Continuity and Differentiability in ACL2?

Ruben Gamboa

Logical Information Machines, Inc.

ruben@lim.com

March 31, 1999

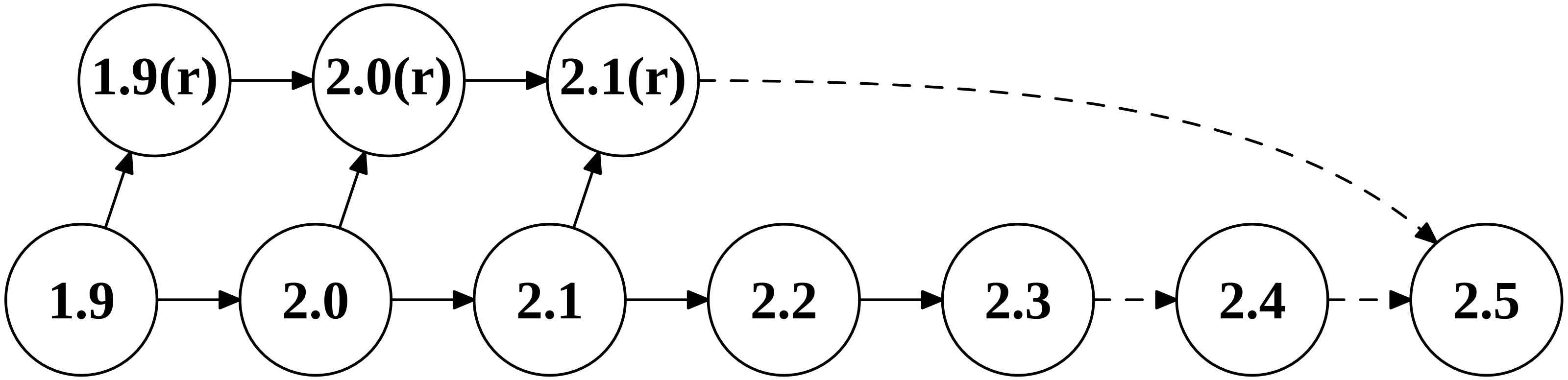
Outline

- Introduction to ACL2 2.1(r)
- Continuity in ACL2
- Differentiability in ACL2
- Is it a toy?
- Concluding Remarks

Introduction to ACL2 2.1(r)

- History and future of ACL2 2.1(r)
- Axioms of non-standard analysis
- New built-in functions in ACL2 2.1(r)
- Non-standard definition principle
- Transfer principle
- Non-standard induction principle

History and Future of ACL2 2.1(r)



The non-standard reals

- ϵ is *small* if and only if $|\epsilon| < |x|$ for all *standard* reals $x \neq 0$.
- There are non-zero *small* numbers (infinitesimals).
- $X \neq 0$ is *large* if (and only if) $1/X$ is *small*.
- A number x is *limited* if and only if it is not *large*.
- If x is *limited*, $x = {}^*x + \epsilon$ where ${}^*x$ is *standard* and ϵ is *small*. ${}^*x$ is called the *standard part* of x.

Axioms of non-standard analysis

- *small*, *limited*, *large*, and *standard* 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 $x < y$, then ${}^*x \leq {}^*y$.
- A standard set (e.g., function) is uniquely determined by its standard elements.
- A *classical* property is true of all numbers if and only if it is true of all `standard` numbers.

New built-in functions in ACL2 2.1(r)

ACL2 2.1(r) extends the number system of ACL2 to include the irrational real and complex numbers.

- `realp`, `complexp`
- `realfix`
- `floor1`

Built-in non-standard functions in ACL2 2.1(r)

- `standard-numberp`
- `i-small`, `i-limited`, `i-large`
- `i-close`
- `standard-part`
- `i-large-integer`

Non-standard definition principle

Consider the (non-classical) function

```
(defun ns-sqrt (x)
  (standard-part
   (sqrt-approx x (i-large-integer))))
```

When `x` is `standard-numberp` so is `(ns-sqrt x)`, so the following *classical* function can be defined (implicitly):

```
(defun-std sqrt (x)
  (ns-sqrt x))
```

Transfer principle

Given the following theorem:

```
(implies (and (standard-numberp x)
              (realp x) (<= 0 x))
         (equal (* (sqrt x) (sqrt x)) x))
```

The transfer principle generalizes the result to all numbers `x`, not just the `standard-numberp`.

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

Non-standard induction principle

Unrestricted use of induction leads to contradictions! E.g., all naturals are `standard-numberp`.

- “Regular” induction is only permitted on *classical* formulas
- If a formula is not classical, induction proves the formula only for standard arguments. Non-standard arguments are handled as separate “base” cases.

Non-standard induction principle: example

To prove the following theorem

```
(defthm limited-factorial
  (implies (i-limited n)
           (i-limited (factorial n))))
```

ACL2 will prove the following "extra base" case, as well as the usual base and induction cases:

```
(implies (and (not (standard-numberp n))
              (i-limited n))
         (i-limited (factorial n)))
```

Continuity in ACL2

- Continuous functions
- Intermediate value theorem
- Extreme value theorems

Continuous functions (why `encapsulate` is our friend)

A continuous function f satisfies these requirements:

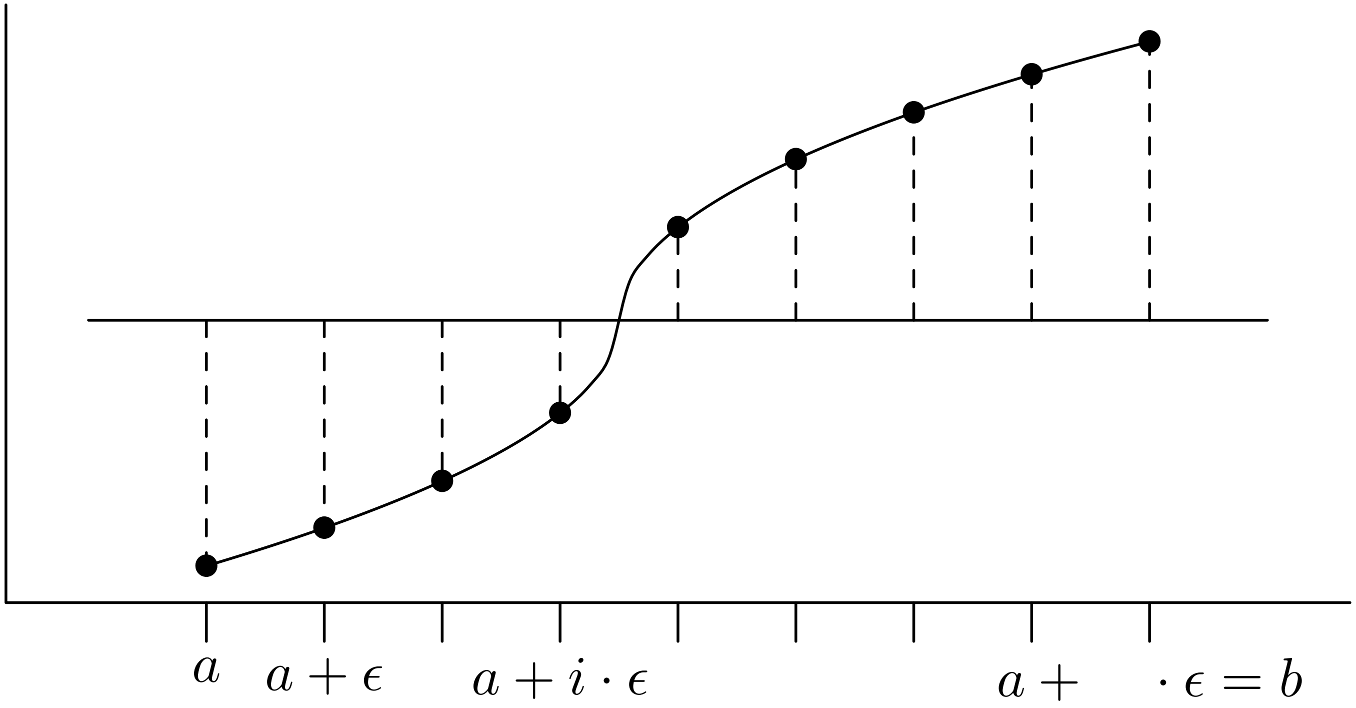
- If x is standard, so is $f(x)$.
- If x is real, so is $f(x)$.
- If x is a standard real and y is a real number close to x, then $f(x)$ is close to $f(y)$.

These properties can be captured in an `encapsulate`.

The constrained continuous functions

```
(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))))
 )
```

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 $x_0 = {}^*(a + i \cdot \epsilon)$. If ϵ is `i-small`, then x_0 is `i-close` to $a + (i + 1) \cdot \epsilon$. By continuity, $f(x_0)$ is `i-close` to both $f(a + i \cdot \epsilon)$ and $f(a + (i + 1) \cdot \epsilon)$. Since $f(x_0)$ is `standard-numberp`, $f(x_0)$ must be z.

The intermediate value theorem in ACL2

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

```
(defun find-zero-n (a z i n eps)
  (declare (xargs :measure
                  (nfix (1+ (- n i)))))
  (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)))
```

IVT in ACL2: `find-zero-n`

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

- `(< (rcfn` x_i`) z)`
- `(<= z (rcfn (+` x_i `eps)))`
- `(<= a` x_i`)`
- `(<=` x_i `(+ a (* (- n i) eps)))`

where x_i is the value of `(find-zero-n a z i n eps)`.

IVT in ACL2: `find-zero`

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
                    (i-large-integer)
                    (/ (- b a)
                       (i-large-integer))))
    0))
```

IVT in ACL2: `find-zero`

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 ...))))
```

IVT in ACL2: `find-zero`

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)))
```

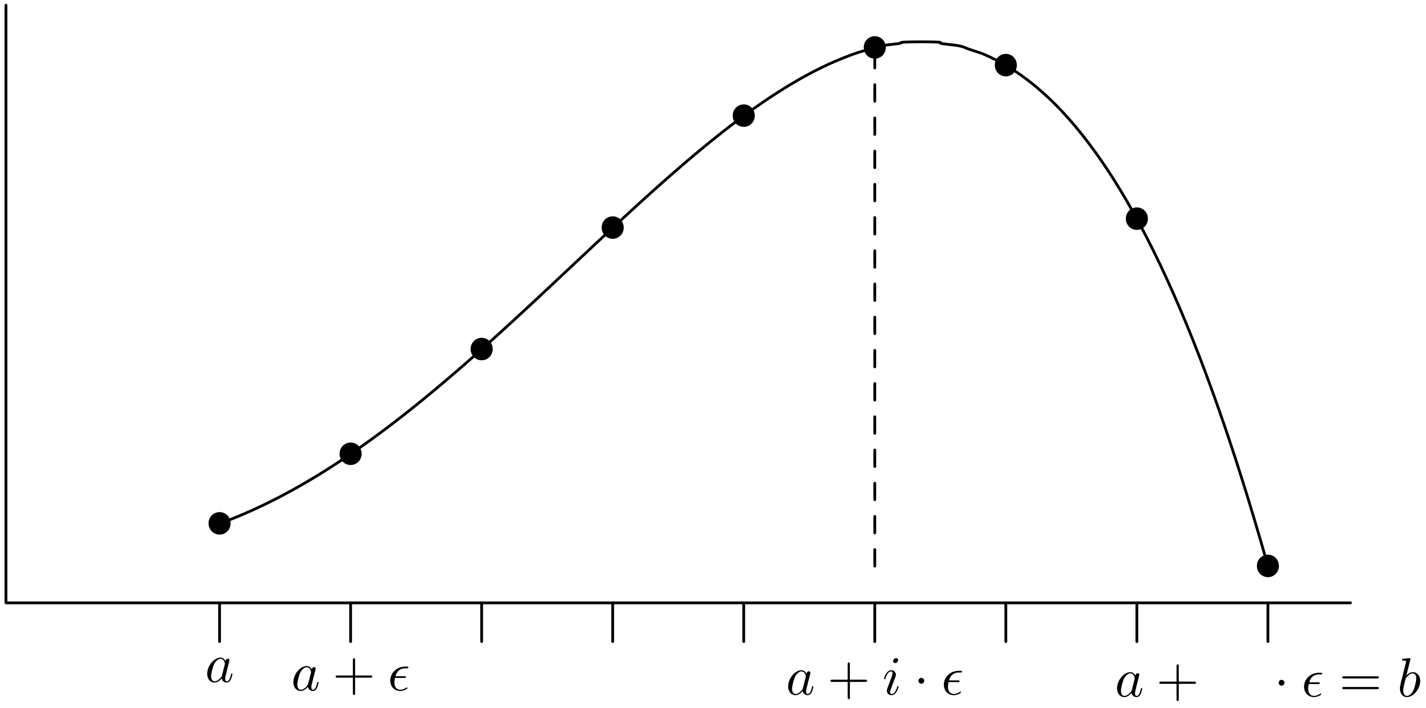
Similarly for the following:

- `(<= 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))))
```

Extreme (maximum) value theorem



Maximum value theorem

For a given $\epsilon > 0$, the value i is chosen to that $f(a + i \cdot \epsilon) \geq f(a + k \cdot \epsilon)$ for any other $a + k \cdot \epsilon$ in the grid. Note, that also means that ${}^*f(a + i \cdot \epsilon) \geq {}^*f(a + k \cdot \epsilon)$.

Let $x_0 = {}^*(a + i \cdot \epsilon)$. Since x_0 is `i-close` to $a + i \cdot \epsilon$, $f(x_0)$ is `i-close` to $f(a + i \cdot \epsilon)$, and since $f(x_0)$ is `standard-numberp`, it follows that $f(x_0) = {}^*f(a + i \cdot \epsilon)$. Thus, $f(x_0) \geq {}^*f(a + k \cdot \epsilon)$. If $y \in [a, b]$ is `standard-numberp`, it must be `i-close` to one of the $a + k \cdot \epsilon$, and $f(y) = {}^*f(a + k \cdot \epsilon)$. So, for any `standard-numberp` $y \in [a, b]$, $f(x_0) \geq f(y)$. By transfer, for any $y \in [a, b]$, $f(x_0) \geq f(y)$.

Maximum value theorem: `find-max-rcfn-x-n`

Find the grid point x_i with the maximum value of $f(x_i)$.

```
(defun find-max-rcfn-x-n (a max-x i n eps)
  (if (and (integerp i) (integerp n)
           (<= i n)
           (realp a)
           (realp eps) (< 0 eps))
      (if (> (rcfn (+ a (* i eps)))
             (rcfn max-x))
          (find-max-rcfn-x-n a (+ a (* i eps))
                             (1+ i) n eps)
        (find-max-rcfn-x-n a max-x
                           (1+ i) n eps))
    max-x))
```

Maximum value theorem: `find-max-rcfn-x-n`

The following are easy to prove:

- ```
  (<= (rcfn (+ a (* k eps)))
      (rcfn (find-max-rcfn-x-n a max-x
                               i n eps)))
  ```
- ```
  (<= a (find-max-rcfn-x-n a a 0 n eps))
  ```
- ```
  (<= (find-max-rcfn-x-n a max-x i n eps)
      (+ a (* n eps)))
  ```

The variable `n` is an arbitrary positive integer, `i` and `k` are integers between 0 and `n`, and `eps` is chosen to that `(+ a (* n eps))` is equal to `b`.
  ```

Maximum value theorem: `find-max-rcfn-x`

```
(defun-std find-max-rcfn-x (a b)
  (if (and (realp a) (realp b) (< a b))
      (standard-part
       (find-max-rcfn-x-n a a 0
                          (i-large-integer)
                          (/ (- b a)
                             (i-large-integer))
       ))
    0))
```

Notice: we must first prove that `find-max-rcfn` is `i-limited`.

Maximum value theorem: `find-max-rcfn-x`

```
(defthm-std find-max-rcfn-x->=-a
  (implies (and (realp a)
                (realp b)
                (< a b))
           (<= a (find-max-rcfn-x a b))))
```

Similarly, `(<= (find-max-rcfn-x a b) b)`.

Maximum value theorem: `find-max-rcfn-x`

```
(defthm find-max-rcfn-is-maximum-of-grid
  (implies (and (realp a) (standard-numberp a)
                (realp b) (standard-numberp b)
                (< a b)
                (integerp k)
                (<= 0 k) (<= k (i-large-integer)))
           (<= (standard-part
                (rcfn (+ a
                         (* k (/ (- b a)
                                 (i-large-integer))
                          ))))
               (rcfn (find-max-rcfn-x a b)))))
```

We need the `(standard-part (rcfn ...))` in the conclusion.

Maximum value theorem: `in-grid`

For an arbitrary $x \in [a, b]$, we can find an index i so that x is close to $a + i\epsilon$:

```
(i-close x
         (+ a (* (in-grid a x i n eps)
                 eps)))
```

It follows from continuity that

```
(i-close (rcfn x)
         (rcfn (+ a (* (in-grid a x i n eps)
                       eps))))
```

That means the two terms above have the same `standard-part`!

Maximum value theorem

For a `standard-numberp`, `(standard-part (rcfn x))` is just `(rcfn x)`. Therefore,

```
(<= (rcfn x)
    (rcfn (find-max-rcfn-x a b)))
```

Using the transfer principle:

```
(defthm-std find-max-rcfn-is-maximum
  (implies (and (realp a) (realp b) (< a b)
                (realp x) (<= a x) (<= x b))
           (<= (rcfn x)
               (rcfn (find-max-rcfn-x a b)))))
```

Differentiability in ACL2

- Differentiable functions
- Rolle's theorem
- Mean value theorem

Differentiable functions

A function f is differentiable if it satisfies the following:

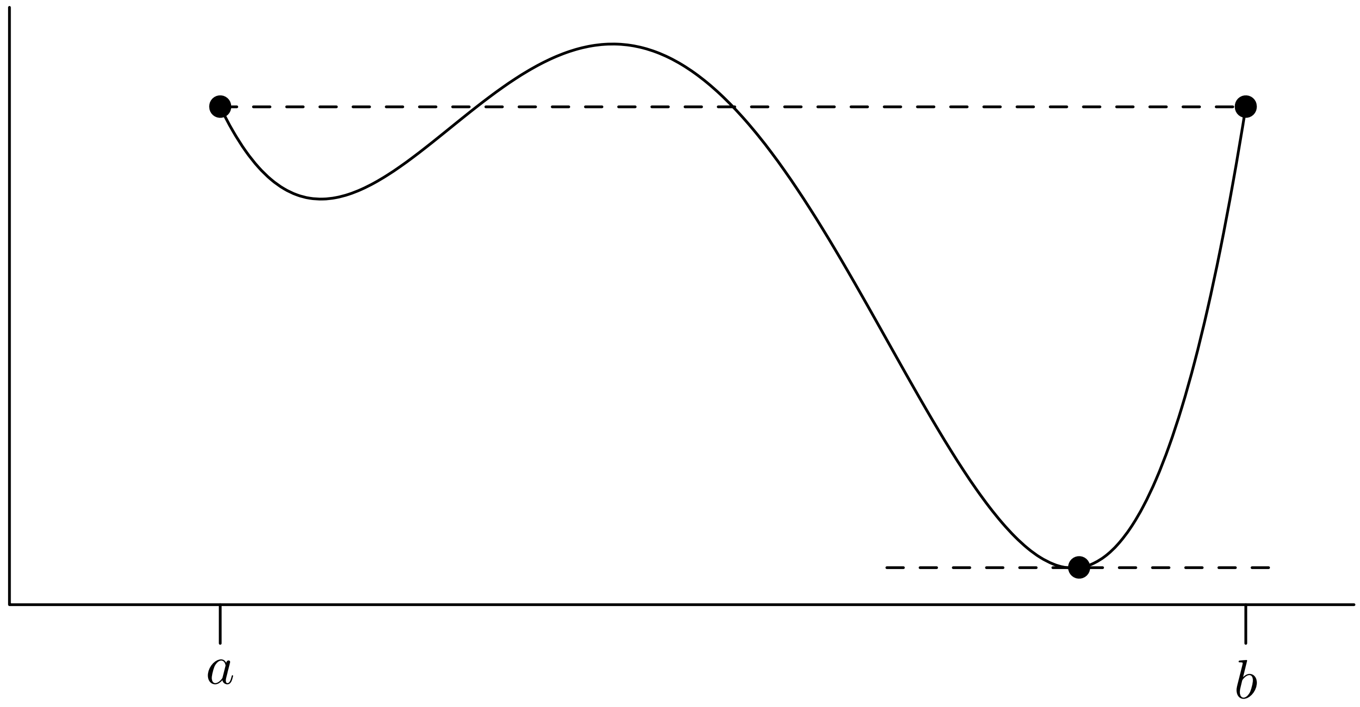
- If x is standard, so is $f(x)$.
- If x is real, so is $f(x)$.
- If x is a standard real and y_1 and y_2 are real numbers close to x, then $\frac{f(x)-f(y_1)}{x-y_1}$ is limited and close to $\frac{f(x)-f(y_2)}{x-y_2}$.

Differentiable functions

```
(encapsulate
 ((rdfn (x) t))
 ...
 (defthm rdfn-differentiable
   (implies (and (standard-numberp x)
                 (realp x) (realp y1) (realp y2)
                 (i-close x y1) (not (= x y1))
                 (i-close x y2) (not (= x y2)))
            (and (i-limited (diff-rdfn x y1))
                 (i-close (diff-rdfn x y1)
                          (diff-rdfn x y2))))))
```

where `(diff-rdfn x y)` is $\frac{f(x)-f(y)}{x-y}$.

Rolle's theorem



Rolle's theorem

The usual trick for Rolle's theorem is to use the extreme theorem to find an interior critical point — i.e., a minimum or maximum. The derivative at this point must be zero.

The first step is to prove that **rdfn** is continuous on $[a, b]$. I.e., we must show that differentiable functions are also continuous.

```
(defthm rdfn-continuous
  (implies (and (realp x) (standard-numberp x)
                (realp y) (i-close x y))
           (i-close (rdfn x) (rdfn y))))
```

Rolle's theorem: rolles-critical-point

```
(defun rolles-critical-point (a b)
  (if (equal (rdfn (find-min-rdfn-x a b))
             (rdfn (find-max-rdfn-x a b)))
      (/ (+ a b) 2)
    (if (equal (rdfn (find-min-rdfn-x a b))
               (rdfn a))
        (find-max-rdfn-x a b)
      (find-min-rdfn-x a b))))
```

The functions `find-min-rdfn-x` and `find-max-rdfn-x` are justified by `:functional-instantiation`.

Rolle's theorem

Let m and M be the minimum and maximum values of `rdfn` on $[a, b]$ respectively, and let c be the critical point. Then,

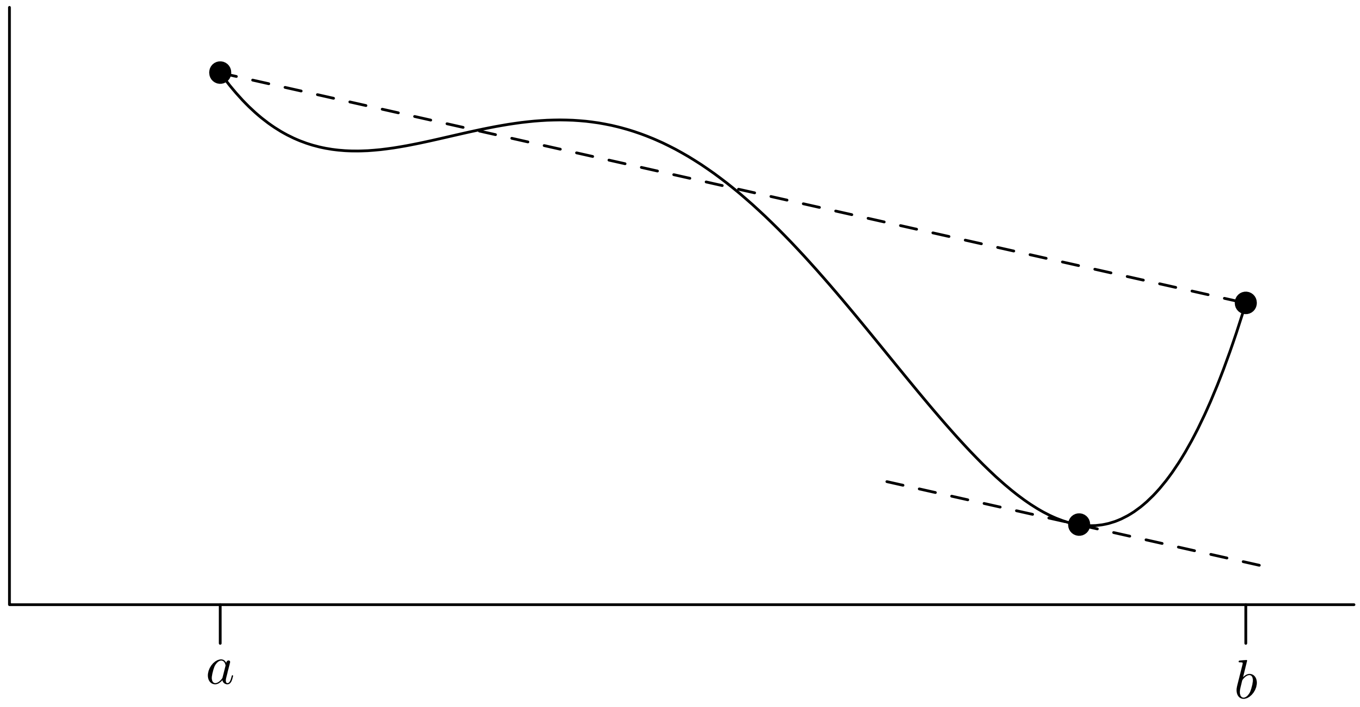
- if $m = M$, `rdfn` is a constant function, so the value of any differential at c will be zero
- at m, the value of any left-differential is negative and the value of any right-differential is positive
- at M, the value of any left-differential is positive and the value of any right-differential is negative

Since `rdfn` is differentiable at c, all the differentials at c must be `i-close`. I.e., the `standard-part` of an arbitrary differential must be zero.

Rolle's theorem

```
(defun differential-rdfn (x eps)
  (/ (- (rdfn x) (rdfn (+ x eps))) (- eps)))
(defun derivative-rdfn (x)
  (standard-part
   (differential-rdfn x (/ (i-large-integer)))))
(defthm rolles-theorem
  (implies (and (realp a) (standard-numberp a)
                (realp b) (standard-numberp b)
                (= (rdfn a) (rdfn b))
                (< a b))
           (equal (derivative-rdfn
                   (rolles-critical-point a b))
                  0)))
```

Mean value theorem



Mean value theorem: `rdfn2`

The trick is to define a new function g so that $g' = f' - m$ and $g(a) = g(b)$. From Rolle's theorem, $g'(c) = f'(c) - m = 0$ so $f'(c) = m$.

```
(defun rdfn2 (x)
  (+ (rdfn x)
     (- (* (- (rdfn (b)) (rdfn (a)))
           (- x (a))
           (/ (- (b) (a)))))
     (- (rdfn (a)))))
```

Notice: `rdfn2` depends on the endpoints of $[a, b]$ as well as on `rdfn`.

Mean value theorem: `rdfn2`

To invoke Rolle's theorem on **`rdfn2`**, we must show that it is a continuous function. This is trivial, but the following theorem shows why the range $[a, b]$ is given in an **`encapsulate`** instead of as extra arguments to **`rdfn2`**.

```
(defthm rdfn2-standard
  (implies (standard-numberp x)
           (standard-numberp (rdfn2 x))))
```

Mean value theorem

Invoking Rolle's theorem gives the following:

```
(defthm rolles-theorem-2
  (implies (and (realp a) (standard-numberp a)
                (realp b) (standard-numberp b)
                (= (rdfn2 a) (rdfn2 b))
                (< a b))
           (equal (derivative-rdfn2
                   (rolles-critical-point-2 a b))
                  0)))
(defthm rolles-theorem-2-specialized
  (equal (derivative-rdfn2
          (rolles-critical-point-2 (a) (b)))
         0))
```

Mean value theorem

```
(defthm mvt-theorem
  (equal (derivative-rdfn
           (rolles-critical-point-2 (a) (b)))
         (/ (- (rdfn (b)) (rdfn (a)))
            (- (b) (a)))))
```

Is it a toy?

The jury is still out, until more people use it to prove interesting theorems. Here is a (partial) list of theorems proved so far.

- $x \geq 0 \Rightarrow \sqrt{x} \cdot \sqrt{x} = x$
- $e^{x+y} = e^x \cdot e^y$
- e^x is continuous
- $e^{i\pi} = -1$
- $\sin^2(x) + \cos^2(x) = 1$
- $\cos(\pi/2) = 0$ (using IVT)
- $\pi \approx 3.14159265358979323846$
- fundamental theorem of calculus (Matt)

Concluding Remarks

- Using non-standard analysis, it is possible to reason about the real numbers *in the context of ACL2*
- Induction played a major role in some of the proofs (IVT and EVT)
- With `encapsulate` we can reason about classes of function; e.g. "all differentiable functions are continuous"
- Unfortunately, to use `encapsulate` we need to write mundane functions (and justifying theorems), e.g., `find-zero-n`, `limited-find-zero-body`, `find-zero`.
- Two powerful tandems: `encapsulate` and `defthm`, and `defun-std` and `defthm-std`