# Non-Standard Analysis in ACL2

Ruben A. Gamboa* (ruben@acm.org)
*Loop One, Inc.*
*515 Congress Ave., Suite 2500*
*Austin, TX 78701*

Matt Kaufmann† (matt.kaufmann@amd.com)
*Advanced Micro Devices, Inc.*
*5900 E. Ben White Blvd., M/S 625*
*Austin, TX 78741*

February 23, 2001

**Abstract.** ACL2 refers to a mathematical logic based on applicative Common Lisp, as well as to an automated theorem prover for this logic. The numeric system of ACL2 reflects that of Common Lisp, including the rational and complex-rational numbers and excluding the real and complex irrationals. In conjunction with the arithmetic completion axioms, this numeric type system makes it possible to prove the non-existence of specific irrational numbers, such as $\sqrt{2}$. This paper describes ACL2(r), a version of ACL2 with support for the real and complex numbers. The modifications are based on non-standard analysis, which interacts better with the discrete flavor of ACL2 than does traditional analysis.

**Keywords:** Non-standard analysis, automated theorem proving with the reals

## 1. Introduction

Non-standard analysis, introduced by Abraham Robinson [31] in the 1960s, gave rigorous foundations to the sort of reasoning about infinitesimal quantities that was used by Leibniz back when he co-invented the calculus and is still used today by engineers and scientists when applying calculus. A feature of many arguments using non-standard analysis is the use of mathematical induction and recursion in place of standard limit and compactness arguments.

In this paper we describe the integration of non-standard analysis into a semi-automated theorem prover, ACL2, that is particularly well-suited to the carrying out of proofs involving induction and recursion. ACL2 stands for "A Computational Logic for Applicative Common Lisp," and denotes both a logic [22, 24] and a theorem prover for this logic; see [20]. The ACL2 logic is closely related to the Boyer-Moore

* Work performed while this author pursued a Ph.D. from the University of Texas at Austin.

† Partially supported by the Office of Naval Research, Contract N00014-94-C-0193.

logic [6, 7], sharing Lisp's syntactic style, as well as its basic rules of inference, namely propositional calculus, equality rules, induction, and the introduction of new total recursive functions and constrained functions. However, ACL2 is faithful to the semantics of the applicative subset of Common Lisp, as defined for example in [18]. Support for quantifiers is very limited in both the Boyer-Moore logic and the logic of ACL2. It is possible to reason about generic functions in both logics, even though they are both strictly first-order; see [5, 24].

A significant difference between the Boyer-Moore logic and that of ACL2 lies in the treatment of numbers. The Boyer-Moore logic recognizes only the integers, whereas ACL2 has built-in support for all rational numbers in the complex plane. However, the ACL2 predicate characterizing numbers enumerates all the possible numeric types, explicitly ruling out the irrationals. This may seem to be a reasonable limitation if one views ACL2 as a vehicle for proving theorems about Common Lisp functions. Note specifically that although Common Lisp's `realp` function recognizes floating-point numbers, floating-point arithmetic does not conform to the rational field axioms, e.g., in that domain it is not necessarily the case that $(x - y) + y$ is equal to $x$. There is good reason, therefore, to eschew the irrationals and reason only about the rationals, explicitly treating floating-point numbers as limited-precision rationals where they are needed. This approach has yielded impressive results for microprocessors, such as the mechanical verifications of the AMD-K5™ microprocessor[1] floating-point division [8, 26] and square root [34] microcode, and hardware implementation of square root and other algorithms at the register-transfer level on the AMD Athlon™ microprocessor [33].

However, even in these projects the lack of support for the irrationals exacts a price. For example, in [34] Russinoff observes that the lack of the irrational numbers in ACL2 prevented him from mechanically proving the square root microcode against the exact IEEE specification. Instead, he had to prove a theorem using rational numbers, and then show outside of ACL2 why the verified theorem was equivalent to the IEEE specification.

We noted above, when considering the equation $(x-y)+y = x$, that familiar properties of rational functions may depend on their being interpreted over the rational numbers rather than floating-point numbers. Thus, the equation above is easily proved in ACL2. However, if we are interested in properties of transcendental functions then we can find ourselves stymied in ACL2 when approximating rational functions do

[1] AMD, the AMD logo and combinations thereof, AMD-K5, and AMD Athlon are trademarks of Advanced Micro Devices, Inc.

not have those properties. A simple example is sufficient to illustrate this point: consider the equation $\sqrt{x^4} = x^2$. This equation follows from the fact that $\sqrt{u^2} = |u|$, which however may be false when $sqrt(u)$ is used in place of $\sqrt{u}$. This example is actually not as contrived as it may seem. The correctness of the Fast Fourier Transform depends on the algebraic properties of the complex roots of unity, for example the fact that $w^{n/2} = -1$, where $w_n$ is the $\mathrm{n}^{\mathrm{th}}$ complex root of 1 [14]. However, this key fact is not necessarily true of a given rational approximation to the $\mathrm{n}^{\mathrm{th}}$ complex root function. To reason effectively about these functions, it is important to reason about the abstract mathematical functions first, and only then to consider a Common Lisp rational or floating-point approximation.

Since ACL2 is a first-order logic with only limited support for quantification, it seems far too weak to reason about the reals. Consider, for example, the least upper bound axiom, which states that every bounded set of real numbers has a least upper bound. This simple fact can not be expressed in ACL2, since ACL2 does not have a first-class notion of infinite set. In this paper, we will present a brief overview of non-standard analysis, first presented by Robinson as a formalization of infinitesimal calculus [31]. The main contribution of this paper will be to show that non-standard analysis provides a suitable framework for reasoning about the irrationals in ACL2. Moreover, this formalization should also be applicable to other theorem provers with support for induction.

Consider the proof of a simple theorem from analysis — given that $f$ and $g$ are continuous functions at a point $x_0$, the product $f \cdot g$ is also continuous at $x_0$. In a traditional analysis context, we would proceed as follows. Let $\epsilon > 0$. Since $f$ and $g$ are continuous at $x_0$, $f$ and $g$ are bounded around some neighborhood $N$ of $x_0$, say $|f(x)| < F$ and $|g(x)| < G$ for all $x \in N$. Let $\delta > 0$ be sufficiently small so that for all $x$ with $|x - x_0| < \delta$, we have for all $x \in N$: $|f(x_0) - f(x)| < \epsilon/2G$, and $|g(x_0) - g(x)| < \epsilon/2F$. With some algebraic simplification, it follows that $|f(x_0)g(x_0) - f(x)g(x)| < \epsilon$, completing the proof.

The mechanical verification of this argument in ACL2 would be considerably complicated as ACL2 would not be able to instantiate $\delta$, $F$, or $G$ automatically. ACL2 instantiates free variables by choosing an appropriate term from the current clause, but in this case the required terms do not occur elsewhere in the proof.

This same theorem can be proved more directly using non-standard analysis. The function $f$ is continuous at a "standard" point $x_0$ if $f(x_0)$ is "infinitely close" to $f(x_0 + \delta)$ for every "infinitely small" $\delta$. Consider $f(x_0 + \delta) \cdot g(x_0 + \delta)$. Since $f$ and $g$ are continuous, $f(x_0 + \delta)$ is infinitely

close to $f$. That is, it is equal to $f(x_0) + \epsilon_1$ for some infinitely small $\epsilon_1$. Similarly, $g(x_0 + \delta)$ is equal to $g(x_0) + \epsilon_2$ for some infinitely small $\epsilon_2$. Using nothing more than algebraic manipulation, we can conclude that $f(x_0+\delta) \cdot g(x_0+\delta) = f(x_0) \cdot g(x_0) + \epsilon_1 g(x_0) + \epsilon_2 f(x_0) + \epsilon_1\epsilon_2$. But $\epsilon_1 g(x_0)$ must be infinitely small, since it is the product of an infinitely small number with a standard number. Similarly, the other two terms involving $\epsilon$ are seen to be infinitely small. This shows that $f(x_0 + \delta) \cdot g(x_0+\delta)$ is infinitely close to $f(x_0)\cdot g(x_0)$; that is, that $f \cdot g$ is continuous at $x_0$.

The argument using non-standard analysis is much simpler, especially from an automated reasoning perspective. At no point does a free variable need to be instantiated. In fact, we may suspect the proof is too easy. Is it really true that $\epsilon_1 \cdot g(x)$ is small? It should be false if $g(x) = 1/\epsilon_1$, for example. The difficulty of determining when a particular term is insignificant led historically to the distrust of "infinitesimal analysis," and hence to the development of traditional analysis. It was only in the 1960's that Robinson showed how the notion of infinitesimals could be treated rigorously [31].

In the next section, we will provide an axiomatic introduction to non-standard analysis. This will lay the theoretical foundation for the introduction of non-standard analysis into ACL2, presented in Section 3. In Section 4, we present a complete ACL2 example illustrating how a transcendental function can be defined and reasoned about in ACL2 using non-standard analysis. Section 5 contains some concluding remarks. Sections 3 and 4 present ACL2 functions and expressions in their native Common Lisp syntax. We assume the reader is sufficiently familiar with a Lisp dialect to read these formulas.

## 2. An Introduction to Non-Standard Analysis

The formalism of non-standard analysis in ACL2 follows the axiomatic approach pioneered by Nelson in Internal Set Theory (IST) [28]. There are several good introductions to this approach to non-standard analysis, for example [30, 9, 27]. In this section, we select from known definitions and results in order to develop the basics needed to understand the material that follows. However, this section is not intended to be a comprehensive introduction to non-standard analysis.

Non-standard analysis changes our intuitive understanding of the number line in the following ways. The integers are divided into two groups. The *standard* integers include all the familiar integers: 0, $\pm 1$, $\pm 2$, .... There is at least one non-*standard* integer $N$. Moreover, $\pm N$, $\pm(N \pm 1)$, $\pm(N \pm 2)$, ..., $\pm(N \pm k)$ are also non-*standard* for any *standard*

integer $k$. Notice in particular that there is no least non-*standard* integer. A number is called *i-large* if it is larger in magnitude than all *standard* numbers; otherwise, the number is considered *i-limited*. Notice that the *i-limited* integers are precisely the same as the *standard* integers. The *i-* prefix in the preceding terms is a reminder that these are mathematically precise notions, unlike their informal counterparts, e.g. "large" or "small." It should be read as "ideally," as in "ideally large" or "ideally small," to distinguish these notions from the modern sense of infinity [9].

Non-standard analysis is typically unintuitive at first glance, but perhaps the following "picture" will help. Imagine a total order constructed as follows: for each rational number put a copy of the integers in its place. Formally, consider the Cartesian product $S$ of the rationals and integers ordered lexicographically: $\langle q, n\rangle < \langle q', n'\rangle$ if and only if $q < q'$ or $[q = q'$ and $n < n']$. The *standard* integers $\mathbb{Z}_0$ is the set of pairs $\langle 0, n\rangle$. An interesting fact is that addition, subtraction, and multiplication can be extended from the *standard* integers to $S$ so that their usual properties (such as commutative, associative, and distributive laws) still hold, though we do not prove that here. Traditionalists (among non-standard analysts!) would consider the "actual" integers to be the *standard* ones, which are isomorphically embedded into $S$. Nelson's approach, which we adopt below, is to axiomatize the integers and a notion of *standard* in such a manner that $S$ may be viewed as a model of these axioms, with $\mathbb{Z}_0$ interpreting the predicate *standard*. Of course Nelson and other non-standard analysts do not stop with the integers; but the two viewpoints above can be extended from the integers to the reals.

The corresponding picture for the reals is a little more complex. Certainly, there are *i-large* numbers, such as the *i-large* integer $N$, as well as $N/2$, $\sqrt{N}$, etc. As is the case with the integers, all reals larger in magnitude than $N$ are also *i-large*, as is any number $N - x$ for *i-limited* $x$. Moreover, there are reals smaller in magnitude than any positive *standard* real, such as $1/N$. Such numbers are called *i-small*. We call two numbers *i-close* if their difference is *i-small*. Every *i-limited* real is *i-close* to a *standard* real. That is, if $x$ is *i-limited*, it can be written as $x = x^* + \epsilon$, where $x^*$ is *standard* and $\epsilon$ is *i-small*. The number $x^*$ is called the *standard-part* of $x$.

The formal underpinning of non-standard analysis can start with the undefined unary predicate *standard*, introduced to the language of set theory containing just $\{\in\}$. It is reasonable to ask whether an arbitrary set is *standard*. Since all mathematical constructs can be built in the language of set theory, it is possible to ask whether an arbitrary mathematical structure, e.g., a function, is *standard*.

The predicate *standard* is special in that it is not considered to be set-forming. The familiar specification axiom schema from ZF set theory allows the construction of a set $S' = \{x \in S \mid P(x)\}$ from an arbitrary predicate $P$ provided $S$ is a set. However, no such set can be constructed based on the predicate *standard*, nor on any predicate defined using *standard*. We refer to *standard* and all predicates built using *standard* as *non-classical* predicates. The original predicates are referred to as *classical*[2]. The precise restriction is that only classical predicates can be used to define sets using the specification axiom schema of set theory. Note, this does not disallow any construction that was possible in the theory before the introduction of the predicate *standard*.

We may refer to a formula or to a function definition as *classical* or *non-classical*. This is a purely syntactic notion, depending on whether the formula in question is defined using the predicate *standard* or some other non-classical predicate. We may describe the *meaning* of a formula or of a function definition, which is a set or function (respectively), as either *standard* or not. It is a non-trivial observation that a classically defined function is necessarily *standard*. However, the converse of this observation is not true. For example, the function $f(x)$ defined as $standard(x) \vee \neg standard(x)$ is not classically defined, but it is *standard* since it is equivalent to the classical function *true*. We will shortly see examples of *standard* sets and functions being constructed using non-classical primitives. It is best, therefore, to keep in mind the distinction between the purely syntactic notion of classical and the mathematical notion of *standard*.

There are three axioms governing the use of the predicate *standard*. The first axiom is the idealization axiom. This states that for any classical binary relation $R(x, y)$, the following statements are equivalent:

- For every *standard*, finite set $F$ there is a $y$ so that $R(x, y)$ is true for all $x$ in $F$.
- There is a $y$ so that $R(x, y)$ holds for all *standard* $x$.

As an example, let $R$ be the $<$ relation. It is certainly the case that there is an upper bound for every finite set of reals. The idealization axiom asserts the existence of a, necessarily non-*standard*, real $y$ that is greater than all *standard* reals.

A second axiom is the standardization axiom, which states that for any *standard* set $S$ and any property $P$, whether classical or not, there is a unique *standard* subset $S'$ of $S$ such that for any *standard* element $x$, $x \in S'$ if and only if $x \in S$ and $P(x)$ is true. This powerful axiom is

[2] Nelson [27] refers to classical predicates as *internal*, and the rest as *external*.

an analogue of the specification axiom, allowing a non-classical property $P$ to be used in set construction. Note, however, that the axiom does not guarantee anything about the non-*standard* elements of $S'$. In particular, $S'$ may contain a non-*standard* element $x$ for which $P(x)$ is false, or it may fail to contain a non-*standard* element $x \in S$ for which $P(x)$ is true. For example, let $P$ be the property *standard*, and consider the set $T = \{x \in \mathbb{R} \mid \mathit{standard}(x)\}$. Since *standard* is a non-classical predicate, $T$ is not an admissible set. However, the standardization principle guarantees the existence of a unique *standard* set $T$ so that, for *standard* $x$, $x \in T$ if and only if $x \in \mathbb{R}$ and *standard*$(x)$. That is, $T$ is a *standard* subset of the reals containing all the *standard* reals. Since $\mathbb{R}$ is itself a subset of $\mathbb{R}$ containing all *standard* reals, it follows that $T = \mathbb{R}$, since the axiom guarantees the subset $T$ is unique.

Although the specification axiom schema can not be used to create sets, it is convenient to consider set-like objects formed by $S = \{x \mid P(x)\}$ where $P(x)$ is a non-classical property. For such an $S$, the standardization axiom justifies the construction of a set ${}^\circ S$, which for $S \subset \mathbb{R}$ is given by ${}^\circ S = \{x \in \mathbb{R} \mid P(x)\}$. ${}^\circ S$ is the unique *standard* set that agrees with $S$ on all *standard* elements. It is referred to as the *shadow* of $S$. A similar construction can be performed when $S$ is not composed entirely of real numbers. It is only necessary to find a *standard* supserset of $S$, e.g., the complex numbers; the value of ${}^\circ S$ is then easily seen to be independent of the choice of superset.

The final axiom is called the transfer axiom. It states that a *classical* predicate $P(x)$ referencing only *standard* constants is true for all $x$ if it is true for all *standard* x. This axiom implies that if a *classical* predicate $P(x)$ with only *standard* constants is satisfied by some $x_0$, it must be satisfied by some *standard* $x_1$. In particular, if the predicate $P$ can be satisfied by only one element, this (unique) element must be *standard*. Examples of such elements include 0, 1, $\pi$, $\mathbb{R}$, etc.

Forgetting to restrict the specification axiom to classical properties is a common mistake made when learning to use non-standard analysis. This is often a subtle mistake. Consider, for example, the following argument. 0 is a *standard* natural number. If $n$ is a natural number and $n$ is *standard*, so is $n+1$. (This follows from the transfer principle, since $n+1$ is uniquely determined and $n$ and 1 are *standard*.) Appealing to the principle of induction, therefore, we can conclude that all the natural numbers are *standard*. This is false. To understand the error, recall that the induction principle is based on the well-foundedness of the naturals: every non-empty set of naturals has a least element. Induction is sound, because the induction hypothesis guarantees the set of counter-examples to the theorem can not have a least element, hence it must be empty. However, in this case the set of counterexamples is

$S = \{n \in \mathbb{N} \mid \neg \mathit{standard}(n)\}$, and since *standard* is not a classical property, this set is not well-formed. What this means is that the principle of induction can not be used to prove non-classical properties. As was the case with the specification axiom, notice that this restriction does not invalidate any proof that was possible before the introduction of the *standard* predicate.

Using the concept of a shadow set, it is possible to derive a weaker induction principle that is applicable to any predicate. Let $P$ be a classical or non-classical property defined over the natural numbers $n \in \mathbb{N}$, and further assume that $P(0)$ and $P(n) \Rightarrow P(n+1)$ have been established. Let $S = {}^\circ\{n \in \mathbb{N} \mid P(n)\}$. That is, $S$ is the *shadow* of the "set" of naturals satisfying $P$. We observe that $S$ is the set of all naturals, since $S$ is a *classical* set, and therefore membership in $S$ can be established using the classical induction principle. But since $S$ is a shadow set, we can only conclude that $P(n)$ is true for *standard* n. Therefore, the non-standard induction principle can conclude $\mathit{standard}(n) \Rightarrow P(n)$ from $P(0)$ and $P(n) \Rightarrow P(n+1)$ for any property $P$.

The concept of shadows allows more powerful constructions. Consider a non-classically defined function $f : \mathbb{R} \to \mathbb{R}$. The function $f$ is a set of tuples $(x, f(x))$, with the restriction that no two tuples have the same first element. Observe, the shadow of this set ${}^\circ f$ is also a function. For if there exist $x$ together with distinct $y$ and $z$ for which $(x, y)$ and $(x, z)$ both belong to ${}^\circ f$, then by the (contrapositive of the) transfer axiom there are *standard* such $x$, $y$, and $z$; but then $(x, y)$ and $(x, z)$ are distinct standard elements of ${}^\circ f$ and hence, by definition of ${}^\circ f$, of $f$, which contradicts the assumption that $f$ is a function. Since $f$ and its shadow have the same *standard* elements, it follows that given any function $f$ such that $f(x)$ is *standard* whenever $x$ is, it is possible to implicitly define a *standard* function $g$ so that $g(x) = f(x)$ for all *standard* $x$; namely, $g$ is the shadow of $f$. As in the case with all shadow constructions, it is not possible to say what the value of $g$ is for a non-*standard* $x$, except by indirect means. For example, if we can establish that $g(x) = x^2$ for all *standard* $x$, then using the transfer principle we find that $g(x) = x^2$ for *all* $x$, even though $f(x)$ may not be $x^2$ for a non-*standard* $x$.

As we have seen, the *standard* predicate and the idealization, standardization, and transfer principles are surprisingly powerful. Using them, we can formalize the intuitive notions introduced at the beginning of this section. We call a number *i-small* if it is smaller in magnitude than all positive *standard* numbers. That is, $\epsilon$ is *i-small* if $|\epsilon| < x$ is true for all *standard* $x > 0$. Clearly 0 is *i-small*, but it is not the only *i-small* number. Recall, using the *idealization* axiom, we discovered a number $y_>$ that is greater than all *standard* reals.

Consequently, $1/y_>$ is smaller in magnitude than all non-zero *standard* reals, and so it is *i-small*. Similarly, a number $x$ is called *i-large* if it is larger in magnitude than all *standard* numbers. The number $y_>$ serves as an example. It is clear that $y$ is *i-large* if and only if $y \neq 0$ and $1/y$ is *i-small*, and that $y$ is *i-small* if and only if $y = 0$ or $1/y$ is *i-large*. A number that is not *i-large* is called *i-limited*. All *standard* numbers are *i-limited*. Two numbers $x$ and $y$ are *i-close* when $x - y$ is *i-small*. Since 0 is the only *standard i-small* number, it follows that two *standard* numbers are *i-close* if and only if they are equal.

In addition, we can prove the existence of a function *standard-part*, which assigns a *standard* number *i-close* to each *i-limited* real. That is, for *i-limited* $x$, *standard-part*$(x)$ is *standard* and *i-close* to $x$. The number *standard-part*$(x)$ can be defined as the supremum of $^\circ\{y \in \mathbb{R} \mid y \leq x\}$. This set is bounded above: for since $x$ is *i-limited*, there must be a *standard* number $M$ with $|x| \leq M$. From the transfer principle, we know that *standard-part*$(x)$ is *standard*, since it is the supremum of a *standard* set. That *standard-part*$(x)$ is *i-close* to $x$ follows from the fact that for any *standard* $c > 0$, $x -$ *standard-part*$(x) \geq c$ implies that $x \geq$ *standard-part*$(x) + c$ and so *standard-part*$(x) + c$ is in $^\circ\{y \in \mathbb{R} \mid y \leq x\}$, since it is *standard* and at most $x$, but then *standard-part*$(x)$ would not be a supremum of this set. Similarly, if *standard-part*$(x) - x \geq c$, we have that *standard-part*$(x) - c \geq x$ and so $^\circ\{y \in \mathbb{R} \mid y \leq x\}$ would be bounded by *standard-part*$(x) - c$, again contradicting *standard-part*$(x)$ as the supremum of the set. Therefore, *standard-part*$(x)$ and $x$ are *i-close*. Since *standard-part*$(x)$ is *standard*, it follows that it is the unique *standard* number *i-close* to $x$.

The power of non-standard analysis becomes evident when we realize that these new predicates possess simple algebraic properties. For example, $x + y$ is *i-small* if both $x$ and $y$ are *i-small*. If $x$ is *i-limited* and $\epsilon$ is *i-small*, $\epsilon \cdot x$ is *i-small* and for $x, \epsilon \neq 0$, $x/\epsilon$ is *i-large*. If $x$ is *i-close* to $y$ and $y$ is *i-close* to $z$, then $x$ is *i-close* to $z$.

The power becomes more obvious when traditional notions from analysis are written in the language of non-standard analysis. For example, a *standard* sequence $\{a_n\}$ converges to the *standard* point $A$ iff $A$ is *i-close* to $a_N$ for all *i-large* natural numbers $N$. A *standard* function $f$ is continuous at a *standard* point $x$ iff $f(y)$ is *i-close* to $f(x)$ for all $y$ *i-close* to $x$. These definitions are easier to use than the traditional definitions from analysis. Consider the function $f(x) = \sin(1/x)$. It is a classical result from analysis that this function can not be extended continuously at $x = 0$. The traditional proof is to find two sequences $\{a_n\}$ and $\{b_n\}$ converging to 0, so that $\{f(a_n)\}$ and $\{f(b_n)\}$ converge to different values. In non-standard analysis, the argument is considerably more direct. Consider $x_0 = \frac{1}{2\pi n}$ and $x_1 = \frac{1}{2\pi n + \pi/2}$, where $n$ is an *i-large*

integer. Then $x_0$ and $x_1$ are *i-close* (both being *i-small*), but $f(x_0) = 0$ is not *i-close* to $f(x_1) = 1$. Therefore, no value of $f(0)$ can be *i-close* to $f(y)$ for all $y$ *i-close* to 0, and hence $f$ can not be continuously extended at $x = 0$.

## 3. Adding Non-Standard Analysis into ACL2

In this section, we introduce ACL2(r)[3], a modified version of ACL2 with support for non-standard analysis.

Before introducing non-standard analysis into ACL2, it is necessary to extend the ACL2 numeric system to include the irrationals. This can be accomplished by adding the new type recognizers `realp` and `complexp`. In addition, the ACL2 arithmetic axioms need to be modified to account for these new types. In many cases, this can be accomplished by substituting `realp` for `rationalp` and `complexp` for `complex-rationalp`. For example, the `Positive` axiom, stating that if $x$ and $y$ are positive rationals then $x \cdot y$ is a positive rational, is trivially extended to the reals. In other cases, it is more appropriate to add a new axiom corresponding to the original one. For example, it is a built-in axiom that the product of two rationals is a rational. Rather than weakening this axiom by changing rational to real, it is better to add the corresponding closure axiom for the reals. Of course, a few axioms can not be extended to the irrationals, such as the axiom defining the numerator and denominator of a rational number.

In addition to containing basic arithmetic axioms, ACL2 also defines many useful arithmetic functions, such as `abs`, `floor`, and `trunc`. All of these functions need to be extended to accept irrational arguments. This is trivial in the case of functions like `abs`. However, the functions `floor` and `trunc` are defined using `integer-quotient`, which is axiomatized to perform division by repeated subtraction. For example, the floor of 17/2 is found by dividing 2 into 17 using repeated subtraction of 2, starting at 17, giving a value of 8. Note in particular that the algorithm does not proceed by repeatedly subtracting 1 from 17/2. The reason is that in the latter case ACL2 would not accept the recursion, since the decreasing "measure" is fractional, hence not well-founded. In the case of rationals, an integer measure can be found by looking at the numerator and denominator of the number; however, a similar trick will not work for the reals. A simple solution to this problem is to introduce a new undefined function `floor1` which is axiomatized to

[3] ACL2(r) is distributed with ACL2 as of ACL2 Version 2.5. It can be obtained from the ACL2 home page: `http://www.cs.utexas.edu/users/moore/acl2/`. The documentation for topic `real` is a useful starting point.

return the correct value of `floor` for an arbitrary number. Of course, it would be easier to axiomatize `floor` directly, but that would prevent the function `floor` from being executable for any arguments. By introducing `floor1`, it is possible to allow ACL2(r) to have an executable version of `floor`, at least for rational constants.

Note, as modified above, ACL2(r) will be able to reason about the irrational numbers, but it can not construct irrational numbers. In particular, there are no irrational constants, and there is no mechanism to allow an ACL2(r) function to return an irrational result given rational arguments. Nevertheless, it can reason effectively about the real and complex numbers. For example, it is a theorem of the unmodified ACL2 that $x \cdot x \neq 2$ for all $x$ [13]. However, this statement is false in ACL2(r). Instead, ACL2(r) can prove that if $x \cdot x = 2$ then $x$ must be real but not rational. But, as described so far, ACL2(r) can not prove that there must be some $x$ with $x \cdot x = 2$. To do that requires knowledge that the real number line is complete. We do this below in ACL2(r) using non-standard analysis.

We begin the treatment of non-standard analysis in ACL2(r) with the following functions: `standard-numberp`, `standard-part`, and `i--large-integer`. We refer to these functions as the primitive non-standard functions in ACL2(r). The function `standard-numberp` tests whether a number is *standard* or not. Note, `standard-numberp` can only be true of ACL2(r) numbers, not arbitrary objects. The function `standard-part` returns the standard part of an *i-limited* real or complex number. The constant `i-large-integer`, as its name suggests, is an integer axiomatized to be *i-large*. The functions `i-small`, `i--large`, `i-limited`, and `i-close` are given explicit definitions in terms of `standard-part`. A number is `i-small` if its `standard-part` is zero; it is `i-large` if its inverse is `i-small`; and it is `i-limited` if it is not `i-large`. Two numbers are `i-close` if their difference is `i-small`.

All of these functions are special in two ways. First, none of the primitive non-standard functions is given an executable definition. Instead, they are all treated as constrained functions, as if they had been introduced using `encapsulate` or `defstub`; ACL2(r) can not evaluate the value of any term involving these functions. Second, ACL2(r) introduces the notion of classical and non-classical functions. These new functions are considered to be non-classical, as are any functions defined in terms of non-classical functions. Note, any ACL2(r) functions (formulas) that are also ACL2 functions (formulas) are necessarily classical. ACL2(r) considers constrained functions to be classical, and only classical functions are allowed as witnesses to or functional instances of constrained functions.

ACL2(r) treats non-classical functions specially. Non-classical functions can not be defined recursively. Moreover, the use of induction on non-classical formulas is restricted. Recall that in internal set theory induction can only be used over the *standard* integers. Similarly, the induction principle in ACL2 can be used to establish the truth of non-classical formulas only for *standard* instances of their variables. The remaining cases are treated separately, similar to the "base" cases. That is, for each variable appearing in the formula, the ACL2(r) induction principle adds the proof obligation that the formula is true for non-*standard* instances of the variable[4]. Details of the non-standard induction principle in ACL2(r) can be found in Appendix A.

Consider the function $2^n$ defined as follows:

```
(defun expt-2 (n)
  (if (and (integerp n) (< 0 n))
      (* 2 (expt-2 (- n 1)))
    1))
```

Suppose we attempt to prove that `expt-2` always returns a *standard* value:

```
(standard-numberp (expt-2 n))
```

The unmodified ACL2 induction principle would reduce this theorem to the following base and induction cases:

```
(implies (not (and (integerp n) (< 0 n)))
         (standard-numberp (expt-2 n)))

(implies (and (and (integerp n) (< 0 n))
              (standard-numberp (expt-2 (- n 1))))
         (standard-numberp (expt-2 n)))
```

In addition, since the original theorem uses the non-classical function `standard-numberp`, ACL2(r) would add the following base case:

```
(implies (not (standard-numberp n))
         (standard-numberp (expt-2 n)))
```

Intuitively, the first two goals prove the theorem for any *standard* $n$, and the last goal proves it for any non-*standard* $n$. Of course, in this case the last goal can not be established, and so ACL2(r) will not prove

[4] Actually, it is only necessary to add this obligation for some of the variables in the term, intuitively those changed by the induction scheme. For example, to prove that $x^n$ is *i-limited*, it is only necessary that $n$ be *standard* and $x$ be *i-limited* but not necessarily *standard*.

that $2^n$ is *standard* for all integers $n$. As can be seen, this modification to the induction principle is crucial for soundness, since the first two induction goals are clearly true.

To see how the modified induction principle works, consider the following theorem, which only requires that `expt-2` return a *standard* value for *standard* arguments:

```
(implies (standard-numberp n)
         (standard-numberp (expt-2 n)))
```

In this case, the base and induction steps are as follows:

```
(implies (and (not (and (integerp n) (< 0 n)))
              (standard-numberp n))
         (standard-numberp (expt-2 n)))

(implies (and (and (integerp n) (< 0 n))
              (standard-numberp n)
              (implies (standard-numberp (- n 1))
                       (standard-numberp
                        (expt-2 (- n 1)))))
         (standard-numberp (expt-2 n)))
```

ACL2(r) can quickly and automatically prove both of these goals. In addition, ACL2(r) adds the following goal, since the theorem is non-classical:

```
(implies (and (not (standard-numberp n))
              (standard-numberp n))
         (standard-numberp (expt-2 n)))
```

ACL2(r) is able to prove this goal, since the hypotheses are obviously contradictory. That completes the proof of the original conjecture. In practice, the only non-classical theorems that can be proved by induction are those that specifically apply to *standard* values, for example by having `standard-numberp` as an explicit hypothesis, as in the `expt-2` example.

ACL2(r) introduces two new events to deal with non-classical formulas. The event `defun-std` is used to define a *standard* function using a non-classical body. Only non-recursive functions may be defined using `defun-std`. The resulting definition is considered classical; that is, the new function symbol may be used in subsequent classical definitions. This event can be justified only when the function body returns a *standard* value for *standard* arguments. In these cases, the function is explicitly defined only for *standard* arguments; that is, the function is

defined by its body only for *standard* arguments. For the remaining arguments, the function is implicitly defined, as being the (unique) *standard* function that agrees with the body for *standard* arguments. That such a function uniquely exists is guaranteed by the principle of standardization, as observed in the remarks about function $^\circ f$ in the previous section.

Consider, for example, the function introduced as follows:

```
(defun-std std-pt (x)
  (standard-part x))
```

This function is accepted, because for *standard* numbers $x$, `(standard-part x)` is also *standard*. It is important to realize that `std-pt` is *not* the same as the function `standard-part`. The *standard* function `std-pt` is only guaranteed to be equal to `standard-part` for *standard* values of `x`. Since `standard-part` returns `x` for these values, we can conclude that `std-pt` is the identity function for *standard* `x`. But, since `std-pt` is a *standard* function, it must also be the identity function for all values of `x`, since two classical functions that have equal values for *standard* arguments must have equal values for all arguments (by the transfer principle), and therefore must be equal functions.

The other event new to ACL2(r) is `defthm-std`, which is an analogue of the transfer principle. Using `defthm-std`, it is possible to prove a theorem by proving it only for *standard* arguments; however, `defthm-std` can only be used to prove classical formulas. For example, suppose we wished to prove the following theorem:

```
(implies (acl2-numberp x)
         (equal (std-pt x) x))
```

Since `std-pt` is a classical function, this theorem can be proved using `defthm-std`. ACL2(r) will prove this theorem by proving the theorem for *standard* numeric values of `x` only. That is, it attempts to prove the following formula instead:

```
(implies (and (standard-numberp x)
              (acl2-numberp x))
         (equal (std-pt x) x))
```

Since `x` is known to be *standard*, we can expand the term `(std-pt x)` using the body of `std-pt`, and the proof is then trivial. Note, the theorem could not have been proved using `defthm` instead of `defthm-std`, since without the hypothesis `(standard-numberp x)`, the term `(std-pt x)` can not be expanded: `std-pt` was introduced using `defun-std`, so its explicit definition is only applicable for *standard* arguments.

Moreover, a similar theorem about `standard-part` instead of `std-pt` could not have been proved even using `defthm-std`, since the resulting formula is not classical.

Using `defun-std` and `defthm-std`, it is possible to introduce irrational functions and to prove theorems about them. Consider the function $\sqrt{x}$. In [13], it is established that the $\sqrt{x}$ function can not be defined in ACL2, although ACL2 can define the function `iter-sqrt` so that if $s$ is equal to `(iter-sqrt` $x$ $\epsilon$`)`, then $0 \leq x - s^2 \leq \epsilon$, for arbitrary positive $\epsilon$ and nonnegative $x$. The function `iter-sqrt` is defined using the bisection method, starting with an initial range equal to $[0, \max(1, x)]$ and terminating when the current range is no wider than $\epsilon$.

We can define the square root function in ACL2(r) as follows:

```
(defun-std sqrt (x)
  (standard-part
   (iter-sqrt x (/ (i-large-integer)))))
```

The proof obligation is to show that the body of this definition has a *standard* value when $x$ is *standard*. Let `s` abbreviate the term `(iter-sqrt x (/ (i-large-integer)))`. The term `s` is bounded by the maximum of `x` and `1`, so it is *i-limited* given that `x` is *i-limited*. Therefore, if `x` is *standard*, the body of `sqrt` is a *standard* number, and we may use `defun-std`. Moreover, for *standard* `x`, the results in [13] show that

- `(<= 0 (- x (* s s)))` and
- `(<= (- x (* s s)) (/ (i-large-integer)))`

are true for non-negative reals `x`. Since `(/ (i-large-integer))` is *i-small*, `x` and `(* s s)` are *i-close*. Hence `x` and `(* (sqrt x) (sqrt x))` are *i-close*; then since they are *standard*, they must be equal. This establishes that for *standard* `x`, `(* (sqrt x) (sqrt x))` is equal to `x`. We can generalize this result using `defthm-std`:

```
(defthm-std sqrt-sqrt
  (implies (and (realp x) (<= 0 x))
           (equal (* (sqrt x) (sqrt x)) x)))
```

In particular, this allows us to identify the real number $x =$ `(sqrt 2)` such that $x^2 = 2$.

We have extended ACL2 while keeping in mind the development in Section 2, but with particular attention to the induction principle as worked out in Appendix A. At a more practical level, we have illustrated how non-standard analysis can be used in ACL2(r) to define

and reason about irrational functions. In the next section, we present a more detailed example of reasoning with non-standard analysis.

## 4. An Extended Example

In this section, we develop a complete example used historically to motivate the development of ACL2(r). Specifically, we will use ACL2(r) to compute the value of $\sin(1/2)$ with an error no larger than $1/645120$. We will also show how more mathematically interesting theorems can be derived, such as $\sin(-x) = -\sin(x)$.

Recall that the Taylor expansion of $\sin(x)$ about $x = 0$ is given by

$$\sin(x) = x - \frac{x^3}{3!} + \cdots + \frac{-1^n \cdot x^{2n+1}}{(2n+1)!} + \cdots$$

We proceed by defining an approximation

$$\begin{aligned} s_N(x) &= x - \frac{x^3}{3!} + \cdots + \frac{-1^N \cdot x^{2N+1}}{(2N+1)!} \\ s(x) &= \textit{standard-part}(s_N(x)) \end{aligned}$$

for some *i-large* integer $N$.

As a warm-up exercise, we check for *i-limited* $x$ that $s(x)$ is *standard* and is independent of the choice of the *i-large* integer $N$. First, observe that for *i-limited* $x$, $s(x)$ is *standard* because $s_N(x)$ is also *i-limited*, which we demonstrate as follows. The first $\lceil |x| \rceil$ terms of $s_N(x)$ are all *i-limited*, hence their sum is *i-limited*. Moreover, the remaining terms form an alternating series, and so their sum is bounded by their first term, which is *i-limited*. Therefore, $s_N(x)$ is *i-limited*. Now, consider $s_M(x)$, where $M > N$. $s_M(x) - s_N(x) = \frac{-1^{N+1} \cdot x^{2N+3}}{(2N+3)!} + \cdots + \frac{-1^M \cdot x^{2M+1}}{(2M+1)!}$, hence $|s_M(x) - s_N(x)| \leq \frac{|x|^{2N+3}}{(2N+3)!}$. But for *i-limited* $x$, this is *i-small* since it is no greater than $(|x|/(2N+3)) \cdot |x|^{\lceil |x| \rceil}$, the product of an *i-small* and an *i-limited* number. Hence, since $s_N(x)$ is *i-limited* and $s_M(x)$ is *i-close* to $s_N(x)$, it follows that *standard-part*$(s_N(x))$ = *standard-part*$(s_M(x))$ for all *i-large* $M$ and $N$.

The function $s(x)$ defined above is *not* identical to the trigonometric sine function. As a simple counterexample, consider $s(\epsilon)$ for some positive, *i-small* $\epsilon$. Clearly, $0 < s_N(\epsilon) < \epsilon$, and so $s(\epsilon) = 0$. But $\sin(\epsilon) \neq 0$, since $0 < \epsilon < \pi$ and there are no roots of sine in the range $(0, \pi)$. However, since $s_N(x)$ is *i-limited* for *i-limited* $x$, it follows that $s(x)$ is *standard* for *standard* $x$, and so we can use `defun-std` to define $^\circ s$, the shadow of $s$ (see Section 2). That is, there is a *standard* function

$\sin(x)$ that coincides with $s(x)$ for all *standard* numbers $x$. The transfer principle guarantees that this standard function is unique.

Why should this unique standard function $^\circ s$ be the same as the trigonometric sine function? The reason is that for any $x$ and any positive $\epsilon$, we can find a positive integer $M_0$ such that $|\sin(x) - s_M(x)| < \epsilon$ for any integer $M \geq M_0$. Moreover, we can choose $M$ so that it is also larger than $N$, the arbitrary *i-large* integer used to define $s(x)$. If $x$ is *standard* and $\epsilon$ is *i-small*, $s_M(x)$ and $\sin(x)$ are *i-close*, since their difference is *i-small*. Since $\sin(x)$ is *standard*, it follows that $\sin(x)$ is equal to the *standard-part* of $s_M(x)$. And as seen earlier, the *standard-part* of $s_M(x)$ is equal to the *standard* part of $s_N(x)$, namely $s(x)$. Since $\sin(x) = {}^\circ s(x)$ for all *standard* $x$, $\sin(x) = {}^\circ s(x)$ for all $x$ by the transfer principle.

We will now use ACL2(r) to formalize the definition of the sine function using the reasoning described above. The key lemma is the fact that $s_N(x)$ is *i-limited* for *i-limited* values of $x$, and this follows from the fact that the Taylor expansion for sine is ultimately an alternating series. We begin by building the necessary theory of alternating series. Sequences are represented as lists, and a series is the sum of a sequence. The sum is computed using the function `sumlist`, defined as follows:

```
(defun sumlist (x)
  (if (consp x)
      (+ (car x) (sumlist (cdr x)))
    0))
```

We are particularly interested in the `sumlist` of alternating sequences, so we develop that theory next. There are two important properties we require of alternating sequences. First, adjacent terms in the sequence have opposite signs. Second, terms in the sequence decrease in magnitude. We define these properties separately, so that we can reason about them independently.

```
(defun opposite-signs-p (x y)
  (or (= x 0)
      (= y 0)
      (equal (sign x) (- (sign y)))))

(defun alternating-sequence-1-p (lst)
  (if (null lst)
      t
    (if (null (cdr lst))
        t
      (and (opposite-signs-p (car lst) (cadr lst))
```

```
            (alternating-sequence-1-p (cdr lst))))))

(defun alternating-sequence-2-p (lst)
  (if (null lst)
      t
    (if (null (cdr lst))
        t
      (and (or (and (equal (car lst) 0)
                    (equal (cadr lst) 0))
               (< (abs (cadr lst))
                  (abs (car lst))))
           (alternating-sequence-2-p (cdr lst))))))

(defun alternating-sequence-p (lst)
  (and (alternating-sequence-1-p lst)
       (alternating-sequence-2-p lst)))
```

Note, the definitions treat zero specially. Zero is considered to be of opposite signs to any number, and adjacent zeros are considered to decrease in magnitude. This allows a sequence with a suffix consisting entirely of zeros to be considered an alternating sequence.

ACL2(r) is able to prove (when the user guides it by posing suitable lemmas) that the sum of an alternating sequence is bounded by its first element. The following lemma is useful in enumerating all the possibilities. Given the alternating sequence $\{a_n\}$, if $a_1$ is positive, it follows that $-a_1 < \sum_2^n a_n \leq 0$. Similarly, if $a_1$ is negative, $-a_1 > \sum_2^n a_n \geq 0$; and if $a_1$ is zero, $\sum_2^n a_n = 0$:

```
(defthm sumlist-alternating-sequence-lemma
  (implies (and (alternating-sequence-p x)
                (real-listp x)
                (consp x))
           (cond ((< 0 (car x))
                  (and (< (- (car x))
                          (sumlist (cdr x)))
                       (<= (sumlist (cdr x)) 0)))
                 ((equal 0 (car x))
                  (and (equal (sumlist x) 0)
                       (equal (sumlist (cdr x)) 0)))
                 ((< (car x) 0)
                  (and (< (sumlist (cdr x))
                          (- (car x)))
                       (<= 0 (sumlist (cdr x)))))))
  :hints ...)
```

Note, ACL2(r) requires the user to provide hints before it is able to prove many of the theorems we present. For ease of presentation, we will omit these hints as done above.

Using this lemma, it is simple to establish the fundamental result of alternating series:

```
(defthm sumlist-alternating-sequence
  (implies (and (alternating-sequence-p x)
                (real-listp x)
                (consp x))
           (<= (abs (sumlist x)) (abs (car x))))
  :hints ...)
```

This result will allow us to show that the sum of an alternating series is *i-small* when the first element of the sequence is *i-small*.

The Taylor expansion for sine can be defined as follows:

```
(defun base-taylor-sin-term (x counter)
  (/ (expt x counter)
     (factorial counter)))

(defun taylor-sin-term (x counter)
  (* (expt -1 counter)
     (base-taylor-sin-term x (1+ (* 2 counter)))))

(defun taylor-sin-list (nterms counter x)
  (if (or (zp nterms)
          (not (integerp counter))
          (< counter 0)
          (not (realp x)))
      nil
    (cons (taylor-sin-term x counter)
          (taylor-sin-list (1- nterms)
                           (1+ counter)
                           x))))
```

The variables `counter` and `nterms` in `taylor-sin-list` are used to keep track of the current term and the numbers of terms remaining, respectively. We would like to define the function `sine` by using the following:

```
(defun-std sine (x)
  (standard-part
   (sumlist (taylor-sin-list (i-large-integer)
                             0
                             x))))
```

However, this definition is inadmissible at this point, because ACL2(r) needs guidance in order to prove that its body is *standard* for all *standard* values of its arguments. What is needed is to show that the value of the `sumlist` is *i-limited* for *i-limited*, or at least *standard*, arguments. To do that, we will show that `taylor-sin-list` can be broken into an *i-limited* prefix and an alternating sequence suffix.

It is easy to establish that the terms in `taylor-sin-list` alternate in sign. A key step in the proof is the following technical lemma, typical of many ACL2 efforts:

```
(defthm taylor-sin-term-x-+1-counter
  (implies (and (integerp counter)
                (<= 0 counter))
           (equal (taylor-sin-term x (+ 1 counter))
                  (* -1 x x
                     (/ (+ 3 (* 2 counter)))
                     (/ (+ 2 (* 2 counter)))
                     (taylor-sin-term x counter))))
  :hints ...)
```

This lemma forces ACL2 to rewrite occurrences of `(taylor-sin-term x (+ 1 counter))` into the given expansion. For real `x` and positive `counter`, it follows that adjacent `taylor-sin-term`s alternate in sign. This is shown by the following lemma:

```
(defthm opposite-signs-p-taylor-sin-term
  (implies (and (integerp counter)
                (<= 0 counter)
                (realp x))
           (opposite-signs-p
            (taylor-sin-term x counter)
            (taylor-sin-term x (1+ counter))))
  :hints ...)
```

It is then trivial for ACL2(r) to prove by induction that the terms in `taylor-sin-list` alternate in sign:

```
(defthm alternating-sequence-1-p-taylor-sin-list
  (alternating-sequence-1-p (taylor-sin-list nterms
                                             counter
                                             x))
  :hints ...)
```

It is harder to show that the terms are (ultimately) decreasing in magnitude. We begin by showing that successive `base-taylor-sin-term` terms decrease, for a sufficiently large `counter`:

```
(defthm abs-base-taylor-sin-term-decreasing
 (implies (and (integerp counter)
               (<= 0 counter)
               (realp x)
               (not (equal x 0))
               (< (abs x) counter))
          (< (abs
              (base-taylor-sin-term x (1+ counter)))
             (abs (base-taylor-sin-term x counter))))
  :hints ...)
```

This result looks at adjacent terms in `base-taylor-sin-term`, but `taylor-sin-term` accesses only the odd elements of `base-taylor--sin-term`. Extending the result above to the case when `counter` is incremented by 2 is trivial.

Using the lemma presented above, ACL2 can prove that successive `taylor-sin-term` terms (ultimately) decrease in magnitude.

```
(defthm abs-taylor-sin-term-decreasing
  (implies (and (integerp counter)
                (<= 0 counter)
                (realp x)
                (not (equal x 0))
                (< (abs x) counter))
           (< (abs (taylor-sin-term x (1+ counter)))
              (abs (taylor-sin-term x counter))))
  :hints ...)
```

In turn, this allows ACL2 to establish that `taylor-sin-list` has an alternating sequence suffix.

```
(defthm alternating-sequence-p-taylor-sin-list
  (implies (< (abs x) counter)
           (alternating-sequence-p
            (taylor-sin-list nterms counter x)))
  :hints ...)
```

This allows us to place a bound on the sumlist of all but the first $\lceil |x| \rceil$ elements of `taylor-sin-list`. We will now show that the sum of these first $\lceil |x| \rceil$ elements must be *i-limited*.

First, we establish that `taylor-sin-term` is *i-limited* for *i-limited* values of `x` and `counter`.

```
(defthm limited-taylor-sin-term
  (implies (and (<= 0 counter)
```

```
                (i-limited counter)
                (i-limited x))
           (i-limited (taylor-sin-term x counter)))
  :hints ...)
```

The term `(taylor-sin-term x counter)` can be broken down into the product of `(expt -1 counter)`, which is either $-1$ or 1 and so clearly *i-limited*, `(/ (factorial counter))` which is a positive number at most equal to 1 and so is also *i-limited*, and `(expt x counter)`.

So it is only necessary to show that `(expt x n)` is *i-limited* when `x` and `n` are *i-limited* and `n` is non-negative[5]:

```
(defthm expt-limited
  (implies (and (<= 0 n)
                (i-limited n)
                (i-limited x))
           (i-limited (expt x n)))
  :hints ...)
```

This is an interesting theorem, because it illustrates the use of induction on a non-classical theorem. Besides the usual proof obligations required by induction in ACL2, ACL2(r) must establish the theorem is true when `n` is non-*standard*. This latter case is trivially true, since `n` is *i-limited* by hypothesis, hence it is either *standard* or not an integer, in which case `(expt x n)` is defined to be 0.

A similar non-standard induction is required to prove the `sumlist` of the terms in `taylor-sin-list` is *i-limited*:

```
(defthm taylor-sin-list-limited-up-to-limited-counter
  (implies (and (i-limited nterms)
                (integerp counter)
                (i-limited counter)
                (i-limited x))
           (i-limited
            (sumlist
             (taylor-sin-list nterms counter x))))
  :hints ...)
```

Aside from the use of non-standard induction, this is a straightforward consequence of `limited-taylor-sin-term`.

We are ready to complete the proof obligation generated by the above definition of `sine` using `defun-std`. First we show that the sum of the first $\lceil |x| \rceil$ terms is *i-limited*, by instantiating the theorem above with $\lceil |x| \rceil$ in place of `counter`:

[5] The counterexample `(expt` $\epsilon$ $-1$`)` illustrates why `n` must be non-negative.

```
(defthm taylor-sin-list-limited-lemma-1
  (implies (and (realp x)
                (i-limited x))
           (i-limited
            (sumlist
             (taylor-sin-list (next-integer (abs x))
                              0
                              x))))
  :hints ...)
```

It remains to show that the `sumlist` of terms in `taylor-sin-list` following the $\lceil |x| \rceil^{\text{th}}$ term is also *i-limited*. This result follows from the fact that the `taylor-sin-list` is an alternating sequence beyond the $\lceil |x| \rceil^{\text{th}}$ term, and moreover that this term is *i-limited*; see `sumlist--alternating-sequence` and `alternating-sequence-p-taylor-sin--list` above.

```
(defthm taylor-sin-list-limited-lemma-2
  (implies (and (integerp nterms)
                (<= 0 nterms)
                (realp x)
                (i-limited x))
           (i-limited (sumlist
                       (taylor-sin-list
                        nterms
                        (next-integer (abs x))
                        x))))
  :hints ...)
```

The only remaining detail is to show how a sequence can be split into a prefix and suffix. This is handled with the following theorems. Their ACL2(r) proofs illustrate induction's important role in non-standard analysis.

```
(defthm taylor-sin-list-split
  (implies (and (integerp n1)      (<= 0 n1)
                (integerp n2)      (<= 0 n2)
                (integerp counter) (<= 0 counter))
           (equal
            (taylor-sin-list (+ n1 n2) counter x)
            (append (taylor-sin-list n1 counter x)
                    (taylor-sin-list n2
                                     (+ counter n1)
                                     x)))))
```

```
  :hints ...)

(defthm sumlist-append
  (equal (sumlist (append x y))
         (+ (sumlist x) (sumlist y)))
  :hints ...)
```

These theorems show that a sequence can be split at any point, and that its sum can be recovered from the sum of the parts. When `n1` is equal to $\lceil |x| \rceil$ and `n2` is equal to `i-large-integer` $- \lceil |x| \rceil$, this splits the `taylor-sin-list` into components that can be processed using the `limited-lemmas` above. This allows ACL2(r) to prove that the sum of a `taylor-sin-list` is *i-limited*:

```
(defthm taylor-sin-list-limited
  (implies (i-limited x)
           (i-limited
            (sumlist
             (taylor-sin-list (i-large-integer)
                              0
                              x))))
  :hints ...)
```

This is precisely the proof obligation needed to use `defun-std` to introduce the sine function, so we can now proceed to do so:

```
(defun-std sine (x)
  (standard-part
   (sumlist (taylor-sin-list (i-large-integer)
                             0
                             x))))
```

It is worth emphasizing that the resulting function `sine` is equal to the (standard) trigonometric sine function, even though ACL2(r) can only open up the definition for *standard* values of `x`.

Next, we can prove some theorems about the sine function. An easy theorem is that $\sin(-x) = -\sin(x)$. To show this, we begin with the analogous result for `taylor-sin-term`:

```
(defthm taylor-sin-term-uminus
  (implies (and (realp x)
                (integerp counter)
                (<= 0 counter))
           (equal (taylor-sin-term (- x) counter)
                  (- (taylor-sin-term x counter))))
  :hints ...)
```

This result can be easily generalized to `taylor-sin-list` as follows:

```
(defthm taylor-sin-list-uminus
  (implies (realp x)
           (equal
            (sumlist (taylor-sin-list nterms
                                      counter
                                      (- x)))
            (- (sumlist (taylor-sin-list nterms
                                         counter
                                         x)))))
  :hints ...)
```

The transfer principle can be used to extend this result to the sine function as follows:

```
(defthm-std sine-uminus
  (implies (realp x)
           (equal (sine (- x))
                  (- (sine x))))
  :hints ...)
```

The transfer principle allows ACL2(r) to add `(standard-numberp x)` as a hypothesis to the theorem above. Usage of the transfer principle is justified since the theorem does not refer to any non-standard functions. In particular, notice that the `sine` function is standard, since it was introduced using `defun-std`. Since the hypothesis is now restricted to *standard* numbers, ACL2(r) can expand the body of the `sine` function, and the result trivially follows from the lemma `taylor-sin-list--uminus` and an axiom that `(standard-numberp (- x))` is equal to `(standard-numberp x)` for numeric `x`.

We close this section with an example illustrating computation with the `sine` function. Although direct computation is impossible, since the body of `sine` uses the undefined constant `i-large-integer`, it is possible to find good approximations.

For example, a `taylor-sin-list` can be split into its first three terms and the remainder. Using the theorems already proved, it is easy to show that under suitable restrictions, the remaining terms add up to no more than the fourth term in the `taylor-sin-list`. Indeed, we have proved the following theorem using ACL2(r):

```
(defthm taylor-sin-approx-by-3-error-best
  (implies (and (realp x)
                (< (abs x) 3))
           (<= (standard-part
```

```
                    (abs
                     (- (sumlist (taylor-sin-list
                                  (i-large-integer)
                                  0
                                  x))
                        (sumlist
                         (taylor-sin-list 3 0 x)))))
                  (standard-part
                   (abs (car (taylor-sin-list
                              (- (i-large-integer) 3)
                              3
                              x))))))
      :hints ...)
```

The value of `(sumlist (taylor-sin-list 3 0 1/2))` can be directly computed; it is `1841/3840`. Similarly, the value of `(car (taylor--sin-list (- (i-large-integer) 3) 3 x))`, the fourth term in the Taylor expansion for $sin(1/2)$, can be computed to be `1/645120`. Thus, ACL2(r) can quickly compute a good approximation for `(sine 1/2)`:

```
(defthm sine-one-half
  (<= (abs (- (sine 1/2)
              1841/3840))
      1/645120)
  :hints ...)
```

More trigonometric examples, including approximation schemes to the sine and cosine functions and a proof of the correctness of the Fast Fourier Transform, can be found in [15]. Examples from analysis, including the fundamental theorem of calculus, appear in [16, 19].

## 5. Summary

In this paper, we described ACL2(r), a version of ACL2 with support for non-standard analysis. In particular, we showed how it is possible to reason mechanically about transcendental functions such as the square root and trigonometric functions.

A more traditional approach to mechanizing the real number line is provided by standard analysis. Typically, this involves adding a completeness axiom to the theorem prover. Such an approach is followed by PVS [29, 10], IMPS [11], MIZAR [32, 35], and others. Harrison presents a constructive approach in his dissertation [17], where he takes advantage of the expressive power of HOL. Our approach in ACL2(r) has to differ

from the above because of ACL2's comparatively weak logic — in particular, it offers no support for infinite objects and only minimal support for quantifiers. The trade-off is that ACL2(r) can draw on its relatively powerful proof engine, in particular its support for recursion and induction, which are central to many arguments in non-standard analysis.

Fleuriot and Paulson describe a system using non-standard analysis in geometry [12]. Their experience proving theorems from Newton's *Principia* illustrates the power of non-standard analysis. Newton's intuitive proofs find elegant counterparts mechanized in Isabelle.

Bledsoe developed several theorem provers capable of proving many results from elementary analysis, such as the intermediate value theorem [3, 4]. Ballantyne and Bledsoe describe a version of Bledsoe's theorem prover IMPLY that proves theorems in the theory of non-standard analysis [1, 2]. Using this system, they are able to prove several theorems of elementary analysis, including the equivalence of the "standard" and "non-standard" definitions of the basic analysis concepts. For example, they present a mechanical proof that the traditional definition of sequence convergence is equivalent to the non-standard version. Although there are some clear similarities, ACL2(r) differs significantly from Bledsoe and Ballantyne's prover. Specifically, ACL2(r) benefits from using ACL2's powerful induction engine. Recursion and induction play a central role in many proofs from non-standard analysis. For example, the intermediate value theorem can be proved by defining a step function that approximates the desired continuous function. The intermediate value theorem follows from induction on the number of "steps" used in the interval of interest, to prove that a zero lies at the standard part of a point found by recursion.

It is significant that ACL2(r) can prove any theorems at all from analysis, since its language, with weak support for quantifiers and without infinite sets, may appear too barren for analysis. The results presented here and in [14, 15, 16, 19] show that non-standard analysis is a natural way to reason about the reals not just in the context of ACL2, but in the more general context of a rewrite-based theorem prover, particularly one with support for induction and recursion.

## Acknowledgements

The first author thanks Bob Boyer for his dedicated supervision of his doctoral work. The second author thanks his former manager Joe Hill of EDS, Inc. for providing time for this effort. We also thank J Moore

for collaborating with us on the integration of this work into the ACL2 release.

## Appendix

### A. Soundness of Non-Standard Induction in ACL2(r)

In this section, we present a proof of the soundness of the non-standard induction principle in ACL2(r). The treatment is based on the soundness proof of the Nqthm and ACL2 induction principles given in [6, 22].

The non-standard induction principle is as follows:

Suppose:

- $p$ is a term;
- $r$ is a classical function symbol that denotes a well-founded relation;
- $m$ is a classical function symbol of `n` arguments;
- $x_1, \ldots, x_n$ are distinct variables;
- $q_1, \ldots, q_k$ are terms;
- $h_1, \ldots, h_k$ are positive integers;
- for $1 \leq i \leq k$ and $1 \leq j \leq h_i$, $s_{i,j}$ is a classical substitution, and it is a theorem that

  `(IMPLIES` $q_i$
  `(`$r$ `(`$m\ x_1 \ldots x_n$`)/`$s_{i,j}$ `(`$m\ x_1 \ldots x_n$`)))`

  and
- $y_1, \ldots, y_u$ are the variables occurring in $p$ that are one of the $x_i$ or are changed by the $s_{i,j}$.

Then $p$ is a theorem if

`(IMPLIES (AND (NOT` $q_1$`) ...(NOT` $q_k$`))`
$p$`)`

is a theorem, for each $1 \leq i \leq u$,

`(IMPLIES (NOT (STANDARD-NUMBERP` $y_i$`))`
$p$`)`

is a theorem, and for each $1 \leq i \leq k$,

(IMPLIES (AND $q_i$ $p/s_{i,1}$ ...$p/s_{i,h_i}$)
$p$)

is a theorem.

Compare this to the formal definition of the induction principle of Nqthm or ACL2, found in [6, 7, 22].

To understand why the non-standard induction principle is sound, consider a specific choice $p$, $r$, $m$, $x_i$, $q_i$, $h_i$, $s_{i,j}$ such that the conditions above can be established. Then the following proof in the language of non-standard analysis establishes the validity of $p$.

PROOF: Without loss of generality, assume that the $x_i$ are $X_1$, $X_2$, ..., $X_n$; that $r$ is $R$; that $m$ is $M$; that $X_{n+1}$, $X_{n+2}$, ..., $X_z$ are all of the variables other than $X_1$, $X_2$, ..., $X_n$ occurring free in $p$, the $q_i$, or either component of any pair in any $s_{i,j}$; that $p$ is $(P\ X_1 \ldots X_z)$; that $q_i$ is $(Q_i\ X_1 \ldots X_z)$; that $s_{i,j}$ replaces $X_v$ with some term $d_{i,j,v}$; and that the $Y_i$ are given by $X_1$, $X_2$, ..., $X_u$, for some $n \leq u \leq z$. Note that $d_{i,j,v}$ is equal to $X_v$ for $u < v \leq z$.

Let $RM$ be the function on $u$-tuples defined by

$$(RM\ \langle U_1 \ldots U_u\rangle\ \langle V_1 \ldots V_u\rangle) = (R\ (M\ U_1 \ldots U_n)\ (M\ V_1 \ldots V_n)).$$

Note that $RM$ is classical and well-founded.

Let the tuple $C = \langle C_{u+1}\ C_{u+2} \ldots C_z\rangle$ be a binding for the tuple of variables $\langle X_{u+1}\ X_{u+2} \ldots X_z\rangle$. Define the set $GC$ as the shadow set of all $u$-tuples $U$ for which $(P\ U\ C)$ is false. That is, it is defined as follows:

$$GC = {}^\circ\{\langle U_1 \ldots U_u\rangle \mid (P\ U_1 \ldots U_u\ C_{u+1}\ C_{u+2} \ldots C_z) \text{ is false}\}$$

We will show that $GC$ is empty. Let us complete the proof of the theorem pending the proof of this claim. From the definition of $GC$ (and shadow), it then follows that for any standard tuple $\langle X_1 \ldots X_u\rangle$, $(P\ X_1 \ldots X_u\ C_{u+1} \ldots C_z)$ must be true. If $\langle X_1 \ldots X_u\rangle$ is a non-standard tuple, it follows from the assumptions that $(P\ X_1 \ldots X_u\ C_{u+1} \ldots C_z)$ is also true. This is because if $\langle X_1 \ldots X_u\rangle$ is non-standard, one of the $X_i$ must be non-standard, and then the theorem follows from the hypothesis

(IMPLIES (NOT (STANDARD-NUMBERP $X_i$))
$p$)

This establishes that $(P\ X_1 \ldots X_u\ C_{u+1} \ldots C_z)$ is true for all tuples $\langle X_1 \ldots X_u\rangle$. Since the $C_i$ are arbitrary, for all possible values of the $X_i$, it follows that $(P\ X_1 \ldots X_u\ X_{u+1} \ldots X_z)$ is true. This establishes the validity of $p$. It remains only to show that $GC$ is empty.

Since $GC$ is a standard set, membership in $GC$ can be decided using the classical principle of induction. In particular, if $GC$ is non-empty, it must have an $RM$-minimal tuple. Moreover, $GC$ is a standard set, so by the transfer principle, if it is non-empty, it must have a standard $RM$-minimal tuple. Let $\langle X_1\ X_2 \dots X_u \rangle$ be such a tuple. We obtain a contradiction in each of the two cases to consider.
*Case 1:* Suppose none of the $q_i$ is true. $(P\ X_1 \dots X_u\ C_{u+1} \dots C_z)$ is true by the base case, so $\langle X_1 \dots X_u \rangle$ should not be in $GC$, yielding a contradiction.
*Case 2:* Suppose at least one of the $q_i$ is true. Without loss of generality, assume that the term $(Q_1\ X_1 \dots X_u\ C_{u+1} \dots C_z)$ is true. From the conditions on $r$, $m$, $q$, and $s_{i,j}$, it follows that

$$\begin{gathered}(R\ (M\ d_{1,1,1} \dots d_{1,1,n})\ (M\ X_1 \dots X_n)) \\ (R\ (M\ d_{1,2,1} \dots d_{1,2,n})\ (M\ X_1 \dots X_n)) \\ \vdots \\ (R\ (M\ d_{1,h_1,1} \dots d_{1,h_1,n})\ (M\ X_1 \dots X_n))\end{gathered}$$

are all true. By the definition of $RM$,

$$\begin{gathered}(RM\ \langle d_{1,1,1} \dots d_{1,1,u} \rangle\ \langle X_1 \dots X_u \rangle) \\ (RM\ \langle d_{1,2,1} \dots d_{1,2,u} \rangle\ \langle X_1 \dots X_u \rangle) \\ \vdots \\ (RM\ \langle d_{1,h_1,1} \dots d_{1,h_1,u} \rangle\ \langle X_1 \dots X_u \rangle)\end{gathered}$$

are all true as well. Observe, the terms $d_{1,i,j}$ are all standard, since the $X_i$ are standard, and the substitutions $s_{1,i}$ are assumed classical, hence they return standard values for standard arguments. Since $\langle X_1 \dots X_u \rangle$ is an $RM$-minimal $u$-tuple such that $(p\ U\ C)$ is false and the $d_{1,i,j}$ are all standard, it follows that

$$\begin{gathered}(P\ d_{1,1,1} \dots d_{1,1,u}\ C_{u+1} \dots C_z) \\ (P\ d_{1,2,1} \dots d_{1,2,u}\ C_{u+1} \dots C_z) \\ \vdots \\ (P\ d_{1,h_1,1} \dots d_{1,h_1,u}\ C_{u+1} \dots C_z)\end{gathered}$$

are all true. Hence, $(P\ X_1 \dots X_u\ C_{u+1} \dots C_z)$ follows from the first induction step, contradicting the assumption that $\langle X_1 \dots X_u \rangle$ is in $GC$.

The contradiction above establishes that $GC$ is empty, which as explained above concludes the proof of the soundness of the non-standard induction principle.

## References


1. Ballantyne, A. and W. W. Bledsoe: 1977, 'Automatic Proofs of Theorems in Analysis Using Non-Standard Techniques'. *Journal of the Association for Computing Machinery (JACM)* **24**(3), 353–371.
2. Ballantyne, A. M.: 1991, 'The Metatheorist: Automatic Proofs of Theorems in Analysis Using Non-Standard Techniques, Part II'. In: R. S. Boyer (ed.): *Automated Reasoning: Essays in Honor of Woody Bledsoe*. Kluwer Academic Publishers, pp. 61–75.
3. Bledsoe, W. W.: 1983, 'The UT Natural Deduction Prover'. Technical Report ATP-17B, University of Texas at Austin.
4. Bledsoe, W. W.: 1984, 'Some Automatic Proofs in Analysis'. *Contemporary Mathematics* **29**.
5. Boyer, R. S., D. Goldschlag, M. Kaufmann, and J. S. Moore: 1991, 'Functional Instantiation in First Order Logic'. In: V. Lifschitz (ed.): *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*. pp. 7–26.
6. Boyer, R. S. and J. S. Moore: 1979, *A Computational Logic*. Orlando: Academic Press.
7. Boyer, R. S. and J. S. Moore: 1988, *A Computational Logic Handbook*. San Diego: Academic Press.
8. Brock, B., M. Kaufmann, and J. S. Moore: 1996, 'ACL2 Theorems about Commercial Microprocessors'. In: M. Srivas and A. Camilleri (eds.): *Formal Methods in Computer-Aided Design (FMCAD'96)*. pp. 275–293.
9. Diener, F. and M. Diener (eds.): 1995, *Nonstandard Analysis in Practice*. Springer.
10. Dutertre, B.: 1996, 'Elements of Mathematical Analysis in PVS'. In: *Proceedings of the Ninth International Conference on Theorem Proving in Higher-Order Logics (TPHOL'96)*.
11. Farmer, W. M., J. D. Guttman, and F. J. Thayer: 1993, 'IMPS: An Interactive Mathematical Proof System'. *Journal of Automated Reasoning* **11**(2), 213–248.
12. Fleuriot, J.: 1999, 'A Combination of Geometry Theorem Proving and Nonstandard Analysis with Application to Newton's Principia'. Ph.D. thesis, University of Cambridge.
13. Gamboa, R.: 1996, 'Square Roots in ACL2: A Study in Sonata Form'. Technical Report CS-TR-96-34, University of Texas at Austin.
14. Gamboa, R.: 1998, 'Mechanically Verifying the Correctness of the Fast Fourier Transform in ACL2'. In: J. Rolim (ed.): *Parallel and Distributed Processing*. pp. 796–806.
15. Gamboa, R.: 1999, 'Mechanically Verifying Real-Valued Algorithms in ACL2'. Ph.D. thesis, The University of Texas at Austin.
16. Gamboa, R.: 2000, 'Continuity and Differentiability in ACL2'. In: M. Kaufmann, P. Manolios, and J. S. Moore (eds.): *Computer-Aided Reasoning: ACL2 Case Studies*. Kluwer Academic Press, Chapt. 18.
17. Harrison, J.: 1996, 'Theorem Proving with the Real Numbers'. Ph.D. thesis, University of Cambridge.
18. Jr., G. L. S.: 1990, *Common LISP The Language*. Bedford, Massachusetts: Digital Press, 2nd edition.
19. Kaufmann, M.: 2000, 'Modular Proof: The Fundamental Theorem of Calculus'. In: M. Kaufmann, P. Manolios, and J. S. Moore (eds.): *Computer-Aided Reasoning: ACL2 Case Studies*. Kluwer Academic Press, Chapt. 6.

20. Kaufmann, M., P. Manolios, and J. S. Moore: 2000, *Computer-Aided Reasoning: An Approach*. Kluwer Academic Press.
21. Kaufmann, M. and J. S. Moore, 'ACL2: A Computational Logic for Applicative Common Lisp, The User's Manual'. Available on the world-wide web at `http://www.cs.utexas.edu/users/moore/acl2/acl2-doc.html`.
22. Kaufmann, M. and J. S. Moore, 'A Precise Description of the ACL2 Logic'. Available on the world-wide web at `http://www.cs.utexas.edu/users/moore/publications/km97a.ps.Z`.
23. Kaufmann, M. and J. S. Moore: 1994, 'Design Goals for ACL2'. Technical Report 101, Computational Logic, Inc. See URL `http://www.cs.utexas.edu/-users/moore/publications/acl2-paper s.html#Overviews`.
24. Kaufmann, M. and J. S. Moore: 2001, 'Structured Theory Development for a Mechanized Logic'. *Journal of Automated Reasoning* **26**(1), 161–203.
25. Keisler, H. J.: 1976, *Elementary Calculus*. Boston: Prindle, Weber and Schmidt.
26. Moore, J. S., T. Lynch, and M. Kaufmann: 1998, 'A Mechanically Checked Proof of the AMD5$_K$86 Floating-Point Division Program'. *IEEE Trans. Comp.* **47**(9), 913–926.
27. Nelson, E., 'On-Line Books: Internal Set Theory'. Available on the world-wide web at `http://www.math.princeton.edu/\~nelson/books.html`.
28. Nelson, E.: 1977, 'Internal Set Theory'. *Bulletin of the American Mathematical Society* **83**, 1165–1198.
29. Owre, S., S. Rajan, J. M. Rushby, N. Shankar, and M. K. Srivas: 1996, 'PVS: Combining Specification, Proof Checking, and Model Checking'. In: R. Alur and T. A. Henzinger (eds.): *Computer-Aided Verification, CAV '96*, Vol. 1102 of *Lecture Notes in Computer Science*. New Brunswick, NJ, pp. 411–414.
30. Robert, A.: 1988, *Non-Standard Analysis*. John Wiley.
31. Robinson, A.: 1996, *Non-Standard Analysis*. Princeton University Press.
32. Rudnicki, P.: 1992, 'An Overview of the MIZAR Project'. In: *Proceedings of the 1992 Workshop on Types for Proofs and Programs*.
33. Russinoff, D.: 1998, 'A Mechanically Checked Proof of IEEE Compliance of a Register-Transfer-Level Specification of the AMD-K7 Floating-Point Multiplication, Division, and Square Root Instructions'. *London Mathematical Society Journal of Computation and Mathematics* **1**, 148–200.
34. Russinoff, D.: 1999, 'A Mechanically Checked Proof of Correctness of the AMD-K5 Floating-Point Square Root Microcode'. *Formal Methods in System Design* **14**, 75–125.
35. Trybulec, A.: 1978, 'The Mizar-QC/6000 Logic Information Language'. *Bulletin of the Association for Literary and Linguistic Computing (LLAC)* **6**(2).