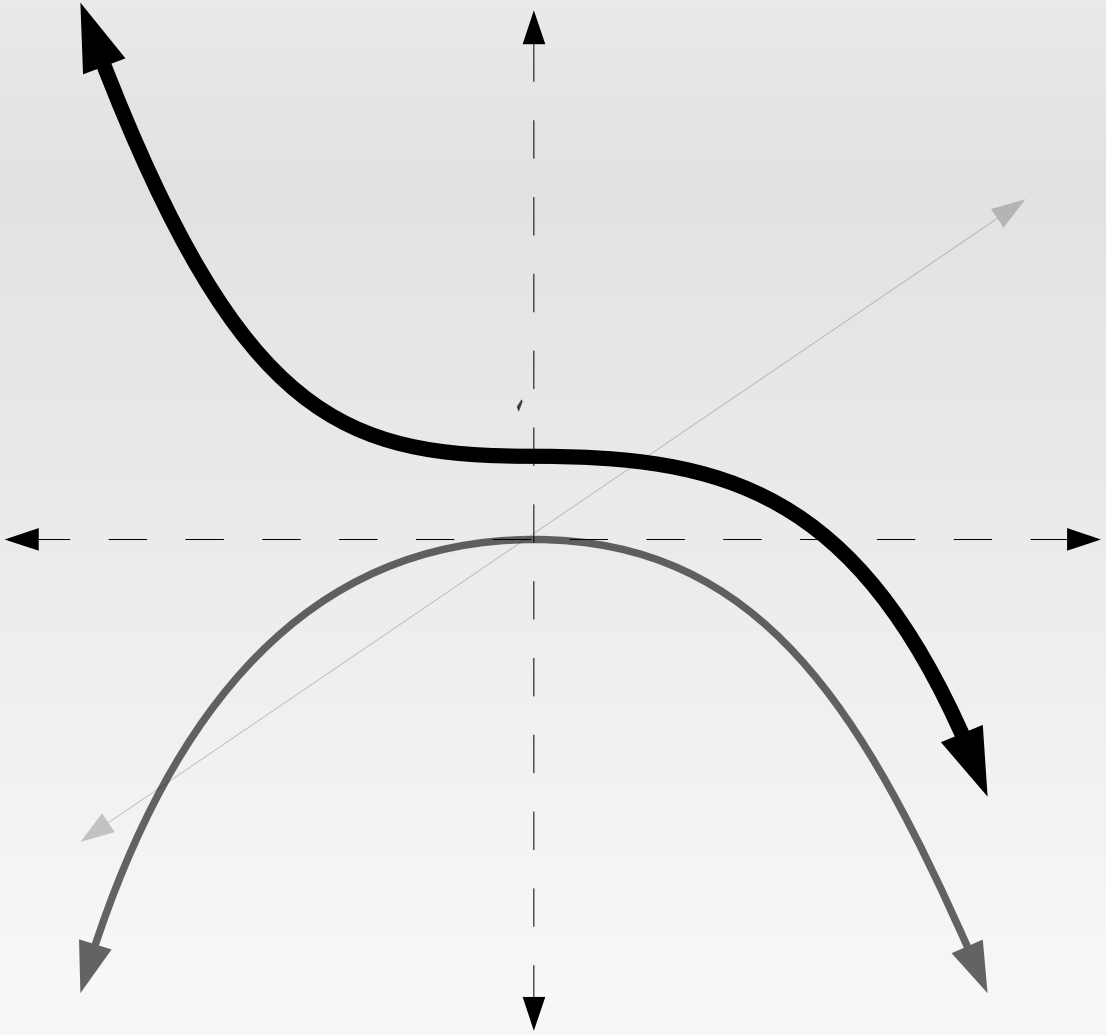
Automatic Differentiation In ACL2



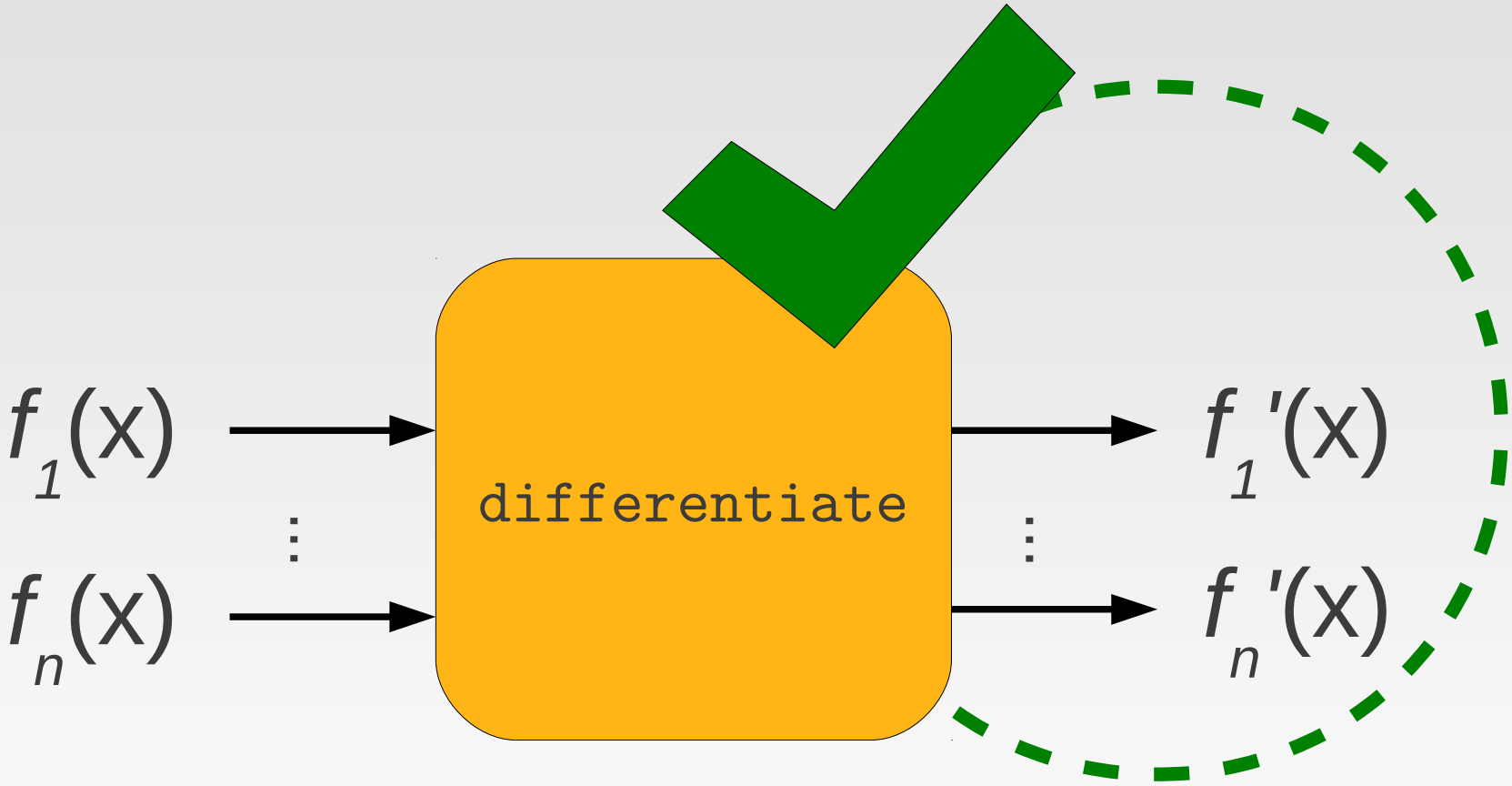
$$\frac{d}{dx} f(x) \approx$$

```
(/ (- (f x) (f y))
   (- x y))
```

Peter Reid – University of Oklahoma
Ruben Gamboa – University of Wyoming

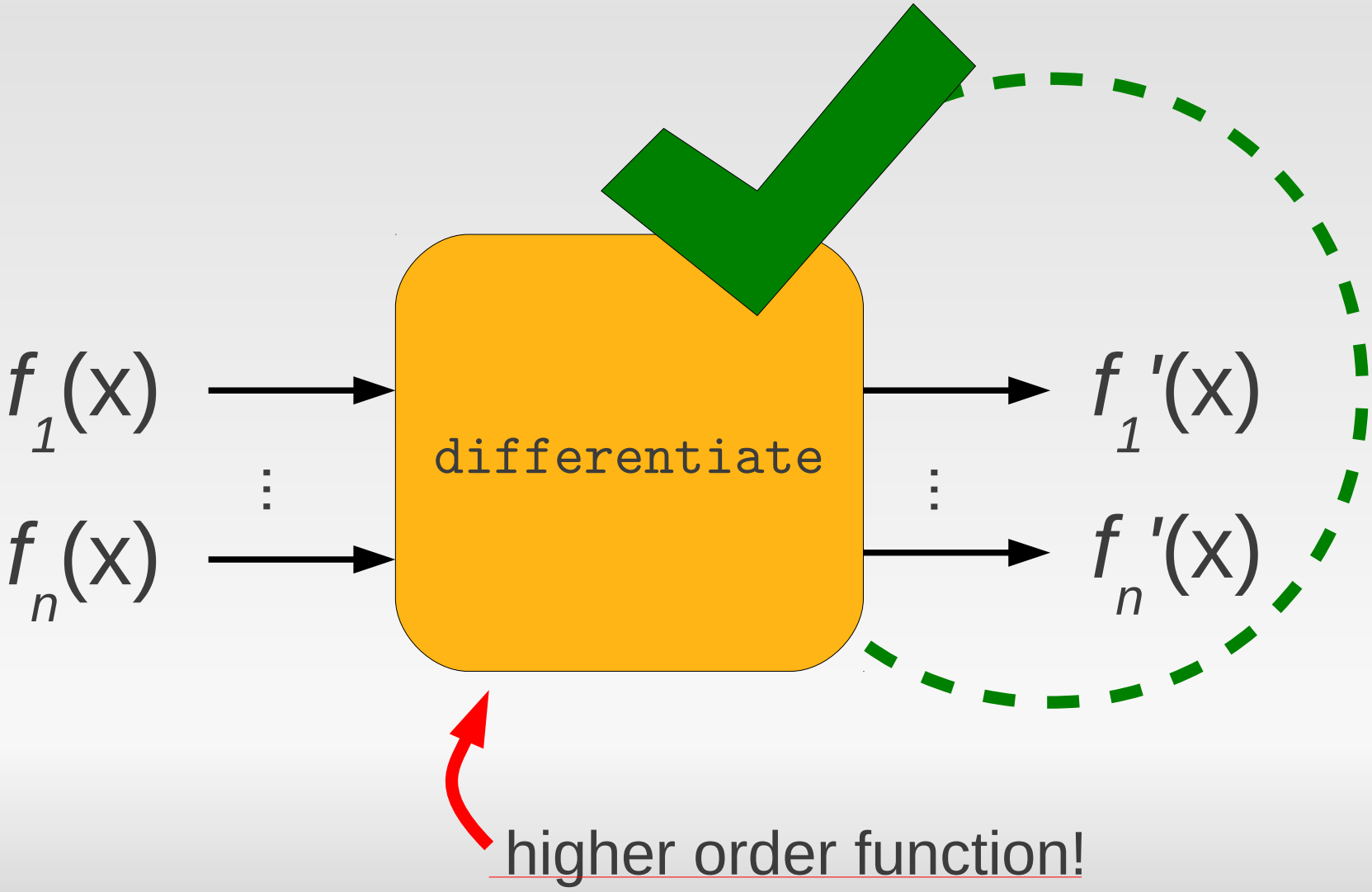
The Goal

- Build a differentiator in ACL2



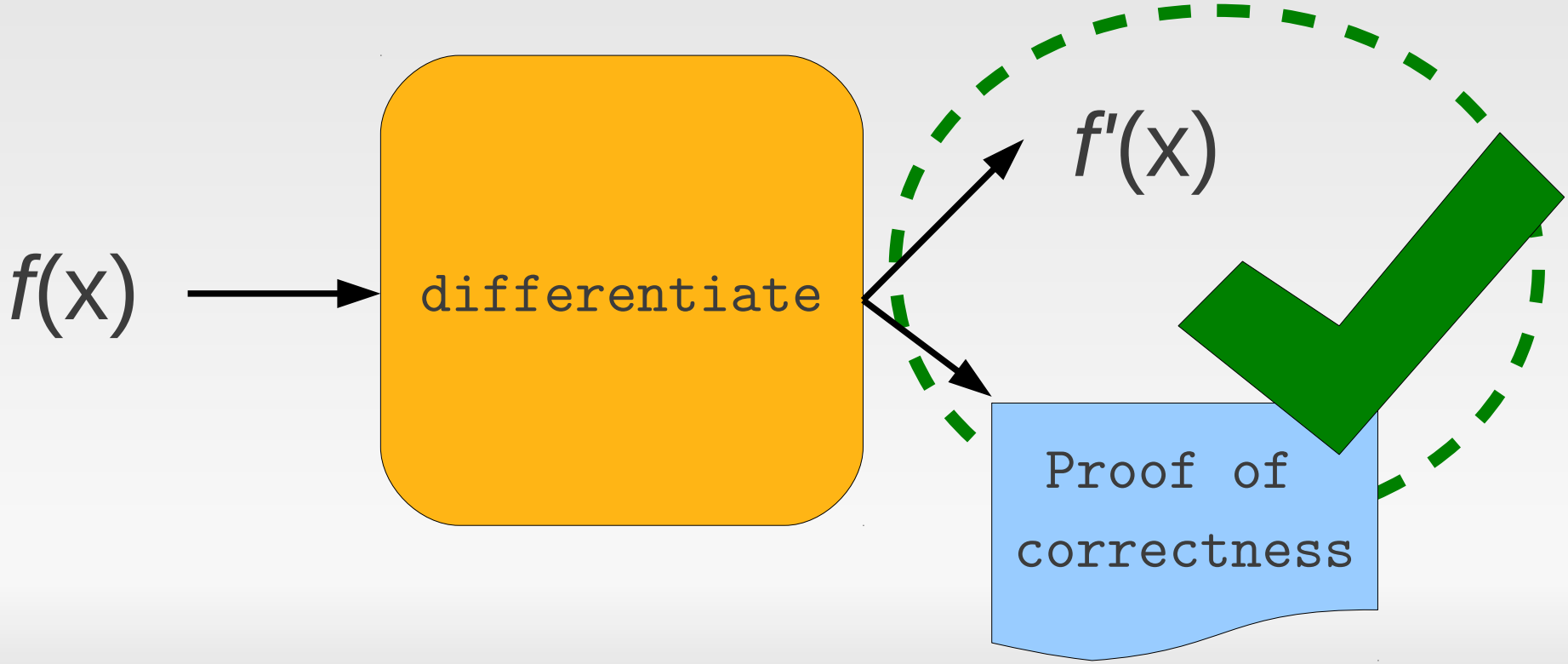
The Goal

- Build a differentiator in ACL2



The Goal

- Build a differentiator in ACL2

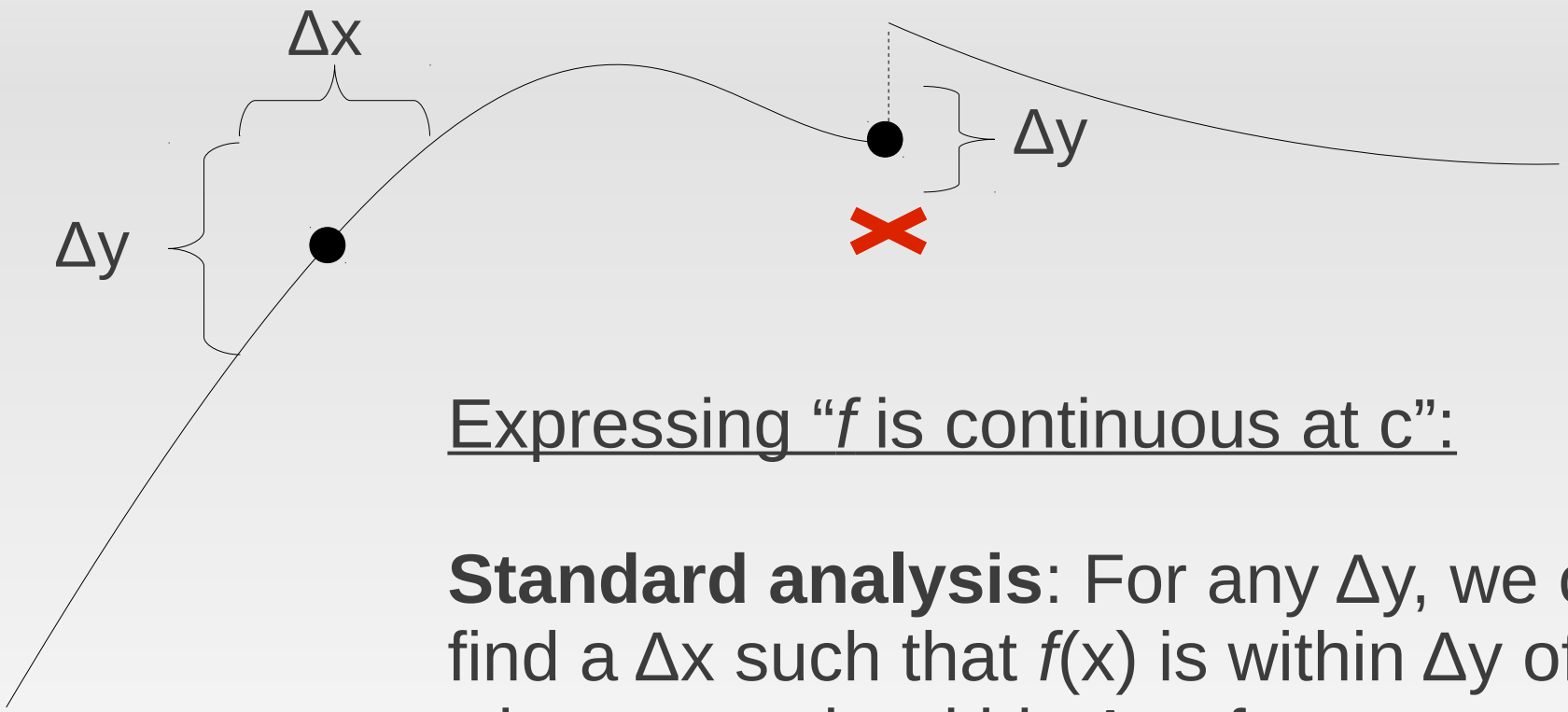


The Goal

- For any elementary function, prove a theorem of this form:

```
(implies (and (DOMAIN-P x)
              (DOMAIN-P y)
              (standardp x)
              (i-close x y)
              (not (equal x y)))
         (i-close (/ (- (F x) (F y))
                     (- (x y)))
                  (F-PRIME x)))
```

Continuity

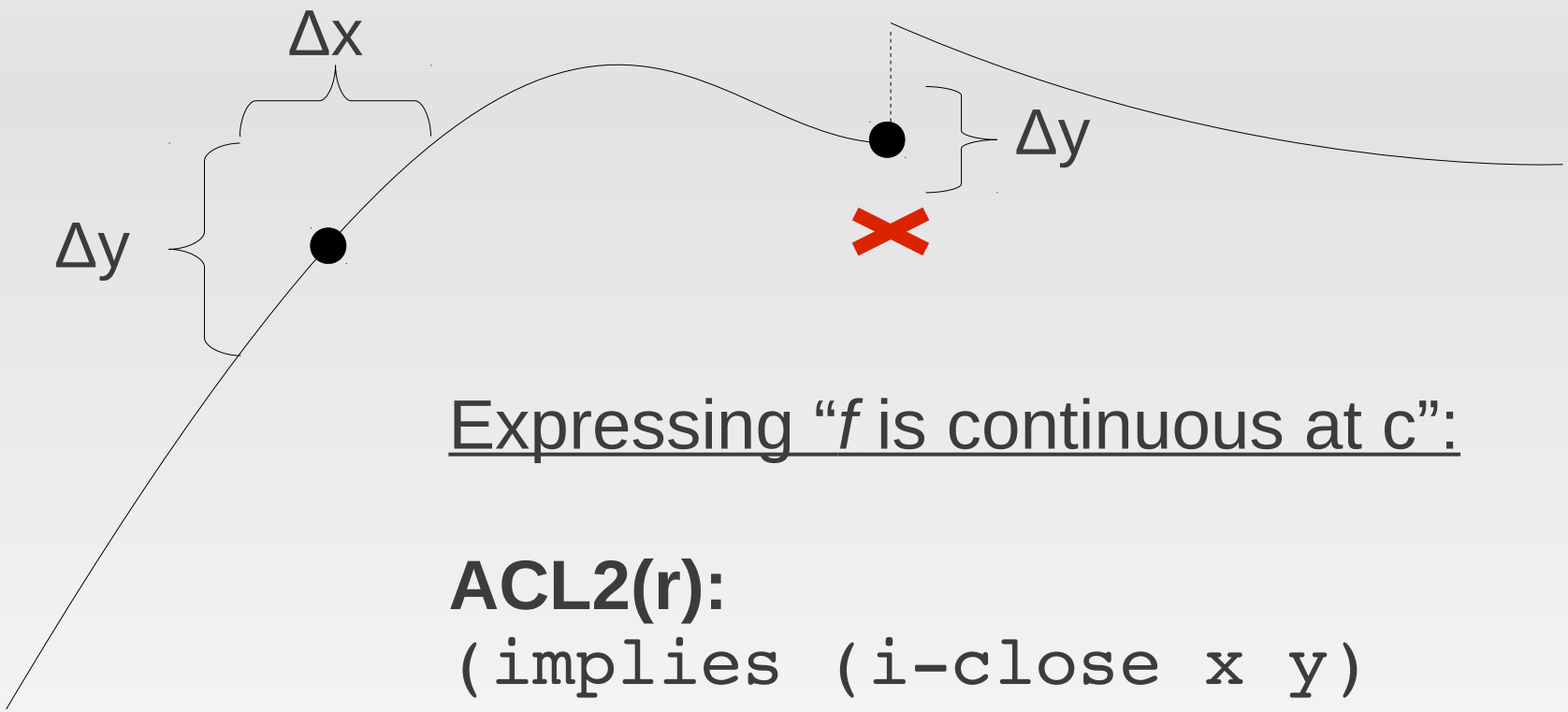


<u>Expressing "*f* is continuous at c":</u>

Standard analysis: For any Δy, we can find a Δx such that *f*(x) is within Δy of *f*(c) whenever x is within Δx of c.

Nonstandard analysis: If x is close to c, then *f*(x) is close to *f*(c).

Continuity

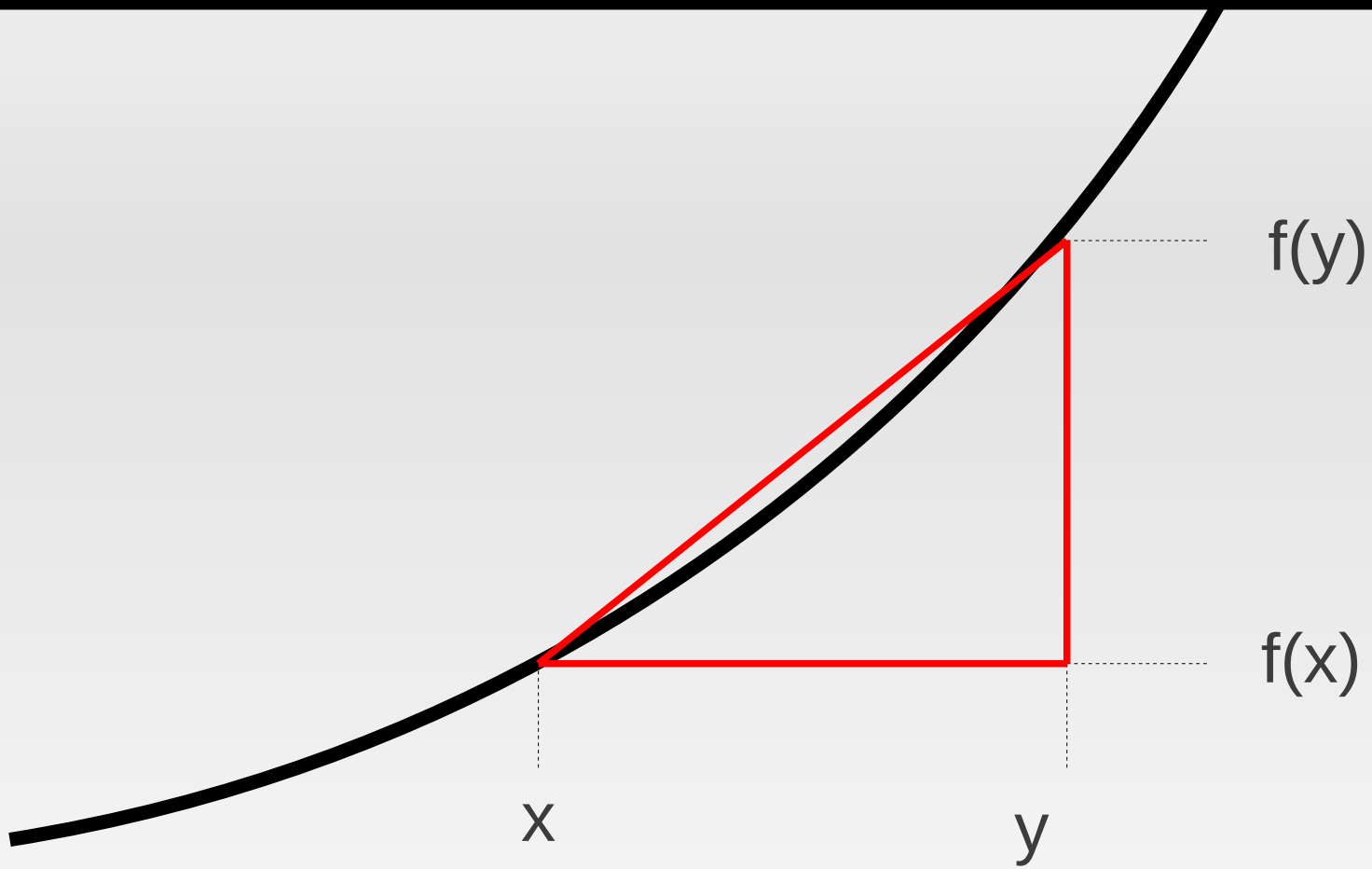


Expressing "*f* is continuous at c":

ACL2(r):

```
(implies (i-close x y)
         (i-close (f x) (f y)))
```

Derivatives



Standard analysis

$$f'(x) = \lim_{y \to x} \frac{f(y) - f(x)}{y - x}$$

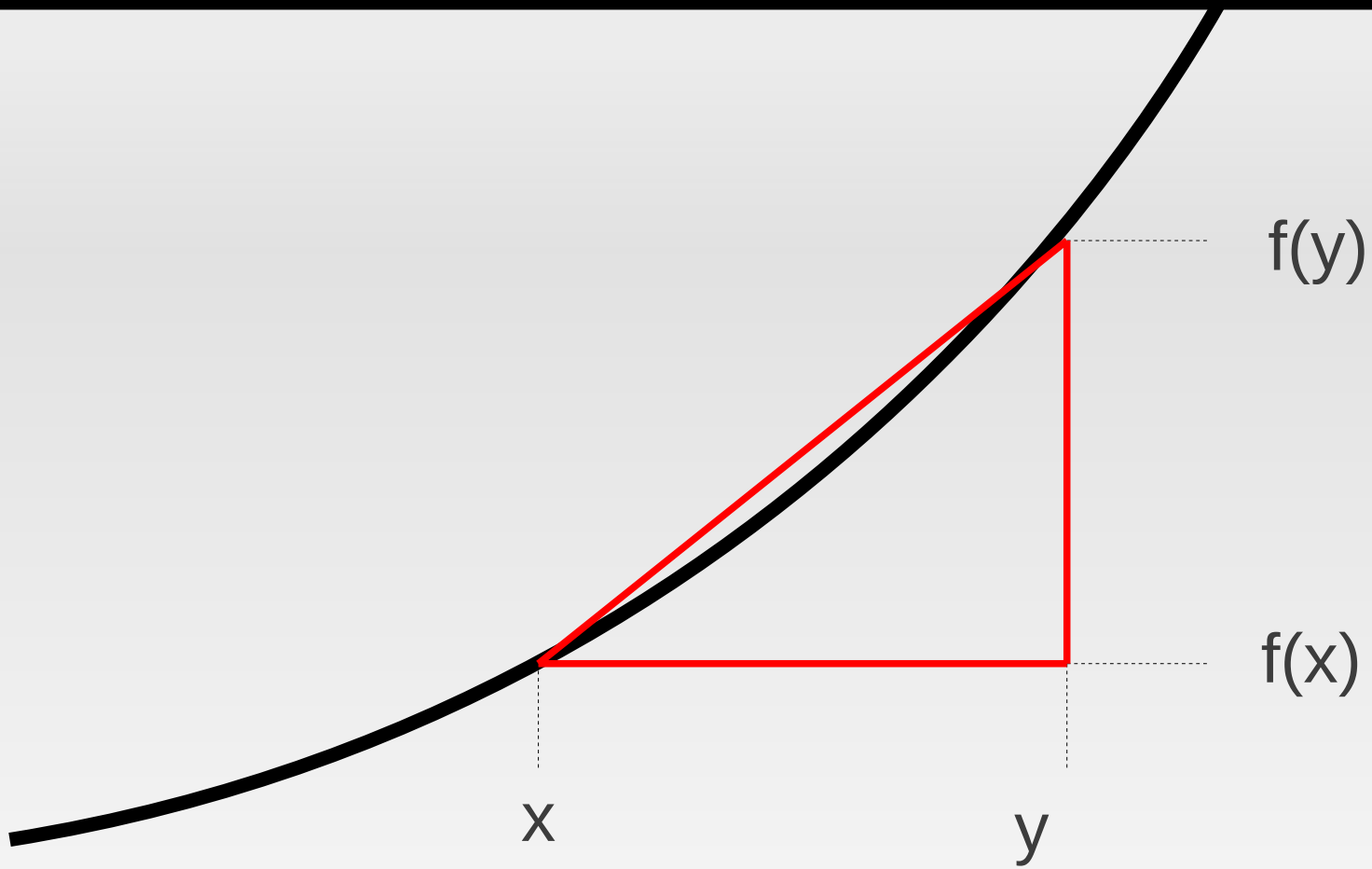
Derivatives

$$f'(x) = \lim_{y \to x} \frac{f(y) - f(x)}{y - x}$$

$$\forall\, \epsilon > 0. \exists \delta > 0. \forall\, y. (|x - y| < \delta) \Rightarrow \left| \frac{f(y) - f(x)}{y - x} - f'(x) \right| < \epsilon$$

Possible but impractical to work with.

Derivatives



Standard analysis

$$f'(x) = \lim_{y \to x} \frac{f(y) - f(x)}{y - x}$$

Nonstandard analysis

$$f'(x) \approx \frac{f(y) - f(x)}{y - x}$$

Nonstandard derivatives

- For x **standard** and y **close** (but not equal) to x

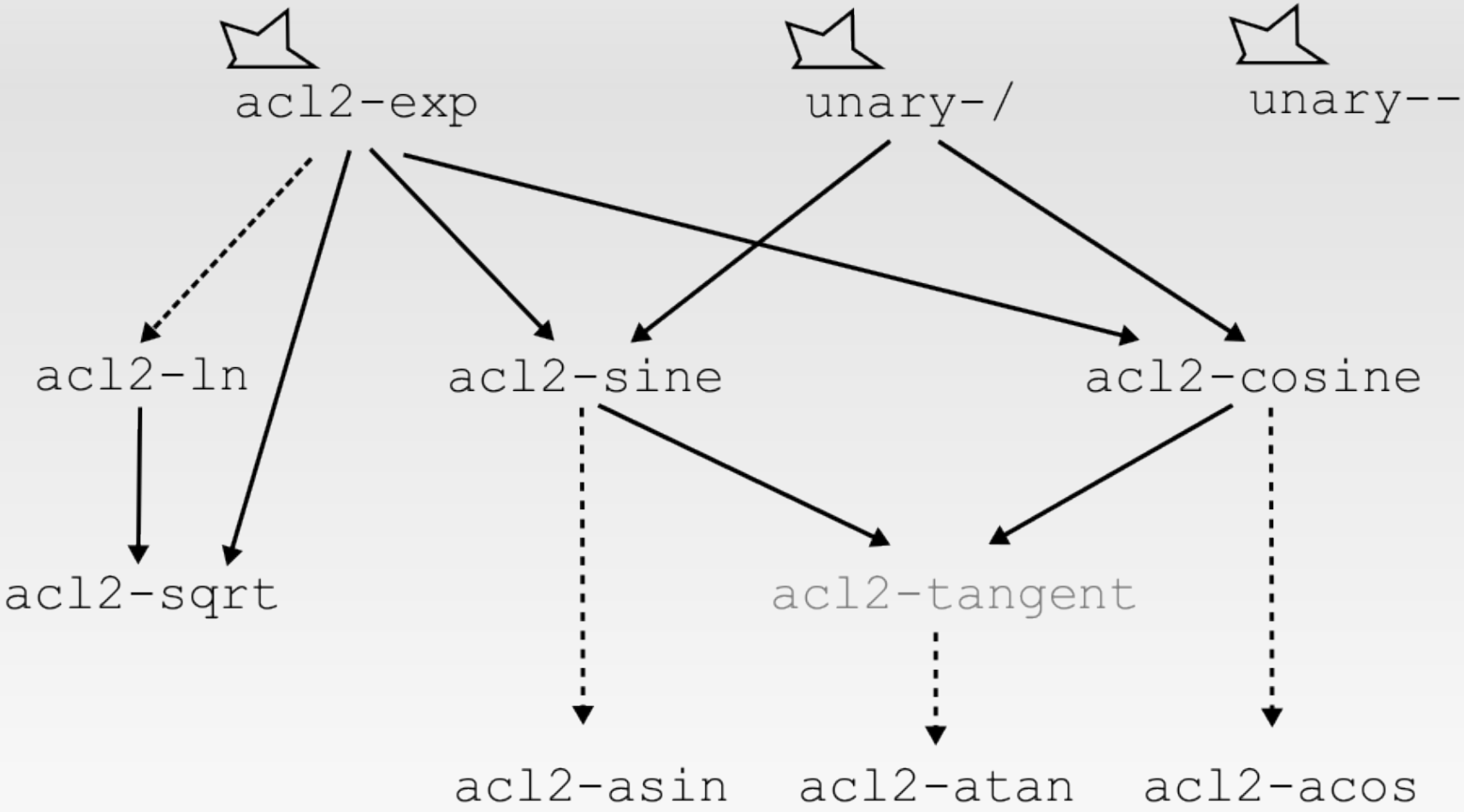
$$f'(x) \approx \frac{f(x) - f(y)}{x - y}$$

```
(implies (and (acl2-numberp x)
              (acl2-numberp y)
              (standardp x)
              (i-close x y)
              (not (equal x y)))
         (i-close (/ (- (f x) (f y))
                     (- (x y)))
                  (f-prime x)))
```

Parts of `defderivative`

1) Simple derivatives
   - e^x, sin(x), √x, etc
2) Algebraic combination theorems
   - (f + g)(x), (f × g)(x), (f ○ g)(x)
3) Macro to assemble them

Base cases


acl2-exp
unary-/
unary--
acl2-ln
acl2-sine
acl2-cosine
acl2-sqrt
acl2-tangent
acl2-asin
acl2-atan
acl2-acos

Combination theorems

- ACL2 is a first-order logic. It cannot directly apply theorems about general functions

  (For example, $(f \times g)' = f'g + fg'$ - the product rule - is a theorem about general functions)

- But: It can follow instructions about applying these sorts of theorems.

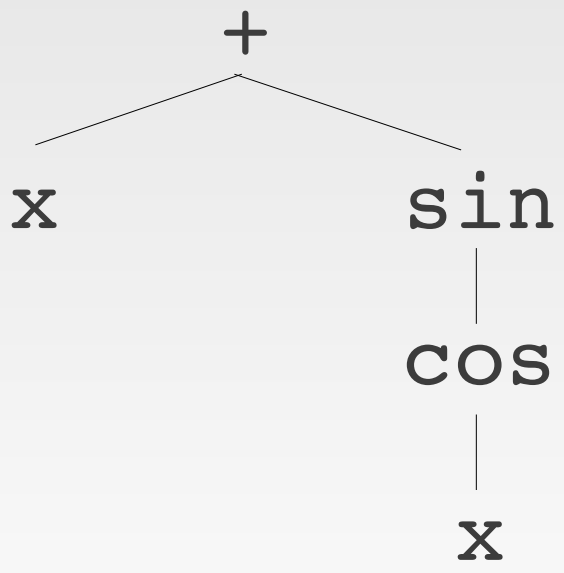
Combination theorems

- Example – the product rule
- Assumptions:
  - f, g, f', and g' are continuous
  - f and g return standard numbers for standard input
  - The derivative of f really is f', and that of g is g'
- Prove*:*
  - ($f \times g$) is continuous
  - ($f \times g$) returns standard numbers for standard input
  - The derivative of ($f \times g$) is $fg' + f'g$

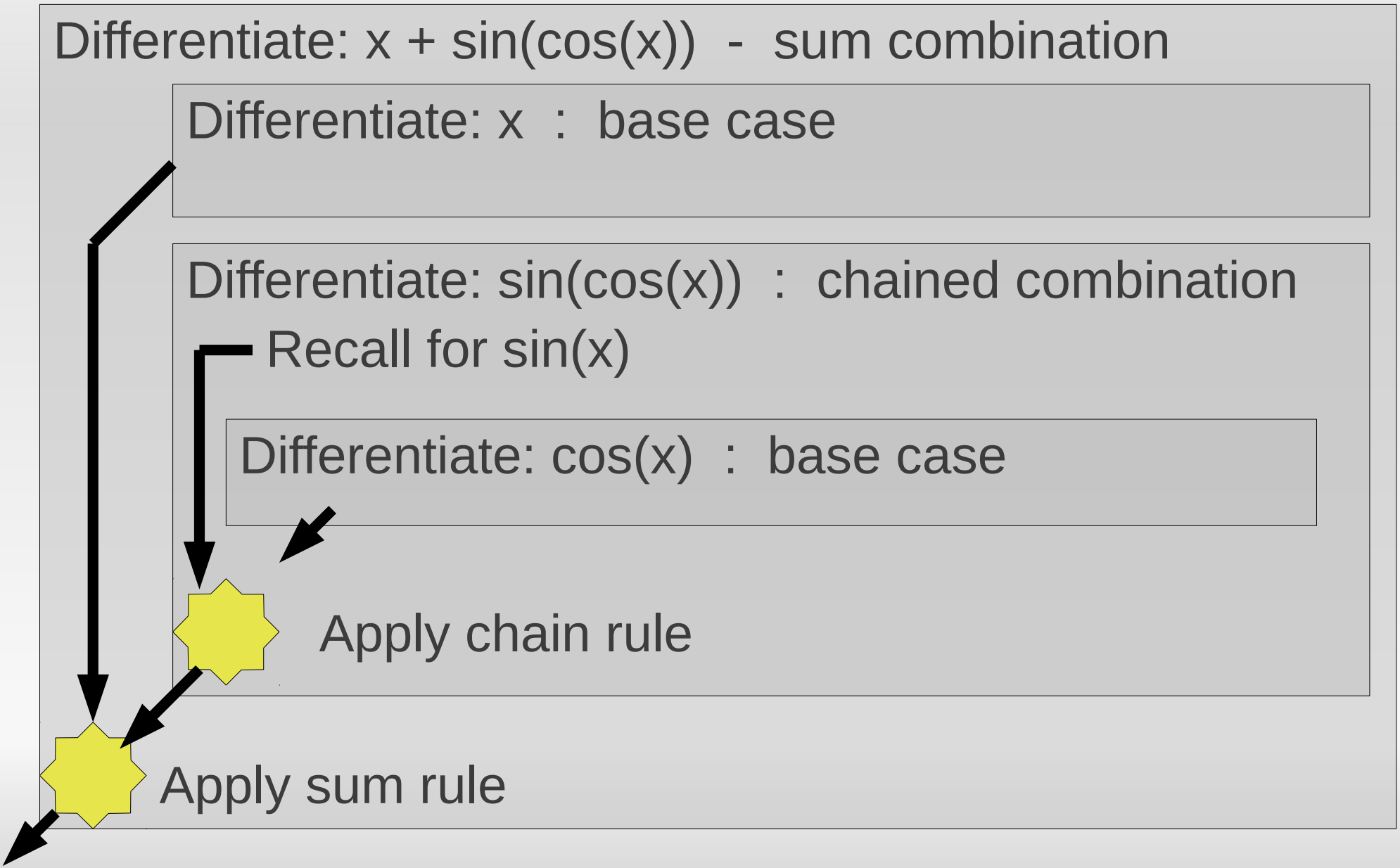
The `defderivative` Macro

- Consider an example: x + sin(cos(x))

```
(+ x (sin (cos x)))
```



The `defderivative` Macro

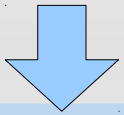


The `defderivative` Macro

- To differentiate `F`, we need:
  - Its derivative, `F-PRIME`
  - Its domain, `DOMAIN-P`
  - A proof that `F` is well behaved
  - A proof that `F-PRIME` is well behaved
  - A proof that `F-PRIME` is actually `F`'s derivative

Result

```
(defderivative sqrt-1+x**2-derivative-lemma
   (acl2-sqrt (+ 1 (* x x))))
```



```
(implies (and (acl2-numberp x)
              (realp (+ 1 (* x x)))
              (< 0 (+ 1 (* x x)))
              (acl2-numberp y)
              (realp (+ 1 (* y y)))
              (< 0 (+ 1 (* y y)))
              (standardp x)
              (i-close x y)
              (not (equal x y)))
         (i-close (/ (- (acl2-sqrt (+ 1 (* x x)))
                        (acl2-sqrt (+ 1 (* y y))))
                     (- x y))
                  (* (/ 1/2 (acl2-sqrt (+ 1 (* x x))))
                     (+ 0 (+ (* x 1) (* x 1)))))))
```

Result

The user can manually rewrite the derivative to a more normal form:

```
(defthm sqrt-1+x**2-derivative
  (implies (and (realp x)
                (realp y)
                (standardp x)
                (i-close x y)
                (not (equal x y)))
           (i-close (/ (- (acl2-sqrt (+ 1 (* x x)))
                          (acl2-sqrt (+ 1 (* y y))))
                       (- x y))
                    (/ x (acl2-sqrt (+ 1 (* x x)))))))
```

Extending the system

Suppose you have some new functions that you want to differentiate:

```
(defun acl2-sinh (x)
  (/ (- (acl2-exp x) (acl2-exp (- x)))
     2))

(defun acl2-cosh (x)
  (/ (+ (acl2-exp x) (acl2-exp (- x)))
     2))
```

Extending the system

Use `defderivative` to differentiate their bodies

```
(defderivative acl2-sinh-lemma
  (/ (- (acl2-exp x) (acl2-exp (- x)))
     2))
```

Extending the system

Modify that derivative to be in the form you expect.

```
(defthm acl2-sinh-derivative
  (implies (and (acl2-numberp x)
                (acl2-numberp y)
                (standardp x)
                (i-close x y)
                (not (equal x y)))
           (i-close (/ (- (acl2-sinh x)
                          (acl2-sinh y))
                       (- X Y))
                    (acl2-cosh x))))
```

Extending the system

Register the new derivative with the system

```
(def-elem-derivative acl2-sinh
  elem-acl2-sinh
  (acl2-numberp x)
  (acl2-cosh x))
```

theorems' names

domain

derivative

Final result

- Differentiate any elementary function
- Handle functions of your own

Built a domain-specific
theorem prover within ACL2(r).

Questions?