp. 51-81, North-Holland, Amsterdam (1986).

[5] M. Beeson, Logic and computation in *Mathpert*: an expert system for learning mathematics, in: Kaltofen, E., and Watt, S. M., *Computers and Mathematics*, pp. 202-214, Springer-Verlag (1989).

[6] M. Beeson, *Mathpert*: a computerized environment for learning algebra, trig, and calculus, *J. Artificial Intelligence and Education* **2** (1990), pp. 1-11.

[7] M. Beeson, *Mathpert*: Computer support for learning algebra, trigonometry, and calculus, in: A. Voronkov (ed.),*Logic Programming and Automated Reasoning*, Lecture Notes in Computer Science **624**, Springer-Verlag (1992).

[8] M. Beeson, Some applications of Gentzen's proof theory in automated deduction, in: Schroeder-Heister, P., *Extension of Logic Programming*, Springer Lecture Notes in Computer Science **475**, pp. 101-156, Springer-Verlag (1991)

[9] W. W. Bledsoe, Some automatic proofs in analysis, pp. 89-118 in W. Bledsoe, and D. Loveland (eds): *Automated Theorem Proving: After 25 years*, volume 29 in the *Contemporary Mathematics* series, AMS, Providence, R. I. (1984).

[10] W. W. Bledsoe, Non-resolution theorem proving, *Artificial Intelligence* **9**: 1-36 (1977).

[11] R. S. Boyer (ed.), *Automated Reasoning*, Kluwer (1991).

[12] Clarke, E., and Zhao, X. [1992], Analytica–an experiment in combining theorem proving and symbolic manipulation, Technical Report CMU-CS-92-1 7, School of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213.

[13] Chou, Shang-Ching, Proving elementary geometry theorems using Wu's algorithm, pp. 243-286 in Bledsoe, W., and Loveland, D. (eds): *Automated Theorem Proving: After 25 years*, volume 29 in the *Contemporary Mathematics* series, AMS, Providence, R. I. (1984).

[14] W. M. Farmer, Theory of Types, *J. Symbolic Logic* **55** (1990) 1269-91.

[15] W. M. Farmer, Guttman, and Thayer, IMPS: an Interactive Mathematical Proof System, *J. Automated Reasoning* **11** (1993) 213-248.

[16] S. Feferman, Logics for termination and correctness of functional programs, in: Y. Moschovakis (ed.), *Logic and Computer Science: Proc. of the '89 MSRI Conference*, Springer-Verlag (1989).

[17] J. Harrison and L. Théry, Extending the HOL theorem prover with a computer algebra system to reason about the reals, in *Higher Order Logic Theorem Proving and its Applications: 6th International Workshop, HUG '93*, pp. 174–184, Lecture Notes in Computer Science **780**, Springer-Verlag (1993).

[18] H. J. Keisler, Probability quantifiers, in: J.Barwise and S.Feferman (eds.) *Model-theoretic Logics*, pp.509-556, Perspectives in Mathematical Logic, Springer-Verlag (1985).

[19] S. Kleene, *Introduction to Metamathematics*, van Nostrand-Reinhold, Princeton, New Jersey (1952).

[20] G. Kreisel, Axiomatizations of non-standard analysis that are conservative extensions of formal systems for classical standard analysis, in: W. A. J. Luxembourg (ed.), *Applications of Model Theory to Algebra, Analysis, and Probability*, pp. 93-106. Holt, Rinehart, and Winston (1967).

[21] E. Nelson, Internal Set Theory: A new approach to nonstandard analysis, *Bull. A. M. S.* **83** (1977) 1165-1198.

[22] E. Nelson, The syntax of nonstandard analysis, *Annals of Pure and Applied Logic* **38** (1988) 123-134.

[23] A. Robinson, *Non-standard Analysis*, North-Holland, Amsterdam (1974).

[24] A. Robinson and E. Zakon, A set-theoretical characterization of enlargements, in W. A. J. Luxembourg (ed.), *Applications of Model Theory to Algebra, Analysis, and Probability*, Holt, New York (1969).

[25] C. L. Siegel, *Topics in Complex Function Theory, vol. I*, Wiley, Interscience, New York (1969).

[26] R. Stoutemeyer, Crimes and misdemeanors in the computer algebra trade, *Notices of the A.M.S.* **38**(7) 779-785, September 1991.