Inverse Functions in ACL2(r)

Ruben Gamboa
Department of Computer Science
University of Wyoming
Laramie, Wyoming
ruben@cs.uwyo.edu

John R. Cowles
Department of Computer Science
University of Wyoming
Laramie, Wyoming
cowles@cs.uwyo.edu

ABSTRACT

ACL2(r) supports the definition of real-valued functions. In this paper, we introduce a theory of inverse functions into ACL2(r). The theory is developed in stages, from an abstract description of inverse functions, to a still abstract but more tractable treatment of the inverse of continuous functions. A macro is introduced to simplify the introduction of concrete inverse functions. We illustrate the approach by defining some inverse functions in ACL2, including the square root, natural logarithm, inverse sine, and inverse cosine functions.

Categories and Subject Descriptors

F.4.1 [**Mathematical Logic**]: Mechanical theorem proving—*Mechanized mathematics*

General Terms

Verification

Keywords

Mechanized math, ACL2, inverse functions, intermediate value theorem, nonstandard analysis

1. INTRODUCTION

Based on non-standard analysis (also known as the calculus of infinitesimals), ACL2(r) adds support for the irrational real and complex numbers to ACL2. Using ACL2(r), it is possible to define transcendental functions, such as e^x, $\sqrt{x}$, and $\sin(x)$. In other words, ACL2(r) can reason about many of the functions introduced in a typical freshman-level course in calculus.

However, inverse functions play an important role in such a course. For example, the exponential function leads to logarithms, and the inverse trigonometric functions lead to transformations between coordinate systems, among other applications. Although some inverse functions have been defined in ACL2(r), such as $\sqrt{x}$, each such function was introduced completely from first principles. For example, $\sqrt{x}$ was introduced by a direct implementation using a bisection algorithm.

In this paper, we report on a new ACL2(r) book that makes it easier to introduce inverse functions. The book is based on **defchoose** to name the inverse function, provided that such a function exists. The bulk of the book demonstrates that such a function indeed exists, given standard necessary assumptions.

In section 2, we will develop a theory of inverses based on the weakest possible assumptions that guarantee the existence of an inverse function. Although it is general, the resulting theory of inverses is awkward to apply. Consequently, we develop a more practical theory of inverses that is applicable to continuous functions. This theory is presented in section 3. We use this theory to introduce another definition of $\sqrt{x}$, as well as the inverse trigonometric functions, and the natural logarithm functions. Once these functions are defined, we can use them to define more of the familiar functions from calculus, such as a^x. This is described in section 4. The work described in this paper takes advantage of some recent features in ACL2(r). In fact, this is the first significant verification performed with the new version of ACL2(r). So we describe in section 5 our experiences with the new version.

Note: With the exception of the support for the irrationals, ACL2(r) is essentially equivalent to ACL2. For the remainder of this paper, we will use the name ACL2 to refer to both versions of the theorem prover. In the few occassions where the difference between the provers is relevant, we will mention the difference explicitly.

2. A GENERAL THEORY OF INVERSES

The basic (or high school) theory of inverse functions is well known. A function f is invertible if it has the following properties:

- f is onto: $\forall y \exists x. f(x) = y$
- f is 1-1: $\forall x_1, x_2. f(x_1) = f(x_2) \Rightarrow x_1 = x_2$

It is not hard to convince oneself that these conditions are both necessary and sufficient for the existence of an inverse function f^{-1}. Certainly, if f is not onto, then there is some y_0 such that there is no x_0 for which $f(x_0) = y_0$. Similarly, if f is not 1-1, then for some y_0, $f(x_1) = f(x_2) = y_0$ where $x_1 \neq x_2$. For such a y_0, there is no unique choice of $f^{-1}(y_0)$.

This theory is slightly complicated when we consider a specific domain D and range R for the function $f : D \to R$.

.

This is an important consideration, since a function is 1-1 and onto (and hence invertible) only over a specific choice of domain and range. A good example is the square function, which is invertible if we consider the domain and range to be the non-negative reals, but not invertible when we let the domain and range be the entire real number line.

The theory can be translated into ACL2 using encapsulate. The idea is to constrain a function *ifn* together with its domain and range, so that *ifn* is 1-1 and onto over these sets. Since ACL2 does not represent sets directly, we chose to constrain the domain and range with the predicates *ifn-domain-p* and *ifn-range-p*.

The constraint that *ifn* is 1-1 can be written directly in ACL2:

```
(defthm ifn-is-1-1
  (implies (and (ifn-domain-p x1)
                (ifn-domain-p x2)
                (equal (ifn x1) (ifn x2)))
           (equal x1 x2))
  :rule-classes nil)
```

The only nontrivial aspect is the use of the empty rule classes. Notice that the conclusion would be a dangerous rewrite rule; in fact, ACL2 would reject it.

Constraining *ifn* to be onto is trickier. The longstanding tradition in ACL2 (and all the way back to NQTHM) would be to introduce a witness function that would select the x for which the function *ifn* returns y. Of course, if we could do that, we would already have the inverse function defined!

Although not commonly used, a feature of ACL2 is support for quantifiers. The approach is to define a function which essentially names the first-order formula containing the quantifier. It is then possible to use this name inside a theorem.

So the first task is to name the property of ontoness. This can be done with the following **defun-sk** event:

```
(defun-sk ifn-is-onto-predicate (y)
      (exists (x)
              (and (ifn-domain-p x)
                   (equal (ifn x) y))))
```

Once the property is named, it can be used in a theorem such as the following:

```
(defthm ifn-is-onto
  (implies (ifn-range-p y)
           (ifn-is-onto-predicate y))
  :hints (("Goal"
  :use ((:instance ifn-is-onto-predicate-suff
                   (x y) (y y)))))
)
```

The theorem *ifn-is-onto-predicate-suff* mentioned in the hint is a formula that is sufficient to imply that the predicate *ifn-is-onto-predicate* holds for a given x and y.

Using only these constraints, it is possible to introduce the inverse of *ifn*. We can define it with the following **defchoose**:

```
(defchoose ifn-inverse (x) (y)
  (and (ifn-domain-p x)
       (equal (ifn x) y)))
```

This has the effect of choosing a value $x = (\textit{ifn-inverse}\ y)$ such that $(\textit{ifn}\ x) = y$, *provided* that such a value exists in the first place. So it is left only to show that there is such a value (in which case, (*ifn-inverse* y) will serve). Naturally, this derives from the fact that *ifn* is onto:

```
(defthm inverse-exists
  (implies (ifn-range-p y)
           (and (ifn-domain-p (ifn-inverse y))
                (equal (ifn (ifn-inverse y)) y)))
  :hints (("Goal"
           :use ((:instance ifn-inverse
                  (x (ifn-is-onto-predicate-witness y))
                  (y y))
                 (:instance ifn-is-onto-predicate
                  (y y))))))
```

Finally, we demonstrate that the choice of inverse is well-defined, which follows directly from the fact that *ifn* is 1-1.

```
(defthm inverse-unique
  (implies (and (ifn-range-p y)
                (ifn-domain-p x)
                (equal (ifn x) y))
           (equal (ifn-inverse y) x))
  :hints (("Goal"
           :use ((:instance inverse-exists
                            (y y))
                 (:instance ifn-is-1-1
                            (x1 x)
                            (x2 (ifn-inverse y)))
                 ))))
```

The theorem *inverse-unique* is especially important, because it allows us to find individual values of the inverse function. Note that the definition of *ifn-inverse* is completely non-constructive. But using *inverse-unique*, we can say for example that $\sqrt{4} = 2$, since $2 \cdot 2 = 4$.

Other familiar properties of inverses have also been shown, such as the fact that the inverse function is also 1-1 and onto, and that *ifn* is its inverse.

3. INVERSE OF CONTINUOUS FUNCTIONS

The biggest difficulty in applying the theory of inverse functions developed in section 2 is proving that a given function is onto. Take the function $f(x) = x^2$, for example. How do you show that there is an x for which $f(x) = 2$?

For continuous functions, the Intermediate Value Theorem (IVT) gives an answer. The theorem, previously formalized in ACL2 [3], states that if f is continuous over the interval $[a, b]$ and y is a real number such that $f(a) < y < f(b)$, then there is a real number $x \in [a, b]$ such that $f(x) = y$.

If we start with the domain $[a, b]$, the IVT can be applied directly. However, for many functions that we may wish to invert, the choice of interval $[a, b]$ depends on the y that we wish to invert. For example, for $f(x) = x^2$, the correct choice of $[a, b]$ is either $[1, y]$ when $y \geq 1$ or $[0, 1]$ when $y < 1$.

So we choose to constrain four functions in ACL2:

- *icfn* — an invertible, continuous function
- *icfn-domain* — the domain of *icfn*, which is assumed to be an interval
- *icfn-range* — the range, also an interval
- *icfn-inv-interval* — a function that returns an appropriate interval $[a, b]$ for a specific choice of y.

Once these functions are constrained, it is possible to show that *icfn* is onto, using the same definition of onto described in section 2. And using the theorems proved in section 2, it then follows that *icfn* has an inverse.

Most of the functions we will wish to find inverse for are continuous, so we will be instantiating the theorems about *icfn* often. We make this less cumbersome by introducing the macro **definv**, which takes a function f, its domain, range, and the function that generates an interval. In return, it defines *f-inverse*, and it instantiates the theorems establishing that f and *f-inverse* are inverses of each other. The macro **definv** also supports many optional arguments that alter its behavior, e.g., by letting the user provide the name of the inverse function to define.

4. EXAMPLES

We now consider how the theory of inverse functions we have developed can be used to define some common functions.

4.1 Square Root

We start with the square root function, which can be defined by inverting the function $f(x) = x^2$. As mentioned previously, the domain and range of f are both $[0, \infty)$, and the interval $[a, b]$ that contains $\sqrt{y}$ can be defined as $[1, y]$ for $y > 1$ and $[0, 1]$ for $y \leq 1$.

The biggest difficulty in introducing the inverse function is showing that $f(x) = x^2$ is continuous, and this follows from reasonably straightforward algebra. In the framework of non-standard analysis, f is continuous if $f(x) \approx f(x + \epsilon)$ when ϵ is infinitesimal. But this holds, since $(x + \epsilon)^2 = x^2 + 2x\epsilon + \epsilon^2 \approx x^2$, and the terms involving ϵ are infinitesimal.

Once this is established, we can introduce the square root function with the following event:

```
(definv square
        :domain (interval 0 nil)
        :range    (interval 0 nil)
        :inverse-interval square-interval)
```

The function *square* needs no explanation. The function *interval* is the constructor for intervals, and nil in this context represents infinity. The function *square-interval* constructs the appropriate interval for a given y as described previously. The definitions of these functions are omitted for brevity.

In earlier work, we had defined the function *acl2-sqrt* with the following property:

```
(defthm-std sqrt-sqrt
  (implies (and (realp x)
                (≤ 0 x))
           (equal (* (acl2-sqrt x)
                     (acl2-sqrt x))
                  x)))
```

Using the version of *inverse-unique* generated by the macro **definv** for the function *square*, we can show that *acl2-sqrt* is, in fact, identical to the inverse of the *square*:

```
(defthm square-inverse->sqrt
    (implies (inside-interval-p y (interval 0 nil))
             (equal (square-inverse y)
                    (acl2-sqrt y))))
```

This is reassuring evidence that the theory of inverse functions we have developed is effective.

4.2 Inverse Trigonometric Functions

The trigonometric functions are defined in ACL2(r) by taking advantage of the identities

- $\sin(x) = \frac{e^{ix} - e^{-ix}}{2i}$
- $\cos(x) = \frac{e^{ix} + e^{-ix}}{2}$

The exponential function is defined by using its Taylor expansion, namely

- $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$

It is worth pointing out that these definitions hold for complex values of x, not just for the reals.

As before, we can define the inverse functions to sine and cosine after we show that these functions are 1-1 and onto, in appropriate domains. Although the choice of domains is arbitrary, it is traditional to pick $[-\pi/2, \pi/2]$ for sine and $[0, \pi]$ for cosine. The range of both, of course, is $[-1, 1]$.

In earlier work, we had developed a large theory of trigonometry in ACL2(r). In particular, we had shown that cos is continuous, and we had used this fact to define π as twice the first (and only) zero of the cos function in the range $[0, 2]$. However, a significant effort was required to extend these results to show that the trigonometric functions are 1-1 in the appropriate restricted domains.

Once this was demonstrated, however, it was a trivial matter to introduce the inverse functions with the macro **definv**.

```
(definv real-sine
        :f-inverse        acl2-asin
        :domain           (interval (- (/ (acl2-pi) 2))
                                    (/ (acl2-pi) 2))
        :range            (interval -1 1)
        :inverse-interval sine-interval)
```

Note that the function that is inverted is *real-sine* instead of *acl2-sine*. The reason is that the latter is defined over all the ACL2 numbers, but the IVT applies only to real-valued functions. *real-sine* is the restriction of *acl2-sine* over the reals.

4.3 Complex Numbers in Polar Form

We can use the inverse functions to transform complex numbers in Cartesian form to polar coordinates. Specifically, given a point $x+iy \neq 0$, we can represent it in polar coordinates as $r \cdot e^{i\theta}$, where $r = \sqrt{x^2 + y^2}$ and $\theta = \cos^{-1}(x/r)$ or $2\pi - \cos^{-1}(x/r)$, depending on the sign of y. For convenience, we name these functions *radiuspart* and *anglepart*, following the convention established by *realpart* and *imagpart*.

Using the theorems about the inverse trigonometric functions, we can show that this conversion works appropriately:

```
(defthm correctness-of-polar-form
  (equal (* (radiuspart x)
            (acl2-exp (* #c(0 1) (anglepart x))))
         (fix x)))
```

Moreover, we have verified in ACL2(r) that when $x + iy$ is equal to $r \cdot e^{i\theta}$ in polar coordinates, the coordinates satisfy the following familiar properties:

- r is a non-negative real
- $r = 0$ only when $x + iy = 0$

- $r = |x|$ when $y = 0$
- $\theta \in [0, 2\pi)$
- if $y = 0$, $\theta = 0$ or $\theta = \pi$, depending on the sign of x

4.4 Natural Logarithm

The natural logarithm function is special, in that we can extend its definition over all non-zero numbers. To introduce it, we proceed in stages.

First, consider the domain $[0, \infty)$. Over this domain, e^x is increasing and onto $[1, \infty)$. Moreover, given a $y \in [1, \infty)$, the interval $[0, y]$ contains an x such that $e^x = y$. These facts can be established easily from the theory of the function e^x that ships with ACL2(r). Consequently, we can use the macro **definv** to define the inverse function over the range $[1, \infty)$. Call this inverse function $\ln_{\geq 1}$.

Next, we extend this inverse function to cover the range $(0, \infty)$. To do so, observe that when $y \in (0, 1)$, it follows that $\frac{1}{y} \in (1, \infty)$. So we can define $\ln_{>0}(y) = -\ln_{\geq 1} \frac{1}{y}$ for $y \in (0, 1)$. That $\ln_{>0}$ is the inverse of e^x follows from the fact that $e^{-x} = \frac{1}{e^x}$.

Finally, consider a non-zero, complex number $x + iy$. Recall that we can write this number as $r \cdot e^{i\theta}$. Using the properties of e^x, we can establish that $\ln(r \cdot e^{i\theta}) = \ln(r) + i\theta$. Notice that r is real and positive (since $x + iy \neq 0$). So we can find $\ln(r)$ using the function $\ln_{>0}$ defined above.

The resulting logarithm function is defined for all numbers, other than 0. Notice that for negative numbers, we are implicitly using the fact that $e^{i\pi} = -1$.

Many of the familiar properties of the logarithm function have also been proved in ACL2(r), by deriving them from the corresponding properties of the exponential function. In particular, the following were easily shown:

- $\ln(xy) = \ln x + \ln y$
- $\ln \frac{1}{x} = -\ln x$

4.5 General Exponentials

Using the logarithm function defined above, we can introduce the general exponential function a^x. This is defined as $a^x = e^{x \ln a}$, and in ACL2 we named it (*raise a i*).

ACL2 defines the function (*expt a i*) with value a^i for integer exponents i. Using induction and the fact that $e^{x+y} = e^x e^y$, it is easy to show that

```
(defthm raise-expt-for-non-negative-exponents
  (implies (and (integerp i)
                (≤ 0 i))
           (equal (raise a i)
                  (expt a i))))
```

This result can be generalized, using the definition of *expt* and the fact that $e^{-x} = \frac{1}{e^x}$ to include all integer powers:

```
(defthm raise-expt
  (implies (integerp i)
           (equal (raise a i)
                  (expt a i))))
```

What this shows is that the function *raise* is a generalization of the more familiar *expt*.

We can also prove some familiar properties, such as the following:

- $a^{x+y} = a^x a^y$
- $a^{-x} = \frac{1}{a^x}$

We conclude this section by highlighting two beautiful theorems. First, define the constant e by (*acl2-exp* 1). The function *acl2-exp* is defined in ACL2 using the Taylor expansion of e^x. We expect that (*expt x*) should have the value e^x, although that is not verified in ACL2. However, the relationship between e, as defined above, and the function *acl2-exp* can be demonstrated with the following theorem:

```
(defthm raise-acl2-exp
  (implies (acl2-numberp x)
           (equal (raise (acl2-exp 1) x)
                  (acl2-exp x))))
```

Finally, we return to the function $\sqrt{x}$, which began our investigation of inverse functions. The function *raise* gives us yet another way to define it:

```
(defthm raise-sqrt
  (implies (and (realp x)
                (≤ 0 x))
           (equal (raise x 1/2) (acl2-sqrt x))))
```

5. UNVEILING THE NEW (R)

In [4], we formalized the theory of definitional events in ACL2(r). As a result, we were able to show that certain extensions to ACL2(r) were justified, and we have implemented most of these. The extensions are as follows:

- Any ACL2(r) object can be standard, not just numbers.
- If $\phi(x)$ is a classical term with no free variables other than x, then $standard(x) \Rightarrow standard(\phi(x))$ is a theorem.
- Non-classical functions can be introduced with **defchoose**. (Additionally, the story behind the introduction of both classical and non-classical functions was clarified.)
- Non-classical constrained functions can be introduced with **encapsulate**. (The argument justifying these functions also demonstrated a soundness bug in previous versions of ACL2(r).)
- Under certain conditions, it is permissible to accept recursive functions with non-classical bodies.

All items except the last one have been implemented in ACL2(r), although this version of ACL2(r) has not yet been released[1].

In this section, we would like to discuss how some of these changes facilitated the development of the theory described in this paper.

First, the theory of continuous functions in previous versions of ACL2(r) assumed that the functions were defined and continuous over the entire real number line. However, this is inappropriate in the context of inverse functions, where we need to be very careful about the function domain and range. Consequently, we developed a new version of continuity in ACL2(r) that supports domains. The domains are restricted to be intervals (as is common in theorems from elementary analysis). Since we wished to quantify

[1]We anticipate that it will be released officially by the time of the workshop.

over these intervals, we developed a significant theory of intervals in ACL2(r). The definitions and theorems presented in prior sections of this paper already introduced some of the functions dealing with intervals, such as the constructor *interval* and the predicate *inside-interval-p*. The theory also describes subintervals, open intervals, closed intervals, half-open intervals, infinite intervals, etc.

More important, the earlier theory of continuous functions included the following constraint:

```
(defthm rcfn-standard
    (implies (standardp x)
             (standardp (rcfn x))))
```

This constrains was the source of many difficulties reported earlier, e.g. [6]. The problem is that when the function *rcfn* is replaced with a lambda term containing free variables, the constraint is not met. What this means is that it is impossible to reason about continuous functions with more than one variable, *even* when all but one of the variables are held fixed. In the new version of ACL2(r), the constraint is unnecessary, since ACL2(r) can prove it directly, simply from the fact that *rcfn* is a classical function. In the case that *rcfn* is functionally instantiated with a lambda term containing free variables, the free variables will need to be restricted to standard values using typical instantiation tricks, i.e., by replacing the variable with an if expression that implicitly adds the needed hypotheses.

In the current project, not having to prove that functions return standard values for standard arguments made it far easier to work with continuous functions. More generally, it completely removed the need to argue that certain terms were standard from first principles, as was often the case with previous versions of ACL2(r).

Another benefit was the expansion of standard to include non-numeric objects. In particular, we reasoned about standard intervals, which are precisely those with standard endpoints. The behavior of standard intervals when non-standard values were inside it could be investigated properly. For example, if I is a closed, bounded, standard interval and $x \in I$, then ${}^*x \in I$, where ${}^*x$ is the unique standard number that is infinitesimally close to x.

After working with the new ACL2(r), we can state without reservation that it is far easier to use than the old ACL2(r). We are looking forward to continuing work on this project and others involving ACL2(r).

6. CONCLUSIONS

The theory of inverse functions described in this paper is sufficient to introduce all the typical functions discussed in an introductory calculus course. Moreover, ACL2 can reason effectively about these functions, being able to prove most of the familiar properties of these functions.

However, there is much work to be done, and we expect to be working on some of these in the near future. For example, when our macro introduces the function f^{-1}, it shows that $f(f^{-1}(y)) = y$ and that $f^{-1}(f(x)) = x$ and that f^{-1} is 1-1 and onto. However, when f is known to be continuous, it follows that f^{-1} is continuous as well. But our macro is not currently proving this fact.

Another omission concerns derivatives. When the function f is differentiable as well as continuous, the derivative of its inverse is given by $1/f'(f^{-1}(y))$, as long as this quantify is defined. This fact could also be generated and proved automatically by our macro.

Finally, our macro could use some enhancements, particularly in the area of passing hints to the generated proof obligations. As with other verification projects, we have found that hints are often required while discharging the proof obligations that permit the introduction of an inverse function. However, there is no mechanism at the current time to provide these hints. What we are doing currently is to prove the required theorems before invoking the macro, but we plan to provide a more direct mechanism.

7. REFERENCES

[1] R. V. Churchill and J. W. Brown. *Complex Variables and Applications*. McGraw-Hill, fourth edition, 1984.

[2] W. Fulks. *Advanced Calculus: an introduction to analysis*. John Wiley & Sons, third edition, 1978.

[3] R. Gamboa. Continuity and differentiability in ACL2. In M. Kaufmann, P. Manolios, and J S. Moore, editors, *Computer-Aided Reasoning: ACL2 Case Studies*, chapter 18. Kluwer Academic Press, 2000.

[4] R. Gamboa and J. Cowles. Theory extension in ACL2(r). *To appear in Journal of Automated Reasoning*, 2006.

[5] R. Gamboa and M. Kaufmann. Nonstandard analysis in ACL2. *Journal of Automated Reasoning*, 27(4):323–351, November 2001.

[6] R. Gamboa and B. Middleton. Taylor's formula with remainder. In *Proc of the Third International Workshop of the ACL2 Theorem Prover and its Applications (ACL2-2002)*, 2002.

[7] E. R. Heineman and J. D. Tarwater. *Plane Trigonometry*. McGraw-Hill, Inc., seventh edition, 1993.