

Square Roots in ACL2: A Study in Sonata Form

Ruben A. Gamboa*
Computer Sciences Department
The University of Texas at Austin
Taylor Hall 2.124
Austin, TX 78712-1188
ruben@lim.com, ruben@cs.utexas.edu
<http://www.lim.com/~ruben/>

November 18, 1996

Prelude

One of ACL2's extensions over its predecessor Nqthm is the direct support for a richer set of numbers, including the rationals and complex-rationals but not including the irrationals. The absence of the irrationals is strong, in the sense that ACL2's typing mechanism asserts that all numbers are integers, rationals, or complex-rationals, and all other objects are indistinguishable from zero, as far as arithmetic operations are concerned. This is in contrast to an approach where the irrationals exist, but the axioms are simply too weak to prove much of anything about them that isn't also true about all the real numbers; in other words, a logic roughly equivalent to our everyday understanding of the reals.

This brings up the question of what is the proper way to reason about transcendental functions in ACL2, since strictly speaking ACL2 can only reason about rational functions. In this paper, we explore this issue by focusing on the square root function. We begin by showing that this function does not exist in the ACL2 universe. In particular, we use the absence of the irrationals to *prove* that for all x , $x^2 \neq 2$. This illustrates that when dealing with assertions about the real number line, one best remember that ACL2 can prove them only for the rationals — and the equivalent assertions about the reals may be false.

One can argue, however, that to a computer scientist the rationals should be enough. After all, computers don't support the irrational numbers, either, so we have to approximate any transcendental function by a rational equivalent. We show how this can be done for square root and use ACL2 to prove that our approximation is arbitrarily close to the square root function. Our technique is to prove that a specific function (namely a bisection algorithm) serves as an approximation.

*Author is supported by a salary from LIM International, Inc.

1 Exposition

ACL2 allows the user to reason about entire classes of functions by selecting the relevant properties of these functions, and only allowing these properties to be used in subsequent proofs. However, it does not allow the introduction of partially defined functions. That is, one cannot introduce a function by specifying only its fundamental theorem. For example, the fundamental theorem of square root is that $\forall x, \sqrt{x} \cdot \sqrt{x} = x$; however, ACL2 will not allow the following “definition” of square root:

```
(encapsulate
  ((sqrt (x) t))

  (defthm sqrt-*-sqrt
    (equal (* (sqrt x) (sqrt x)) x)))
```

ACL2 rejects the `encapsulate` event on purely syntactic grounds: It fails to provide a “witness function” for `sqrt` inside the `encapsulate`. That is, ACL2 demands a definition of `sqrt` even if it is local to the `encapsulate`.

There is a good semantic reason for this restriction as well. From the theorem `sqrt-*-sqrt`, one can easily establish that

```
(equal (* (sqrt 2) (sqrt 2)) 2)
```

However, this is a spurious assertion, since ACL2’s universe does not include the irrationals, so it seems possible that the following is *also* a theorem:

```
(not (equal (* x x) 2))
```

Such a situation would render ACL2 unsound. In section 2.1, we will show that ACL2 does in fact prove that the square of no object is equal to 2, thus defeating any attempt to introduce a square root function. In section 2.2, however, we will show that we can define an approximation scheme to square root. That is, for any acceptable error tolerance, we can find a rational function that approximates square root within this error bound.

2 Development

2.1 First Treatment

To prove that

```
(not (equal (* x x) 2))
```

is a theorem in ACL2, we proceed by ruling out suitable candidates for `x`. The first step is the most interesting from a mathematical viewpoint. We will show that

```
(implies (rationalp x)
  (not (equal (* x x) 2)))
```

We do this by following the classic proof of the irrationality of $\sqrt{2}$. Subsequently, we can rule out the complex rationals, since all their squares are either complex or negative. Since all other objects (i.e., non-numbers) in ACL2 have zero squares, this will complete the proof.

Let us begin by considering the rationals and show that none of them can be equal to $\sqrt{2}$. On paper, one would proceed as follows. Suppose $\sqrt{2}$ is rational. Then, we have that for some relatively prime integers p and q , $(\frac{p}{q})^2 = 2$. Now, this implies that $p^2 = 2q^2$ and so p^2 is even. But since p is an integer, this implies that p must be even as well. That is, there is some integer p' with $p = 2p'$. But then, it follows that $4p'^2 = 2q^2$, and hence q^2 is even. Again, this implies that q is even. But then, p and q are not relatively prime as claimed, and therefore $\sqrt{2}$ cannot be rational.

The first step in formalizing this argument is to prove the following lemmas:

```
(defthm even-square-implies-even
  (implies (and (integerp p)
                (divisible (* p p) 2))
           (divisible p 2)))
```

```
(defthm even-implies-square-multiple-of-4
  (implies (and (integerp p)
                (divisible p 2))
           (divisible (* p p) 4)))
```

Given the equation $p^2 = 2q^2$, we conclude from `even-square-implies-even` that p is even, from `even-implies-square-multiple-of-4` that q^2 is even, and finally from `even-square-implies-even` again that q is even. That is, we can prove the following lemmas in ACL2:

```
(defthm lemma-1.1
  (implies (and (integerp p)
                (integerp q)
                (equal (* p p) (* 2 (* q q))))
           (divisible q 2)))
```

```
(defthm lemma-1.2
  (implies (and (integerp p)
                (integerp q)
                (equal (* p p) (* 2 (* q q))))
           (divisible p 2)))
```

To complete the argument, we need the key property that p and q are relatively prime, or equivalently that $\frac{p}{q}$ is expressed in lowest terms. ACL2 provides the built-in functions `numerator` and `denominator` that have this property. So, we proceed by showing how we can apply `lemma-1.2` to the `numerator` and `denominator` of an allegedly rational square root of 2:

```
(defthm lemma-1.3
  (implies (and (rationalp x)
                (equal (* x x) 2))
            (equal (* (numerator x) (numerator x))
                  (* 2 (* (denominator x)
                          (denominator x))))))
```

All these results can be succinctly combined into a single ACL2 lemma as follows:

```
(defthm lemma-1.4
  (implies (and (rationalp x)
                (equal (* x x) 2))
            (and (divisible (numerator x) 2)
                 (divisible (denominator x) 2))))
```

At this point, we've dispensed with the mathematics. We have shown that if x is rational and its square is equal to 2, then its numerator and denominator are not relatively prime. But this is absurd, since we know we can always write a rational in lowest terms.

However, ACL2 failed to prove the following simple fact:

```
(defthm lemma-1.5
  (implies (and (divisible (numerator x) 2)
                (divisible (denominator x) 2))
            (not (rationalp x))))
```

Odd indeed, since the documentation explicitly states that the `numerator` and `denominator` functions always return the answer in lowest terms. After some experimentation, we discovered the reason that ACL2 was skeptical of the fact above is that the relevant axiom about `numerator` and `denominator` is not enabled in the ACL2 prover. The result is that while `numerator` and `denominator` are guaranteed to be in lowest terms by the ACL2 *logic*, the ACL2 *prover* is unaware of the fact. The disabled axiom (found in the file `axioms.lisp` of the ACL2 source distribution), is as follows:

```
(defaxiom Lowest-Terms
  (implies (and (integerp n)
                (rationalp x)
                (integerp r)
                (integerp q)
                (< 0 n)
                (equal (numerator x) (* n r))
                (equal (denominator x) (* n q)))
            (equal n 1))
  :rule-classes nil)
```

Notice the `:rule-classes nil` directive which effectively disables this axiom in the prover. To work around this, we need to explicitly ask the ACL2 prover to invoke this axiom. We do this by providing a hint as follows:

```

(defthm lemma-1.5
  (implies (and (divisible (numerator x) 2)
                (divisible (denominator x) 2))
            (not (rationalp x)))
  :hints (("Goal"
           :use (:instance Lowest-Terms
                    (n 2)
                    (r (/ (numerator x) 2))
                    (q (/ (denominator x) 2))))))

```

From lemma-1.4 and lemma-1.5, ACL2 can now easily deduce that the square root of two cannot be rational:

```

(defthm sqrt-2-is-not-rationalp
  (implies (rationalp x)
            (not (equal (* x x) 2))))

```

Now that we have ruled out the rationals, we can turn our attention to the remaining numbers in ACL2, namely the complex rationals. A complex rational has the form $a+bi$, where $b \neq 0$ and a and b are rationals. None of these objects can be the square root of 2, because their squares all have the form $a^2 - b^2 + 2abi$, and for that to be equal to 2, a must be zero. But then, the square of bi is equal to $-b^2$, which is negative since b is rational.

Formalizing this argument in ACL2 is not too hard. The major stumbling block is that the key axiom concerning complex arithmetic is disabled. So, we have to begin by defining complex squares. The relevant axiom is as follows:

```

(defaxiom complex-definition
  (implies (and (rationalp x)
                (rationalp y))
            (equal (complex x y)
                    (+ x (* #c(0 1) y))))
  :rule-classes nil)

```

where the term `#c(0 1)` is ACL2's version of $i = \sqrt{-1}$. We use this axiom in the following lemma, which instructs ACL2 on how to square complex rationals:

```

(defthm complex-square-definition
  (implies (and (rationalp x)
                (rationalp y))
            (equal (* (complex x y) (complex x y))
                    (complex (- (* x x) (* y y))
                              (+ (* x y) (* x y)))))
  :hints (("Goal"
           :use (:instance complex-definition))))

```

Once we have this fact, it becomes simple to prove that the square of `(complex x y)` is rational if and only if `(* x y)`, and therefore `x`, is equal to zero:

```
(defthm complex-squares-rational-iff-imaginary
  (implies (and (complex-rationalp w)
                (rationalp (* w w)))
            (equal (realpart w) 0)))
```

where the `realpart` returns the real component of a complex rational. ACL2 can also prove that all complex rationals with zero `realpart`, that is to say pure imaginary numbers, have negative squares:

```
(defthm imaginary-squares-are-negative
  (implies (and (complex-rationalp w)
                (equal (realpart w) 0))
            (< (* w w) 0)))
```

Putting these results together, we have been able to prove that no complex rational can be the square root of two:

```
(defthm sqrt-2-is-not-complex-rationalp
  (implies (complex-rationalp x)
            (not (equal (* x x) 2))))
```

At this point, we have considered all the possible ACL2 numbers and found that none of them are equal to $\sqrt{2}$. ACL2 can dispense of the remaining objects using simple type reasoning, thus we can now prove the following strong theorem:

```
(defthm there-is-no-sqrt-2
  (not (equal (* x x) 2)))
```

2.2 Second Treatment

We have shown that the fundamental theorem of square roots is inconsistent with the axioms of ACL2. However, it is possible to prove weaker versions of this theorem. One such approach is to require that it only hold when both x and \sqrt{x} are rational; at other points, no claims are made about the function, so it is free to take on any value, say zero. That such a function exists in the ACL2 logic is clear, since for rational p/q in least terms, $\sqrt{p/q}$ is given by \sqrt{p}/\sqrt{q} and since p and q are relatively prime, this is rational if and only if \sqrt{p} and \sqrt{q} are integers. Unfortunately, the likelihood that an arbitrary integer has an integer square root is small, so this would only cover a small fraction of the rationals, and the modified theorem would be too weak.

A better alternative is to substitute closeness for strict equality. For example, we may require that $|\sqrt{x} \cdot \sqrt{x} - x| < \epsilon$ for some $\epsilon > 0$. There are many different approximation schemes that can be used to come close to the square root function. A natural approach is to use an iterative algorithm, such as Newton's method, but proving its convergence rate within the ACL2 logic can be cumbersome. From a reasoning perspective, a more natural approximation scheme is the bisection algorithm, since it is easy to predict its convergence rate; i.e., the uncertainty is precisely halved at each step.

The convergence result is actually interesting, since the result \sqrt{x} to which we are converging is not necessarily in the ACL2 universe, so we are not able to guarantee something similar to $|\hat{x} - \sqrt{x}| < \epsilon$. It is for this reason, that we chose $|\hat{x}^2 - x| < \epsilon$ as our convergence criterion, although we will have to prove a variant of the former along the way.

We begin by defining the iterative square root function. Because its convergence isn't obvious and ACL2 only accepts definitions whose termination it can prove, we chose to divorce the convergence result from the terminating condition of the iterative function. The definition is given below:

```
(defun iterate-sqrt-range (low high x num-iters)
  (if (<= (nfix num-iters) 0)
      (cons (rfix low) (rfix high))
      (let ((mid (/ (+ low high) 2)))
          (if (<= (* mid mid) x)
              (iterate-sqrt-range mid high x (1- num-iters))
              (iterate-sqrt-range low mid x (1- num-iters))))))
```

The function iterates on the `high-low` range. That is, it returns the new `high-low` values after iterating `num-iter` times. We will use the `low` value as the approximation to \sqrt{x} .

We split the convergence result into two parts. First, we show that if we let `num-iters` be large enough, the difference between the final `high` and `low` is arbitrarily small. Then, we show that if the `high` and `low` are very close to each other, `(* low low)` is very close to `x`.

Before proceeding, however, we need to establish some basic properties of `iterate-sqrt-range`. For starters, we need to show that if the original `high-low` range is not vacuous, then neither is the final `high-low` range after iterating any number of times:

```
(defthm iterate-sqrt-range-reduces-range
  (implies (and (rationalp low)
                (rationalp high)
                (< low high))
            (< (car (iterate-sqrt-range low high x num-iters))
                (cdr (iterate-sqrt-range low high x
                                          num-iters)))))
```

A particularly crucial lemma is that the final `high` estimate is not larger than the initial one. That is, no iteration can increase the current upper estimate.

```
(defthm iterate-sqrt-range-non-increasing-upper-range
  (implies (and (rationalp low)
                (rationalp high)
                (< low high))
            (<= (cdr (iterate-sqrt-range low high x
                                          num-iters))
                 high)))
```

Nonetheless, the final `high` estimate is large enough that it doesn't cross below the square root of `x`:

```
(defthm iterate-sqrt-range-upper-sqrt-x
  (implies (and (rationalp low)
                (rationalp high)
                (rationalp x)
                (<= x (* high high)))
    (<= x
      (* (cdr (iterate-sqrt-range low high x
                                  num-iters))
         (cdr (iterate-sqrt-range low high x
                                  num-iters))))))
```

This gives us a tight bound on how far the values of `high` can range. Similarly, we can prove the analogous theorems for the `low` bound of the range.

With these lemmas, we are now ready to make use of the continuity of x^2 to show that if the `high-low` range is small enough, then the range of their squares is arbitrarily small. Specifically, for any $\epsilon > 0$ and $a > 0$, we can find a δ so that for a `b` with $0 < b - a < \delta$, $b^2 - a^2 < \epsilon$. In fact, algebraic manipulation will show that this is true for any $\delta \leq \epsilon / (a + b)$. Moreover, since we're interested in ranges $[a, b]$ such that $a \leq \sqrt{x} \leq b$, we can replace b^2 by `x` to conclude that $x - a^2 < \epsilon$. The continuity condition can be stated as follows:

```
(defthm sqrt-epsilon-delta-aux-4
  (implies (and (rationalp a)
                (rationalp b)
                (rationalp x)
                (rationalp epsilon)
                (<= 0 a)
                (< a b)
                (<= x (* b b))
                (< (- b a) delta)
                (<= delta (/ epsilon (+ b a))))
    (< (- x (* a a)) epsilon)))
```

Unfortunately, this result is unsatisfying because it is stated in terms of $(+ b a)$, which will correspond to the *final* `high-low` estimates or our approximation, and we would rather find δ from the original estimates or guesses. Since we know that the `high` estimates are monotonically decreasing, but the `low` estimates are increasing, we can't readily conclude anything about the sum of the final `high` and `low`. However, we can proceed by observing that the claim remains true for $\delta \leq \epsilon / 2b$, since for $0 \leq a \leq b$, $\epsilon / 2b \leq \epsilon / (a + b)$. Now, δ will only depend on the final `high` estimate, and since we know `high` is monotonically decreasing we can replace the final `high` estimate with the initial guess. This is important, because it lets us compute a priori the number of iterations that will be required. Combining these observations gives the first half of the convergence result:

```

(defthm iter-sqrt-epsilon-delta
  (implies (and (rationalp low)
                (rationalp high)
                (rationalp epsilon)
                (rationalp delta)
                (rationalp x)
                (< 0 epsilon)
                (<= 0 low)
                (< low high)
                (<= x (* high high))
                (<= delta (/ epsilon (+ high high))))
    (let ((range (iterate-sqrt-range low high x
                                     num-iters)))
      (implies (< (- (cdr range) (car range)) delta)
                (< (- x (* (car range) (car range)))
                    epsilon))))))

```

Now, it only remains to be shown that if we iterate long enough, the final **high-low** estimate will be sufficiently close together so that the theorem above can apply. We start out by defining `guess-num-iters` which computes the required number of iterations for a specific x and ϵ . Since the iteration scheme halves the estimate at each step, it turns out that we only need to know the size of the initial estimate (which is derived from x). The routine, which essentially computes the log of the initial range, is given below:

```

(defun guess-num-iters-aux (range num-iters)
  (if (and (integerp range)
           (integerp num-iters)
           (> num-iters 0)
           (> range (2-to-the-n num-iters)))
      (guess-num-iters-aux range (1+ num-iters))
      (1+ (nfix num-iters))))
(defmacro guess-num-iters (range delta)
  `(guess-num-iters-aux (ceiling ,range ,delta) 1))

```

where the `2-to-the-n` function returns 2^n for non-negative integer n . Before proving that `guess-num-iters` does return a sufficiently large value for any choice of `range` and `epsilon`, we consider how `iterate-sqrt-range` reduces the **high-low** range after a number of iterations, specifically, it halves the range at each step:

```

(defthm iterate-sqrt-reduces-range-size
  (implies (and (<= (* low low) x)
                (<= x (* high high))
                (rationalp low)
                (rationalp high)
                (integerp num-iters))

```

```

(let ((range (iterate-sqrt-range low high x
                                num-iters)))
      (equal (- (cdr range) (car range))
              (/ (- high low)
                 (2-to-the-n num-iters)))))

```

With this result and much algebraic rewriting, we can prove the second half of the convergence theorem, namely that by iterating up to `guess-num-iters` the final `high-low` range is sufficiently small for the previous convergence theorem to apply:

```

(defthm iterate-sqrt-range-reduces-range-size-to-delta
  (implies (and (rationalp high)
                 (rationalp low)
                 (rationalp delta)
                 (< 0 delta)
                 (< low high)
                 (<= (* low low) x)
                 (<= x (* high high)))
            (let ((range (iterate-sqrt-range
                          low
                          high
                          x
                          (guess-num-iters (- high low)
                                             delta))))
              (< (- (cdr range) (car range)) delta))))

```

The only thing left is the choice of appropriate starting values for `high` and `low`. Given an $x > 0$, we let the initial range be $[0, x]$ if $x > 1$, and $[0, 1]$ otherwise. It is clear that this range includes \sqrt{x} , is non-trivial, and includes only non-negative numbers, hence we can use it to begin the iteration.

The resulting ACL2 function that approximates square root can be defined as follows:

```

(defun iter-sqrt (x epsilon)
  (if (and (rationalp x)
           (<= 0 x))
      (let ((low 0)
            (high (if (> x 1) x 1)))
        (let ((range (iterate-sqrt-range
                      low high x
                      (guess-num-iters (- high low)
                                         (/ epsilon
                                             (+ high high))))))
          (car range)))
      nil))

```

With little more than propositional reasoning, ACL2 can now prove the main convergence result:

```

(defthm convergence-of-iter-sqrt
  (implies (and (rationalp x)
                (rationalp epsilon)
                (< 0 epsilon)
                (<= 0 x))
    (and (<= (* (iter-sqrt x epsilon)
                (iter-sqrt x epsilon))
          x)
         (< (- x (* (iter-sqrt x epsilon)
                    (iter-sqrt x epsilon)))
              epsilon))))

```

3 Recapitulation

We began this project to explore how ACL2 can be used to reason about real-valued functions. Although the ACL2 universe includes only the rationals, most of the ACL2 arithmetic axioms — algebraic equalities and inequalities — are true of both the rationals and irrationals, so it is tempting to believe that by exercising some care ACL2 proofs can be generalized to include the irrationals. This views the ACL2 logic as essentially unaware of the reals, but consistent with them.

However, in this paper we showed how dangerous this can be, since rather than ACL2 being too weak to prove that $\sqrt{2}$ exists, it is actually so strong it can prove $\sqrt{2}$ does *not* exist. Any attempt to generalize an ACL2 proof about a rational analogue of the square root function needs to be tempered with the fact that the existence of the real square root function is inconsistent with the axioms of ACL2.

However, for many real functions, we can construct rational approximation schemes. We showed how such a scheme can be defined for the square root function, and also how ACL2 can prove the convergence of the approximation scheme to square root.

4 Coda

All the theorems described here have been mechanically verified by ACL2 version 1.8. The relevant ACL2 books are in the public domain, and they can be accessed <http://www.lim.com/~ruben/research/acl2/sqrt/>. The files contain considerably more detail than that discussed in this paper, not only in the forms of auxiliary lemmas (some of them with large syntax to semantics quotients) that are needed by ACL2 in its proof discovery but also in the form of hints and directives to help guide the ACL2 prover.

Acknowledgments

We would like to thank Bob Boyer for being extraordinarily patient with us as we've tried to learn first Nqthm and then ACL2, as well as for helping us with the structure of this paper; J Moore, Matt Kaufmann and the rest of the ACL2 developers for giving us a wonderful toy to play with; Matt again for showing us that playing with ACL2 can be fun; and Ludwig von Beethoven for immeasurably enriching mankind.

References

- [BM79] R. S. Boyer and J S. Moore. *A Computational Logic*. Academic Press, Orlando, 1979.
- [BM88] R. S. Boyer and J S. Moore. *A Computational Logic Handbook*. Academic Press, San Diego, 1988.
- [CB84] R. V. Churchill and J. W. Brown. *Complex Variables and Applications*. McGraw-Hill, fourth edition, 1984.
- [CLR90] T. H. Corman, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*, chapter 32. McGraw-Hill, New York, 1990.
- [KM] M. Kaufmann and J S. Moore. *ACL2: A Computational Logic for Applicative Common Lisp, The User's Manual*.
- [KM94] M. Kaufmann and J S. Moore. Design goals for ACL2. Technical Report 101, Computational Logic, Inc., 1994.
- [KP96] M. Kaufmann and P. Pecchiari. Interaction with the Boyer-Moore theorem prover: A tutorial study using the arithmetic-geometric mean theorem. *Journal of Automated Reasoning*, 16(1-2):181–222, 1996.
- [Str91] G. Strang. *Calculus*. Wellesley-Cambridge Press, Wellesley, 1991.