Using MOP to implement reflectable function objects

Lightning Talk

Jim Newton

10th European Lisp Symposium

3-4 April 2017





Too many functions, I get confused, need help debugging.

```
(defmethod map-pxls ((p-pxl-iters parallel-pxl-iters)
                     &key unary aggregate
                       (mk-unary (lambda () unary)))
  (let ((0-ary-functions
         (mapcar (lambda (p-pxl-iter
                          &aux (unary (funcall mk-unary)))
                   (lambda ()
                     (map-pxls p-pxl-iter :unary unary)))
                 (pxl-iters p-pxl-iters))))
    (lparallel:preduce (lambda (v f)
                         (funcall aggregate v (funcall f)))
                       0-ary-functions)))
```

Declaring function types doesn't help. Compiler ignores content.

```
(defmethod map-pxls ((p-pxl-iters parallel-pxl-iters)
                     &key unary aggregate
                       (mk-unary (lambda () unary)))

  (declare (type (function (t) t) unary)
           (type (function () (function (t) t)) mk-unary)
           (type (function (t t) t) aggregate))

  (let ((0-ary-functions
          (mapcar (lambda (p-pxl-iter
                           &aux (unary (funcall mk-unary)))
                    (lambda ()
                      (map-pxls p-pxl-iter :unary unary)))
                  (pxl-iters p-pxl-iters))))
  ...
```

Forbidden to make run-time type assertions.

```
(defmethod map-pxls ((p-pxl-iters parallel-pxl-iters)
                     &key unary aggregate
                       (mk-unary (lambda () unary)))
  (let ((0-ary-functions
         (mapcar (lambda (p-pxl-iter
                          &aux (unary (funcall mk-unary)))
                   (assert (typep (function () t) mk-unary))
                   (assert (typep (function (t) t) unary))
                   (lambda ()
                     (map-pxls p-pxl-iter :unary unary)))
                 (pxl-iters p-pxl-iters))))
    (lparallel:preduce (lambda (v f)
                         (funcall aggregate v (funcall f)))
                       0-ary-functions)))
```

CL specification of *Function* TYPEP

An error of type error is signaled if type-specifier is `values`*, or a type specifier list whose first element is either* `function` *or* `values`*.*

```
(assert (typep (function (t) t) unary))
```

MOP based, limited solution

```
(defclass reflectable-function ()
  ((function :initarg :function :type function)
   (body :initarg :body)
   (lambda-list :initarg :lambda-list))
  (:metaclass closer-mop::funcallable-standard-class))

(defmethod initialize-instance
      :after ((self reflectable-function) &key)
  (closer-mop:set-funcallable-instance-function
     self (slot-value self 'function)))

(defmacro refl-lambda (lambda-list &rest body)
  `(make-instance 'reflectable-function
                  :function (lambda ,lambda-list ,@body)
                  :body ',body
                  :lambda-list ',lambda-list))
```

MOP based, limited solution

```
(defun refl-2-ary (obj)
  (and (typep obj 'reflectable-function)
       (= 2 (length (slot-value obj 'lambda-list)))))

(deftype refl-2-ary ()
  `(and reflectable-function
        (satisfies refl-2-ary)))

;; find max red in image
(map-pxls p-iters
          :unary (refl-lambda (pxl)
                   (red pxl))
          :aggregate (refl-lambda (v2 v2)
                       (max v1 v2)))
```

```
(defmethod map-pxls ((p-pxl-iters parallel-pxl-iters)
                     &key unary aggregate
                       (mk-unary (refl-lambda () unary)))
 (declare (type refl-1-ary unary)
          (type refl-0-ary mk-unary)
          (type refl-2-ary aggregate))
   (let ((0-ary-functions
           (mapcar (lambda (p-pxl-iter
                            &aux (unary (funcall mk-unary)))
                     (declare (type refl-1-ary unary))
                     (refl-lambda ()
                       (map-pxls p-pxl-iter :unary unary)))
                   (pxl-iters p-pxl-iters))))
     (lparallel:preduce (lambda (v f)
                          (declare (type refl-0-ary f))
                          (funcall aggregate v (funcall f)))
                        0-ary-functions)))
```

Future Research in Function type inspection

- Argument list is a sequence of objects
- We'd like to make assertions w.r.t. types of declared arguments
- Which brings us back to the topic of RTE
- Also links back to smarter user interaction in CLIMB GUI