Unified `for` Loops

Version 7.2

Michael M. MacLeod <mmmacleo at ucsd dot edu>

May 5, 2019

```
(require unified-for)
```
package: `unified-for`

This package consolidates the various flavors of `for` iteration—`for`, `for/list`, `for/vector`, `for/fold`, and so on—into an extensible *unified for* macro that compiles directly to efficient named let code,

The unified for macro extends the functionality of the traditional loop constructs through user-definable §1 “Iterators” and §2 “Accumulators”. It also allows identifiers to be bound with match patterns.

```
(for maybe-accumulator (iterator-clause ...) body ...+)

   maybe-accumulator =
                     | accumulator-id
                     | (accumulator-id arg-form ...)

     iterator-clause = [maybe-match-patterns (iterator-id arg-form ...)]

maybe-match-patterns = var-id ...
                     | match-pattern-expr ...
```

Iteratively evaluates *body*s.

Examples:

```
> (for ([x (from-range 9)])
    (display x))
012345678
> (for (to-vector #:length 4)
       ([(app real-part r) (from-vector #(1+2i 3+4i 5+6i 7+8i))])
    (* r 2))
'#(2 6 10 14)
```

```
> (let ([table #hash((a . 0) (b . 1) (c . 2) (d . 3) (e . 4))])
    (for (to-fold [even-valued-keys empty])
         ([key value (from-hash table)])
      (if (even? value)
          (cons key even-valued-keys)
          even-valued-keys)))
'(c e a)
```

1 Iterators

2 Accumulators

```
(to-list lst)

  lst : list?
```

```
(to-vector length-option)

    length-option =
                  | expandable-option
                  | fixed-option

expandable-option = #:grow-from initial-capacity-expr
                  | #:grow-from initial-capacity-expr growth-option

     fixed-option = #:length length-expr
                  | #:length length-expr #:fill fill-expr

    growth-option = #:by multiplier-expr
                  | #:with growth-proc

  initial-capacity-expr : exact-positive-integer?
  length-expr : exact-nonnegative-integer?
  fill-expr : any/c
  multiplier-expr : (and/c exact-integer? (>=/c 2))
                (->i ([old-size exact-positive-integer?])
  growth-proc :      [new-size (old-size)
                      (and/c exact-integer? (>/c old-size))])
```