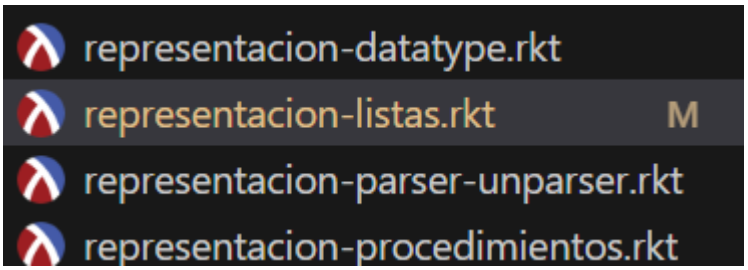
TALLER 1

FLP

Kevin Andres Bejarano - 2067678
Juan David Gutierrez Florez - 2060104
Johan Sebastián Laverde pineda - 2266278
Johan Sebastian Acosta Restrepo 2380393

2024 II
Universidad del valle
Sede Tuluá

4. Use una estructura sencilla para entregar el taller, se recomienda tener 4 archivos:
   - representacion-listas.rkt
   - representacion-procedimientos.rkt
   - representacion-datatype.rkt
   - parser-unparser.rkt



Desarrollo
Constructores:

```
;;Constructores

;Constructor1 chip prim
;<chip prim> := prim_or
;               chip-or()
;            := prim and
;                chip-and()
;            := prim_not
;               chip-not()
;             := prim_xor
;               chip-xor()
;            := prim_nand
;               chip-xor()


(define chip_or
  (lambda ()
      '(chip_or())
    )
)

(define chip_and
  (lambda ()
      '(chip_and())
    )
)
```

```
;;Constructor2 chip
 ;  <chip> := <chip prim>
 ;            prim-chip (chip-prim)
 ;          := chip ( --> {(port)}* )
 ;            ( <-- {(port)}* )
 ;            <circuito>
 ;             comp-chip (in, out, circ)

(define prim-chip
  (lambda (chip-prim)
      (list 'chip-prim chip-prim
    )
))



;comp-chip

(define comp-chip

  (lambda (in out circ)

    (list 'comp-chip in out circ)

  )
 )
```

```
;constructor4 circuitos

;<circuito> := circsimple ({cable }*) ({cable}*)
;             <chip>
;              simple-circuit(in out chip)
;          := circcomp <circuito> {<circuito>}+
;              input{cable}*
;              output{cable}*
;              complex-circuit(circ lcircs in out)



;;simple-circuit

(define simple-circuit

  (lambda (in out chip)

     (list 'simple-circuit in out chip)

  )
)

;;complex-circuit

(define complex-circuit
  (lambda (circ lcircs in out)

     (list 'complex-circuit circ lcircs in out)

  )
)
```

Observadores:

Predicados:

```
;;Predicados

(define chip-prim?
  (lambda (n)
    (equal? (car n) 'chip-prim)))

(define prim-chip?
  (lambda (n)
    (equal? (car n) 'prim-chip)))

(define comp-chip?
  (lambda (n)
    (equal? (car n) 'comp-chip)))

(define simple-circuit?
  (lambda (n)
    (equal? (car n) 'simple-circuit)))

(define complex-circuit?
  (lambda (n)
    (equal? (car n) 'complex-circuit)))
```

Extractores:

```
;;Extractores

;get->type:  devuelve el tipo de circuito o chip con el que se identifica
;ejemplo: (get->type (simple-circuit '(e f) '(g) (prim-chip (chip_or)))) -> 'simple-circuit

(define get->type
  (lambda (c)
    (car c)
  )
 )

;simple-cir->in: Devuelve el/los puertos de entrada de un circuito simple
(define simple-cir->in
  (lambda(c)
      (cadr c)
  )
)

;simple-cir->out: Devuelve el/los puertos de salida de un circuito simple
(define simple-cir->out
  (lambda(c)
      (caddr c)
  )
)

;simple-cir->chip: Devuelve el chip de de un circuito simple
(define simple-cir->chip
  (lambda(c)
      (cadddr c)
  )
)

;complex-cir->circ: Devuelve el circuito de entrada de un circuito complejo
(define complex-cir->circ
  (lambda(c)
      (cadr c)
  )
)
```

```
;complex-cir->lcircs: Devuelve una lista de circuitos de entrada de un circuito complej
(define complex-cir->lcircs
  (lambda(c)
      (caddr c)
  )
)

;complex-cir->in: Devuelve la entrada de un circuito complejo
(define complex-cir->in
  (lambda(c)
      (cadddr c)
))

;complex-cir->out: Devuelve la salida circuito de un circuito complejo
(define complex-cir->out
  (lambda(c)
      (car(cddddr c))
))

;-------------------------------------------------
```

Área del programador:

Circuito 1:

```
;;ejem1

(define cir1 (comp-chip

  '(INA INB INC IND)

  '(OUTA)

  (complex-circuit

    (simple-circuit '(a b) '(e)
      (prim-chip (chip_and)))

    (list
      (simple-circuit '(c d) '(f)
        (prim-chip (chip_and))

      )

      (simple-circuit '(e f) '(g)
        (prim-chip (chip_or))
      )

    )

    '(a b c d)

    '(g))

  ))
```

Circuito 2:

```
;;ejem2

(define cir2


 (complex-circuit
 (simple-circuit
  ' (m n o p)
  ' (e f)
  (comp-chip
   '(INA INB INC IND)
   '(OUTE OUTF)
   (complex-circuit
    (simple-circuit ' (a b) ' (e) (prim-chip (chip_and)))
    (list
     (simple-circuit ' (c d) ' (f) (prim-chip (chip_and))))
    ' (a b c d)
    ' (e f))

   ))

 (list
  (simple-circuit
   ' (e f)
   ' (z)
   (comp-chip
    '(INE INF)
    '(OUTA)
    (simple-circuit ' (e f) ' (g) (prim-chip (chip_or)))

    )

   ))

 ' (m n o p)
 ' (z)))
```

Circuito 3:

```
;; ejem3: Un circuito que utiliza XOR y NAND para combinar tres entradas
(define cir3
  (simple-circuit
    '(A B C)          ; Entradas del circuito
    '(OUT)            ; Salida del circuito
    (comp-chip
      '(A B C)
      '(OUT)
      (complex-circuit
        (simple-circuit '(A B) '(w1) (prim-chip (chip_xor))) ;
        (list
          (simple-circuit '(w1 C) '(w2) (prim-chip (chip_nand))) ;
          (simple-circuit '(w2) '(OUT) (prim-chip (chip_or))))  ;
        '(A B C)
        '(OUT)))))     ; Salida del circuito complejo
```

Circuito 4:

```
;; ejem4: Circuito de un sumador de medio usando XOR y AND
(define cir4
  (simple-circuit
    '(A B)            ; Entradas del circuito (dos bits)
    '(SUM CARRY)      ; Salidas del circuito (suma y acarreo)
    (comp-chip
      '(A B)          ; Entradas del chip
      '(SUM CARRY)    ; Salidas del chip
      (complex-circuit
        (simple-circuit '(A B) '(SUM) (prim-chip (chip_xor)))    ; XOR
        (list
          (simple-circuit '(A B) '(CARRY) (prim-chip (chip_and)))) ; AND
        '(A B)
        '(SUM CARRY)))))
```

Circuito 5:

```
(define cir5
  (comp-chip
    '(INA INB INC IND)
    '(OUT)
    (complex-circuit
      ;; Parte 1: AND entre INA y INB
      (simple-circuit '(INA INB) '(w1) (prim-chip (chip_and)))  ; w1 = INA AND INB

      ;; Parte 2: AND entre INC e IND
      (list
        (simple-circuit '(INC IND) '(w2) (prim-chip (chip_and))) ; w2 = INC AND IND

        ;; Parte 3: OR entre los dos resultados anteriores
        (simple-circuit '(w1 w2) '(OUT) (prim-chip (chip_or)))   ; OUT = w1 OR w2
      )
      '(INA INB INC IND)
      '(OUT)
    )
  )
)
```

Evidencias:

```
(define cir1 (comp-chip

  '(INA INB INC IND)

  '(OUTA)

  (complex-circuit

    (simple-circuit '(a b) '(e)
      (prim-chip (chip_and)))

    (list
      (simple-circuit '(c d) '(f)
        (prim-chip (chip_and))

       )

      (simple-circuit '(e f) '(g)
```

```
Welcome to DrRacket, version 8.13 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
> cir1
#<procedure:...-procedimientos.rkt:130:4>
>
```

```
> (complex-circuit->in cir2)
(m n o p)
>
```

```
> (simple-circuit? cir4)
#t
>
```

```
> (simple-circuit->chip cir4)
#<procedure:...-procedimientos.rkt:130:4>
>
```

```
> (chip? cir1)
#t
>
```

```
> (chip? cir2)
#f
>
```

```
> (circuit? cir2)
#t
> |
```

```
> (circuit? cir2)
#t
> |
```

```
> (chip-prim? cir3 )
#f
>
```

```
> (chip-prim? chipP)
#t
> |
```

```
(define chipP (chip_and))
```

Parse

```
 1 #lang eopl
124       [(eqv? (car exp) 'prim-chip)
125        (prim-chip (cadr exp))]  ;; Extrae el tipo de chip primitivo
126       [(eqv? (car exp) 'comp-chip)
127        (comp-chip
128          (cadr exp)             ;; Lista de entradas
129          (caddr exp)            ;; Lista de salidas
130          (parse (cadddr exp)))]))  ;; Parsea el circuito interno
131
132 (define parse
133   (lambda (exp)
134     (cond
135       ;; Si 'exp' es un circuito simple
136       [(eqv? (car exp) 'simple-circuit)
137        (simple-circuit
138          (cadr exp)                          ;; Lista de entradas
139          (caddr exp)                         ;; Lista de salidas
140          (parse-chip (cadddr exp)))]         ;; Parsea el chip asociado
141
142       ;; Si 'exp' es un circuito complejo
143       [(eqv? (car exp) 'complex-circuit)
144        (complex-circuit
145          (parse (cadr exp))                  ;; Parsea el circuito interno
146          (map parse (caddr exp))             ;; Parsea cada circuito en la lista
147          (cadddr exp)                         ;; Lista de entradas
148          (cadddr (cdr exp)))]))))            ;; Lista de salidas
149
150
151
```

```
Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
> parse cir5
#<procedure:parse>
#(struct:comp-chip
  (INA INB INC IND)
  (OUT)
  #(struct:complex-circuit
    #(struct:simple-circuit (INA INB) (w1) #(struct:prim-chip #(struct:chip_and)))
    (#(struct:simple-circuit (INC IND) (w2) #(struct:prim-chip #(struct:chip_and))) #(struct:simple-circuit (w1 w2) (OUT) #(struct:prim-chip #(struct:chip_or))))
    (INA INB INC IND)
    (OUT)))
>
```

Parse-chip

```
  1 | #lang eopl
109 |     (comp-chip
110 |      '(INE INF)
111 |      '(OUTA)
112 |      (simple-circuit '(e f) '(g) (prim-chip (chip_or)))
113 |
114 |      )
115 |
116 |     ))
117 |
118 |  '(m n o p)
119 |  '(z)))
120 |
121 |  (define parse-chip
122 |    (lambda (exp)
123 |      (cond
124 |        [(eqv? (car exp) 'prim-chip)
125 |         (prim-chip (cadr exp))]  ;; Extrae el tipo de chip primitivo
126 |        [(eqv? (car exp) 'comp-chip)
127 |         (comp-chip
128 |           (cadr exp)               ;; Lista de entradas
129 |           (caddr exp)              ;; Lista de salidas
130 |           (parse (cadddr exp)))]))))  ;; Parsea el circuito interno
131 |
132 |  (define parse
133 |    (lambda (exp)
134 |      (cond
135 |        ;; Si 'exp' es un circuito simple
136 |        [(eqv? (car exp) 'simple-circuit)
```

```
Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
> parse-chip cir5
#<procedure:parse-chip>
#(struct:comp-chip
  (INA INB INC IND)
  (OUT)
  #(struct:complex-circuit
    #(struct:simple-circuit (INA INB) (w1) #(struct:prim-chip #(struct:chip_and)))
    (#(struct:simple-circuit (INC IND) (w2) #(struct:prim-chip #(struct:chip_and))) #(struct:simple-circuit (w1 w2) (OUT) #(struct:prim-chip #(struct:chip_or))))
    (INA INB INC IND)
    (OUT)))
>
```

Ejemplo2 parse

```
  1 | #lang eopl
158 |       '(OUT)
159 |       (complex-circuit
160 |         (simple-circuit '(A B) '(w1) (prim-chip (chip_xor))) ;
161 |         (list
162 |           (simple-circuit '(w1 C) '(w2) (prim-chip (chip_nand))) ;
163 |           (simple-circuit '(w2) '(OUT) (prim-chip (chip_or))))  ;
164 |         '(A B C)
165 |         '(OUT)))))    ; Salida del circuito complejo
166 |
167 |  ;; ejem4: Circuito de un sumador de medio usando XOR y AND
168 |  (define cir4
169 |    (simple-circuit
170 |      '(A B)          ; Entradas del circuito (dos bits)
171 |      '(SUM CARRY)    ; Salidas del circuito (suma y acarreo)
172 |      (comp-chip
173 |        '(A B)        ; Entradas del chip
174 |        '(SUM CARRY)  ; Salidas del chip
175 |        (complex-circuit
176 |          (simple-circuit '(A B) '(SUM) (prim-chip (chip_xor)))   ; XOR
177 |          (list
178 |            (simple-circuit '(A B) '(CARRY) (prim-chip (chip_and)))) ; AND
179 |          '(A B)
180 |          '(SUM CARRY)))))
181 |
182 |
183 |  (define cir5
184 |    (comp-chip
185 |      '(INA INB INC IND)
```

```
Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
> parse cir4
#<procedure:parse>
#(struct:simple-circuit
  (A B)
  (SUM CARRY)
  #(struct:comp-chip
    (A B)
    (SUM CARRY)
    #(struct:complex-circuit #(struct:simple-circuit (A B) (SUM) #(struct:prim-chip #(struct:chip_xor))) (#(struct:simple-circuit (A B) (CARRY) #(struct:prim-chip #(struct:chip_and)))) (A B) (SUM CARRY))))
>
```

Parse ejemplo3

```
  1 | #lang eopl
125 |        (prim-chip (cadr exp))]  ;; Extrae el tipo de chip primitivo
126 |       [(eqv? (car exp) 'comp-chip)
127 |        (comp-chip
128 |          (cadr exp)               ;; Lista de entradas
129 |          (caddr exp)              ;; Lista de salidas
130 |          (parse (cadddr exp)))]))  ;; Parsea el circuito interno
131 |
132 |  (define parse
133 |    (lambda (exp)
134 |      (cond
135 |        ;; Si 'exp' es un circuito simple
136 |        [(eqv? (car exp) 'simple-circuit)
137 |         (simple-circuit
138 |           (cadr exp)                          ;; Lista de entradas
139 |           (caddr exp)                         ;; Lista de salidas
140 |           (parse-chip (cadddr exp)))]         ;; Parsea el chip asociado
141 |
142 |        ;; Si 'exp' es un circuito complejo
143 |        [(eqv? (car exp) 'complex-circuit)
144 |         (complex-circuit
145 |           (parse (cadr exp))                  ;; Parsea el circuito interno
146 |           (map parse (caddr exp))             ;; Parsea cada circuito en la lista
147 |           (cadddr exp)                         ;; Lista de entradas
148 |           (cadddr (cdr exp)))]))              ;; Lista de salidas
149 |
150 |
151 |  ;; ejem3: Un circuito que utiliza XOR y NAND para combinar tres entradas
152 |  (define cir3
```

```
Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
> parse cir3
#<procedure:parse>
#(struct:simple-circuit
  (A B C)
  (OUT)
  #(struct:comp-chip
    (A B C)
    (OUT)
    #(struct:complex-circuit
      #(struct:simple-circuit (A B) (w1) #(struct:prim-chip #(struct:chip_xor)))
      (#(struct:simple-circuit (w1 C) (w2) #(struct:prim-chip #(struct:chip_nand))) #(struct:simple-circuit (w2) (OUT) #(struct:prim-chip #(struct:chip_or))))
      (A B C)
      (OUT))))
>
```

parse-chip Ejemplo 3

```
#lang eopl
    '(INE INF)
    '(OUTA)
    (simple-circuit '(e f) '(g) (prim-chip (chip_or)))

    )

  ))

'(m n o p)
'(z)))

(define parse-chip
  (lambda (exp)
    (cond
      [(eqv? (car exp) 'prim-chip)
       (prim-chip (cadr exp))]  ;; Extrae el tipo de chip primitivo
      [(eqv? (car exp) 'comp-chip)
       (comp-chip
         (cadr exp)              ;; Lista de entradas
         (caddr exp)             ;; Lista de salidas
         (parse (cadddr exp)))])))  ;; Parsea el circuito interno

(define parse
  (lambda (exp)
    (cond
      ;; Si 'exp' es un circuito simple
      [(eqv? (car exp) 'simple-circuit)
       (simple-circuit
```

```
Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
> parse-chip cir3
#<procedure:parse-chip>
#(struct:simple-circuit
  (A B C)
  (OUT)
  #(struct:comp-chip
    (A B C)
    (OUT)
    #(struct:complex-circuit
      #(struct:simple-circuit (A B) (w1) #(struct:prim-chip #(struct:chip_xor)))
      (#(struct:simple-circuit (w1 C) (w2) #(struct:prim-chip #(struct:chip_nand))) #(struct:simple-circuit (w2) (OUT) #(struct:prim-chip #(struct:chip_or))))
      (A B C)
      (OUT))))
>
```

Unparse-chip ejemplo3

```
#lang eopl
  (lambda (chip-datatype)
    (cond
      ['(prim-chip? chip-datatype)
       (list 'prim-chip '(unparse-chip-prim '(prim-chip->ent chip-datatype)))]  ; Usa el unparser de chip-prim
      ['(comp-chip? chip-datatype)
       (list 'comp-chip
             '(comp-chip->in chip-datatype)
             '(comp-chip->out chip-datatype)
             '(unparse-circuit (comp-chip->circ chip-datatype)))])))  ; Cambia a unparse-circuit

(define unparse
  (lambda (circuito-datatype)
    (cond
      ['(simple-circuit? circuito-datatype)
       (list 'simple-circuit
             '(simple-cir->in circuito-datatype)
             '(simple-cir->out circuito-datatype)
             (unparse-chip '(simple-cir->chip circuito-datatype)))]
      ['(complex-circuit? circuito-datatype)
       (list 'complex-circuit
             (unparse '(complex-cir->circ circuito-datatype))
             (map unparse '(complex-cir->lcircs circuito-datatype))
             '(complex-cir->in circuito-datatype)
             '(complex-cir->out circuito-datatype))]
      [else ('error "Tipo no reconocido")])))


;; ejem3: Un circuito que utiliza XOR y NAND para combinar tres entradas
```

```
Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
> unparse cir3
#<procedure:unparse>
#(struct:simple-circuit
  (A B C)
  (OUT)
  #(struct:comp-chip
    (A B C)
    (OUT)
    #(struct:complex-circuit
      #(struct:simple-circuit (A B) (w1) #(struct:prim-chip #(struct:chip_xor)))
      (#(struct:simple-circuit (w1 C) (w2) #(struct:prim-chip #(struct:chip_nand))) #(struct:simple-circuit (w2) (OUT) #(struct:prim-chip #(struct:chip_or))))
      (A B C)
      (OUT))))
>
```

unparse ejemplo3

```
1  #lang eopl
152  (lambda (chip-datatype)
153    (cond
154      ['(prim-chip? chip-datatype)
155       (list 'prim-chip '(unparse-chip-prim '(prim-chip->ent chip-datatype)))]  ; Usa el unparser de chip-prim
156      ['(comp-chip? chip-datatype)
157       (list 'comp-chip
158             '(comp-chip->in chip-datatype)
159             '(comp-chip->out chip-datatype)
160             '(unparse-circuit (comp-chip->circ chip-datatype)))])))  ; Cambia a unparse-circuit
161
162  (define unparse
163    (lambda (circuito-datatype)
164      (cond
165        ['(simple-circuit? circuito-datatype)
166         (list 'simple-circuit
167               '(simple-cir->in circuito-datatype)
168               '(simple-cir->out circuito-datatype)
169               (unparse-chip '(simple-cir->chip circuito-datatype)))]
170        ['(complex-circuit? circuito-datatype)
171         (list 'complex-circuit
172               (unparse '(complex-cir->circ circuito-datatype))
173               (map unparse '(complex-cir->lcircs circuito-datatype))
174               '(complex-cir->in circuito-datatype)
175               '(complex-cir->out circuito-datatype))]
176        [else ('error "Tipo no reconocido")])))
177
178
179
180  ;; ejem3: Un circuito que utiliza XOR y NAND para combinar tres entradas
```

```
Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
> unparse cir3
#<procedure:unparse>
#(struct:simple-circuit
  (A B C)
  (OUT)
  #(struct:comp-chip
    (A B C)
    (OUT)
    #(struct:complex-circuit
      #(struct:simple-circuit (A B) (w1) #(struct:prim-chip #(struct:chip_xor)))
      (#(struct:simple-circuit (w1 C) (w2) #(struct:prim-chip #(struct:chip_nand))) #(struct:simple-circuit (w2) (OUT) #(struct:prim-chip #(struct:chip_or))))
      (A B C)
      (OUT))))
>
```

unparse ejemplo4

```
1  #lang eopl
152  (lambda (chip-datatype)
153    (cond
154      ['(prim-chip? chip-datatype)
155       (list 'prim-chip '(unparse-chip-prim '(prim-chip->ent chip-datatype)))]  ; Usa el unparser de chip-prim
156      ['(comp-chip? chip-datatype)
157       (list 'comp-chip
158             '(comp-chip->in chip-datatype)
159             '(comp-chip->out chip-datatype)
160             '(unparse-circuit (comp-chip->circ chip-datatype)))])))  ; Cambia a unparse-circuit
161
162  (define unparse
163    (lambda (circuito-datatype)
164      (cond
165        ['(simple-circuit? circuito-datatype)
166         (list 'simple-circuit
167               '(simple-cir->in circuito-datatype)
168               '(simple-cir->out circuito-datatype)
169               (unparse-chip '(simple-cir->chip circuito-datatype)))]
170        ['(complex-circuit? circuito-datatype)
171         (list 'complex-circuit
172               (unparse '(complex-cir->circ circuito-datatype))
173               (map unparse '(complex-cir->lcircs circuito-datatype))
174               '(complex-cir->in circuito-datatype)
175               '(complex-cir->out circuito-datatype))]
176        [else ('error "Tipo no reconocido")])))
177
178
179
180  ;; ejem3: Un circuito que utiliza XOR y NAND para combinar tres entradas
```

```
Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
> unparse cir4
#<procedure:unparse>
#(struct:simple-circuit
  (A B)
  (SUM CARRY)
  #(struct:comp-chip
    (A B)
    (SUM CARRY)
    #(struct:complex-circuit #(struct:simple-circuit (A B) (SUM) #(struct:prim-chip #(struct:chip_xor))) (#(struct:simple-circuit (A B) (CARRY) #(struct:prim-chip #(struct:chip_and)))) (A B) (SUM CARRY))))
>
```

unparse-chip ejemplo4

```
152  (lambda (chip-datatype)
153    (cond
154      ['(prim-chip? chip-datatype)
155       (list 'prim-chip '(unparse-chip-prim '(prim-chip->ent chip-datatype)))]  ; Usa el unparser de chip-prim
156      ['(comp-chip? chip-datatype)
157       (list 'comp-chip
158             '(comp-chip->in chip-datatype)
159             '(comp-chip->out chip-datatype)
160             '(unparse-circuit (comp-chip->circ chip-datatype)))])))  ; Cambia a unparse-circuit
161
162  (define unparse
163    (lambda (circuito-datatype)
164      (cond
165        ['(simple-circuit? circuito-datatype)
```

```
Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
> unparse-chip cir4
#<procedure:unparse-chip>
#(struct:simple-circuit
  (A B)
  (SUM CARRY)
  #(struct:comp-chip
    (A B)
    (SUM CARRY)
    #(struct:complex-circuit #(struct:simple-circuit (A B) (SUM) #(struct:prim-chip #(struct:chip_xor))) (#(struct:simple-circuit (A B) (CARRY) #(struct:prim-chip #(struct:chip_and)))) (A B) (SUM CARRY))))
>
```

Unparse ejemplo 5

```
1  #lang eopl
204        (complex-circuit
205          (simple-circuit '(A B) '(SUM) (prim-chip (chip_xor)))   ; XOR
206          (list
207            (simple-circuit '(A B) '(CARRY) (prim-chip (chip_and)))) ; AND
208          '(A B)
209          '(SUM CARRY)))))
210
211
212  (define cir5
213    (comp-chip
214      '(INA INB INC IND)
215      '(OUT)
216      (complex-circuit
217        ;; Parte 1: AND entre INA y INB
218        (simple-circuit '(INA INB) '(w1) (prim-chip (chip_and)))  ; w1 = INA AND INB
219
220        ;; Parte 2: AND entre INC e IND
221        (list
222          (simple-circuit '(INC IND) '(w2) (prim-chip (chip_and))) ; w2 = INC AND IND
223
224          ;; Parte 3: OR entre los dos resultados anteriores
225          (simple-circuit '(w1 w2) '(OUT) (prim-chip (chip_or)))   ; OUT = w1 OR w2
226        )
227        '(INA INB INC IND)
228        '(OUT)
229      )
230    )
231  )
```

```
Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
> unparse cir5
#<procedure:unparse>
#(struct:comp-chip
  (INA INB INC IND)
  (OUT)
  #(struct:complex-circuit
    #(struct:simple-circuit (INA INB) (w1) #(struct:prim-chip #(struct:chip_and)))
    (#(struct:simple-circuit (INC IND) (w2) #(struct:prim-chip #(struct:chip_and))) #(struct:simple-circuit (w1 w2) (OUT) #(struct:prim-chip #(struct:chip_or))))
    (INA INB INC IND)
    (OUT)))
>
```

Unparse-chip ejemplo5

```
1  #lang eopl
204        (complex-circuit
205          (simple-circuit '(A B) '(SUM) (prim-chip (chip_xor)))   ; XOR
206          (list
207            (simple-circuit '(A B) '(CARRY) (prim-chip (chip_and)))) ; AND
208          '(A B)
209          '(SUM CARRY)))))
210
211
212  (define cir5
213    (comp-chip
214      '(INA INB INC IND)
215      '(OUT)
216      (complex-circuit
217        ;; Parte 1: AND entre INA y INB
218        (simple-circuit '(INA INB) '(w1) (prim-chip (chip_and)))  ; w1 = INA AND INB
219
220        ;; Parte 2: AND entre INC e IND
221        (list
222          (simple-circuit '(INC IND) '(w2) (prim-chip (chip_and))) ; w2 = INC AND IND
223
224          ;; Parte 3: OR entre los dos resultados anteriores
225          (simple-circuit '(w1 w2) '(OUT) (prim-chip (chip_or)))   ; OUT = w1 OR w2
226        )
227        '(INA INB INC IND)
228        '(OUT)
229      )
230    )
231  )
```

```
Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
> unparse-chip cir5
#<procedure:unparse-chip>
#(struct:comp-chip
  (INA INB INC IND)
  (OUT)
  #(struct:complex-circuit
    #(struct:simple-circuit (INA INB) (w1) #(struct:prim-chip #(struct:chip_and)))
    (#(struct:simple-circuit (INC IND) (w2) #(struct:prim-chip #(struct:chip_and))) #(struct:simple-circuit (w1 w2) (OUT) #(struct:prim-chip #(struct:chip_or))))
    (INA INB INC IND)
    (OUT)))
>
```