PRISMA 2009 code-box and iteration

Kilian Sprotte

July 4, 2009

Looping over a single list to get started.

```
(iter
 (for x in list)
 (collect x))
```

Looping over a single list to get started.

```
list ← '(1 2 3 4 5 6)

(iter
 (for x in list)
 (collect x))
```

Looping over a single list to get started.

```
list ← '(1 2 3 4 5 6)

(iter
 (for x in list)
 (collect x))
```

$\Rightarrow$ (1 2 3 4 5 6)

Looping over a single list to get started.

But ITERATE is more powerful than LOOP!

```
list ← '(1 2 3 4 5 6)
```

```
(iter
 (for x in list)
 (when (evenp x)
  (collect x)))
```

$\Rightarrow$ (2 4 6)

Looping over a single list to get started.

But ITERATE is more powerful than LOOP! Really!

```
list ← '(1 2 3 4 5 6)
```

```
(iter
 (for x in list)
 (dotimes (i x)
   (when (evenp x)
    (collect x))))
```

⇒ (2 2 4 4 4 4 6 6 6 6 6 6)

Looping over two lists in parallel.

ITERATE clauses can be combined.

```
(iter
 (for x in list-a)
 (for y in list-b)
 (collect (list x y)))
```

Looping over two lists in parallel.

ITERATE clauses can be combined.

$$\texttt{list-a} \leftarrow '(1\ 2\ 3\ 4\ 5)$$
$$\texttt{list-b} \leftarrow '(a\ b\ c)$$

```
(iter
 (for x in list-a)
 (for y in list-b)
 (collect (list x y)))
```

Looping over two lists in parallel.

ITERATE clauses can be combined.
When the first clause is exausted, the process terminates.

```
list-a ← '(1 2 3 4 5)
list-b ← '(a b c)
```

```
(iter
 (for x in list-a)
 (for y in list-b)
 (collect (list x y)))
```

$\Rightarrow$ ((1 a) (2 b) (3 c))

Looping over two lists in parallel.

ITERATE clauses can be combined.
When the first clause is exausted, the process terminates.
Let's add another one.

```
list-a ← '(1 2 3 4 5)
list-b ← '(a b c)
```

```
(iter
 (for x in list-a)
 (for y in list-b)
 (repeat 2)
 (collect (list x y)))
```

$\Rightarrow$ ((1 a) (2 b))

Looping over two lists in parallel.

Extending the lists in a circular way.

```
list-a ← '(1 2 3 4 5)
list-b ← '(a b c)
```

```
(iter
 (for x in-circular-list list-a)
 (for y in-circular-list list-b)
 (repeat 11)
 (collect (list x y)))
```

⇒ ((1 a) (2 b) (3 c) (4 a)
(5 b) (1 c) (2 a) (3 b)
(4 c) (5 a) (1 b))

Looping over two lists in parallel. CODE-BOX!

ITERATE clauses can be combined.
When the first clause is exausted, the process terminates.
Let's add another one.

```
list-a ← '(1 2 3 4 5)
list-b ← '(a b c)
```

```
(iter
 (for x in list-a)
 (for y in list-b)
 (repeat 2)
 (collect (list x y)))
```

$\Rightarrow$ ((1 a) (2 b))

xs → dxs

The famous function for working on pitches and intervals, for instance.

```
(iter
 (for x in list)
 (for prev-x previous x)
 (when prev-x
  (collect (- x prev-x))))
```

xs → dxs

The famous function for working on pitches and intervals, for instance.

```
list ← '(1 3 0 5 4 7)
```

```
(iter
 (for x in list)
 (for prev-x previous x)
 (when prev-x
  (collect (- x prev-x))))
```

xs → dxs

The famous function for working on pitches and intervals, for instance.

```
list ← '(1 3 0 5 4 7)
```

```
(iter
 (for x in list)
 (for prev-x previous x)
 (when prev-x
  (collect (- x prev-x))))
```

$\Rightarrow$ (2 −3 5 −1 3)

x ⟶ dx (variation using first-iteration-p)

```
list ← '(1 3 0 5 4 7)
```

```
(iter
 (for x in list)
 (for prev-x previous x)
 (unless (first-iteration-p)
  (collect (- x prev-x))))
```

$\Rightarrow$ (2 −3 5 −1 3)

x ⟶ dx (variation using for-wind)

```
list ← '(1 3 0 5 4 7)

(iter
 (for-wind (prev-x x) in list)
 (collect (- x prev-x)))
```

⇒ (2 −3 5 −1 3)

dxs → xs

```
(iter
 (for dx in list)
 (when (first-iteration-p)
  (collect start))
 (collect (incf start dx)))
```

dxs ⟶ xs

```
start ← 0
list ← '(1 2 3 2 4 1)

(iter
 (for dx in list)
 (when (first-iteration-p)
  (collect start))
 (collect (incf start dx)))
```

dxs ⟶ xs

```
start ← 0
list ← '(1 2 3 2 4 1)

(iter
 (for dx in list)
 (when (first-iteration-p)
  (collect start))
 (collect (incf start dx)))
```

⇒ (0 1 3 6 8 12 13)

for-wind

FOR-WIND has more options.

```
list ← '(1 2 3 4 5 6 7 8 9)

(iter
 (for-wind (a b c) in list)
 (collect (list a b c)))

    ⇒ ((1 2 3) (2 3 4) (3 4 5)
 (4 5 6) (5 6 7) (6 7 8)
 (7 8 9))
```

for-wind

FOR-WIND has more options.

```
list ← '(1 2 3 4 5 6 7 8 9)

(iter
 (for-wind (a b c) in list by 2)
 (collect (list a b c)))

    ⇒ ((1 2 3) (3 4 5) (5 6 7)
(7 8 9))
```

for-wind

FOR-WIND has more options.

```
list ← '(1 2 3 4 5 6 7 8 9)
```

```
(iter
 (for-wind (a b c) in list by 5)
 (collect (list a b c)))
```

$\Rightarrow$ ((1 2 3) (6 7 8))

for-wind

FOR-WIND has more options.

```
list ← '(1 2 3 4 5 6 7 8 9)

(iter
 (for-wind (a b) in list)
 (for-wind (x y z) in list)
 (collect (list (list a b) (list x y z))))

    ⇒ (((1 2) (1 2 3))
 ((2 3) (2 3 4))
 ((3 4) (3 4 5))
 ((4 5) (4 5 6))
 ((5 6) (5 6 7))
 ((6 7) (6 7 8))
 ((7 8) (7 8 9)))
```

destructuring list elements

Processing a list of duration/pitch events.

```
list ← '((2 60) (1 55) (1 61) (3 66))
transposition ← 11
```

```
(iter
 (for (dur midi) in list)
 (collect (list dur (+ transposition midi))))
```

⇒ ((2 71) (1 66) (1 72) (3 77))

using random

```
(iter
 (repeat n)
 (collect x))
```

using random

$$\texttt{x} \leftarrow (\mathit{random}\ 100)$$
$$\texttt{n} \leftarrow 5$$

```
(iter
 (repeat n)
 (collect x))
```

using random

```
x ← (random 100)
n ← 5

(iter
 (repeat n)
 (collect x))

⇒ (98 98 98 98 98)
```

using random

```
!x ← (random 100)
n ← 5

(iter
 (repeat n)
 (collect !x))
```

⇒ (85 26 66 59 97)