Generalized comprehensions for Lisp

Sven-Olof Nyström

Uppsala Universitet

`svenolof at csd.uu.se`

How to express iteration in Lisp?

- Do-macro
  Mard to write, hard to read
- Loop-macro
  Powerful, but is it Lisp?
- Tail-recursive functions
  Might run out of memory
- Prog + go
  Does not belong in this century!
- map, reduce, dolist, dotimes, . . .
  Not a general solution

An alternative . . .

- List comprehensions
- In Erlang, Haskell, Python, . . .
- Inspired by mathematical notation

$$\{x * x \mid x \in S, x \text{ odd}, x < 5\}$$

- Powerful, convenient, popular
- But only allows operations on lists

Goals

- Implement list comprehensions in Lisp
- Extend it to handle vectors, arrays, hashtables, . . .
- Extend it to match the loop macro
- Make it extensible

The Lisp implementation

Example:

```
(collect (list) ((* x x))
  (in x '(1 2 3 4 5 6 7 8)))
```

Three components:

1. A collection type (decribes the object beeing built)
2. A list of expressions (giving values to be inserted)
3. One or more clauses (describe iteration)

Clauses . . .

- Iterating over a list l
  `(in x l)`
  (x is bound to each element of the list)
- Iterating over a vector v
  `(in x v)`
  (x is bound to each element of the vector)
- Iterating over a hash table h
  `(in (k v) h)`
  (Variables k and v are bound to each key-value pair of h)

More clauses . . .

- Filter

  `(when b)`

  Only consider cases when `b` holds

- Termination

  `(while b)`

  Stop the entire iteration if b does not hold

- Side effect

  `(do s)`

  Evaluate s for side-effects

More clauses . . .

- Computing values
  ```
  (step v init-exp test-exp next-exp)
  ```
  A for-loop
- Running clauses in parallel

```
  (for (step i 0 (< i 10) (+ i 1))
       (in x l))
```

Bind x to the first ten elements of the list l

Collection types

Simple collection types

- `list`
  Build a list
- `vector`
  Build a vector
- `t`
  Last value inserted
- `nil`
  Don’t collect
- `sum`
  Sum. . .
- `(reduce f)`
  Combine inserted values using function `f`

Complex collection types (Examples)

- `hash-table`
  Insert the values as keys in the table
- `(hash-table t)`
  Collect key-value pairs. If many pairs have the same key, keep the last
- `(hash-table list)`
  Build a hash-table which maps each key to a list of values
- (and so on . . .)

Complex collection types (Examples)

- `(array t (10))`
  Build a one-dimensional array of ten elements. Needs two values, an index and something to be inserted.
- `(array sum (10 10))`
  A two-dimensional array where values are combined by addition. Nice for matrix multiplication. Needs three values, two indices and a value to be added.