Laboratorio de Lenguajes de Programación

Sistema de Macros

Manuel Soto Romero

Universidad Nacional Autónoma de México
Facultad de Ciencias

16 de noviembre de 2016

Definición de macro

Definición de macro

¿Qué es UN macro?

Definición de macro

¿Qué es UN macro?

Definición. (Macro)

Un macro es una nueva forma sintáctica que pueden traducirse a formas ya existentes mediante transformadores.

Ejemplo

Ejemplo

Definiendo or

Supongamos que no existe una primitiva para la operación lógica ”or” y queremos implementarla en términos de la estructura de decisión `if`.

Ejemplo

Definiendo or

Supongamos que no existe una primitiva para la operación lógica ”or” y queremos implementarla en términos de la estructura de decisión `if`.

Solución:

Ejemplo

Definiendo or

Supongamos que no existe una primitiva para la operación lógica ”or” y queremos implementarla en términos de la estructura de decisión `if`.

Solución:

```
(define (or x y)
   (if x x y))
```

Ejemplo

Definiendo or

Supongamos que no existe una primitiva para la operación lógica ”or” y queremos implementarla en términos de la estructura de decisión `if`.

Solución:

```
(define (or x y)
   (if x x y))
```

¿Tiene algún problema la implementación anterior?

Ejemplo

Definiendo or

Supongamos que no existe una primitiva para la operación lógica ”or” y queremos implementarla en términos de la estructura de decisión `if`.

Solución:

```
(define (or x y)
   (if x x y))
```

¿Tiene algún problema la implementación anterior?

¿Qué pasa si ejecutamos la siguiente expresión?

```
(or 3 (/ 2 0))
```

Ejemplo

Definiendo or

Supongamos que no existe una primitiva para la operación lógica ”or” y queremos implementarla en términos de la estructura de decisión `if`.

Solución:

```
(define (or x y)
   (if x x y))
```

¿Tiene algún problema la implementación anterior?

¿Qué pasa si ejecutamos la siguiente expresión?

```
(or 3 (/ 2 0))
```

El parámetro formal `y` siempre es evaluado sin siquiera llegar a transformar la expresión en `if`. Este error se da en **tiempo de ejecución**. Queremos entonces, transformar en **tiempo de compilación**.

Macros en Racket

Macros en Racket

Para definir macros en Racket se usa la primitiva `define-syntax-rule`.

Macros en Racket

Para definir macros en Racket se usa la primitiva `define-syntax-rule`.

```
(define-syntax-rule (or x y)
   (if x x y))
```

Macros en Racket

Para definir macros en Racket se usa la primitiva `define-syntax-rule`.

```
(define-syntax-rule (or x y)
   (if x x y))
```

¡Ya funciona!

Macros en Racket

Para definir macros en Racket se usa la primitiva `define-syntax-rule`.

```
(define-syntax-rule (or x y)
   (if x x y))
```

¡Ya funciona!

Pero... ¿y la complejidad?

¿Qué pasa si ejecutamos la siguiente expresión?

```
(or (begin (display 3) 3) (/ 2 0))
```

Macros en Racket

Para definir macros en Racket se usa la primitiva `define-syntax-rule`.

```
(define-syntax-rule (or x y)
   (if x x y))
```

¡Ya funciona!

Pero... ¿y la complejidad?
¿Qué pasa si ejecutamos la siguiente expresión?

```
(or (begin (display 3) 3) (/ 2 0))
```

El parámetro formal `x` se evalúa dos veces.

Macros en Racket

Para definir macros en Racket se usa la primitiva `define-syntax-rule`.

```
(define-syntax-rule (or x y)
   (if x x y))
```

¡Ya funciona!

Pero... ¿y la complejidad?
¿Qué pasa si ejecutamos la siguiente expresión?

```
(or (begin (display 3) 3) (/ 2 0))
```

El parámetro formal `x` se evalúa dos veces.

Sólo hay que aplicar *la vieja confiable...*

Macros en Racket

Para definir macros en Racket se usa la primitiva `define-syntax-rule`.

```
(define-syntax-rule (or x y)
   (if x x y))
```

¡Ya funciona!

Pero... ¿y la complejidad?
¿Qué pasa si ejecutamos la siguiente expresión?

```
(or (begin (display 3) 3) (/ 2 0))
```

El parámetro formal `x` se evalúa dos veces.

Sólo hay que aplicar *la vieja confiable...*

```
(define-syntax-rule (or x y)
   (let ([z x])
      (if z z y)))
```

Otro ejemplo

Otro ejemplo

Macro swap

Definamos ahora un macro que intercambie dos valores:

Otro ejemplo

Macro swap

Definamos ahora un macro que intercambie dos valores:

Solución:

Otro ejemplo

Macro swap

Definamos ahora un macro que intercambie dos valores:

Solución:

```
(define-syntax-rule (or x y)
   (let ([tmp x])
      (set! x y)
      (set! y tmp)))
```

La magia detrás de los macros

La magia detrás de los macros

¿Qué pasa cuando compilamos código con definiciones de macros y su uso?

La magia detrás de los macros

¿Qué pasa cuando compilamos código con definiciones de macros y su uso?

¡Sustitución textual!

La magia detrás de los macros

¿Qué pasa cuando compilamos código con definiciones de macros y su uso?

¡Sustitución textual!

Código original

```
(define x 17)
(define y 29)
(swap x y)
x
y
```

La magia detrás de los macros

¿Qué pasa cuando compilamos código con definiciones de macros y su uso?

¡Sustitución textual!

Código original

```
(define x 17)
(define y 29)
(swap x y)
x
y
```

Código con sustitución textual (después de llamar al macro)

```
(define x 17)
(define y 29)
(let ([tmp x])
   (set! x y) (set! y tmp))
x
y
```

Diferencias entre sistemas de macros

Diferencias entre sistemas de macros

¿Qué pasa si usamos nuestro macro con una variable llamada ”tmp”?

Diferencias entre sistemas de macros

¿Qué pasa si usamos nuestro macro con una variable llamada ”tmp”?

Racket

Diferencias entre sistemas de macros

¿Qué pasa si usamos nuestro macro con una variable llamada ”tmp”?

Racket

Diferencias entre sistemas de macros

¿Qué pasa si usamos nuestro macro con una variable llamada ”tmp”?

Racket

C

Diferencias entre sistemas de macros

¿Qué pasa si usamos nuestro macro con una variable llamada ”tmp”?

Racket

C

Diferencias entre sistemas de macros

¿Qué pasa si usamos nuestro macro con una variable llamada ”tmp”?

Racket

C

¿Por qué pasa esto?

Higiene

Higiene

Definición. (Higiene)

Decimos que el sistema de macros de un lenguaje de programación es higiénico si éste asegura que no haya colisiones entre los identificadores.

Higiene

Definición. (Higiene)

Decimos que el sistema de macros de un lenguaje de programación es higiénico si éste asegura que no haya colisiones entre los identificadores.

Macros más complejos

Macros más complejos

Al igual que las funciona, al crear macros podemos usar la ya conocida técnica de **caza de patrones** para definir macros más complejos de acuerdo a un patrón.

Macros más complejos

Al igual que las funciona, al crear macros podemos usar la ya conocida técnica de **caza de patrones** para definir macros más complejos de acuerdo a un patrón.

```
(define-syntax or
   (syntax-rules ()
      [(or) #f]
      [(or x) x]
      [(or x y) (let ([z x]) (if z z y))]
      [(or x ...) (or x (or y ...))]))
```

Macros más complejos

Al igual que las funciona, al crear macros podemos usar la ya conocida técnica de **caza de patrones** para definir macros más complejos de acuerdo a un patrón.

```
(define-syntax or
   (syntax-rules ()
      [(or) #f]
      [(or x) x]
      [(or x y) (let ([z x]) (if z z y))]
      [(or x ...) (or x (or y ...))]))
```

Nota: La primitiva `define-syntax-rule` para definir macros es por sí misma un macro que hace una transformación a expresiones construidas con `define-syntax` y `syntax-rules`.

Macros más complejos

Al igual que las funciona, al crear macros podemos usar la ya conocida técnica de **caza de patrones** para definir macros más complejos de acuerdo a un patrón.

```
(define-syntax or
   (syntax-rules ()
      [(or) #f]
      [(or x) x]
      [(or x y) (let ([z x]) (if z z y))]
      [(or x ...) (or x (or y ...))]))
```

Nota: La primitiva `define-syntax-rule` para definir macros es por sí misma un macro que hace una transformación a expresiones construidas con `define-syntax` y `syntax-rules`.

Nota: El segundo parámetro de `syntax-rules` es una lista que contendrá las palabras reservadas (*keywords*) involucradas en la expresión.

Lectura recomendada

Autor: Greg Hendershott

http://www.greghendershott.com/fear-of-macros/