Reporte Taller 1 FLP

Andres Felipe Rojas Sanchez
2160328

Juan Camilo Gutierrez Viveros
2159874

Universidad del valle
Sede Tuluá

Ejemplos implementación listas

se crearon varios ejemplos para cada chip y circuito, quedando de la siguiente forma el codigo:

```
;; observadores de chips primitivos
(define or-chip ((prim-chip) chip-or 'A))
(define not-chip ((prim-chip) chip-not 'B))
(define xor-chip ((prim-chip) chip-xor 'C))

;; observadores de circuitos simples
(define circuito-simple1 ((circ_simple) '(cable1 cable2) '(cable3 cable4) or-chip))
(define circuito-simple2 ((circ_simple) '(cableA cableB) '(cableC cableD) not-chip))
(define circuito-simple3 ((circ_simple) '(cablex cabley) '(cablew cablez) xor-chip))

;; observadores de chips
(define chip1 ((comp-chip) '(port1 port2) '(port3 port4) circuito-simple1))
(define chip2 ((comp-chip) '(portA portB) '(portC portD) circuito-simple2))
(define chip3 ((comp-chip) '(portX portY) '(portW portZ) circuito-simple3))

;;observadores de circuitos complejos

(define circuito-complejo1 ((circ-comp) circuito-simple1 (list circuito-simple2) '(ABD CFG) '(DBA GFC)))
(define circuito-complejo2 ((circ-comp) circuito-simple3 (list circuito-simple2) '(XYZ JKL) '(ZYX LKJ)))
```

para imprimirlos en consola se usó:

```
(display chip1)
(newline)
(display chip2)
(newline)
(display chip3)
(newline)
(display circuito-simple1)
(newline)
(display circuito-simple2)
(newline)
(display circuito-complejo1)
(newline)
(display circuito-complejo2)
```

y el resultado después de la compilación es:

```
[Running] racket "d:\programacion\codigos\taller1-flp24\representacion-listas.rkt"
(comp-chip (port1 port2) (port3 port4) (simple-circuit (cable1 cable2) (cable3 cable4) (chip-or A)))
(comp-chip (portA portB) (portC portD) (simple-circuit (cableA cableB) (cableC cableD) (chip-not B)))
(comp-chip (portX portY) (portW portZ) (simple-circuit (cablex cabley) (cablew cablez) (chip-xor C)))
(simple-circuit (cable1 cable2) (cable3 cable4) (chip-or A))
(simple-circuit (cableA cableB) (cableC cableD) (chip-not B))
(circ-comp (simple-circuit (cable1 cable2) (cable3 cable4) (chip-or A)) ((simple-circuit (cableA cableB) (cableC cableD) (chip-not B))) (ABD CFG) (DBA
GFC))
(circ-comp (simple-circuit (cablex cabley) (cablew cablez) (chip-xor C)) ((simple-circuit (cableA cableB) (cableC cableD) (chip-not B))) (XYZ JKL) (ZYX
LKJ))
```

Ejemplos implementación procedimientos:

se crearon varios ejemplos para cada chip y circuito, quedando de la siguiente forma el código

```
;; Observadores de chips primitivos
(define or-chip (prim-chip chip-or 'A))
(define not-chip (prim-chip chip-not 'B))
(define xor-chip (prim-chip chip-xor 'C))

;; Observadores de circuitos simples
(define circuito-simple1 (circ-simple '(cable1 cable2) '(cable3 cable4) or-chip))
(define circuito-simple2 (circ-simple '(cableA cableB) '(cableC cableD) not-chip))
(define circuito-simple3 (circ-simple '(cablex cabley) '(cablew cablez) xor-chip))

;; Observadores de chips
(define chip1 (comp-chip '(port1 port2) '(port3 port4) circuito-simple1))
(define chip2 (comp-chip '(portA portB) '(portC portD) circuito-simple2))
(define chip3 (comp-chip '(portX portY) '(portW portZ) circuito-simple3))

;; Observadores de circuitos complejos
(define circuito-complejo1 (circ-comp circuito-simple1 (list circuito-simple2) '(ABD CFG) '(DBA GFC)))
(define circuito-complejo2 (circ-comp circuito-simple3 (list circuito-simple2) '(XYZ JKL) '(ZYX LKJ)))
```

:
para imprimirlos en consola se usó:

```
(imprimir-comp-chip chip1)
(imprimir-comp-chip chip2)
(imprimir-comp-chip chip3)
(imprimir-circuito circuito-simple1)
(imprimir-circuito circuito-simple2)
(imprimir-circuito circuito-complejo1)
(imprimir-circuito circuito-complejo2)
```

y el resultado fue:

```
Chip compuesto - Entradas: (port1 port2)
Salidas: (port3 port4)
Circuito: Circuito simple - Entradas: (cable1 cable2)
Salidas: (cable3 cable4)
Chip: Chip tipo: chip-or
Símbolo: A


Chip compuesto - Entradas: (portA portB)
Salidas: (portC portD)
Circuito: Circuito simple - Entradas: (cableA cableB)
Salidas: (cableC cableD)
Chip: Chip tipo: chip-not
Símbolo: B


Chip compuesto - Entradas: (portX portY)
Salidas: (portW portZ)
Circuito: Circuito simple - Entradas: (cablex cabley)
Salidas: (cablew cablez)
Chip: Chip tipo: chip-xor
Símbolo: C


Circuito simple - Entradas: (cable1 cable2)
Salidas: (cable3 cable4)
Chip: Chip tipo: chip-or
Símbolo: A
```

```
 Circuito simple - Entradas: (cableA cableB)
 Salidas: (cableC cableD)
 Chip: Chip tipo: chip-not
 Símbolo: B

 Circuito compuesto - Entradas: (ABD CFG)
 Salidas: (DBA GFC)
 Circuito principal: Circuito simple - Entradas: (cable1 cable2)
 Salidas: (cable3 cable4)
 Chip: Chip tipo: chip-or
 Símbolo: A


 Circuitos secundarios: Circuito simple - Entradas: (cableA cableB)
 Salidas: (cableC cableD)
 Chip: Chip tipo: chip-not
 Símbolo: B


 Circuito compuesto - Entradas: (XYZ JKL)
 Salidas: (ZYX LKJ)
 Circuito principal: Circuito simple - Entradas: (cablex cabley)
 Salidas: (cablew cablez)
 Chip: Chip tipo: chip-xor
 Símbolo: C
```

```
Circuitos secundarios: Circuito simple - Entradas: (cableA cableB)
Salidas: (cableC cableD)
Chip: Chip tipo: chip-not
Símbolo: B



[Done] exited with code=0 in 0.676 seconds
```

Ejemplo Representación Datatype

En base a la gramática vista en el código base, se implementó el define-datatype de cada componente circuito, chip y chip primitivo de la siguiente manera:

```
(define-datatype circuito circuito?
  (simple-circuit (in (list-of symbol?))
                  (out (list-of symbol?))
                  (chip chip?))
  (complex-circuit (circ circuito?)
                   (lcircs (list-of circuito?))
                   (input (list-of symbol?))
                   (output (list-of symbol?))))

(define-datatype chip chip?
  (prim-chip (chip-prim chip-prim?))
  (comp-chip (in (list-of symbol?))
             (out (list-of symbol?))
             (circ circuito?)))

(define-datatype chip-prim chip-prim?
  (chip-and)
  (chip-or)
  (chip-not)
  (chip-xor)
  (chip-nand)
  (chip-nor)
  (chip-xnor))
```

Se definieron los siguientes ejemplos para la prueba del datatype:

```
;; Ejemplo 2: circuito simple con un chip primitivo
(define ejemplo-2
  (simple-circuit
    '(x y z)
    '(out)
    (prim-chip (chip-and))
  )
)

;; Ejemplo 3: circuito complejo con dos circuitos simples
(define ejemplo-3
  (complex-circuit
    (simple-circuit
      '(a b)
      '(c)
      (prim-chip (chip-or))
    )
    (list
      (simple-circuit
        '(d e)
        '(f)
        (prim-chip (chip-xor))
      )
    )
    '(a b d e)
    '(c f)
  )
)
```

```
;; Ejemplo 4: circuito complejo con un chip complejo
(define ejemplo-4
  (complex-circuit
    (simple-circuit
      '(g h)
      '(i)
      (comp-chip
        '(IN1 IN2)
        '(OUT1)
        (simple-circuit
          '(j k)
          '(l)
          (prim-chip (chip-not))
        )
      )
    )
    (list)
    '(g h)
    '(i)
  )
```

```
;; Ejemplo 5: Circuitos complejos anidados
(define ejemplo-5
  (complex-circuit
    (complex-circuit
      (simple-circuit
        '(m n)
        '(o)
        (prim-chip (chip-and))
      )
      (list
        (simple-circuit
          '(p q)
          '(r)
          (prim-chip (chip-or))
        )
      )
      '(m n p q)
      '(o r)
    )
    (list)
    '(m n p q)
    '(o r)
  )
```

Se hicieron pruebas de los anteriores ejemplos definidos de esta manera:

```
(display (circuito? ejemplo-2))
(newline)
(display (circuito? ejemplo-3))
(newline)
(display (circuito? ejemplo-4))
(newline)
(display (circuito? ejemplo-5))
(newline)
(display ejemplo-5)
(newline)
(display ejemplo-4)
(newline)
(display ejemplo-3)
(newline)
(display ejemplo-2)
```

Y las pruebas arrojaron lo siguiente:

```
#t
#t
#t
#t
#(struct:complex-circuit #(struct:complex-circuit #(struct:simple-circuit (m n) (o) #(struct:prim-chip #(struct:chip-and))) (#(struct:simple-circuit (p q) (r)
#(struct:prim-chip #(struct:chip-or)))) (m n p q) (o r)) () (m n p q) (o r))
#(struct:complex-circuit #(struct:simple-circuit (g h) (i) #(struct:comp-chip (IN1 IN2) (OUT1) #(struct:simple-circuit (j k) (l) #(struct:prim-chip
#(struct:chip-not))))) () (g h) (i))
#(struct:complex-circuit #(struct:simple-circuit (a b) (c) #(struct:prim-chip #(struct:chip-or))) (#(struct:simple-circuit (d e) (f) #(struct:prim-chip
#(struct:chip-xor)))) (a b d e) (c f))
#(struct:simple-circuit (x y z) (out) #(struct:prim-chip #(struct:chip-and)))
>
```

Ejemplo representacion parse-unparse

Se usó el mismo datatype del ejemplo anterior:

```
(define-datatype circuito circuito?
  (simple-circuit (in (list-of symbol?))
                  (out (list-of symbol?))
                  (chip chip?))
  (complex-circuit (circ circuito?)
                   (lcircs (list-of circuito?))
                   (input (list-of symbol?))
                   (output (list-of symbol?))))

(define-datatype chip chip?
  (prim-chip (chip-prim chip-prim?))
  (comp-chip (in (list-of symbol?))
             (out (list-of symbol?))
             (circ circuito?)))

(define-datatype chip-prim chip-prim?
  (chip-and)
  (chip-or)
  (chip-not)
  (chip-xor)
  (chip-nand)
  (chip-nor)
  (chip-xnor))
```

Se implementó la siguiente función parse:

```
;; parse
(define parse
  (lambda (expr)
    (cond
      [(circuito? expr)
       (cases circuito? expr
         [(simple-circuit in out chip)
          (list 'simple-circuit in out (parse chip))]
         [(complex-circuit circ lcircs input output)
          (list 'complex-circuit (parse circ) (map parse lcircs) input output)])]
      [(chip? expr)
       (cases chip? expr
         [(prim-chip chip-prim)
          (list 'prim-chip (parse-chip-prim chip-prim))]
         [(comp-chip in out circ)
          (list 'comp-chip in out (parse circ))])]
      [(chip-prim? expr)
       (cases chip-prim? expr
         [(chip-and) 'chip-and]
         [(chip-or) 'chip-or]
         [(chip-not) 'chip-not]
         [(chip-xor) 'chip-xor]
         [(chip-nand) 'chip-nand]
         [(chip-nor) 'chip-nor]
         [(chip-xnor) 'chip-xnor])])))
```

Se implementó la siguiente función unparse:

```
;; unparse
(define unparse
  (lambda (circuit)
    (cond
      [(circuito? circuit)
       (cases circuito? circuit
         [(simple-circuit in out chip)
          (simple-circuit in out (unparse chip))]
         [(complex-circuit circ lcircs input output)
          (complex-circuit (unparse circ) (map unparse lcircs) input output)])]
      [(chip? circuit)
       (cases chip? circuit
         [(prim-chip chip-prim)
          (prim-chip (unparse-chip-prim chip-prim))]
         [(comp-chip in out circ)
          (comp-chip in out (unparse circ))])]
      [(chip-prim? circuit)
       (cases chip-prim? circuit
         [(chip-and) (chip-and)]
         [(chip-or) (chip-or)]
         [(chip-not) (chip-not)]
         [(chip-xor) (chip-xor)]
         [(chip-nand) (chip-nand)]
         [(chip-nor) (chip-nor)]
         [(chip-xnor) (chip-xnor)])])))
```