Random Testing in 321
Test Cases So Far

Each test relates a particular input to a particular output.

(test (bound-ids
    (with 'x (id 'y) (id 'x)))
' (x))
(test (binding-ids
    (with 'x (id 'y) (id 'x)))
' (x))
Property-based Testing

But we can only write so many tests by hand.

To find additional bugs, we can automate testing.

We start with what we hope is a fact about our program.

For example,

“If bound-ids says 'x' is bound,
  then binding-ids says 'x' is binding.”
Property Violation

If we can find some WAE for which the property doesn’t hold ... 

\[
\text{(define a-WAE ...)} \\
\text{(bound-ids a-WAE) } \Rightarrow '(x) \\
\text{(binding-ids a-WAE) } \Rightarrow '() \\
\]

... we’ve found a bug.
Property Testing

We can test this property in the usual style.

; bound=>binding? : WAE -> boolean
; checks if bound ids are also binding
(define (bound=>binding? e) ...)

(test (bound=>binding? (id 'x))
  true)

(test (bound=>binding?
    (with 'x (num 0) (id 'x)))
  true)

Expected result is always true, so if we had lots of WAEs, then we’d have lots of tests.
Automated Property Testing

Write a program to generate test inputs!
Random WAEs

; random-WAE: -> WAE
(define (random-WAE)
  (case (random 5)
    [(0) (num (random-nat))]
    [(1) (id (random-symbol))]
    [(2) (add (random-WAE) (random-WAE))]
    [(3) (sub (random-WAE) (random-WAE))]
    [(4) (with (random-symbol)
               (random-WAE)
               (random-WAE))])))

Watch out – that code is buggy.... (read on for why)
Random WAEs

; random-nat: -> nat
(define (random-nat)
  (case (random 2)
    [(0) 0]
    [(1) (add1 (random-nat))]))

; random-symbol: -> symbol
(define (random-symbol)
  (random-elem '(x y z a b c)))

; random-elem: (listof X) -> X
(define (random-elem xs)
  (list-ref xs (random (length xs))))
Generation Strategy

To build a WAE,

- 1/5 of the time, build a number
- 1/5 of the time, build a symbol
- 3/5 of the time, first build two more WAEs
Expected Progress

On average, we “reduce” the problem from

Generate 1 WAE.

to

Generate 1.2 WAEs.

since $1.2 = (2/5)*0 + (3/5)*2$
Height Bound

Limit WAE size by bounding tree height.

; random-WAE/b: nat -> WAE
(define (random-WAE/b h)
  (case (random (if (zero? h) 2 5))
    [(0) (num (random-nat))]
    [(1) (id (random-symbol))]
    [(2) (add (random-WAE/b (sub1 h))
                (random-WAE/b (sub1 h)))]
    [(3) (sub (random-WAE/b (sub1 h))
               (random-WAE/b (sub1 h)))]
    [(4) (with (random-symbol)
               (random-WAE/b (sub1 h))
               (random-WAE/b (sub1 h)))]))

(Alternatively, tweak weights.)
; bound=>binding: WAE -> boolean
(define (bound=>binding e)
   (sublist? (bound-ids e) (binding-ids e)))

; sublist?: (listof X) (listof X) -> boolean
; Expects xs and ys to be sorted and have no dups.
(define (sublist? xs ys)
  (cond [(null? xs) #t]
    [(null? ys) #f]
    [(equal? (car xs) (car ys))
     (sublist? (cdr xs) (cdr ys))]
    [else (sublist? xs (cdr ys))])))
Running Tests

; test-bound=>binding: nat nat -> (or 'passed WAE)
(define (test-bound=>binding size attempts)
  (if (zero? attempts)
    'passed
    (let ([test-input (random-WAE/b size)])
      (if (bound=>binding test-input)
        (test-bound=>binding
         size
         (sub1 attempts))
        test-input))))
(test-bound=>binding 5 1000)
HW2 Test Results

We ran random tests on last year’s HW2 submissions.

○ Received 99 submissions
○ Tested 6 properties
○ Found a bug in 53 out of those 99 submissions
Interpreter Properties

• Interpreter does not crash
• Produces same result as another implementation (e.g., DrRacket)
• Type checker accurately predicts result (later)
• Program equivalences hold
With Elimination Example

For example, we should be able to replace a `with` with a new function.

```
{with {x (+ 7 2)}
  (+ x x)}

{deffun {f x}
  (+ x x)}
{f (+ 7 2)}
```
With Elimination Rule, an Attempt

In general,

{...
    {with {an-id a-wae}
        another-wae}
    ...
}
With Elimination Rule, an Attempt

In general,

\[
\begin{align*}
\{ & \ldots \\
& \{ \text{with } \{ \text{an-id a-wae} \} \\
& \quad \text{another-wae} \} \\
\ldots \} \quad \rightarrow \quad \\
& \{ \text{deffun } \{ \text{new-id an-id} \} \\
& \quad \text{another-wae} \} \\
& \{ \ldots \\
& \quad \{ \text{new-id a-wae} \} \\
& \quad \ldots \} 
\end{align*}
\]
With Elimination Rule, an Attempt

In general,

Different free variables!

{...  
 {with {an-id a-wae} another-wae}  
...}  

{deffun {new-id an-id} another-wae}  
{...  
 {new-id a-wae}  
...}
Rule Example

\{\text{with} \ \{x \ {+ \ 2 \ 7}\}\}
\{\text{with} \ \{y \ {+ \ x \ x}\}\}
\{+ \ x \ y\}\}\}

\rightarrow

\{\text{deffun} \ \{f \ y\}\}
\{+ \ x \ y\}\}\}
\{\text{with} \ \{x \ {+ \ 2 \ 7}\}\}
\{f \ {+ \ x \ x}\}\}\}
Rule Example

\[
\{\text{with } \{x \{+ 2 7\}\} \\
\{\text{with } \{y \{+ x x\}\} \\
\{+ x y\}\}\}\}
\]

\[
\{\text{def un } \{f y\} \\
\{+ x y\}\}
\{\text{with } \{x \{+ 2 7\}\} \\
\{f \{+ x x\}\}\}\}
\]
With Elimination, Fixed

Pass free variables of `another-wae` as arguments.

```scheme
{deffun {new-id an-id
         id₁ ...}
     another-wae}
{...      
  {new-id a-wae
   id₁ ...}
...}
```

where

```scheme
(equal?
  (free-ids another-wae)
  (list id₁ ...))
```
Rule Example

x becomes a parameter of f

\[
\begin{align*}
\{\text{with } \{x \{+ 2 7\}\} \\
\{\text{with } \{y \{+ x x\}\} \\
\{+ x y\}\}\} & \quad \rightarrow \quad \{\text{deffun } \{f y x\} \\
\{+ x y\}\} \\
\{\text{with } \{x \{+ 2 7\}\} \\
\{f \{+ x x\} x\}\} &
\end{align*}
\]