Condicionales

Cecilia Manzino

19 de marzo de 2024

Constructor **if**

Vimos que el constructor if tiene ésta forma:

```
( if  ⟨condición⟩   ; es una proposición
      ⟨exp1 ⟩       ; si la condición es V
      ⟨exp2 ⟩)      ; si la condición es F
```

Llamaremos **paso a paso** al modelo que usaremos como mecanismo de evaluación en DrRacket.

Leyes de reducción del **if**

Ley 1:

```
        (if #t a b)
 ==     definición de if (ley 1)
        a
```

Ley 2:

```
        (if #f a b)
 ==     definición de if (ley 2)
        b
```

Ejemplo de reducción

```
        (complementarios? 30 60)
==      def. de función
        (if (= 90 (+ 30 60) ) "C" "NC"))
==      def. de +
        (if (= 90 90) "C" "NC"))
==      def. de =
        (if #t "C" "NC"))
==      definición de if (ley 1)
        "C"
```

Significado de operadores booleanos

¿Qué expresiones condicionales sirven para reformular los operadores and y or?

(and exp_1 exp_2)	?
(or exp_1 exp_2)	?

Significado de operadores booleanos

¿Qué expresiones condicionales sirven para reformular los operadores and y or?

(and exp_1 exp_2)	(if exp_1 exp_2 #f)
(or exp_1 exp_2)	?

Significado de operadores booleanos

¿Qué expresiones condicionales sirven para reformular los operadores and y or?

(and exp_1 exp_2)	(if exp_1 exp_2 #f)
(or exp_1 exp_2)	(if exp_1 #t exp_2)

Ejemplo

Definir un predicado que dadas 3 proposiciones determine si alguna es cierta.

- Con el operador if:

```
; pred1? : Boolean Boolean Boolean -> Boolean
(define (pred1? e1 e2 e3)
  (if  e1
       #t
       (if e2 #t e3)))))
```

- Con operadores booleanos:

```
; pred1? : Boolean Boolean Boolean -> Boolean
(define  (pred1? e1 e2 e3)
         (or e1 e2 e3))
```

Condicionales múltiples

Si un programa tiene que tomar más de una decisión usaremos la sentencia cond en lugar de varios if anidados.

```
(cond   [Condición-1     Resultado-1]
        [Condición-2     Resultado-2]
        ...
        [Condición-n     Resultado-n])
```

Condicionales múltiples

- En lugar de una sóla condición se tienen n condiciones booleanas.
- Cada condición está asociada a un resultado, que puede ser un valor de cualquier tipo.
- Evaluación:
  - Si *Condición-1* evalúa a #t, toda la expresión evalúa a *Resultado-1*.
  - Si *Condición-1* evalúa a #f y *Condición-2* evalúa a #t, toda la expresión evalúa a *Resultado-2*.
  - Se evalúan las condiciones en orden hasta que una evalúe a #t.
  - Si todas evalúan a #f se produce un error.

Reglas de evaluación

```
(cond   [#t              Resultado-1]
        [Condición-2     Resultado-2]
        ...
        [Condición-n     Resultado-n])
```

== definición de cond (ley 1)

Resultado-1

Reglas de evaluación

```
(cond   [#f              Resultado-1]
        [Condición-2     Resultado-2]
        ...
        [Condición-n     Resultado-n])
```

== definición de cond (ley 2)

```
(cond   [Condición-2     Resultado-2]
        ...
        [Condición-n     Resultado-n])
```

Ejemplo

El juego *FizzBuzz* es un juego para niños donde sentados en una ronda van diciendo los números del 1 a n de manera alternada. Cuando alguien debe decir un número que sea múltiplo de 3, en su lugar debe decir la palabra "Fizz", en el caso de que sea múltiplo de 5, debe decir "Buzz" y debe decir "FizzBuzz" si el número es mútiplo de ambos.

Definir una función que dado un número n devuelva la palabra que deba decir el niño que está en la posición n de la ronda.

Una solución

```
; div? : Number Number -> Boolean
(define (div? x y) (= 0 (modulo x y)))

; fizzbuzz : Number -> String
(define (fizzbuzz n)
        (cond   [(div? n 15) "fizzbuzz"]
                [(div? n 3) "fizz"]
                [(div? n 5) "buzz"]
                [else (number->string n )]))
```