$\boxed{5}$ Iteration and Macros

$\boxed{5.1}$ Iteration, DO

initial value

variable

next value

```
(DO ((v1 i1 n1)
     (v2 i2 n2)
     . . .      )
   (end-test answer))
```

$$\Longrightarrow$$

$$\begin{pmatrix} \mathtt{v}_1 \\ \mathtt{v}_2 \\ \cdots \end{pmatrix} \longleftarrow \begin{pmatrix} \mathtt{i}_1 \\ \mathtt{i}_2 \\ \cdots \end{pmatrix}$$

while (NOT end-test):

$$\begin{pmatrix} \mathtt{v}_1 \\ \mathtt{v}_2 \\ \cdots \end{pmatrix} \longleftarrow \begin{pmatrix} \mathtt{n}_1 \\ \mathtt{n}_2 \\ \cdots \end{pmatrix}$$

return answer

```
(SUM m n) ==>
  (DO ((I m (+ I 1))
       (SUM 0 (+ SUM I)))
    ((> I n) SUM))
```

$\boxed{5.2}$ Iteration, DO Implementation

```
(DO ((v1 i1 n1)
     (v2 i2 n2)
     . . .     )
  (end-test answer))
```

$\Longrightarrow$

```
(LET ((LOOP-F #'(LAMBDA (LF v1 v2 ...)
                   (COND (end-test answer)
                         (T (FUNCALL LF LF n1  n2 ...))))))
  (FUNCALL LOOP-F LOOP-F i1 i2 ...))
```

Note absence of #' in FUNCALLs.

5.3 Backquote Notation

`(x . y) ≡ (CONS `x `y)

`(,x . y) ≡ (CONS x `y)

`(,x . ,y) ≡ (CONS x y)

`(x . ,y) ≡ (CONS `x y)

`a ≡ 'a if a ∈ atom

E.g.: `(+ ,(+ 5 2) ,(- 5 2))

≡ `(+ . (,(+ 5 2) . (,(- 5 2) . NIL)))

≡ (CONS `+ `(,(+ 5 2) . (,(- 5 2) . NIL)))

≡ (CONS `+ (CONS (+ 5 2) `(,(- 5 2) . NIL)))

≡ (CONS `+ (CONS (+ 5 2) (CONS (- 5 2) `NIL)))

≡ (CONS '+ (CONS (+ 5 2) (CONS (- 5 2) 'NIL)))

⟹ (CONS '+ (CONS 7 (CONS 3 'NIL))) ⟹ '(+ 7 3)

5.4 Macros, MY-DO

Function: RR applied <u>after</u> arguments are evaluated.
Macro: RR applied <u>before</u> arguments are evaluated.

```
(DEFMACRO MY-DO (((V1 I1 N1)         ; (MY-DO ((V1 I1 N1)
                  (V2 I2 N2))        ;         (V2 I2 N2))
                 (END-TEST ANSWER)) ;     (END-TEST ANSWER))
  `(LET ((LOOP-F #'(LAMBDA (LF ,V1 ,V2)
                     (COND (,END-TEST ,ANSWER)
                           (T (FUNCALL LF LF ,N1 ,N2))))))
     (FUNCALL LOOP-F LOOP-F ,I1 ,I2)))
```

Comments:

1. The function that is the value of LOOP-F must call itself, so it does this by having a first argument which is the function itself. When it calls itself, it must pass this first argument along.

2. A DEFUN has an argument list with is mapped to the actual arguments of a function call to assign names.

   A DEFMACRO has a similar argument list, but in this case the argument list can have sublists. Thus the DEFMACRO argument list:

```
(((V1 I1 N1)
  (V2 I2 N1))
 (END-TEST ANSWER))
```

   matches against the macro call:

```
(MY-DO ((I m (+ I 1))
        (SUM 0 (+ SUM I)))
 ((> I n) SUM))
```

   to bind the argument names to the following s-expressions:

V1	to	I	V2	to	SUM
I1	to	m	I2	to	0
N1	to	(+ I 1)	N2	to	(+ SUM I)
END-TEST	to	(> I n)	ANSWER	to	SUM

$\boxed{5.5}$ Tail Recursion

RR: `(RANDOM-SUM 0)` ⇒ `0`

`(RANDOM-SUM n)` ⇒ `(+ (RANDOM 2) (RANDOM-SUM (- n 1)))`

`(RANDOM-SUM 1000000)` has 1000000 simultaneously active calls.

RR: `(RANDOM-SUM 0 sum)` ⇒ `sum`

`(RANDOM-SUM n sum)` ⇒

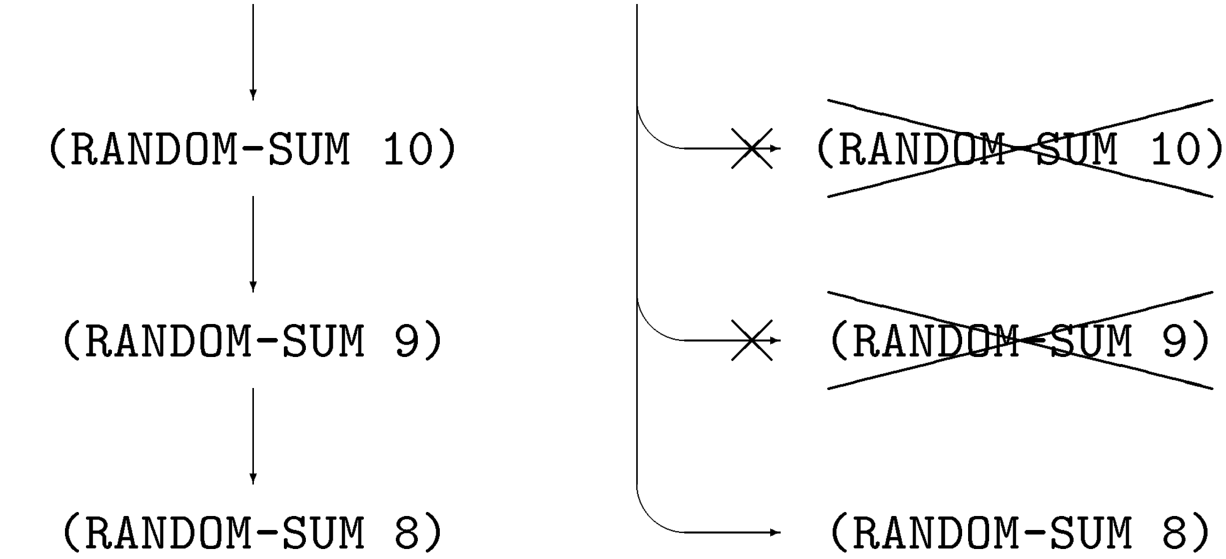
`(RANDOM-SUM (- n 1) (+ (RANDOM 2) sum)))`

`(RANDOM-SUM 1000000)` may have only 1 active call at any time.

Because the second `RANDOM-SUM` is ‘tail recursive’ and only calls itself as the very last thing it does, it can get rid of its old call as it makes the new call.

5.6 Tail Recursive Call Trees

Not Tail Recursive | Tail Recursive



Tail Recursion is an optimization performed by the compiler.

Comments:

3. For a DO the <u>iteration variables</u> are the variables defined by the DO.

   For a recursion the iteration variables can be introduced as &OPTIONAL variables, with the default values of these variables being the initial values of the iteration variables in the loop.

4. Using tail recursion is actually more powerful than using DO. The trouble with DO is that the expressions which produce the next values of the iteration variables are fixed for all iterations. With recursion, different expressions can be used for different situations within the preceding iteration.

   This is another way of saying that in a function with several tail recursive calls, the argument lists in the different tail recursive calls can be different. If they are different, then the tail recursive function cannot be translated in a simple way into a DO.