DoubleCheck Your Theorems

Carl Eastlund
cce@ccs.neu.edu

Northeastern University
Boston, Massachusetts
A Tale Of Two Students
Happy Student

`; s qr : Int -> Int`
`(defun s qr (x)`
  `(* x x))`

`; All squares are nonnegative.`
`(defthm s qr>=0`
  `(implies (integerp x)`
    `  (>= (s qr x) 0)))`

Welcome to DrScheme, version 4.1 [3m].
Language: Dracula v4.2.
> (s qr 2)
4
>
Happy Student

```lisp
;; sqr : Int -> Int
(defun sqr (x)
 (* x x))

;; All squares are nonnegative.
(defthm sqr>=0
 (implies (integerp x)
  (>= (sqr x) 0)))
```
Happy Student

;; sqr : Int -> Int
(defun sqr (x)
  (* x x))

;; All squares are nonnegative.
(defthm sqr>=0
  (implies (integerp x)
    (>= (sqr x) 0)))

Q.E.D.

Summary
Form:  ( DEPTHM SQR>=0 ...)
Rules: ((:DEFINITION NOT)
        (:DEFINITION SQR)
Sad Student

;;; sqr : Int -> Int
(defun sqr (x)
  x)

;;; All squares are nonnegative.
(defun sqr>=0
  (implies (integerp x)
    (>= (sqr x) 0)))

(IMPLIES (INTEGERP X) (<= 0 X)).

Name the formula above *1.

No induction schemes are suggested by *1. Consequently, the proof attempt has failed.
Sad Student

```lisp
;; sqr : Int -> Int
(defun sqr (x)
  x)

;; Unit tests:
(check-expect (sqr 0) 0)
(check-expect (sqr 2) 4)
```

Ran 2 checks.
1 of the 2 checks failed.

Actual value 2 differs from 4, the expected value.

In /Users/cce/Desktop/sqr.lisp at line 10 column 0
Sad Student

;; sqr : Int -> Int
(defun sqr (x)
  (+ x x))

;; Unit tests:
(check-expect (sqr 0) 0)
(check-expect (sqr 2) 4)

Welcome to DrScheme, version 4.1 [3m].
Language: Dracula v4.2.
All tests passed!
Mad Student

;;; sqr : Int -> Int
(defun sqr (x)
  (+ x x))

;;; Unit tests:
(check-expect (sqr 0) 0)
(check-expect (sqr 2) 4)

;;; All squares are nonnegative
(defthm sqr>=0
  (implies (integerp x)
           (>= (sqr x) 0)))

(IMPLIES (INTEGERP X) (<= 0 (+ X X))).

Name the formula above *1.

No induction schemes are suggested by *1. Consequently, the proof attempt has failed.
Mad Student
Mad Student

;;; sqr : Int -> Int
(defun sqr (x)
  (+ x x))

;;; Unit tests:
(check-expect (sqr 0) 0)
(check-expect (sqr 2) 4)
(check-expect (let ((x '-30)) (>= (sqr x) 0)) t)

Language: Dracula
> (sqr -30)
-60
>

Ran 3 checks.
1 of the 3 checks failed.

Actual value nil differs from t, the expected value.
In /Users/cce/Desktop/sqr.lisp at line 12 column 0
Mad Student

```lisp
;; sqr : Int -> Int
(defun sqr (x)
  (+ x x))

;; Unit tests:
(check-expect (sqr 0) 0)
(check-expect (sqr 2) 4)
(check-expect (sqr -30) 900)
```

Test Results

Ran 3 checks.
1 of the 3 checks failed.

Actual value -60 differs from 900, the expected value.

In /Users/cce/Desktop/sqr.lisp at line 12 column 0
Another Happy Student

```lisp
;;; sqr : Int -> Int
(defun sqr (x)
  (* x x))

;;; Unit tests:
(check-expect (sqr 0) 0)
(check-expect (sqr 2) 4)
(check-expect (sqr -30) 900)

;;; All squares are nonnegative.
(defunproperty sqr>=0
  (x :where (integerp x)
    :value (random-integer))
  (>= (sqr x) 0))

Language: Dracula v4.2.
All tests passed!
> 
```
Check and DoubleCheck
Check

(defthm sqr>=0

  (implies (integerp x)
    (>= (sqr x) 0)))
DoubleCheck

(include-book "doublecheck" :dir :teachpacks)

(defproperty sqr>=0
  (x)
  (implies (integerp x)
    (>= (sqr x) 0)))

(generate-properties)
(include-book "doublecheck" :dir :teachpacks)

(defproperty sqr>=0
  (x :value (random-integer)
    :where (integerp x))
  (>= (sqr x) 0))

(generate-properties)
(include-book "doublecheck" :dir :teachpacks)

(defproperty sqr>=0 :repeat 1000 :limit 5000
  (x :value (random-integer)
    :where (integerp x))
  (>= (sqr x) 0))

(generate-properties)
(defthm sqr>=0

  (implies (integerp x)
    (>= (sqr x) 0)))
Random Generators

(random-boolean)  (random-natural)
(random-char)     (random-integer)
(random-string)  (random-rational)
(random-symbol)  (random-number)

(random-atom)     (random-data-size)
(random-sexp)     (random-between lo hi)

(random-element-of lst)
Random Generators

(random-list-of expr [:size size])
(random-sexp-of expr [:size size])

(random-case expr [:weight weight] ...)

(defrandom name (arg ...) body)
Random Generators

(random-list-of expr [:size size])
(random-sexp-of expr [:size size])

(random-case expr [:weight weight] ...)

(defrandom name (arg ...) body)

; random-multiset : [Listof X] -> [Listof X]
(defrandom random-multiset (elements)
  (random-case
    nil :weight 1/4
    (cons (random-element-of elements)
      (random-multiset elements)))))
RealityCheck

• Random testing
  ◦ based on SchemeUnit
  ◦ `defproperty` constructs test suite & all cases
  ◦ `generate-properties` runs all suites
  ◦ random values pulled from lazy stream

• Theorem proving
  ◦ macro-expands to `defthm`
  ◦ generators are vacuous, program-mode
Other Approaches


Thank You.