A Cantor Trio: Denumerability, the Reals, and the Real Algebraic Numbers

Ruben Gamboa and John Cowles

University of Wyoming, Laramie, WY, USA
{ruben,cowles}@uwyo.edu
http://www.cs.uwyo.edu/~ruben

Abstract. We present a formalization in ACL2(r) of three proofs originally done by Cantor. The first two are different proofs of the non-denumerability of the reals. The first, which was described by Cantor in 1874, relies on the completeness of the real numbers, in the form that any infinite chain of closed, bounded intervals has a non-empty intersection. The second proof uses Cantor's celebrated diagonalization argument, which did not appear until 1891. The third proof is of the existence of real transcendental (i.e., non-algebraic) numbers. It also appeared in Cantor's 1874 paper, as a corollary to the non-denumerability of the reals. What Cantor ingeniously showed is that the algebraic numbers are denumerable, so every open interval must contain at least one transcendental number.

Key words: ACL2, nonstandard analysis, non-denumerability of the reals, denumerability of algebraic numbers

1 Introduction

In an important paper first published in 1874[1, 2] and later popularized by Dunham among others [3], Cantor presented a new proof of the existence of transcendental numbers, those numbers that are not the root of any polynomial with rational coefficients. Cantor's proof was quite unlike Liouville's earlier demonstration of a transcendental number. While Liouville actually constructed a transcendental number (indeed, a whole family of them), Cantor used a counting argument to show that the set of real numbers must include many transcendental numbers.

Cantor's counting argument proceeds as follows. First, he showed that no sequence $\{x_1, x_2, \dots\}$ of real numbers can completely enumerate all the numbers in an open interval (a, b). That is, there must be some $x \in (a, b)$ such that $x \notin \{x_1, x_2, \dots\}$. Second, he constructed an enumeration of the algebraic numbers, that is, all the roots of all polynomials with rational coefficients. Since the algebraic numbers could be placed in a sequence, it followed that every non-empty open interval must contain at least one number that is not among the algebraic numbers, i.e., a transcendental number.

Although Cantor's 1874 paper emphasized the application to transcendental numbers, the more revolutionary result was the non-denumerability of the real numbers! In 1891, Cantor proved, by diagonalization, that non-denumerable sets exist. This diagonalization proof is easily adapted to showing that the reals are non-denumerable, which is the proof commonly presented today [4, 2].

We present a formalization of Cantor's two proofs of the non-denumerability of the reals in ACL2(r). In addition, we present a formalization of Cantor's application of this theorem to the existence of transcendental numbers. The formalizations rely on some uncommon techniques of ALC2(r). So we begin the presentation in Sect. 2 by briefly describing ACL2(r) and the main techniques on which the proofs rely. We follow in Sect. 3 with the formalization of Cantor's two proofs of the non-denumerability of the reals. Then in Sect. 4 we present Cantor's enumeration of the algebraic numbers, which immediately establishes the existence of an infinite number of transcendental numbers. Finally, we present some concluding remarks in Sect. 5.

2 Background: ACL2(r)

In this section, we briefly describe ACL2(r), a variant of ACL2 with support for the real numbers, including an overview of nonstandard analysis, the foundational theory of ACL2(r). Our description is limited to those elements of ACL2 and ACL2(r) that are needed for the main results described later in this paper. Readers familiar with ACL2 or ACL2(r) may wish to skip this section.

In the tradition of the Boyer-Moore family of theorem provers, ACL2 is a first-order theorem prover with a syntax similar to Common Lisp's [5]. The primary inference rules are term rewriting with equality (and with respect to other equivalence relations) and induction up to ϵ_0. ACL2 supports the explicit introduction of new function symbols via explicit definitions as well as implicitly via constraints, using the events[1] `defun` and `encapsulate`, respectively. In addition, ACL2 permits the introduction of "choice" functions via Skolem axioms using the `defchoose` event. For example, let $\phi(x, y)$ be a formula whose only free variables are x and y. Then the Skolem axiom introducing the function f from the formula $\phi(x, y)$ with respect to y is

$$(\forall x, y)(\phi(x, y) \Rightarrow \phi(x, f(x)))$$

What this axiom states is that the new function f can "choose" an appropriate y for a given x as long as such a y exists. For example, if $\phi(x, y)$ states that $y^2 = x$, then the choice function will select one of $\pm\sqrt{x}$ for a non-negative real x. What it does for other values of x is unspecified.

Choice functions in ACL2 are also used to define expressions that capture the essence of existential and universal quantifiers in the event `defun-sk`. For example, the following event captures the concept that an object has a square root:

[1] An ACL2 "event" is the unit of interaction between the user and the theorem prover, e.g., a command.

```
(defun-sk exists-square-root (x)
 (exists (y)
   (and (realp y)
        (equal (* y y) x))))
```

These "quantification" functions can then be used in theorems, such as the following, which states that non-negative reals have square roots:

```
(defthm nonneg-reals-have-square-roots
 (implies (and (realp x) (<= 0 x))
          (exists-square-root x)))
```

Choice functions in ACL2 are also used to justify the definition of "partial" functions with the event `defpun` [6]. The basic idea behind `defpun` is that under certain circumstances a recursive expression can be used to define a function symbol, even when there is no guarantee that the function terminates for all inputs. For example, consider the following function, which returns the next highest prime number.

```
(defpun next-prime (n)
 (if (primep (1+ n))
     (1+ n)
   (next-prime (1+ n))))
```

A naive attempt to define `next-prime` by replacing `defpun` with `defun` will fail, because ACL2 is unable to find a measure that decreases in the recursive call. However, using `defpun`, the definition is admitted. The function `next-prime` defined in this way is not truly partial, because it has a value for every input. Rather, it is underspecified, because its value is only known for certain input values. E.g., it would be possible to prove that `(next-prime 80)` is equal to 83, but it would not be possible to prove what `(next-prime 1/2)` is equal to, even though it must surely be equal to *something*, since ACL2 is a logic of total functions.

ACL2(r) modifies the base ACL2 theorem prover by introducing notions from nonstandard analysis, as axiomatized by Nelson [7, 8]. In Nelson's formulation of nonstandard analysis, the real numbers can be further characterized as standard, small, limited, or large. The standard reals include all the real numbers that can be uniquely characterized, such as 0, 1, π, $\sqrt{2}$, etc. Small reals, also called infinitesimals, are those that are smaller in magnitude than any non-zero standard real. Zero is the only standard number that is also small, but there are other small numbers. Necessarily, there are also large numbers, namely those that are larger in magnitude than all standard reals. Real numbers x that are not large are called limited, and they can always be written as $x = {}^*x + \epsilon$, where ${}^*x$ is standard and ϵ is small. The number ${}^*x$ is called the standard part of x. Finally, two numbers are said to be close if their difference is small. It is important to note that all the usual algebraic properties of the real numbers are still true in nonstandard analysis. E.g., $x \cdot 1/x = 1$ for all non-zero x, whether x is small, large, limited, standard, etc. Also, the properties close, small, and so on

have nice algebraic properties. E.g., the sum of two small numbers is small, the product of a small and a limited number is small, and the standard part of the sum of two numbers is the sum of their standard parts.

Formulas and functions in nonstandard analysis are said to be classical if they do not mention any of the "new" functions of nonstandard analysis, i.e., standard, large, etc. Thus, all functions of traditional analysis, such as square root and sine, are classical. One of the most important principles of nonstandard analysis is the transfer principle, which states that any first-order classical formula that is true of all standard values must also be true of all values. That is, in order to prove that a classical formula $P(x)$ is true of all x, it is sufficient to prove that $standard(x) \Rightarrow P(x)$. This principle is captured in the ACL2(r) event `defthm-std`. This same principle also justifies an indirect definitional principle, where only the values of $f(x)$ for standard values of x are specified. Under certain conditions, this is sufficient to define f as the only classical function that maps x to $f(x)$ for all standard values of x.

We conclude this section by mentioning that many of the traditional notions of analysis can be stated more naturally in nonstandard analysis. For example, we say that a sequence of real numbers $\{a_1, a_2, \dots\}$ converges to a value A if and only if a_N is close to A for all large values of N. This is remarkably simpler than the traditional "epsilon" definition of convergence.

3 Non-Denumerability of the Reals

In this section, we present two proofs of the non-denumerability of the reals. We start with Cantor's 1874 proof, based on the completeness of the real number line [1]. Then we formalize the more familiar proof based on his 1891 paper [4]. In both cases, we present first a mathematical description of the proof that we actually formalized in ACL2(r), and then we present highlights from the formalization.

Note that this is not the first formalization of the non-denumerability of the reals! As of this writing, there are five others in Freek Wiedijk's list, Formalizing 100 Theorems [9]—and this list is not comprehensive. However, our interest here is not just to prove that the reals are non-denumerable, but to formalize *Cantor's actual arguments* in ACL2(r).

3.1 The First Proof: Using the Completeness of the Reals

3.1.1 The Informal Argument Consider a sequence $\{s_n\}$ of real numbers and a real interval (a, b). Cantor showed that there must be at least one $x \in (a, b)$ such that $x \notin \{s_n\}$. The following argument is a slight variant of Cantor's argument, which we formalized in ACL2(r).

First, construct an infinite chain of nested, closed, and bounded intervals as follows. Let $[a_0, b_0] = [a, b]$. The interval $[a_1, b_1] \subsetneq [a_0, b_0]$ is then defined as $[s_{i_1}, s_{j_1}]$, where i_1 and j_1 are chosen to be the smallest indexes such that $i_1 < j_1$ and $s_{i_1} < s_{j_1}$ are both in (a_0, b_0)—as long as such indexes can be found. Repeat

this process, so that i_n and j_n as the smallest indexes such that $j_{n-1} < i_n < j_n$ and $s_{i_n} < s_{j_n}$ are both in (a_{n-1}, b_{n-1})[2]. By construction, if appropriate indexes can be found at every step, the intervals $[a_n, b_n]$ form an infinite chain of nested, closed, and bounded intervals.

However, the construction can fail at a step if no i_n and j_n can be found such that $j_{n-1} < i_n < j_n$ and s_{i_n} and s_{j_n} are both in (a_{n-1}, b_{n-1}). But if this is the case, we have found a point in $[a_{n-1}, b_{n-1}] \subset (a, b)$ that cannot be one of the $\{s_n\}$, as desired. This is because either (i) for all $i_n > j_{n-1}$, $s_{i_n} \notin (a_{n-1}, b_{n-1})$ or (ii) if i_n is the first index such that $i_n > j_{n-1}$ and $s_{i_n} \in (a_{n-1}, b_{n-1})$, then for all $j_n > i_n$ if $s_{j_n} \in (a_{n-1}, b_{n-1})$, then $s_{j_n} \leq s_{i_n}$. In case (ii) holds, then for all $j_n > i_n$, $s_{j_n} \notin (s_{i_n}, b_{n-1})$.

Conversely, suppose the construction does succeed in building an infinite chain of nested intervals. Then there is some point x such that $x \in [a_n, b_n]$ for all n. The claim is that x is not any of the s_n. This follows by considering the possible values of the indexes i_n and j_n. First, it's clear that $i_1 \geq 1$ and $j_n \geq 2$, since these are the first two indexes that can be considered; in general, $i_n \geq 2n - 1$ and $j_n \geq 2n$. Second, we observe that for i such that $j_1 < i < i_2$, $s_i \notin (a_1, b_1)$, since i_2 is chosen to be the first i in that interval. Moreover, for i such that $i_1 < i < j_1$, we also know that $s_i \notin (a_1, b_1)$, since j_1 is the first index such that $j_1 > i_1$ and $s_{j_1} \in [a_0, b_0]$. Combining and generalizing these facts, it follows that for i in the range $i_n \leq i < i_{n+1}$, $s_i \notin (a_n, b_n)$. This statement can be used to show, by induction, that for all i in the range $1 \leq i < i_{n+1}$, $s_i \notin (a_n, b_n)$.

Finally, the two observations above can be combined to observe that if $1 \leq i < 2n+1$, $s_i \notin (a_n, b_n)$, since $i_{n+1} \geq 2(n+1)-1$. In particular, $s_n \notin (a_n, b_n)$ for any n. Since $(a_n, b_n) \supsetneq [a_{n+1}, b_{n+1}]$, it follows that $s_n \neq x$, since x was chosen previously such that $x \in [a_n, b_n]$ for all n.

3.1.2 The Completeness of the Reals Cantor's proof makes use of the fact that the real numbers are complete, in the form that a sequence of nested, closed, bounded intervals has a non-empty intersection. It is necessary, therefore, to formalize this result in ACL2(r). To do so, we introduce the constrained function `nested-interval`, which represents a sequence of closed intervals, i.e., a mapping from each positive integer n to a pair of real numbers a_n and b_n such that $a_n \leq b_n$. In addition, the intervals are constrained to form a nested chain by requiring that $a_m \leq a_n \leq b_n \leq b_m$ whenever $m \leq n$.

We limit ourselves to standard sequences of nested closed and bounded intervals, since the transfer principle of nonstandard analysis permits us to generalize this result to all sequences later. Since the sequence $\{[a_n, b_n]\}$ is standard, it follows that both a_1 and b_1 are standard. Moreover, since the intervals are nested, we find that $a_1 \leq a_n \leq b_n \leq b_1$ for all n. In particular, $|a_n| \leq \max(|a_1|, |b_1|)$, and this implies that a_n must be limited for all values of n.

[2] Cantor's original proof does not require that $i_n < j_n$. Rather, Cantor finds the next two sequence points in the interval, then chooses i_n and j_n so that $s_{i_n} < s_{j_n}$. That is the only difference between his proof and the one formalized in ACL2(r).

Now, let N be an arbitrary large positive integer—the ACL2(r) constant `i-large-integer` serves this purpose. Since a_N is limited, we can define $A \equiv {}^*a_N$. Notice that A is necessarily standard, since it is the standard part of a limited number. Moreover, for all standard n, $n < N$ (since all standard integers are less than all large integers), and since the intervals are nested, it follows that $a_n \leq a_N$. Taking standard parts of both sides, we can conclude that $a_n \leq {}^*a_N = A$, and using the transfer principle we conclude that $a_n \leq A$ for all n (standard or not).

Similarly, notice that $a_n \leq b_n$ for all n, so that ${}^*a_n \leq {}^*b_n$. Taking standard parts of both sides, it follows that $A \leq b_n$ for all standard values of n, and hence for all values n by the transfer principle. What this means is that we have found a real number A such that $a_n \leq A \leq b_n$ for all n; i.e., $A \in [a_n, b_n]$ for all n, and hence the intersection of the intervals $[a_n, b_n]$ is not empty. This result is summarized in the following ACL2(r) theorem:

```
(defthm standard-part-car-interval-in-intersection
 (and (realp (standard-part-car-interval-large))
      (implies (posp n)
               (and (<= (car (nested-interval n))
                        (standard-part-car-interval-large))
                    (<= (standard-part-car-interval-large)
                        (cdr (nested-interval n))))))
 :hints ...)
```

This argument depends crucially on the use of the transfer principle to show that $a_n \leq A = {}^*a_N \leq b_n$ for all n. However, the transfer principle only applies to classical statements, which this statement is not, since it uses the function standard part. The reason we can do this is that we can define two versions of A, one using `defun` and the other `defun-std`.

```
(defun standard-part-car-interval-large ()
 (standard-part (car (nested-interval (i-large-integer)))))

(defun-std standard-part-car-interval-large-classical ()
 (standard-part-car-interval-large))
```

As explained in the introduction, the version that uses `defun-std` is classical, but its definition is only equal to the expression in the body when the arguments to the function are standard—a condition that is vacuously true in this case, so ACL2(r) can prove that these two definitions are equivalent.

As it turns out, this is the only step in this first proof that uses the non-standard analysis features of ACL2(r). The remainder of the proof could just as easily be carried out in ACL2 (with the exception that it refers to real numbers, not just the rationals).

3.1.3 Constructing the Chain of Nested, Closed, and Bounded Intervals We now consider some of the highlights of the formalization of the

construction of the chain of intervals. The sequence $\{s_n\}$ itself is formalized by defining a constrained function `seq` whose only constraint is that it maps the positive integers to real numbers.

The construction repeatedly looks for the smallest index i such that $i \geq n$ and $s_i \in [a, b]$, for some choice of a, b, and n. This is implemented by the function `next-index-in-range`:

```
(defpun next-index-in-range (n A B)
 (if (in-range (seq n) A B)
     n
   (next-index-in-range (1+ n) A B)))
```

Of course, there is no guarantee that such an i can be found, so the function `next-index-in-range` is not guaranteed to always terminate as written. Thus, it can only be admitted into ACL2(r) by the use of `defpun` instead of `defun`. However, this also means that to reason about `next-index-in-range`, we have to consider the possibility that it fails for a given `n`, `A`, and `B`.

To do so requires the use of existential quantifiers, which we can do with `defun-sk`. The following function, for example, is used to determine which values of `n`, `A`, and `B` lead to success:

```
(defun-sk exists-next-index-in-range (n A B)
 (exists m
         (and (posp m)
              (<= n m)
              (in-range (seq m) A B))))
```

It is then possible to define the function `cantor-sequence-indexes` which returns the nth interval in the construction, or `nil` if no such interval can be found.

Now, suppose that `cantor-sequence-indexes` ever returns `nil`; i.e., that the construction of nested intervals stops after a finite number of iterations. This means that `next-index-in-range` must have been false for some choice of n, A, and B. In this case, we find a point $x \notin \{s_n\}$ as follows. First, we observe that given the choice of n, none of the s_i with $i > n$ can be in $[A, B]$. This means that at most a finite number of points (i.e., n) in the sequence can be in $[A, B]$. But then it is easy to find a point $x \in (A, B)$ that is not one of these n points. The simple, recursive function `counter-example` does just that.

So now suppose that `cantor-sequence-indexes` never returns `nil`; i.e., that the construction of nested intervals continues ad infinitum. It can be easily shown that the resulting sequence of intervals satisfies all the constraints of an infinite chain of nested, closed, bounded intervals, as defined in Sect.3.1.2. Thus, the theorems of that section apply to `cantor-sequence-indexes`, and we can conclude using the principle of functional instantiation that there is some point that is in each of the intervals.

At this point, the remainder of the proof can be carried out. The only difficult portion is the proof that for i in the range $1 \leq i < i_{n+1}$, $s_i \notin [a_n, b_n]$. This was

done using natural induction on n, with the key lemmas being that the theorem holds for i in the range $i_n \leq i < i_{n+1}$, and that the intervals are nested, so that if $s_i \notin [a_{n-1}, b_{n-1}]$, then it trivially follows that $s_i \notin [a_n, b_n]$.

The final statement of the theorem makes use of the (limited) support for quantifiers in ACL2(r). Because this support does not extend directly to nested quantifications, it is necessary to introduce several functions to express the result. First, the function `exists-in-sequence` captures the notion that x is one of the $\{s_n\}$. Similarly, the function `exists-in-interval-but-not-in-sequence` states that x is in the interval $[a, b]$ but is *not* one of the $\{s_n\}$. This uses `exists-in-sequence` to capture that nested quantification. With that, the final statement of the theorem is that `exists-in-interval-but-not-in-sequence` holds (over an arbitrary interval).

```
(defun-sk exists-in-sequence (x)
 (exists i
         (and (posp i)
              (equal (seq i) x))))

(defun-sk exists-in-interval-but-not-in-sequence (A B)
 (exists x
         (and (realp x)
              (< A x)
              (< x B)
              (not (exists-in-sequence x)))))

(defthm reals-are-not-countable
 (exists-in-interval-but-not-in-sequence (a) (b))
 :hints ...)
```

3.2 The Second Proof: Using Diagonalization

3.2.1 The Informal Argument Cantor's second proof of the non-denumerability of the real numbers is based on diagonilization. The familiar idea is as follows. As before, let $\{s_n\}$ be a sequence of real numbers, but this time further assume that $s_n \in [0, 1]$.

Now, any number x such that $x \in [0, 1]$ can be written as a sequence of digits, e.g., $x = 0.d_1d_2d_3\ldots$, where each digit d_i is an integer from 0 to 9. This expansion of x into digits follows from the fact that x can be written in the form $x = \sum_{i=1}^{\infty} \frac{d_i}{10^i}$.

Obviously, if two numbers have the same expansions they are equal to each other. However, it is possible for two different expansions to result in the the same number, e.g., $0.1999\ldots = 0.2000\ldots$. This strictly technical difficulty prevents us from casually swapping between the number and its representation as a sequence of digits, but this difficulty can be addressed in a number of different ways. We chose to address it by considering how different two expansions have to be in order for them to represent two different numbers.

Suppose $x = \sum_{i=1}^{\infty} \frac{d_i}{10^i}$ and $y = \sum_{i=1}^{\infty} \frac{e_i}{10^i}$, where each of the d_i and e_i are digits, and suppose that k is such that $d_k \neq e_k$. Obviously, we can divide the expansion of x as follows, and similarly for y:

$$x = \sum_{i=1}^{\infty} \frac{d_i}{10^i} = \left(\sum_{i=1}^{k-1} \frac{d_i}{10^i} \right) + \frac{d_k}{10^k} + \left(\sum_{i=k+1}^{\infty} \frac{d_i}{10^i} \right) = L_x + \frac{d_k}{10^k} + R_x,$$

where L_x and R_x (and similarly L_y and R_y) are introduced as shorthands for the respective sums. We can now find bounds for the two sums on the right. For instance, since $d_i \leq 9$ for all i, it follows that $R_x = \sum_{i=k+1}^{\infty} \frac{d_i}{10^i} \leq 1/10^k$. This also gives us a (rough, but sufficient) estimate of the maximum difference between R_x and R_y, i.e., $|R_x - R_y| \leq 2/10^k$. Similarly, we can consider possible differences between L_x and L_y. This time, we find a minimum difference, i.e., $|L_x - L_y| \geq 10/10^k$, unless $L_x = L_y$. This follows, because if L_x and L_y are different, then the minimum difference is when only the least significant digits differ and then only by 1, which yields a minimum difference of $1/10^{k-1}$.

So when $d_k \neq e_k$, we have

$$|x - y| \leq |L_x - L_y| + |d_k - e_k| + |R_x - R_y|.$$

We have an upper bound for $|R_x - R_y|$, so if $|d_k - e_k|$ is large enough, the difference between R_x and R_y will be insufficient to make x and y equal to each other. That's enough to show that if $L_x = L_y$, then $x \neq y$. So suppose $L_x \neq L_y$. Again, we have a lower bound for the difference, so as long as $|d_k - e_k|$ is small enough, the difference will be insufficient to make x and y equal. Thus, as long as d_k is sufficiently different from e_k (i.e., $3 \leq |d_k - e_k| \leq 7$), we can show that $x \neq y$.

ACL2(r) does not support infinite computations, such as $x = \sum_{i=1}^{\infty} \frac{d_i}{10^i}$. Instead, we use the standard part of a partial sum up to a large integer. So, if N is an arbitrary, fixed, large integer, we can say $x = {}^*\sum_{i=1}^{N} \frac{d_i}{10^i}$. As before, we can split this sum into three parts, so that $x = {}^*(L_x + d_k + R_x)$, where L_x is as before and R_x is similar, but with upper limit N instead of ∞. We can limit ourselves to standard x and k, since the transfer principle will carry over the results to all x and k. When x is standard, so are L_x and d_k (as these are finite sums), so $x = L_x + d_k + {}^*R_x$. Earlier, the upper and lower bounds on L_x and R_x were enough to show that if d_k is sufficiently different from e_k, then $x \neq y$. But the argument is more subtle in the nonstandard case: It is not enough to show that the sums $L_x + d_k + R_x$ and $L_y + e_k + R_y$ differ, because two numbers may be different even though their standard parts are the same. So what we need to show is that these two sums have different standard parts, or equivalently that they are not close to each other. We can do so by observing that if d_k is sufficiently different from e_k, then $|(L_x + d_k + R_x) - (L_y + e_k + R_y)| \geq 2/10^k$. Since k is standard, $2/10^k$ is not small. This means that $L_x + d_k + R_x$ and $L_y + e_k + R_y$ must have different standard parts, so $x \neq y$.

We have established that every number $x \in [0, 1]$ can be converted into a sequence of digits, and that whenever two sequences of digits are "sufficiently

different" at a given position, the numbers that correspond to those sequences are different. That is all we need to carry out Cantor's diagonalization argument.

Start with the sequence $\{s_n\}$ and convert each s_n into a sequence of digits, $s_n = 0.d_{n,1}d_{n,2}d_{n,3}\ldots$. Then construct a new sequence $\{t_n\}$ by choosing t_n to be sufficiently different from $d_{n,n}$—for example, let $t_n = 7$ if $d_{n,n} < 5$, and $t_n = 2$ otherwise. Then the sequence $\{t_n\}$ is sufficiently different than the sequence (in k) $\{d_{n,k}\}$ in the nth digit, so if t is the number corresponding to $\{t_n\}$, we have that $t \neq s_n$ for all n.

3.2.2 Remarks on the ACL2(r) Formalization The formalization of this argument in ACL2(r) is mostly straightforward. The function `digit-seq` is constrained to map positive integers to digits, and `digit-seq-sum` converts a portion of this sequence into a number in $[0, 1]$. Then we can define the limit of this sum as follows:

```
(defun-std digit-seq-sum-limit ()
 (standard-part (digit-seq-sum 1 (i-large-integer))))
```

To prove that different (enough) sequences correspond to different numbers, we introduce a second constrained function `digit-seq-2` with its own partial sum and limit functions. Then we can carry out the argument as before and show that these limits must be different.

```
(defthm different-enough-digits-implies-different-numbers-of-limit
 (implies (and (posp i)
               (<= (abs (- (digit-seq i) (digit-seq-2 i))) 7)
               (>= (abs (- (digit-seq i) (digit-seq-2 i))) 3))
          (not (equal (digit-seq-sum-limit)
                      (digit-seq-2-sum-limit))))
 :hints ...)
```

To complete the proof, it is only necessary to convert each s_n in the sequence into a sequence of digits, and this can be done with the function `nth-digit`, which is defined as $|x \cdot 10^n| \bmod 10$. As before, we define the summation functions `nth-digit-seq-sum` and `nth-digit-seq-sum-limit`, which take partial sums and their limit, respectively. The important lemma is that the limit of these partial sums is the same as the original number that was taken apart by `nth-digit`. This lemma can be proved by finding an upper bound on the difference between the original number and the partial sum up to an arbitrary index k. Now that we can convert a number to a sequence of digits and vice versa, the rest of the proof goes through easily, yielding a second version of the non-denumerability of the reals:

```
(defthm diag-seq-sum-limit-not-in-sequence
 (and (realp (diag-seq-sum-limit))
      (<= 0 (diag-seq-sum-limit))
      (<= (diag-seq-sum-limit) 1)
```

```
     (implies (posp i)
              (not (equal (diag-seq-sum-limit) (seq i)))))
:hints ...)
```

The statement of `diag-seq-sum-limit-not-in-sequence` is typical of theorems in ACL2(r), as it avoids the use of quantifiers. E.g., instead of saying that *some* $x \in [0,1]$ is not among the $\{s_n\}$, the theorem explicitly names a specific x that is not among the $\{s_n\}$. The way `diag-seq-sum-limit-not-in-sequence` is stated is very much in the tradition of ACL2, as exposed in [5] and [10]. Of course, it is trivial to restate this result using quantifiers, in which case, the final statement of the theorem is almost identical to that in Sect. 3.1. The only difference is that the theorem in this section is specialized for the interval $[0,1]$, whereas in Sect. 3.1 an arbitrary open interval was permitted.

We close this section by mentioning some differences in the ACL2(r) formalizations of Cantor's two proofs. When we started this project, we were not certain that the second proof could be carried out in ACL2(r), since the argument about the equivalence of sequences and numbers appeared to be significantly different than the usual arguments that have been formalized in ACL2(r). In contrast, we expected the first proof to be much easier to formalize in ACL2(r), since it was based on the notion of completeness, which is directly embedded in ACL2(r) with the function `standard-part`. However, the reverse turned out to be the case.

The first proof limited the use of the nonstandard features of ACL2(r) to the proof that the real numbers are complete. In contrast, the second proof used these features extensively, as they are needed to reason about the equivalence of numbers and infinite sums, as well as the fact that different sums correspond to different numbers. However, the arguments in Cantor's diagonalization proof translated more directly to ACL2(r), very much in the Boyer-Moore tradition. We believe the main reason is that the first proof relied on universally quantified hypotheses—which required the explicit use of quantifiers in ACL2(r)—as well as partially defined functions. Nevertheless, we are pleased to report that the admittedly limited support for quantifiers and partial functions in ACL2(r) was sufficient to formalize both proofs.

4 Existence of Trasncendental Numbers

We conclude this paper with a formalization of Cantor's proof of the existence of transcendental numbers. This turns out to be a corollary of the nondenumerability of the reals, since Cantor proceeds by showing how the algebraic numbers can be enumerated. Some aspects of the proof could be simplified significantly by using more modern arguments. For instance, the fact that the set of polynomials with integer coefficients is denumerable follows directly from the denumerability of words from a finite (even denumerable) alphabet. However, we avoid these modern notions, since our goal is to follow Cantor's argument closely.

4.1 The Informal Argument

A number x is algebraic if there is some nontrivial polynomial P with rational coefficients such that $P(x) = 0$; otherwise, x is called transcendental. It is sufficient to consider polynomials Q with integer coefficients, because if there exists some nontrivial polynomial P with rational coefficients such that $P(x) = 0$, then there must also exists a nontrivial polynomial Q with integer coefficients such that $Q(x) = 0$—just let $Q(x) = q \cdot P(x)$, where q is the product of the denominators of the coefficients of P.

So we wish to show that there is some real number x such that $P(x) \neq 0$ for all polynomials P with integer coefficients. We do so with a counting argument as follows. First, define the height[3] of the polynomial $P = \sum_{i=0}^{n} a_i x^i$ of degree n to be $h(P) = n - 1 + \sum_{i=0}^{n} |a_i|$. Clearly, x^h is of degree h, so there is at least one polynomial for each positive height h. More important, there are only a finite number of polynomials for each height h. This follows, because any polynomial of degree greater than h will have height greater than h. Moreover, a polynomial with a coefficient greater than h or less than $-h$ will have height greater than h. So at most $(2h + 1)^{h+1}$ polynomials can be of height h.

This means that we can enumerate all the polynomials of height h, and this leads to an enumeration of all polynomials with integer coefficients. Simply enumerate the (finite) polynomials of height 1, then the (finite) polynomials of height 2, and so on.

The next step is to use this plan to enumerate the algebraic numbers instead of the polynomials. Simply enumerate the roots of the (finite) polynomials of height 1, then the roots of the (finite) polynomials of height 2, and so on.

Finally, we observe that no sequence of real numbers can completely cover the interval $(0, 1)$ (or any other non-trivial interval), as shown in Sect. 3. That means there is an $x \in (0, 1)$ such that x is not algebraic. I.e., we have shown that there exists at least one transcendental number (and indeed many more).

4.2 Formalizing Polynomials

The first step in the ACL2(r) proof is a formalization of polynomials. We represent polynomials using lists, so that the polynomial $3x^3 + 2x - 6$ is represented as `(-6 2 0 3)`. This allows us to define the function `eval-polynomial` that evaluates a polynomial at a point. The root of a polynomial is then defined as a number x such that `eval-polynomial` returns 0. We can now define what we mean by an algebraic number; i.e, one that is the root of some polynomial with rational coefficients. The definition uses the support for quantifiers in ACL2(r):

```
(defun-sk algebraic-numberp (x)
 (exists poly
         (and (rational-polynomial-p poly)
              (non-trivial-polynomial-p poly)
              (polynomial-root-p poly x))))
```

[3] There are different notions of "polynomial height"; the one used here is due to Cantor.

We will need the fact that a polynomial of degree n has at most n roots. We prove this by dividing a polynomial P by $x - a$ whenever $P(a) = 0$. An important lemma is that the resulting quotient is of degree one less than P, as long as P is of degree at least 1 (what we call a "non-trivial" polynomial). Another important lemma is that if $P(b) = 0$ and $a \neq b$, then $Q(b) = 0$ where Q is the quotient polynomial, i.e., $Q(x) = P(x)/(x - a)$. Once these facts are known, we can show by induction that if P is of degree n, then it has at most n roots.

We now have the tools to find a list containing all the roots of a given polynomial. Although it would be possible to compute many of the (algebraic) roots, it is sufficient to use ACL2(r)'s choice functions to successively add a new root to an existing list of roots.

```
(defchoose choose-new-root (x) (poly roots)
 (and (polynomial-root-p poly x)
      (not (member x roots))))
```

It is then a simple matter to define the function `find-roots-of-poly` which chooses all the roots of a given polynomial. It is trivial to show that `find-roots-of-poly` is of length at most equal to the degree of the polynomial, and that if x is a root of the polymonial, then it must be in the result of `find-roots-of-poly`.

4.3 Enumerating the Algebraic Numbers

We now turn our attention to the enumeration of the algebraic numbers. The first step is to enumerate all the polynomials of height h, and the function `generate-polys-with-height` is defined to do so. The definition is typical of combinatorial functions. I.e., a polynomial p is of degree at most n and height h if either

- $p = ax$ for some constant a and $h = |a|$, or
- p is of degree at most $n - 1$ and height h, or
- $p = ax^n + p'$ where $a \neq 0$ and p' is of degree at most $n-1$ and height $h - |a|$.

Since the degree n and leading coefficient a have bounds as explained above, this definition can be implemented recursively. However, the function `generate-polys-with-height` is difficult to introduce into ACL2(r), as many cases need to be considered and it is not obvious why the function terminates—which must be proven before the definition can be accepted. The exact form of the definition is not important, so we omit it here[4]. Instead, we mention the key theorem, namely that the function is guaranteed to generate all the polynomials of the given height.

```
(defthm generate-polys-with-height-valid
 (implies (and (integer-polynomial-p poly)
               (non-trivial-polynomial-p poly))
```

[4] Interested readers can refer to the supporting materials.

```
          (member poly (generate-polys-with-height
                        (polynomial-height poly))))
 :hints ...)
```

We can enumerate all the algebraic numbers: We have an enumeration of the polynomials with integer coefficients, so we simply need to find the roots of each polynomial, using `find-roots-of-poly`.

```
(defun enumerate-roots-of-polys (polys)
 (if (consp polys)
     (append (pad-list (length (car polys))
                       (find-roots-of-poly (car polys)))
             (enumerate-roots-of-polys (cdr polys)))
   nil))
```

The function `pad-list` is there for a technical reason. It simply adds zeros to the list of roots, in order to ensure that the list of roots for a polynomial of degree n has $n + 1$ elements, even though the polynomial has fewer (or no) real roots. What this means is that the enumeration returns more than just the list of roots, but this is unimportant. What matters is that all the roots of the polynomials are accounted for. The padding simply makes it easier to associate the ith element of the list of roots with the jth polynomial.

The two functions `enumerate-roots-of-polys` and `generate-polys-with-height` can be used to define `enumerate-roots-of-polys-of-height`, which enumerates the roots of polynomials of the given height. In turn, this can be generalized into `enumerate-roots-of-polys-up-to-height`, which returns all the roots of polynomials of height 1, then those of height 2, and so on, up to a chosen limit:

```
(defun enumerate-roots-of-polys-up-to-height (height)
 (if (zp height)
     nil
   (append (enumerate-roots-of-polys-up-to-height (1- height))
           (enumerate-roots-of-polys-of-height height))))
```

The definition of `enumerate-roots-of-polys-up-to-height` was carefully chosen so that the algebraic numbers come in a predictable order. I.e., calling this function with a higher limit returns additional roots at the *end of* the list, not in the front. We say that the enumeration of `enumerate-roots-of-polys-up-to-height` is monotonic.

Intuitively, if we call `enumerate-roots-of-polys-up-to-height` repeatedly, we will generate all the roots of all polynomials with integer coefficients; i.e., we can enumerate the algebraic numbers. But we have to make this explicit. I.e., we have to define a mapping from the positive integers into the algebraic numbers, and we have to be able to produce the index n such that the mapping yields a particular algebraic number. The mapping can be defined as follows:

```
(defun algebraic-number-sequence (idx)
 (if (posp idx)
     (nth (1- idx) (enumerate-roots-of-polys-up-to-height idx))
   0))
```

Note that the definition uses `enumerate-roots-of-polys-up-to-height` to enumerate all the polynomials up to height `idx`. This works, because of the properties of `enumerate-roots-of-polys-up-to-height` mentioned above. First, we know that the roots of polynomials of height h is non-empty, since there is at least one polynomial of height h (namely x^h) and we are padding the list of roots in the definition of `enumerate-roots-of-polys`. This means that there are at least `idx` roots of polynomials with height at most `idx`, so the call to `nth` in the definition returns a valid element. Second, since the enumeration of `enumerate-roots-of-polys-of-height` is monotonic, it does not matter that the call to `enumerate-roots-of-polys-of-height` uses a height limit (`idx`) that is almost certainly larger than necessary, since there are bound to be many more than one root at each height!

What remains is to show that if `root` is a root of some polynomial with integer coefficients, say `poly`, then there is an index n such that `root` is the nth element in the sequence `algebraic-number-sequence`. We already know that `root` is in `enumerate-roots-of-polys-of-height` h, where h is the height of `poly`, and we can find the index k of `root` in this list using a simple recursive function. Now, let M be the length of `enumerate-roots-of-polys-up-to-height` $h-1$. Then $M+k$ is the index of `root` in `enumerate-roots-of-polys-up-to-height` h' for any $h' \geq h$. To complete the argument, it is only necessary to observe that $M + k \geq h$, and again this follows because there is at least one root at each height. What this means is that $M + k$ is a suitable choice of n, as the following theorem demonstrates, where `get-index-in-last-list` returns $M + k - 1$:

```
(defthm algebraic-number-sequence-valid
 (implies (and (integer-polynomial-p poly)
               (non-trivial-polynomial-p poly)
               (polynomial-root-p poly root))
          (equal (algebraic-number-sequence
                   (1+ (get-index-in-last-list root poly)))
                 root))
 :hints ...)
```

Now we can prove the existence of transcendental numbers. Using the previous theorem, we can show that the sequence `algebraic-number-sequence` contains all the algebraic numbers. Moreover, `algebraic-number-sequence` satisfies the constraints of the function `seq` defined in Sect. 3.1. That means we can apply the main result of that section to conclude that every open interval (for concreteness, the interval $(0, 1)$) must contain at least one number that is not in the sequence. By definition, this number must be transcendental. The final statement of the theorem is as follows:

```
(defun-sk exists-transcendental-number ()
 (exists x
         (and (realp x)
              (not (algebraic-numberp x)))))

(defthm existence-of-transcendental-numbers
 (exists-transcendental-number)
 :hints ...)
```

5 Conclusions

This paper describes a formalization of Cantor's proofs of the non-denumerability of the continuum, a result listed in Freek Wiedijk's Formalizing 100 Theorems [9]. Following Cantor's 1874 paper, this paper also formalizes his proof of the existence of transcendental numbers. The formalization depends on features of ACL2 that are rarely used in ACL2(r), such as choice functions, explicit quantifiers, and partial (or rather underspecified) functions.

Given the two proofs, it would appear that the formalization based on Cantor's familiar diagonalization argument would be the more difficult to formalize in ACL2(r). However, our experience demonstrates that the diagonalization argument can be formalized more directly in the ACL2 (or Boyer-Moore) tradition. It turns out that the original proof is more difficult to formalize in ACL2(r), as it requires explicit use of quantifiers and choice functions.

References

1. Cantor, G.: On a property of the set of real algebraic numbers. In: From Kant to Hilbert. Volume 2. Oxford University Press (1874) 839–843
2. Ewald, W., ed.: From Kant to Hilbert. Volume 2. Oxford University Press (2005)
3. Dunham, W.: The Calculus Gallery. Princeton (2005)
4. Cantor, G.: On an elementary question in the theory of manifolds. In: From Kant to Hilbert. Volume 2. Oxford University Press (1891) 920–922
5. Kaufmann, M., Moore, J S.: An industrial strength theorem prover for a logic based on Common Lisp. IEEE Transactions on Software Engineering **23**(4) (1997) 203–213
6. Manolios, P., Moore, J S.: Partial functions in ACL2. Journal of Automated Reasoning (JAR) **31** (2003) 107–127
7. Nelson, E.: Internal set theory: A new approach to nonstandard analysis. Bulletin of the American Mathematical Society **83** (1977) 1165–1198
8. Gamboa, R., Kaufmann, M.: Nonstandard analysis in ACL2. Journal of Automated Reasoning **27**(4) (2001) 323–351
9. Wiedijk, F.: Formalizing 100 theorems. `http://www.cs.ru.nl/~freek/100/index.html` (2012)
10. Boyer, R.S., Moore, J S.: A Computational Logic. Academic Press, Orlando (1979)