

Próximo:[Vars locais surpreendentes](#), Anterior:[Momento errado](#), Acima:[Problemas com macros](#) [ Conteúdo][\[Índice\]](#)

## 14.5.2 Avaliando Argumentos de Macro Repetidamente

Ao definir uma macro deve-se atentar para o número de vezes que os argumentos serão avaliados quando a expansão for executada. A macro a seguir (usada para facilitar a iteração) ilustra o problema. Essa macro nos permite escrever uma construção de loop for.

```
(defmacro for (var from init to final do &rest body)
  "Execute um loop \"for\" simples.
Por exemplo, (para i de 1 a 10 faça (imprima i))."
  (list 'let (list (list var init))
        (contras 'enquanto
                  (contras (list '<= var final)
                            (anexar corpo (list (list 'inc var)))))))

(para i de 1 a 3 faça
  (setq quadrado (* ii))
  (princ (formato "\n%d %d" i quadrado)))
→
(deixe ((i 1))
  (enquanto (<= i 3)
    (setq quadrado (* ii))
    (princ (formato "\n%d %d" i quadrado))
    (incluindo eu)))
  -|1 1
  -|2 4
  -|3 9
→ nada
```

Os argumentos `from`, `to`, e `do` nessa macro são açúcar sintático; eles são totalmente ignorados. A ideia é que você escreva palavras de ruído (como `from`, `to` e `do`) nessas posições na chamada da macro.

Aqui está uma definição equivalente simplificada através do uso de backquote:

```
(defmacro for (var from init to final do &rest body)
  "Execute um loop \"for\" simples.
Por exemplo, (para i de 1 a 10 faça (imprima i))."
  `(let (,var ,init)
      (enquanto (<= ,var ,final)
                ,@corpo
                (incluindo ,var))))
```

Ambas as formas dessa definição (com backquote e sem) sofrem do defeito de que `final` é avaliado em cada iteração. Se `final` for uma constante, isso não é um problema. Se for uma forma mais complexa, digamos (`long-complex-calculation x`), isso pode desacelerar significativamente a execução. Se `final` tiver efeitos colaterais, executá-lo mais de uma vez provavelmente está incorreto.

Uma definição de macro bem projetada toma medidas para evitar esse problema produzindo uma expansão que avalia as expressões de argumento exatamente uma vez, a menos que avaliações repetidas façam parte do propósito pretendido da macro. Aqui está uma expansão correta para a `formacro`:

```
(deixe ((i 1)
        (máximo 3))
      (enquanto (<= i max)
        (setq quadrado (* ii))
        (princ (formato "%d %d" i quadrado)))
        (incluso eu)))
```

Aqui está uma definição de macro que cria essa expansão:

```
(defmacro for (var from init to final do &rest body)
  "Execute um loop for simples: (para i de 1 a 10 faça (imprima i))."
  `(let ((,var ,init)
         (máximo, final))
     (enquanto (<= ,var max)
       ,@corpo
       (incluso ,var))))
```

Infelizmente, essa correção apresenta outro problema, descrito na seção a seguir.

Próximo:[Vars locais surpreendentes](#), Anterior:[Momento errado](#), Acima:[Problemas com macros](#) [ [Conteúdo](#) ] [ [Índice](#) ]