

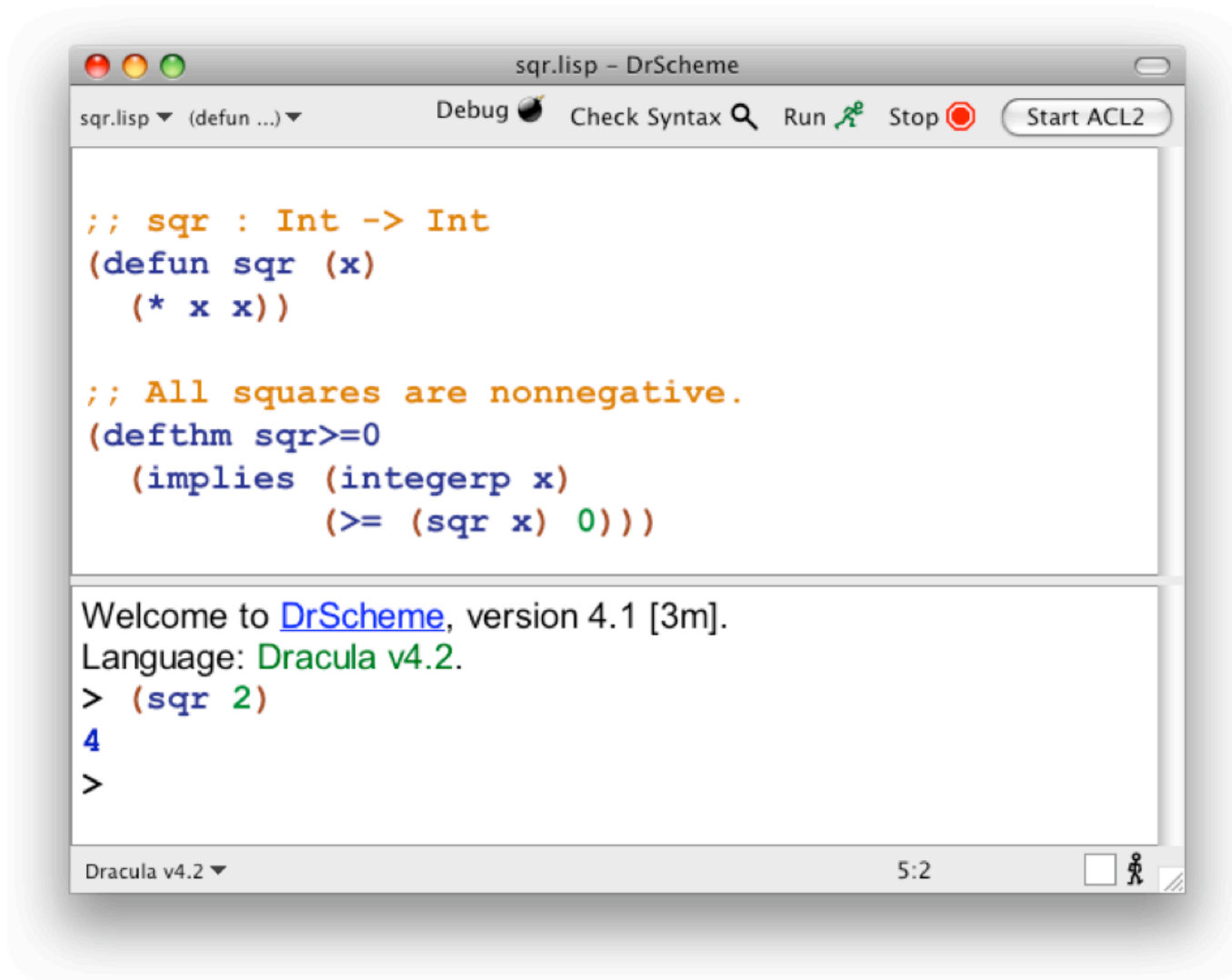
DoubleCheck Your Theorems

Carl Eastlund
cce@ccs.neu.edu

Northeastern University
Boston, Massachusetts

A Tale Of Two Students

Happy Student



```
sqr.lisp - DrScheme
Debug Check Syntax Run Stop Start ACL2

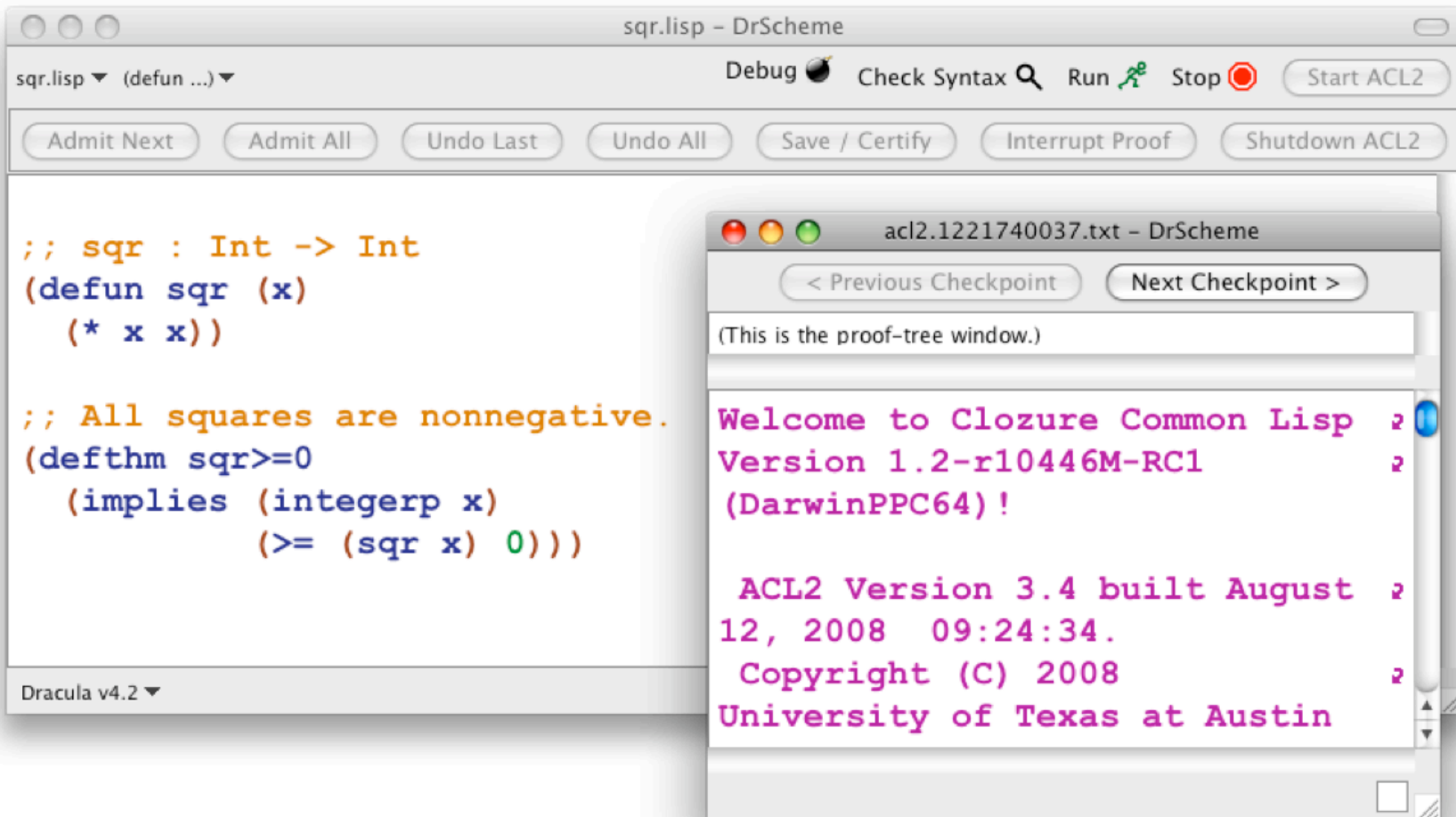
;; sqr : Int -> Int
(defun sqr (x)
  (* x x))

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

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

Dracula v4.2 5:2
```

Happy Student



Happy Student

The screenshot displays the DrScheme IDE interface. The main window, titled "sqr.lisp - DrScheme", shows the following code:

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

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

The code is highlighted in green. The bottom status bar indicates "Dracula v4.2".

The smaller window, titled "acl2.1221740037.txt - DrScheme", shows the proof output:

```
< Previous Checkpoint      Next Checkpoint >
{ DEFTHM SQR<=0 ... }
Q.E.D.

Q.E.D.

Summary
Form:  ( DEFTHM SQR<=0 ... )
Rules: ( (:DEFINITION NOT)
          (:DEFINITION SQR) )
```

Sad Student

The image shows a screenshot of the DrScheme IDE. The main window, titled 'sqr.lisp - DrScheme', contains the following code:

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

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

The IDE interface includes a menu bar with 'Debug', 'Check Syntax', 'Run', and 'Stop' options, and a toolbar with buttons for 'Admit Next', 'Admit All', 'Undo Last', 'Undo All', 'Save / Certify', 'Interrupt Proof', and 'Shutdown ACL2'. A 'Start ACL2' button is also present.

An ACL2 proof window, titled 'acl2.1221740037.txt - DrScheme', is open in the foreground. It displays the following text:

```
( DEFTHM SQR>=0 ... )
***** FAILED *****

(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

The image shows a screenshot of the DrScheme IDE. The main window, titled "sqr.lisp - DrScheme", contains the following code:

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

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

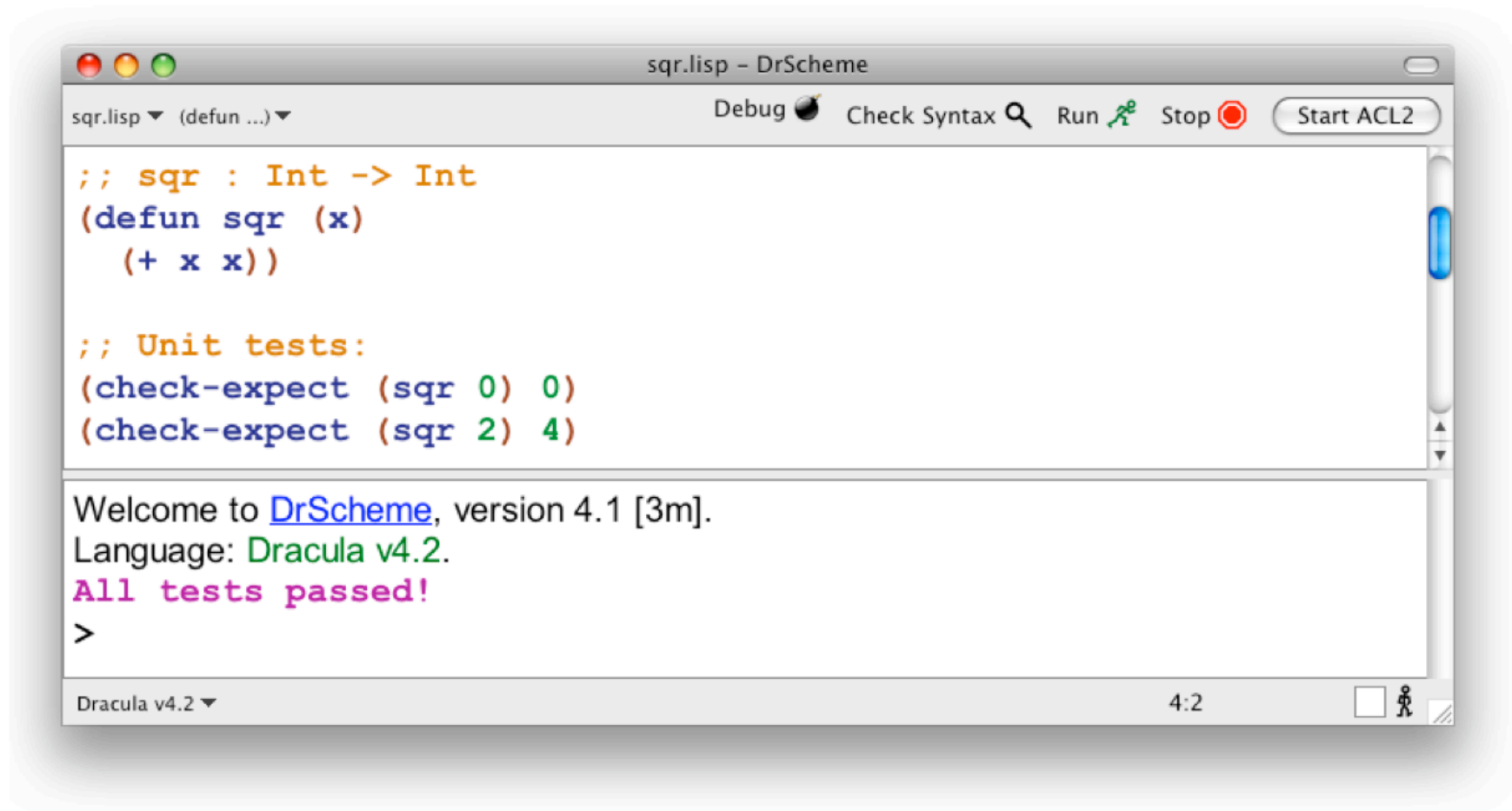
The second test case, `(check-expect (sqr 2) 4)`, is highlighted in pink. Below the code editor, a "Test Results" window is open, displaying the following output:

```
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
```

The "Test Results" window also features "Close" and "Dock" buttons at the bottom right.

Sad Student



The image shows a screenshot of the DrScheme IDE window titled "sqr.lisp - DrScheme". The window has a menu bar with "sqr.lisp" and "(defun ...)" selected. The toolbar includes "Debug", "Check Syntax", "Run", "Stop", and "Start ACL2". The main editor area contains the following code:

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

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

Below the code, the output area displays the following text:

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

The status bar at the bottom shows "Dracula v4.2" on the left, "4:2" in the center, and a small icon on the right.

Mad Student

The image shows a screenshot of the DrScheme IDE. The main window, titled 'sqr.lisp - DrScheme', contains the following code:

```
;; 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)))
```

The code is color-coded: function definitions are in blue, unit tests in green, and the theorem definition in pink. The bottom left corner shows 'Dracula v4.2'.

An inset window, titled 'acl2.1221741071.txt - DrScheme', shows the output of the ACL2 proof attempt:

```
{ DEFTHM SQR>=0 ... }
***** FAILED *****

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

Name the formula above *1.

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

The inset window also features navigation buttons for checkpoints: '< Previous Checkpoint' and 'Next Checkpoint >'.

Mad Student

The image shows a screenshot of the DrScheme IDE. The main window, titled "sqr.lisp - DrScheme", contains the following Scheme code:

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

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

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

The SchemeUnit window, titled "SchemeUnit", displays the execution results for the property `sqr>=0`. It shows a list of 13 test cases, with the 11th case highlighted in red, indicating a failure. The failure message is:

```
Additional information:
key x:
-30
key check-expect:
(check-expect
 (let ((x '-30))
  (>= (sqr x) 0))
 t)
```

Timing information is also shown: cpu: 0; real: 0; gc: 0.

Mad Student

The image shows a screenshot of the DrScheme IDE. The main window, titled "sqr.lisp - DrScheme", contains the following code:

```
;; 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)
```

The third test case is highlighted in red. Below the code editor, a "Test Results" window is open, displaying the following output:

```
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
```

At the bottom of the Test Results window, there are "Close" and "Dock" buttons.

In the bottom-left corner of the IDE, a small window shows the Dracula language version (v4.2) and a REPL prompt with the following interaction:

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

Mad Student

The image shows a screenshot of the DrScheme IDE. The main editor window, titled 'sqr.lisp - DrScheme', contains the following code:

```
;; 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)
```

The third test case, `(check-expect (sqr -30) 900)`, is highlighted in pink. Below the editor, a 'Test Results' window is open, displaying the following output:

```
Welcome to DrScheme
Language: DrScheme
>
Dracula v4.2

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
```

The 'Test Results' window has 'Close' and 'Dock' buttons at the bottom right.

Another Happy Student

The screenshot shows the DrScheme IDE with a file named `sqr.lisp` open. The code defines a function `sqr` and several unit tests. The tests include `check-expect` for specific values and a `defproperty` for a non-negativity property. The status bar at the bottom indicates that all tests passed. Two floating windows are also visible: `SchemeUnit` showing a `DoubleCheck` summary with 50 successes and 1/1 success for `sqr>=0`, and `acl2.1221749229.txt` showing the theorem statement and its proof summary.

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

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

Dracula v4.2

DoubleCheck
Total: 50 successes
Successes (1/1)
[sqr>=0](#)

acl2.1221749229.txt - DrScheme

(DEFTHM SQR>=0 ...)
Q.E.D.

Q.E.D.

Summary
Form: (DEFTHM SQR>=0 ...)
Rules: ((:DEFINITION NOT)
(:DEFINITION SQR)

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)
```


DoubleCheck

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

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

```
(generate-properties)
```

DoubleCheck

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

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

```
(generate-properties)
```

DoubleCheck

```
(defthm      sqr>=0  
  
  (implies  (integerp x)  
    (>= (sqr x) 0)))
```

Random Generators

`(random-boolean)`

`(random-char)`

`(random-string)`

`(random-symbol)`

`(random-atom)`

`(random-sexp)`

`(random-element-of lst)`

`(random-natural)`

`(random-integer)`

`(random-rational)`

`(random-number)`

`(random-data-size)`

`(random-between lo hi)`

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

- **Claessen and Hughes. QuickCheck: a lightweight tool for random testing of Haskell programs. ICFP 2000.**
- **Runciman et al. SmallCheck and Lazy SmallCheck: automatic exhaustive testing for small values. Haskell 2008.**
- **Berghofer and Nipkow. Random testing in Isabelle/HOL. SEFM 2004.**
- **Spiridinov and Khurshid. Pythia: automatic generation of counterexamples for ACL2 using Alloy. ACL2 2007.**
- **Sumners. Checking ACL2 theorems via SAT checking. ACL2 2002.**

Thank You.