

Mechanical Verification of Elementary Calculus Theorems in ACL2

Ruben Gamboa
University of Wyoming
ruben@cs.uwyo.edu

September 13, 2006

Presented at the University of Northern Colorado

Outline

- Non-Standard Analysis
- What is ACL2?
- Intermediate-Value Theorem
- Concluding Remarks

Non-Standard Analysis

An extension of the real number line \mathbb{R} into the “hyperreal” number line ${}^*\mathbb{R}$, containing “infinite” and “infinitesimal” numbers.

$$\begin{aligned} S \subset \mathbb{R} & \implies {}^*S \subset {}^*\mathbb{R} \\ f : \mathbb{R} \rightarrow \mathbb{R} & \implies {}^*f : {}^*\mathbb{R} \rightarrow {}^*\mathbb{R} \\ \text{A theorem } T \text{ about } \mathbb{R} & \iff \text{A theorem } {}^*T \text{ about } {}^*\mathbb{R} \end{aligned}$$

Note: We consider only first-order theorems T in this discussion.

Non-Standard Analysis (Alternative View)

Some real numbers are designated “standard.” There are also standard functions, standard sets, etc.

- To define a set, say which *standard* elements belong to it.
- To define a function, specify how it behaves on *standard* arguments.
- To prove “any” theorem, prove it for all *standard* values.

The Non-Standard View of the Reals

Key concepts: *small, large, limited, close, standard, standard part.*

- $0, 1, 1/2, \pi, e, 2^{32}$, etc. are all *standard*.
- ϵ is *small* iff $|\epsilon| < |x|$ for all *standard* reals $x \neq 0$.
- X is *large* iff $|X| > |x|$ for all *standard* reals x .
- A number x is *limited* iff it is not *large*.
- x and y are *close* if $x - y$ is *small*.
- If x is *limited*, there is exactly one *standard* number that is *close* to x .
- If x is *limited*, $x = {}^*x + \epsilon$ where *x is *standard* and ϵ is *small*. *x is called the *standard part* of x .
- **The big one:** There does exist some *small* number other than 0.

How Can This Be True?

The reason non-standard analysis works is because of compactness:

Compactness Theorem: Suppose S is a set of first-order predicate statements that is inconsistent (i.e., assuming S , you can prove false.) Then there is a *finite* subset of S that is also inconsistent.

Intuition: A proof of a contradiction from S uses at most a finite number of statements from S (since proofs are finite).

A Strange Theory of the Natural Numbers

We will create a consistent set S of first-order statements that demonstrates a non-standard (i.e., strange) model of the naturals.

Let C be a new constant symbol (e.g., 0 or 1 or 10 or ...)

S contains the following (first-order) statements:

- All true statements from arithmetic (or if you prefer, just the Peano Axioms)
- $0 < C$
- $1 < C$
- $1 + 1 < C$
- $1 + 1 + 1 < C$
- ...

A Strange Theory of the Natural Numbers (Cont'd)

- Suppose S (as defined in the previous slide) is inconsistent.
- By compactness, there is a finite subset of S (say S') that is also inconsistent.
- But S' contains only a finite number of the $1 + \dots + 1 < C$ statements.
- Suppose $1 + 1 + 1 + 1 < C$ is the largest such statement.
- But then S' is clearly consistent. Simply interpret C as the number 5.

What this means is that we have a structure that looks just like the natural numbers, except that it contains a number bigger than any of the “standard” natural numbers!

Now, think about $1/C$ in the real number line...

Algebraic Properties of NSA Notions

- *small, limited, large, standard, and standard part* have intuitive algebraic properties.
 - if x and y are *small*, so is $x + y$.
 - if x *small* and y is *limited*, $x \cdot y$ is *small*.
 - if x and y are *standard*, so is $x + y$.
 - if x and y are *limited*, ${}^*(x + y) = x^* + y^*$.
- If $x < y$, then ${}^*x \leq {}^*y$.

Large \neq Infinite; *Small* \neq Zero

Non-standard numbers are still numbers. They obey all the laws of arithmetic. In particular:

- $x + 1 \neq x$, even if x is *large*.
- $x - \epsilon \neq x$, unless $\epsilon = 0$.
- $x \cdot \epsilon$ is not necessarily *small*.

Non-standard numbers are *not* infinite (or infinitely small.) They are *not* ordinals.

Standard vs. Non-Standard Analysis

- Convergence (Cauchy)

- (Traditional) The sequence $\{a_n\}$ converges if for all $\epsilon > 0$, there exists a natural number N so that for all integers n, m greater than N , $|a_n - a_m| < \epsilon$.
- (NSA) The sequence $\{a_n\}$ converges if a_n is *close* to a_m for all *large* natural numbers n and m .

- Continuity

- (Traditional) f is continuous at a point x if for all $\epsilon > 0$, there exists a $\delta > 0$ such that $|f(x) - f(y)| < \epsilon$ whenever $|x - y| < \delta$.
- (NSA) f is continuous at a standard point x if $f(x)$ is *close* to $f(y)$ whenever x is *close* to y .

What is ACL2?

1. A logic based on applicative Common Lisp
2. A programming environment for this logic
3. A (semi-automated) theorem prover for this logic

Recent News: Boyer, Kaufmann, and Moore were awarded the 2006 ACM Software Systems Award for their work on the “Boyer-Moore Family of Theorem Provers” which includes ACL2.

How Computers Prove Theorems

1. Heuristic search — OTTER.
2. Decision procedure — Knuth-Bendix for group theory, linear arithmetic, BDDs for propositional logic, or Wu's geometry prover.
3. Rewriting — RRL, ACL2.

How ACL2 Proves Theorems

```
Pool ← {g}  
while Pool ≠ ∅ ∧ false ∉ Pool do  
  Remove first goal c from Pool  
  if a rule rewrites c into c', and its hypotheses rewrite to true then  
    Pool ← Pool ∪ ({c'} − {true})  
  else if ... then  
    ∴  
  else if c suggests an induction scheme then  
    Pool ← Pool ∪ {base, induction}  
  else  
    break (give up)  
  end if  
end while
```

How ACL2 Proves Theorems (Cont'd)

if $Pool = \emptyset$ **then**

 Add g to known rewrite rules

 return success

else

 return failure

end if

An Example: Peano Arithmetic

- $0 + j = j$
- $s(i) + j = s(i + j)$

```
(defun plus (i j)
  (if (and (integerp i) (< 0 i))
      (1+ (plus (1- i) j))
      j))
```

```
(defthm plus-commutative
  (implies (and (integerp i) (<= 0 i)
                (integerp j) (<= 0 j))
           (equal (plus i j)
                  (plus j i))))
```

Proving plus-commutative

Since no rewrite rules are applicable, ACL2 has to use induction. It chooses an induction scheme on the variable `I` in `(PLUS I J)`. This results in four goals. The first one is the base case:

Subgoal *1/4

```
(IMPLIES (AND (NOT (AND (INTEGERS I) (< 0 I)))
              (INTEGERS I)
              (<= 0 I)
              (INTEGERS J)
              (<= 0 J))
         (EQUAL (PLUS I J) (PLUS J I))) .
```

Essentially, $i \leq 0 \Rightarrow i + j = j + i$.

Subgoal *1/4'

```
(IMPLIES (AND (<= I 0)
              (INTEGERP I)
              (<= 0 I)
              (INTEGERP J)
              (<= 0 J))
         (EQUAL J (PLUS J 0)))
```

This simplifies, using linear arithmetic, to

Subgoal *1/4''

```
(IMPLIES (AND (INTEGERP J) (<= 0 J))
         (EQUAL J (PLUS J 0)))
```

Name the formula above *1.1.

ACL2 will (attempt to) prove the missing lemma $j = j + 0$ later.

The next case is the inductive case.

Subgoal *1/3

```
(IMPLIES (AND (AND (INTEGERP I) (< 0 I))
              (EQUAL (PLUS (+ -1 I) J)
                    (PLUS J (+ -1 I))))
         (INTEGERP I)
         (<= 0 I)
         (INTEGERP J)
         (<= 0 J))
 (EQUAL (PLUS I J) (PLUS J I))) .
```

To proceed, ACL2 will “open up” the definition of (PLUS I J).

Subgoal *1/3''

```
(IMPLIES (AND (INTEGERP I)
              (< 0 I)
              (EQUAL (PLUS (+ -1 I) J)
                    (PLUS J (+ -1 I))))
         (INTEGERP J)
         (<= 0 J))
(EQUAL (+ 1 (PLUS J (+ -1 I)))
       (PLUS J I))) .
```

After opening up the definition of PLUS, ACL2 can use the induction hypothesis.

Subgoal *1/3'''

```
(IMPLIES (AND (INTEGERP I)
              (< 0 I)
              (INTEGERP J)
              (<= 0 J))
         (EQUAL (+ 1 (PLUS (+ -1 I) J))
                (PLUS J I))).
```

Name the formula above *1.2.

This goal $(1 + ((i - 1) + j) = j + i)$ will also be proved by induction later.

The other two goals are base cases, and they are easily proved because the hypotheses are contradictory.

Subgoal *1/2

```
(IMPLIES (AND (AND (INTEGERP I) (< 0 I))
              (< (+ -1 I) 0)
              (INTEGERP I) (<= 0 I)
              (INTEGERP J) (<= 0 J))
         (EQUAL (PLUS I J) (PLUS J I))) .
```

Subgoal *1/1

```
(IMPLIES (AND (AND (INTEGERP I) (< 0 I))
              (NOT (INTEGERP (+ -1 I))))
         (INTEGERP I) (<= 0 I)
         (INTEGERP J) (<= 0 J))
         (EQUAL (PLUS I J) (PLUS J I))) .
```

Now ACL2 returns to the goals that were pushed for induction.

So we now return to *1.2, which is

```
(IMPLIES (AND (INTEGERP I)
              (< 0 I)
              (INTEGERP J)
              (<= 0 J))
         (EQUAL (+ 1 (PLUS (+ -1 I) J))
                (PLUS J I))).
```

Perhaps we can prove *1.2 by induction.

The first subgoal is:

Subgoal *1.2/4

```
(IMPLIES (AND (NOT (AND (INTEGERS J) (< 0 J)))
            (INTEGERS I)
            (< 0 I)
            (INTEGERS J)
            (<= 0 J))
          (EQUAL (+ 1 (PLUS (+ -1 I) J))
                 (PLUS J I))).
```

Subgoal *1.2/4''

```
(IMPLIES (AND (INTEGERS I) (< 0 I))
          (EQUAL (+ 1 (PLUS (+ -1 I) 0)) I)).
```

Name the formula above *1.2.1.

ACL2 is losing control. In fact, the proof will eventually fail with the following:

We therefore turn our attention to *1.2.1, which is

```
(IMPLIES (AND (INTEGERP I) (< 0 I))
          (EQUAL (+ 1 (PLUS (+ -1 I) 0)) I)).
```

No induction schemes are suggested by *1.2.1.
Consequently, the proof attempt has failed.

The solution is to prove a lemma *before* trying to prove `plus-commutative`:

```
(defthm plus-j-0
  (implies (and (integerp j) (<= 0 j))
    (equal (plus j 0) j)))
```

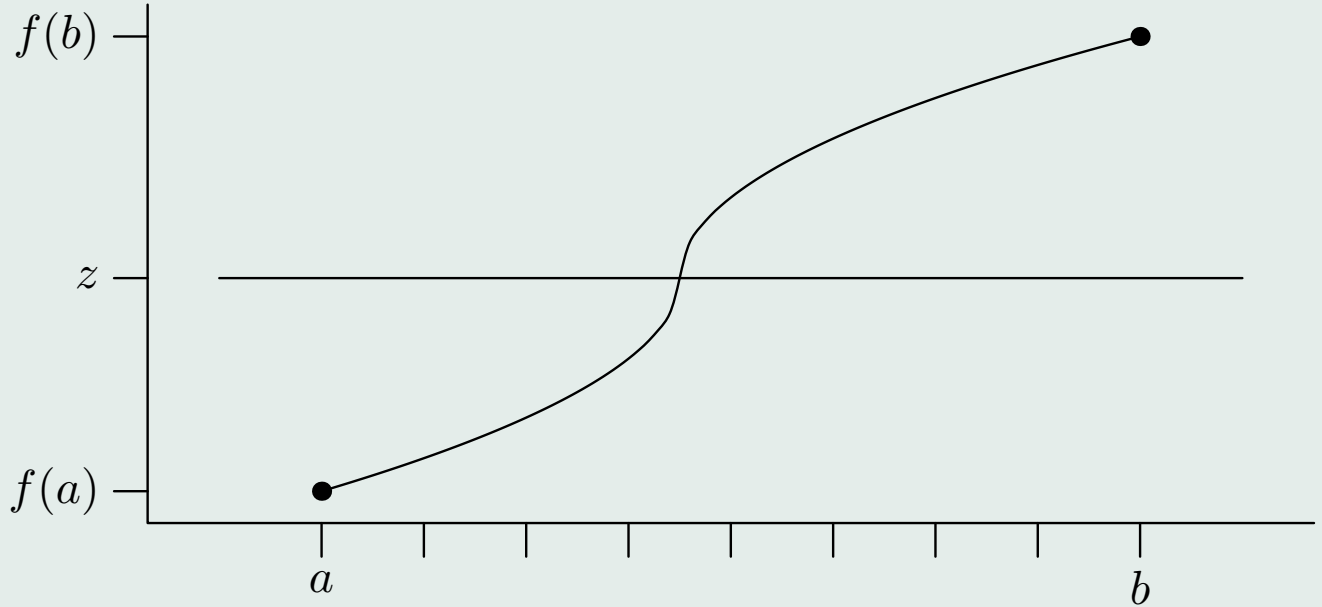
ACL2 will then be able to prove `plus-commutative`.

Things Lacking in ACL2

- No support for quantifiers. All variables are universally quantified.
- Strictly first-order — no sets, no higher-order functions (e.g., limit or derivative).

However, it offers the notion of a constrained function via `encapsulate`, which makes *some* second-order reasoning possible.

The Intermediate Value Theorem



(Standard Real) Continuous Functions

A continuous function f satisfies these requirements:

- If x is standard, so is $f(x)$.
- If x is real, so is $f(x)$.
- If x is a standard real and y is a real number *close* to x , then $f(x)$ is *close* to $f(y)$.

Problem: these are second-order properties — properties of a function f .1

Continuous Functions in ACL2

```
(encapsulate
  ((rcfn (x) t))
  ...
  (defthm rcfn-continuous
    (implies (and (standard-numberp x)
                  (realp x)
                  (i-close x y)
                  (realp y))
              (i-close (rcfn x) (rcfn y))))
  )
```

Outside of the `encapsulate`, the only things known about the function `rcfn` are its constraints, e.g., `rcfn-continuous`.

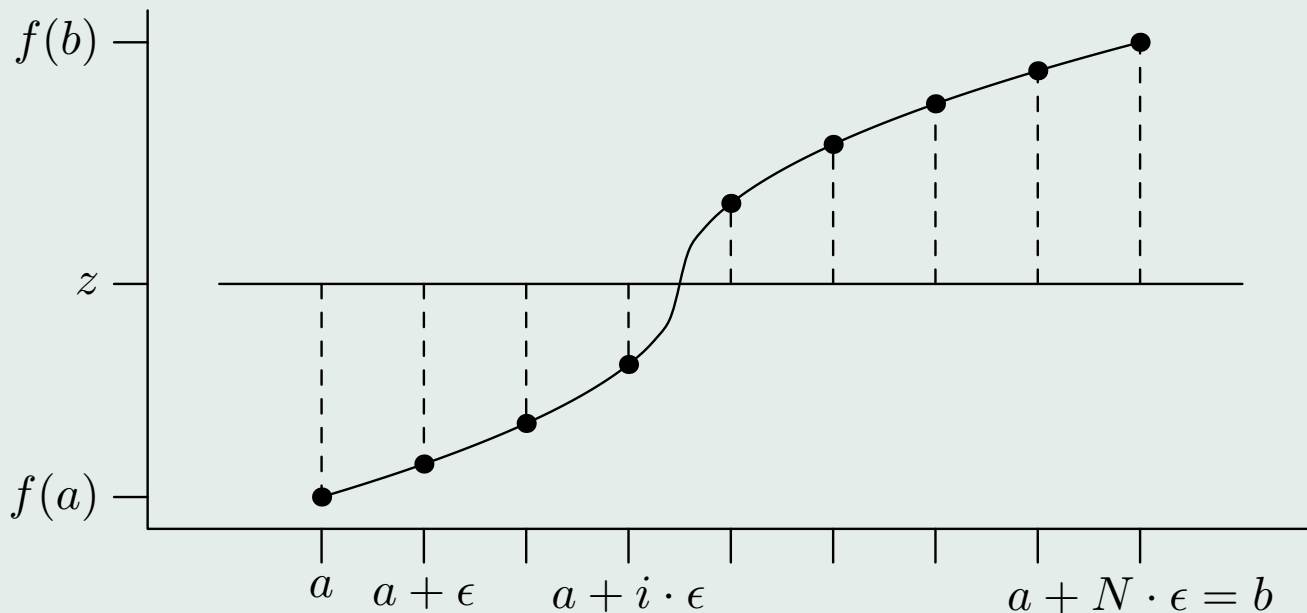
This is rather like saying “Let `rcfn` be a continuous function...” at the beginning of a proof.

Using Non-Standard Analysis in ACL2

Basic Strategy to prove a theorem T about a function f :

1. Define f_n , a discrete approximation to f .
2. Prove (by induction) T_n , a counterpart to T using f_n .
3. Use non-standard analysis to define the function f as the *standard part* of f_n (for *large* values of n).
4. Prove T by using the transfer principle on T_n .

Proving the Intermediate Value Theorem



The Intermediate Value Theorem

For a given $\epsilon > 0$, the value i is chosen so that $f(a + i \cdot \epsilon) < z$ and $f(a + (i + 1) \cdot \epsilon) \geq z$.

- Let $x_i = a + i \cdot \epsilon$.
- x_i is *limited*, so *x_i is *close* to x_i .
- If ϵ is *small*, then *x_i is also *close* to $a + (i + 1) \cdot \epsilon$.
- Therefore, by continuity at *x_i , $f({}^*x_i)$ is *close* to both $f(a + i \cdot \epsilon)$ and $f(a + (i + 1) \cdot \epsilon)$.
- Since $f({}^*x_i)$ is *standard*, $f({}^*x_i)$ must be z .

Step 1: Defining f_n

This function finds the right value of i . It returns $a + i\epsilon$ directly.

```
(defun find-zero-n (a z i n eps)
  (if (and (realp a)
           (integerp i)
           (integerp n)
           (< i n)
           (realp eps)
           (< 0 eps)
           (< (rcfn (+ a eps)) z))
      (find-zero-n (+ a eps) z (1+ i) n eps)
      (realfix a)))
```

Step 2: Proving T_n

It is “easy” to show `find-zero-n` satisfies the following:

- $a \leq x_i$
- $x_i \leq \underbrace{a_i + (n - i)\epsilon}_b$
- $rcfn(x_i) < z$
- $z \leq rcfn(x_i + \epsilon)$

where x_i is the value of `(find-zero-n a z i n eps)`. The proofs proceed by induction on i .

Step 3: Defining f as $*f_n$

The function `find-zero-n` can be used to implicitly define `find-zero` that finds the actual intermediate value.

```
(defun-std find-zero (a b z)
  (if (and (realp a) (realp b) (< a b)
          (realp z))
      (standard-part
       (find-zero-n a z 0 NNN EPS))
      0))
```

where the constant `NNN` is an arbitrary *large* natural, and `EPS` is equal to $NNN/(b - a)$.

Key fact: `find-zero` is a *standard* function. We can define this function by specifying only how it behaves on *standard* arguments, as long as our mapping guarantees that *standard* arguments are mapped to *standard* results.

Step 3: Defining f as $*f_n$

Before the function `find-zero` can be accepted, we need to prove the following:

```
(defthm limited-find-zero-body
  (implies (and (i-limited a)
                (i-limited b)
                (realp b))
           (i-limited (find-zero-n a z 0
                                   ...))))
```

This theorem ensures that the body of `find-zero` returns a *standard* value when its arguments are *standard*.

Step 4: Proving T by Using Transfer on T_n

The properties about `find-zero-n` can be transferred to `find-zero` using `defthm-std`:

```
(defthm-std rcfn-find-zero-<=-z
  (implies (and (realp a) (realp b) (< a b)
                (realp z)
                (< (rcfn a) z))
            (<= (rcfn (find-zero a b z)) z)))
```

Similarly for the following:

- `(<= z (rcfn (find-zero a b z)))`
- `(<= a (find-zero a b z))`
- `(<= (find-zero a b z) b)`

IVT in ACL2

```
(defthm intermediate-value-theorem
  (implies (and (realp a) (realp b) (< a b)
                (realp z)
                (< (rcfn a) z) (< z (rcfn b))))
  (and (realp (find-zero a b z))
        (< a (find-zero a b z))
        (< (find-zero a b z) b)
        (equal (rcfn (find-zero a b z))
                z))))
```

What Else Can We Do?

- Extreme value theorems
- Rolle's theorem
- Mean value theorem
- Geometric series, alternating series, ...
- Taylor's theorem (with Brittany Middleton)
- Differentiable functions are also continuous
- $x \geq 0 \Rightarrow \sqrt{x} \cdot \sqrt{x} = x$

What Else Can We Do? (Cont'd)

- e^x is continuous
- $e^{x+y} = e^x \cdot e^y$
- $\log xy = \log x + \log y$
- $e^{i\pi} = -1$
- $\sin^2(x) + \cos^2(x) = 1$
- $\sin(x + y) = \sin(x) \cos(y) + \cos(x) \sin(y)$
- $\cos(\pi/2) = 0$ (using IVT)
- $\pi \approx 3.14159265358979323846$
- $\int_a^b + \int_b^c = \int_a^c$ (Matt Kaufmann)
- fundamental theorem of calculus (Matt Kaufmann)

In the Future?

We are currently interested in computational number theory, especially as it applies to cryptography.

Some of this requires reasoning about analytic number theory, which we hope to formalize in ACL2.

Concluding Remarks

- Using non-standard analysis, it is possible to reason about the real numbers, *in the context of ACL2* — a quantifier-free, first-order logic.
 - universal quantification
 - first-order terms instead of second-order operators
 - `encapsulate`
- Induction plays a major role in the proofs — and ACL2 is very good at induction.
- Conditional rewriting also plays a major role in the proofs — and ACL2 is very good at conditional rewriting.
- Four-step process to prove analysis theorems in ACL2.