6.945 Spring 2009

Generic Operator Discovery

Laura Harris <lch> and Bryan Newbold <bnewbold>

Once generics are Registered...

```
; As a referesher, in case you forgot...
; these procedures and generic-operator-table are defined in ghelper.scm

(define add1 (make-generic-operator 1 #f 'add1))      ; note the symbols
(define sub1 (make-generic-operator 1 #f 'sub1))
(define double (make-generic-operator 1 #f 'double))
(define square (make-generic-operator 1 #f 'square))
(defhandler add1 (lambda (x) (+ x 1)) number?)
(defhandler sub1 (lambda (x) (- x 1)) number?)
(defhandler double (lambda (x) (* 2 x)) number?)
(defhandler square (lambda (x) (* x x)) number?)
```

... they can be Discovered...

```
; discover:named-opers-for searches *generic-operator-table* for
; the set of named (meaning defined with name symbols) operators
; which can be applied to the arguments

(discover:named-opers-for 4.5)

;Value 84: (double square add1 sub1 thingaling)
```

... and leveraged

```
; discover:satisfy-sequence takes a [predicate?] and [. args];
; it breadth first searches with all applicable operators

(discover:satisfy-sequence (lambda (x) (eq? x 9)) (/ 1 2))

;Value 8:  (((9) square double add1)
            ((9) square add1 inverse))


; The first time I ran this it crashed with a divide by zero...
; Automated bug catching!
```

Assumptions

- Dispatch predicates are *fast*, operator procedures may be *slow*
- Operator “return value” type is indeterminate
- But operators need not be functional

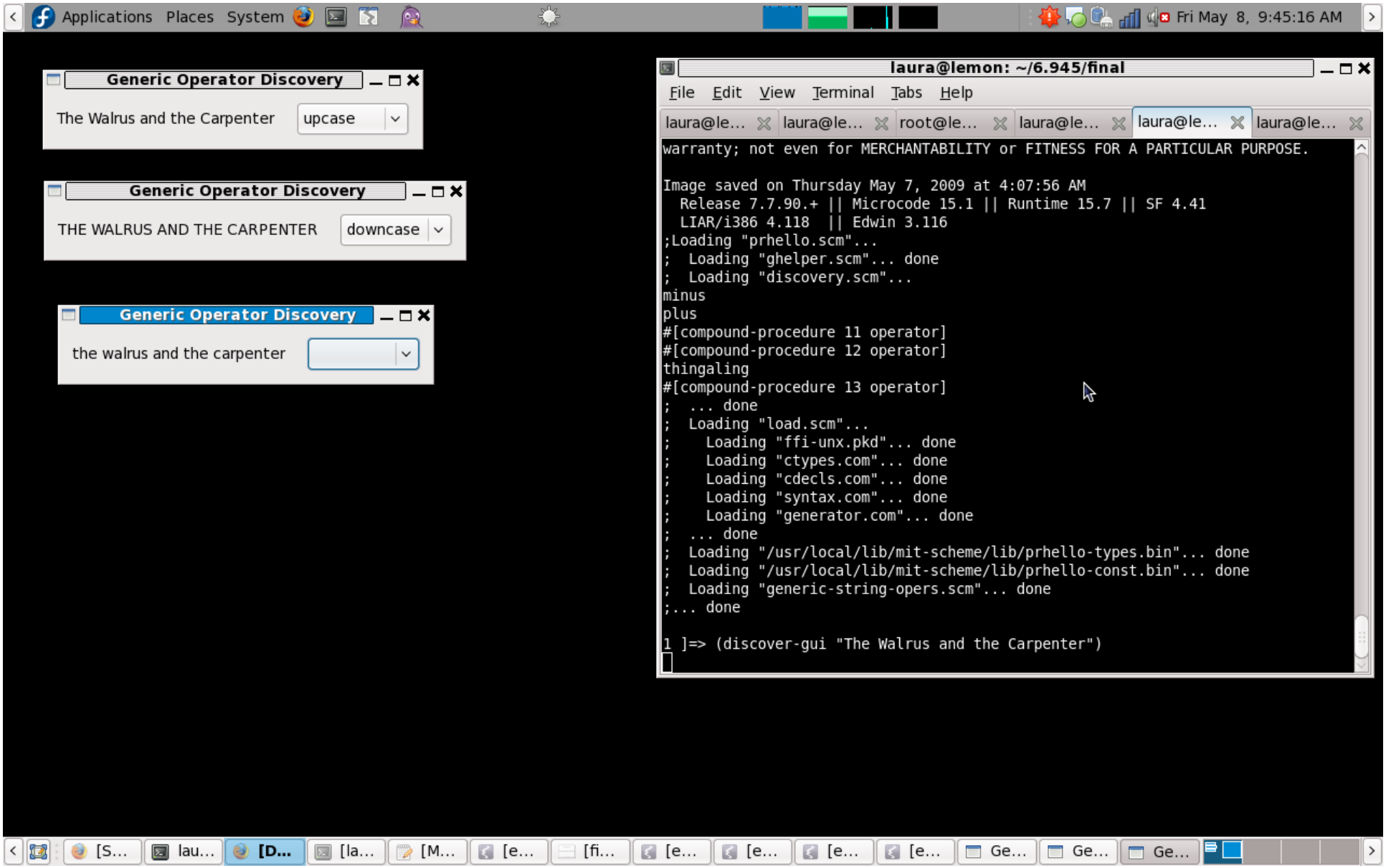
Remarks

- This isn't a good way! Brute force, but...
- Unknown systems with existing operators
- Flexibility, reusability
- Lazily Distribute

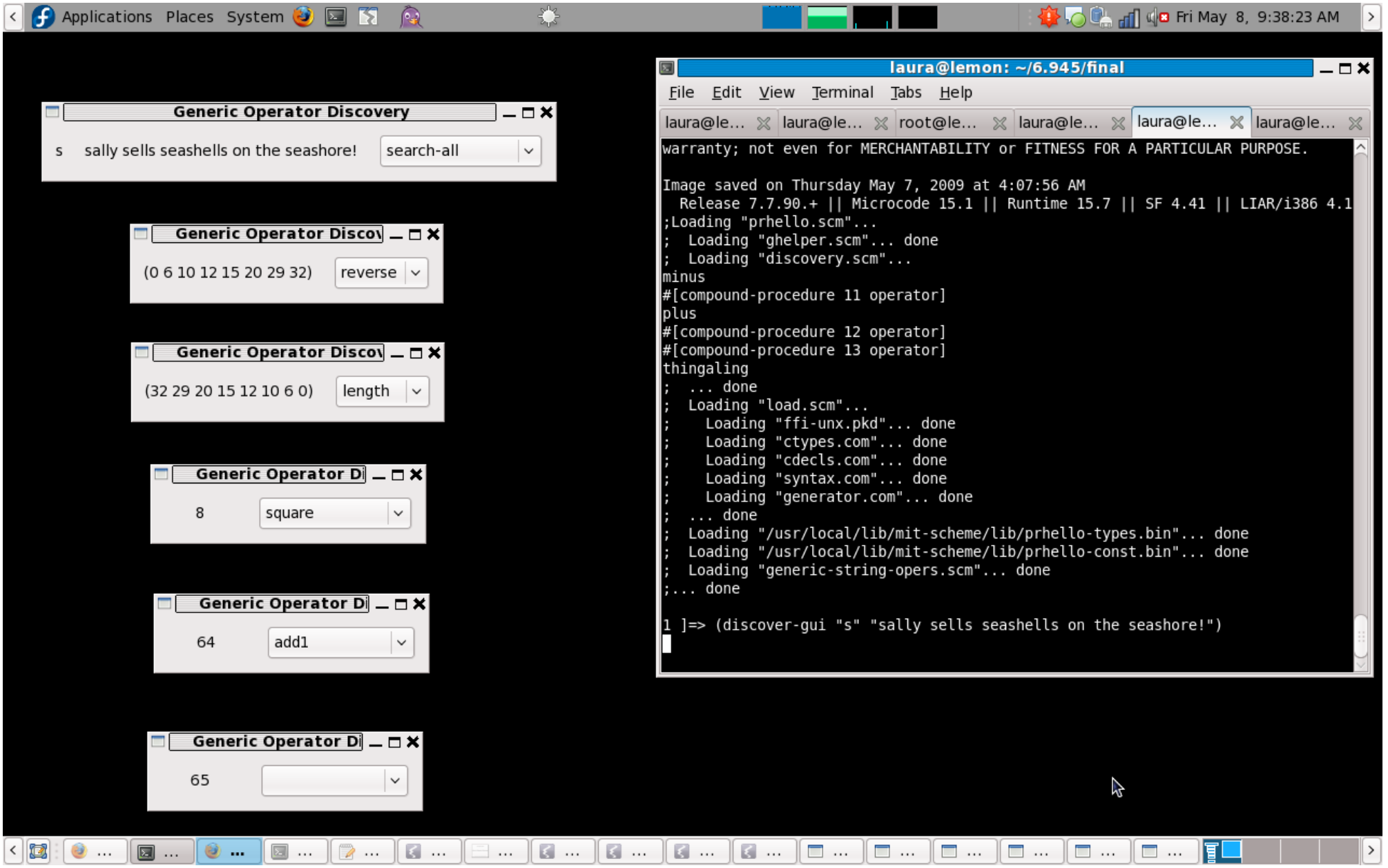
scmutils examples

```
(discover:named-opers-for
     (matrix-by-rows '(1 0 0)
                     '(0 1 0)
                     '(0 0 1)))
;Value: (one-like cos exp conjugate zero? zero-like identity? sin
inexact? type arity invert negate identity-like trace determinant type-
predicate)


(discover:named-opers-for 1 2)
;Value: (+ dot-product expt * gcd atan2 = / apply make-polar - make-
rectangular)


(discover:named-opers-for (compose sin cos))
;Value: (one-like cos acos exp cosh imag-part zero-like abs sinh sin
asin angle magnitude inexact? type arity real-part invert negate
identity-like sqrt log square type-predicate atan1)
```

GUI Screenshot



GUI Screenshot



discovery.scm

```
(define (discover:apply-all-name . args)
  (let ((names (apply discover:named-opers-for args)))
    (map (lambda (x)
           (list (apply discover:apply-name (cons x args)) x))
         names)))


(define (discover:satisfy pred? . args)
  (let try ((objs (list args)))
    (let ((goodies (filter (lambda (x) (apply pred? x)) objs)))
      (if (not (null? goodies))
          (car goodies)
          (try (fold-right append
                           '()
                           (map (lambda (x)
                                  (map list
                                       (apply discover:apply-all x)))
                                objs)))))))
```

discovery.scm

```
; finds all the operators which can be applied to the args; returns a list
; of operators (not the actual procedures; will include duplicate symbols and
; operator stubs for named operators)
(define (discover:opers-for . args)
  (let* ((arity (length args))
         (opers (hash-table->alist *generic-operator-table*))
         (check
          (lambda (op)
            (if (not (eq? arity (cadr op)))
                #f
                (let per-arg ((tree (operator-record-tree (cdr op)))
                              (args args)
                              (fail (lambda () #f)))
                  (let per-pred ((tree tree) (fail fail))
                    (cond ((pair? tree)
                           (if ((caar tree) (car args))
                               (if (pair? (cdr args))
                                   (per-arg (cdar tree)
                                            (cdr args)
                                            (lambda ()
                                              (per-pred (cdr tree) fail)))
                                   #t)
                               (per-pred (cdr tree) fail)))
                          ((null? tree) (fail))
                          (else #t)))))))))
    (map car (filter check opers))))
```

Possible Applications

- File/media conversion and visualization
- Fix data corruption
  - CRC or hash as predicate
  - bitshifts and simple operations as operators
- Exploration of new systems, libraries
- MVC webdesign
    - Views and manipulators generic over models
- Another tool in the toolbox!