TALLER 1

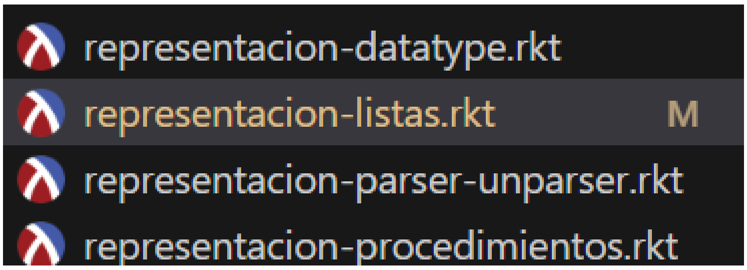
FLP

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



representacion-datatype.rkt

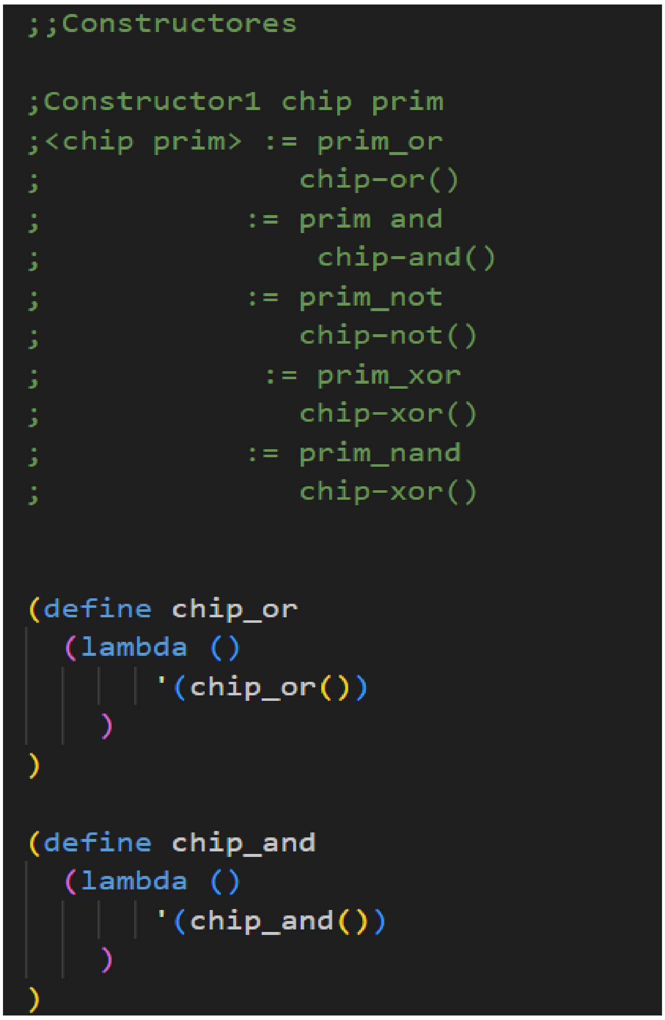
representacion-listas.rkt

representacion-parser-unparser.rkt

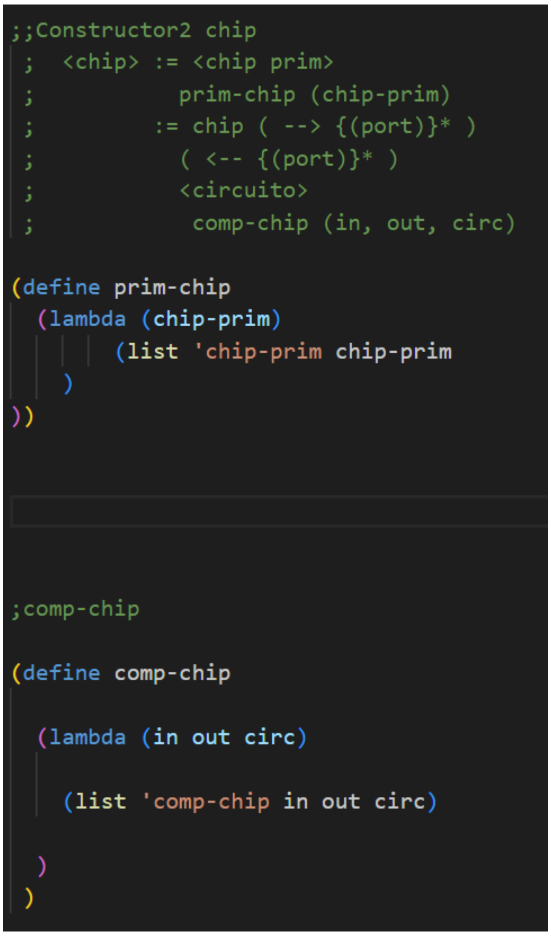
representacion-procedimientos.rkt

Desarrollo

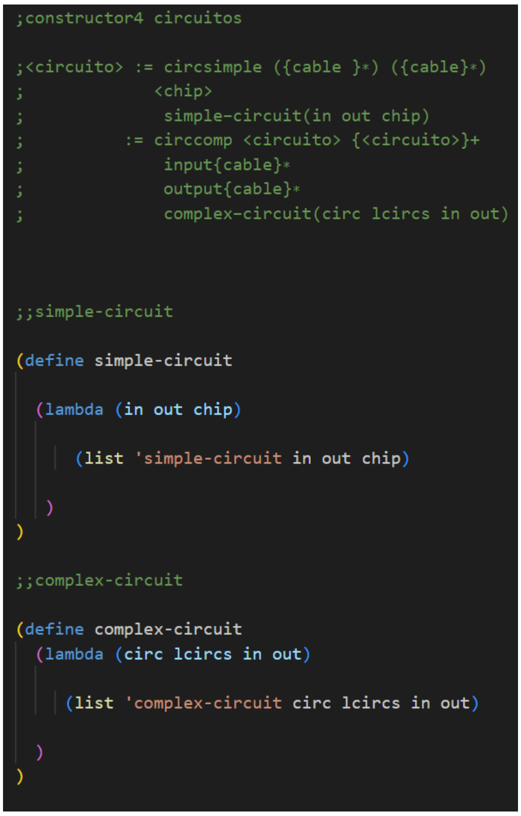
Constructores:



```
;;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())
    )
)
```



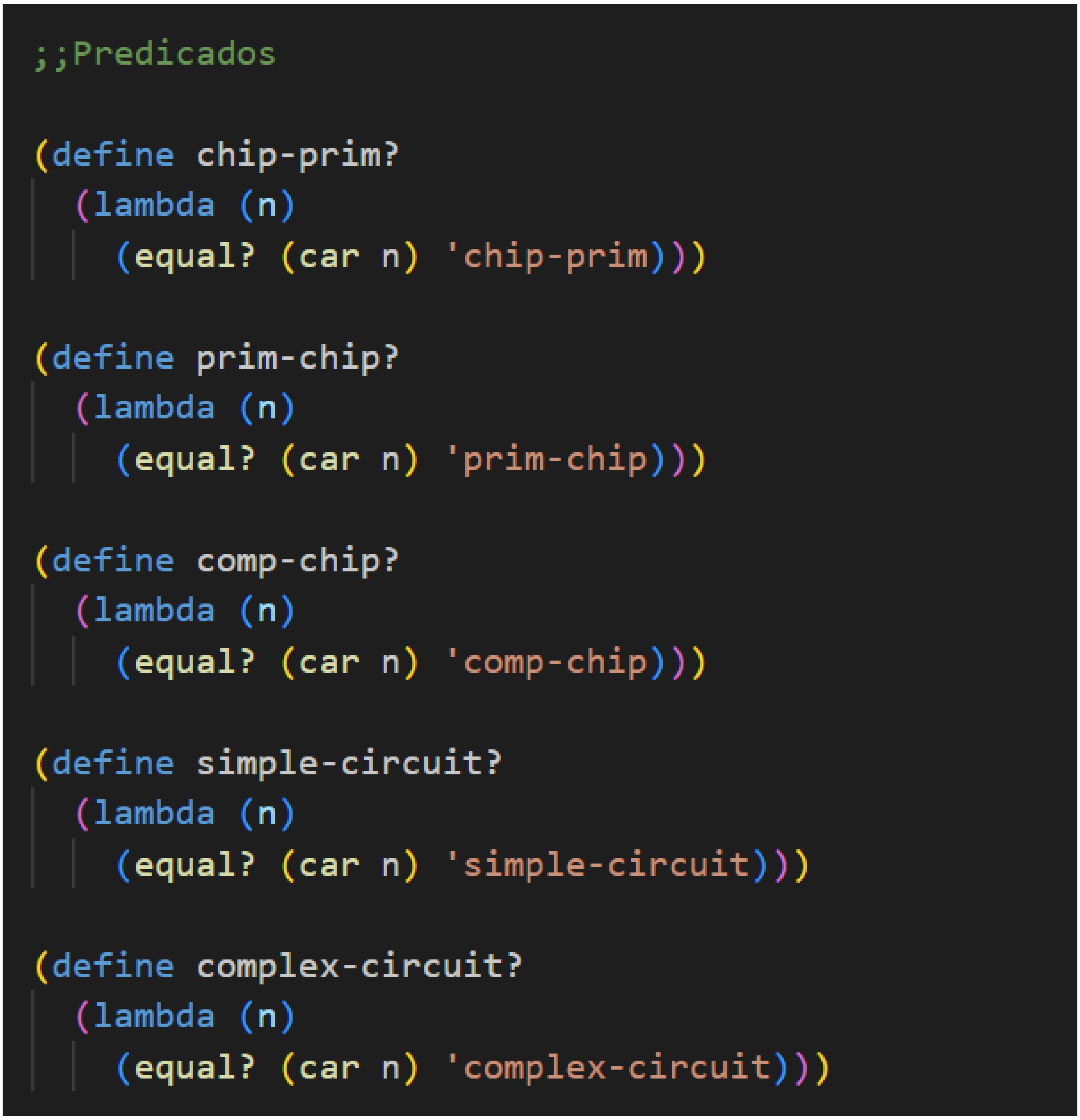
```
;;Constructor2 chip
 ;  <chip> := <chip prim>
 ;           prim-chip (chip-prim)
 ;         := chip ( --> {(port)}* )
 ;           ( <-- {(port)}* )
 ;           <circuito>
 ;            comp-chip (in, out, circ)

(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)

  )
)
```

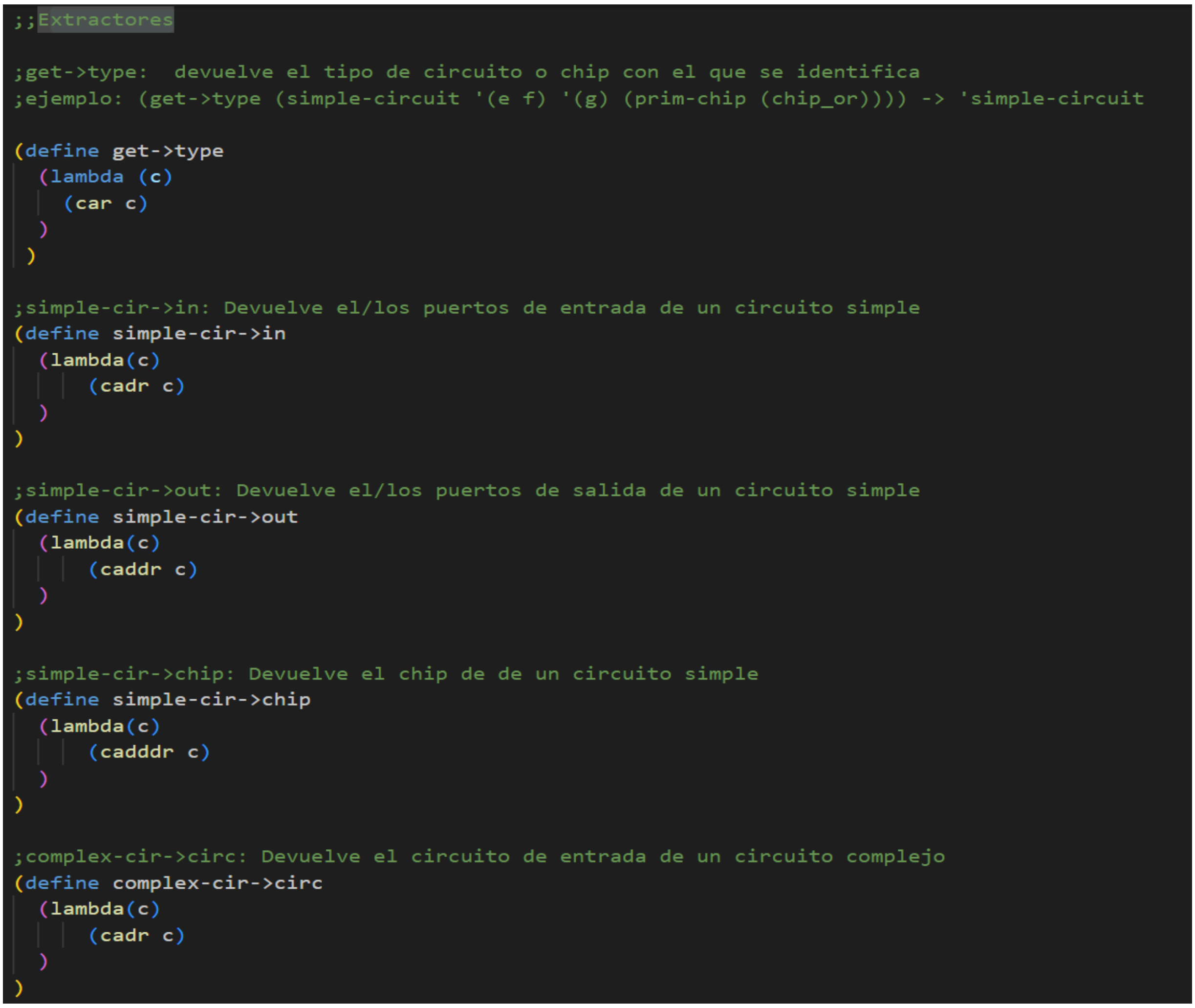
Observadores:

Predicados:

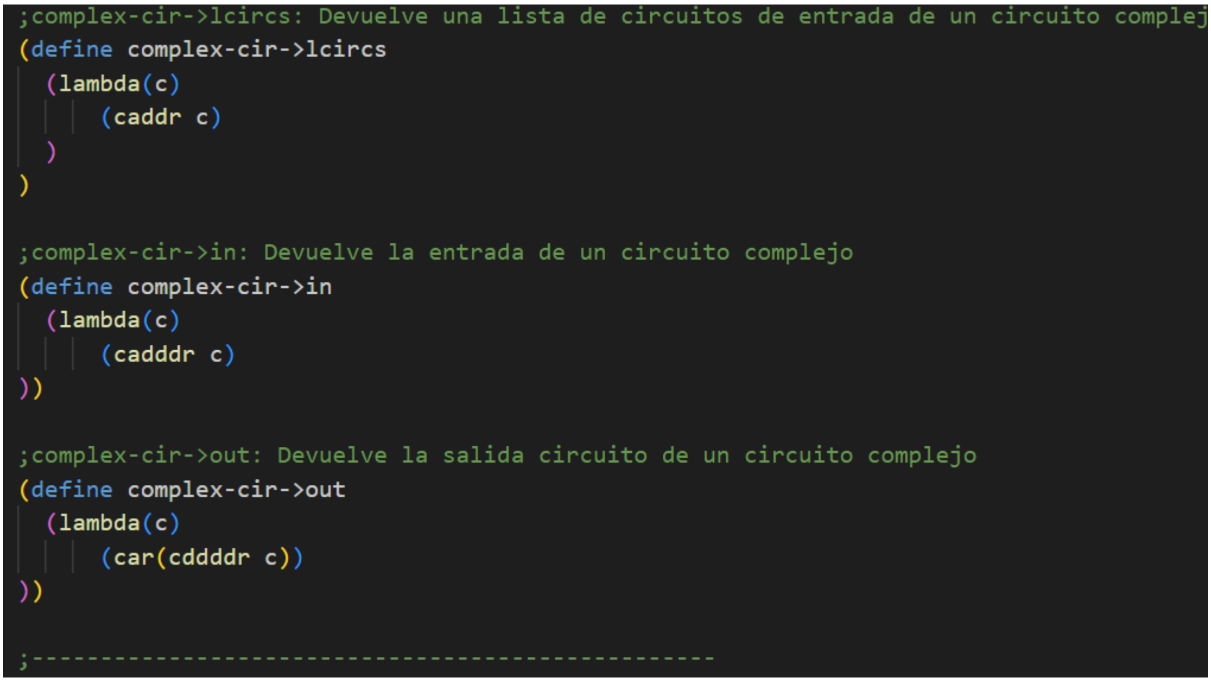


```scheme
;;Predicados

(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:



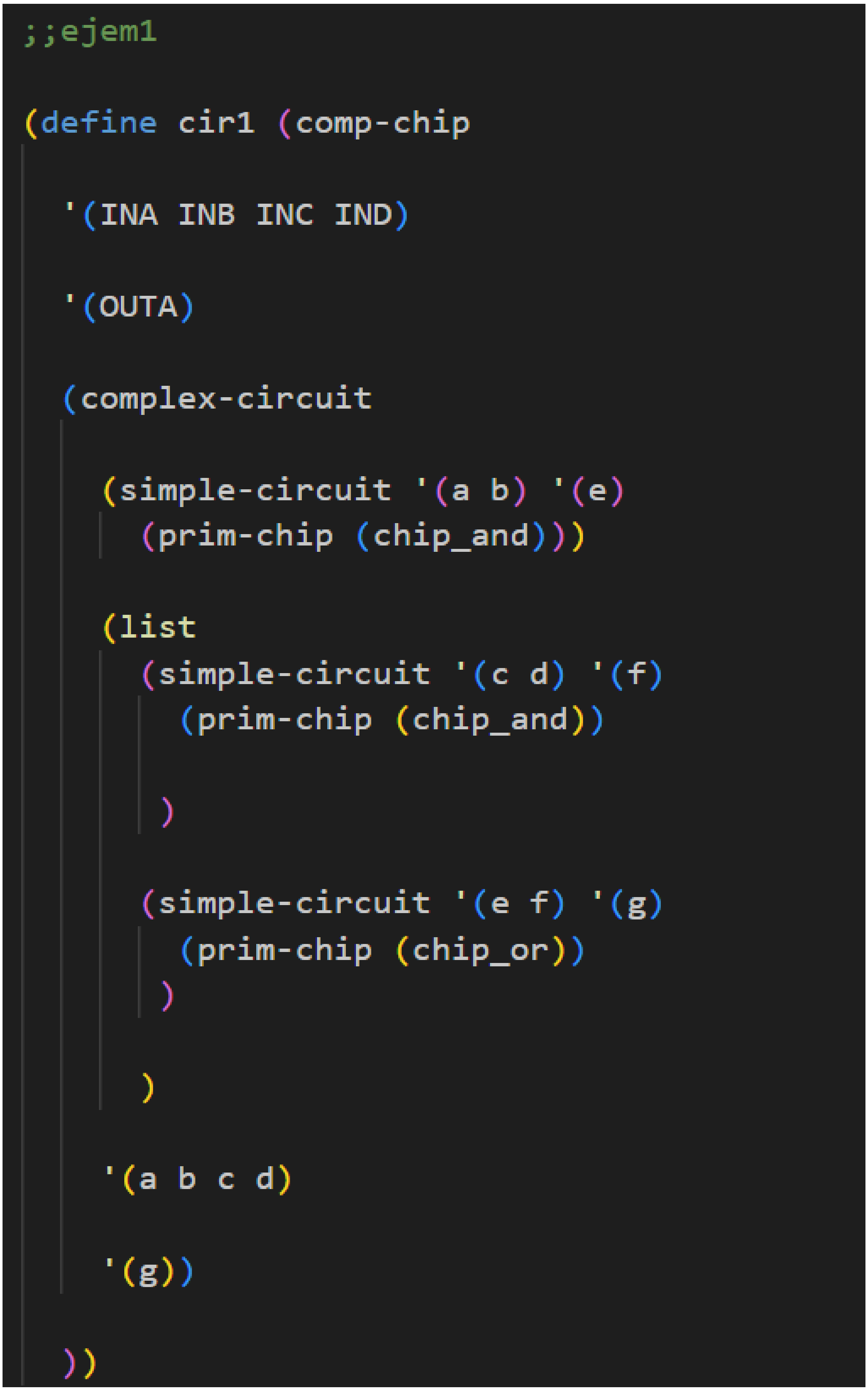
```
;;Extractores

;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))
))

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

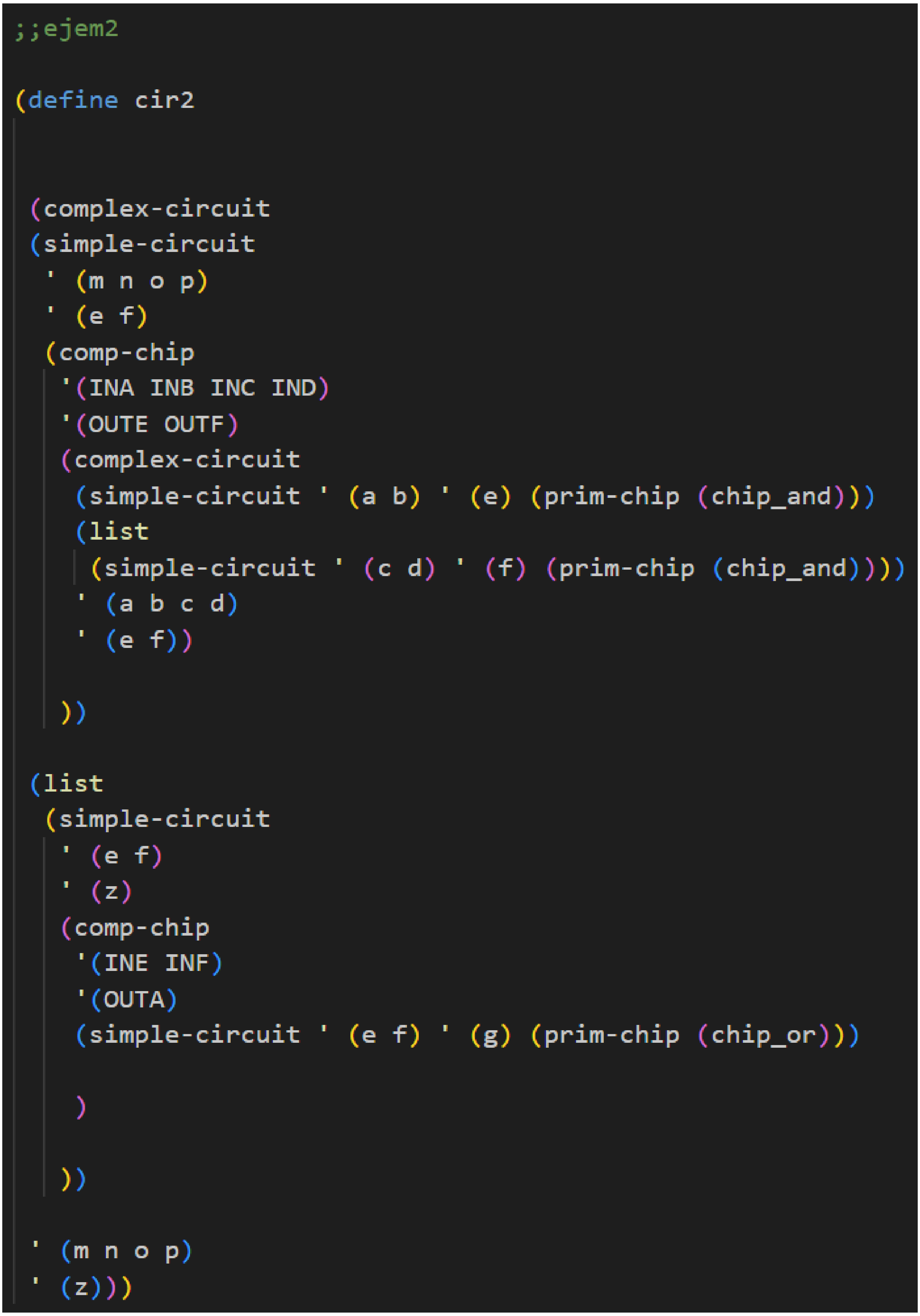
Área del programador:

Circuito 1:



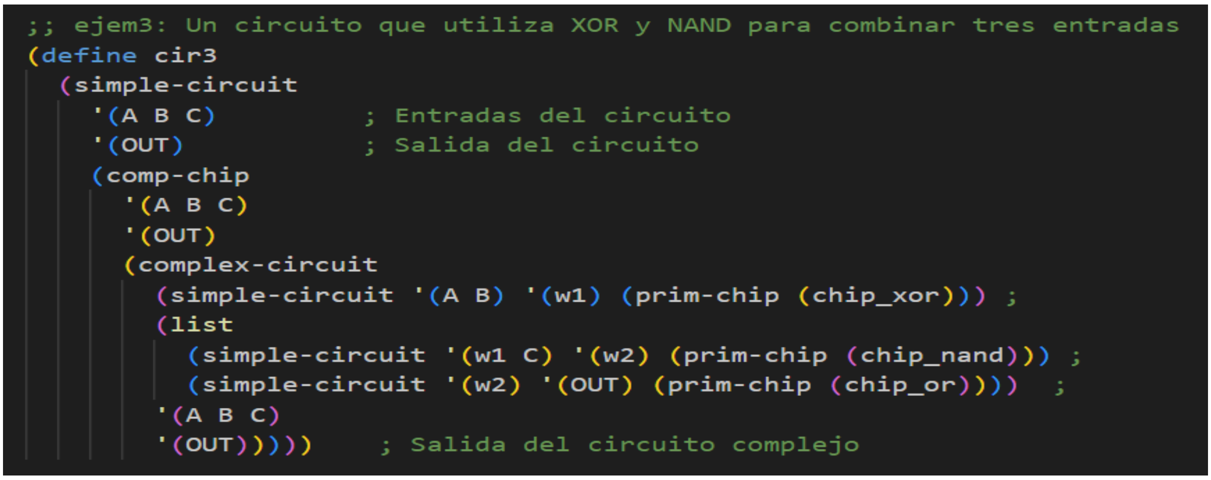
```
;;ejem1

(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:



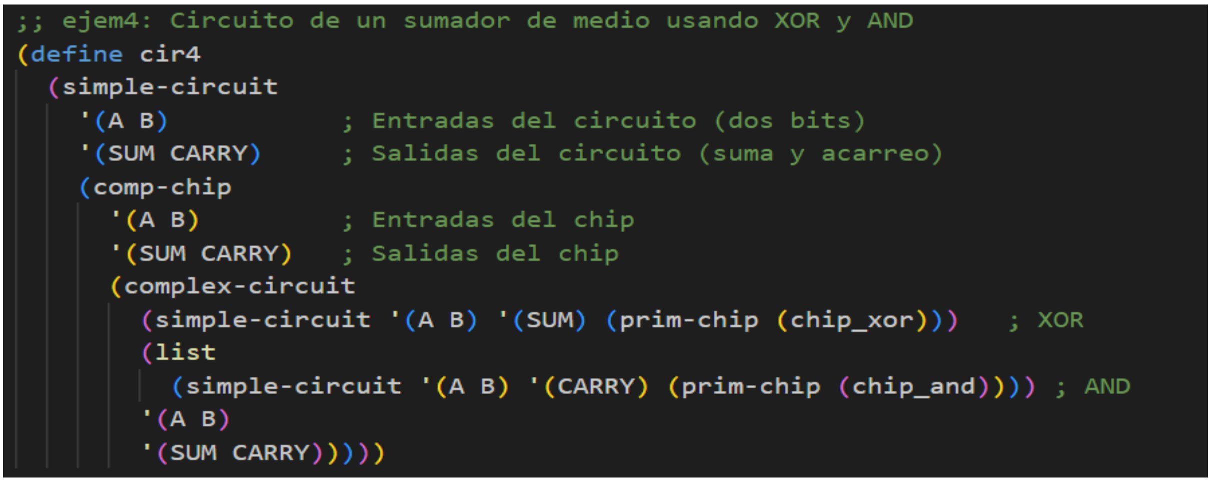
```
;;ejem2

(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:



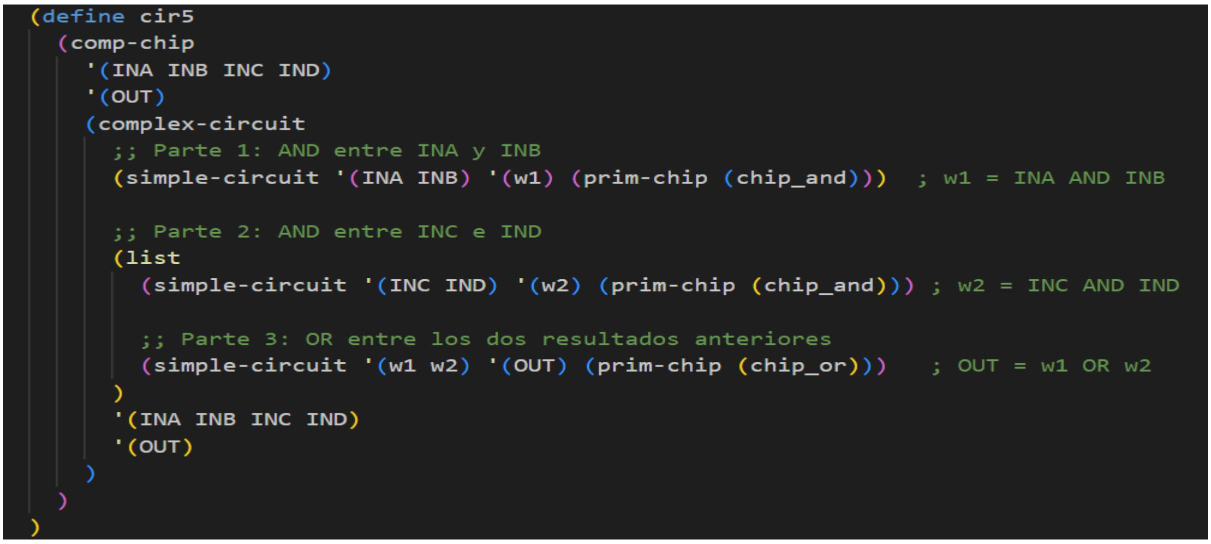
```
;; ejem3: Un circuito que utiliza XOR y NAND para combinar tres entradas
(define cir3
  (simple-circuit
    '(A B C)            ; Entradas del circuito
    '(OUT)              ; Salida del circuito
    (comp-chip
      '(A B C)
      '(OUT)
      (complex-circuit
        (simple-circuit '(A B) '(w1)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



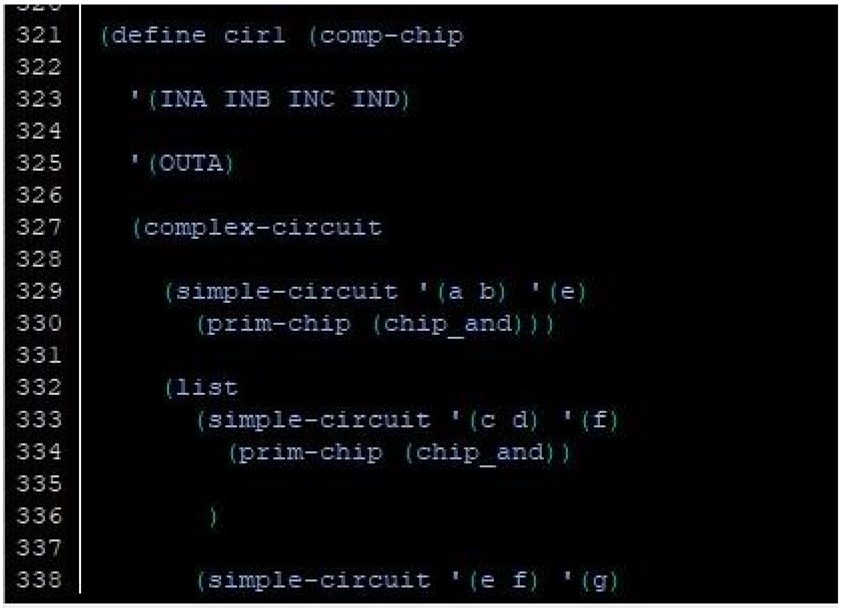
```
;; ejem4: Circuito de un sumador de medio usando XOR y AND
(define cir4
  (simple-circuit
    '(A B)            ; Entradas del circuito (dos bits)
    '(SUM CARRY)      ; Salidas del circuito (suma y acarreo)
    (comp-chip
      '(A B)          ; Entradas del chip
      '(SUM CARRY)    ; Salidas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

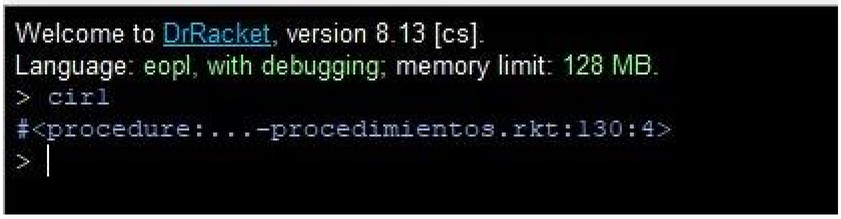


```
(define cir5
  (comp-chip
    '(INA INB INC IND)
    '(OUT)
    (complex-circuit
      ;; Parte 1: AND entre INA y INB
      (simple-circuit '(INA INB) '(w1) (prim-chip (chip_and)))  ; w1 = INA AND INB

      ;; 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)
    )
  )
)
```

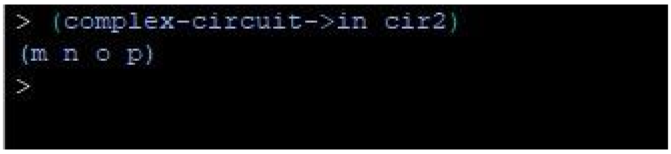
Evidencias:



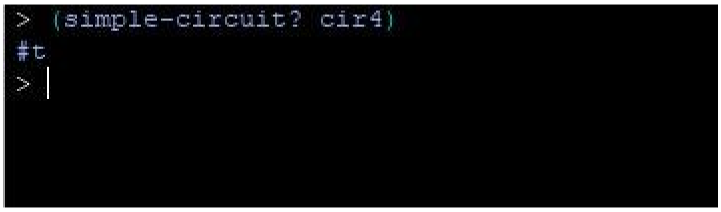
```
(define cir1 (comp-chip

  '(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)
```



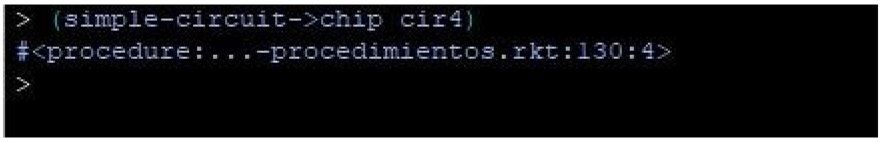
```
Welcome to DrRacket, version 8.13 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
> cir1
#<procedure:...-procedimientos.rkt:130:4>
> 
```



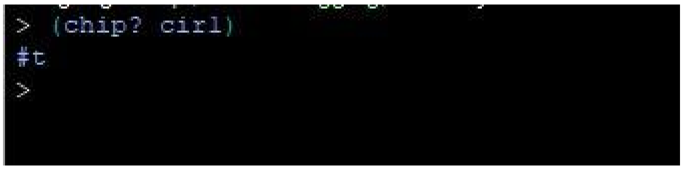
```
> (complex-circuit->in cir2)
(m n o p)
>
```



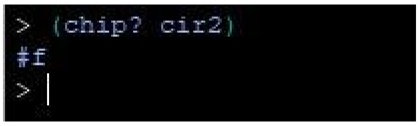
```
> (simple-circuit? cir4)
#t
> 
```



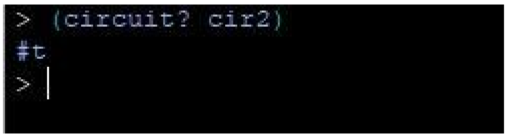
```
> (simple-circuit->chip cir4)
#<procedure:...-procedimientos.rkt:130:4>
>
```



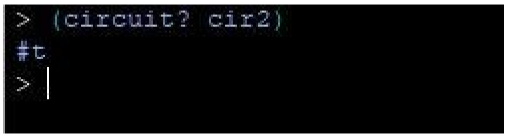
```
> (chip? cir1)
#t
>
```



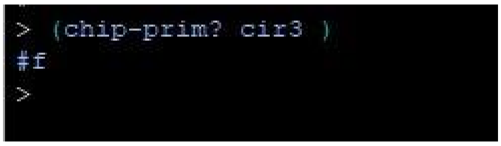
```
> (chip? cir2)
#f
> 
```



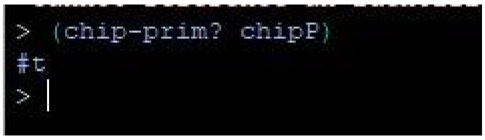
```
> (circuit? cir2)
#t
> |
```



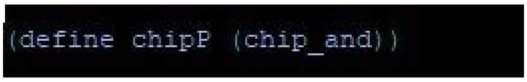
```
> (circuit? cir2)
#t
> |
```



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

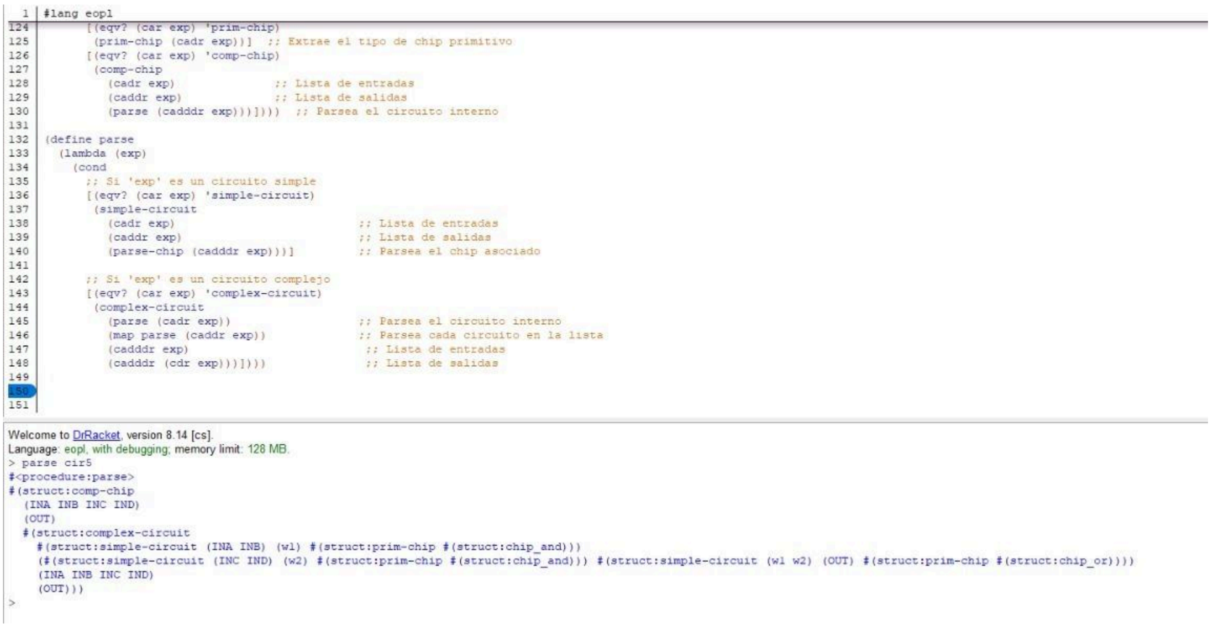


```
> (chip-prim? chipP)
#t
> |
```



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

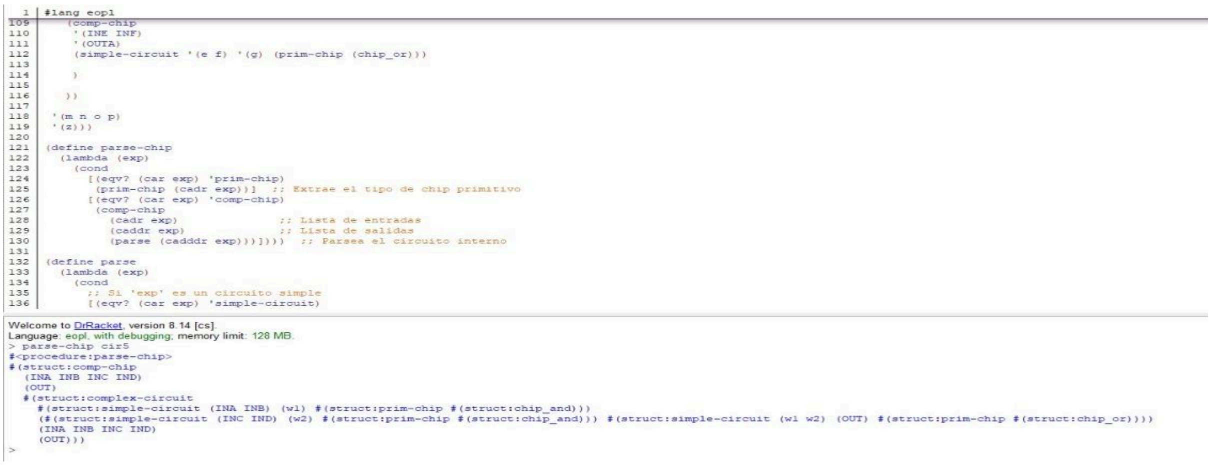
Parse



```
#lang eopl
        [(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
         (cadr exp)                         ;; Lista de entradas
         (caddr exp)                        ;; Lista de salidas
         (parse-chip (cadddr exp)))]        ;; Parsea el chip asociado

      ;; Si 'exp' es un circuito complejo
      [(eqv? (car exp) 'complex-circuit)
       (complex-circuit
         (parse (cadr exp))                 ;; Parsea el circuito interno
         (map parse (caddr exp))            ;; Parsea cada circuito en la lista
         (cadddr exp)                        ;; Lista de entradas
         (cadddr (cdr exp)))])))             ;; Lista de salidas
```

```
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)))
>
```

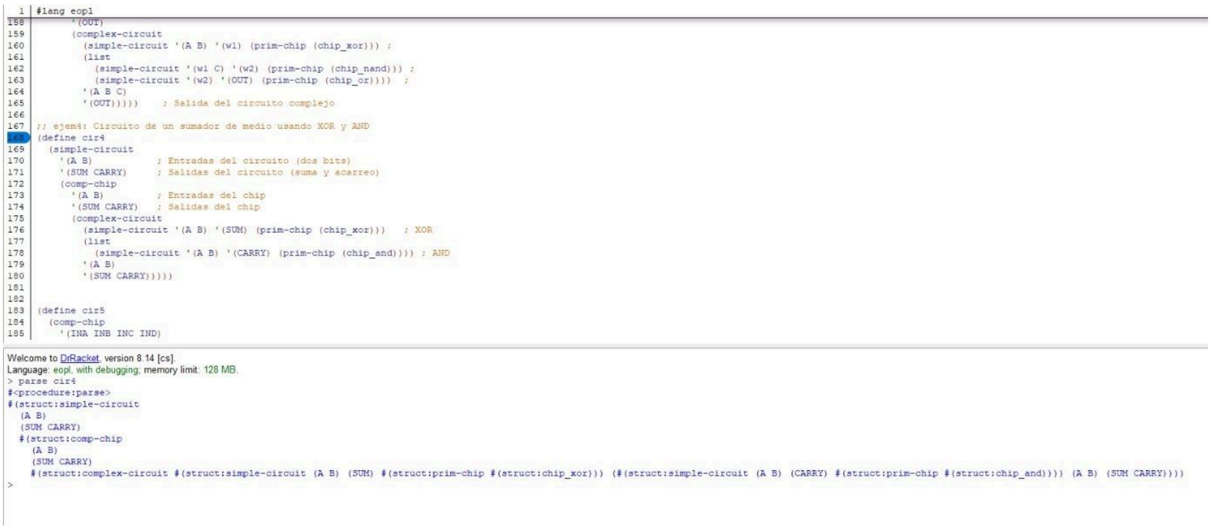
Parse-chip



```
#lang eopl
      (comp-chip
       '(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)
```

```
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)))
>
```

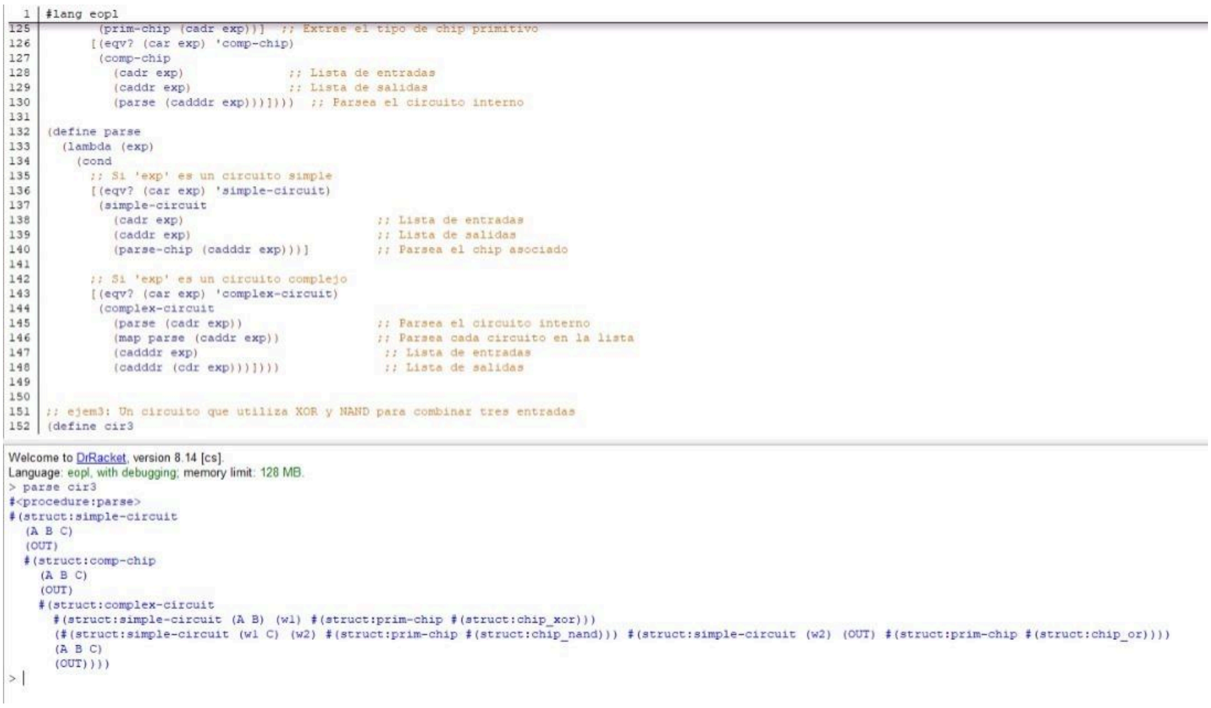
Ejemplo2 parse



```
#lang eopl
        '(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

;; 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)))))


(define cir5
  (comp-chip
    '(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))))
>
```

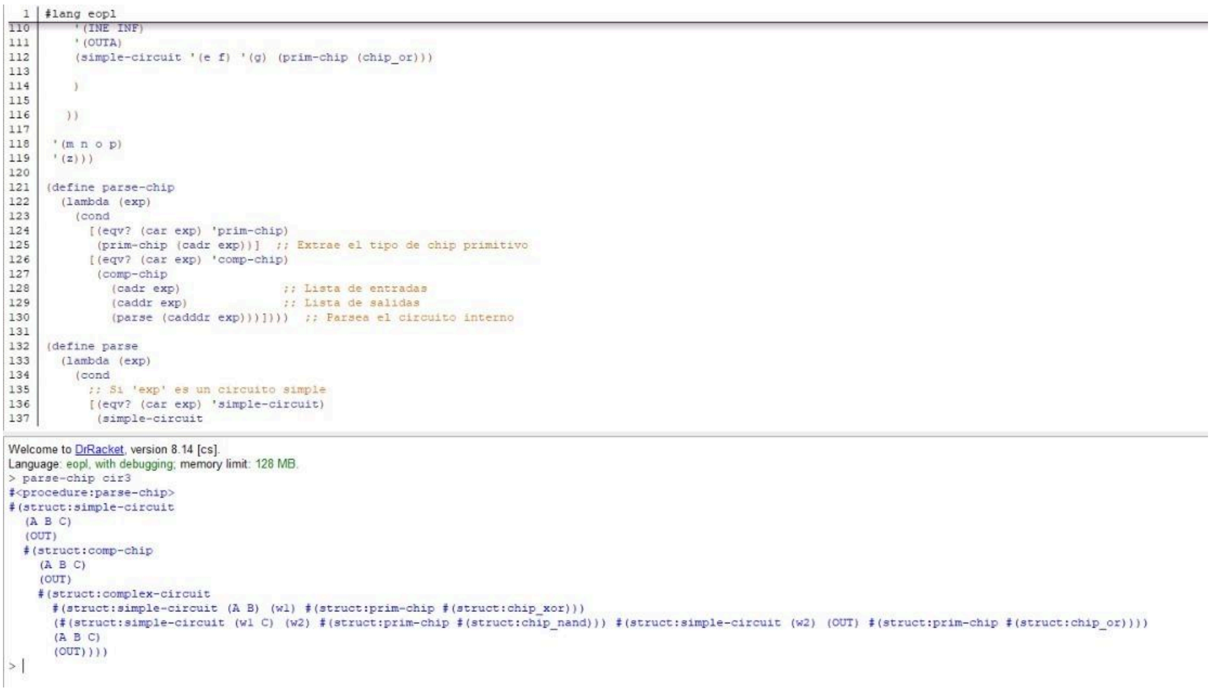
Parse ejemplo3



```
#lang eopl
        (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
          (cadr exp)                          ;; Lista de entradas
          (caddr exp)                         ;; Lista de salidas
          (parse-chip (cadddr exp)))]         ;; Parsea el chip asociado

       ;; Si 'exp' es un circuito complejo
       [(eqv? (car exp) 'complex-circuit)
        (complex-circuit
          (parse (cadr exp))                  ;; Parsea el circuito interno
          (map parse (caddr exp))             ;; Parsea cada circuito en la lista
          (cadddr exp)                         ;; Lista de entradas
          (cadddr (cdr exp)))])))              ;; Lista de salidas


 ;; ejem3: Un circuito que utiliza XOR y NAND para combinar tres entradas
 (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))))
> |
```

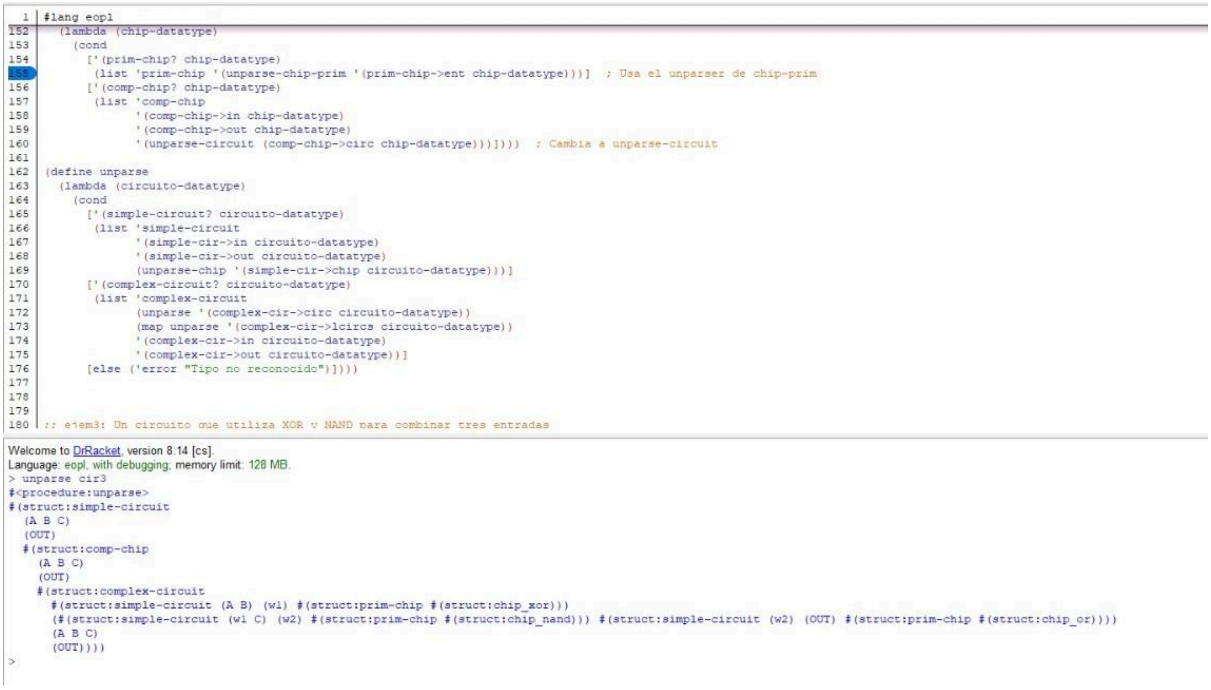
parse-chip Ejemplo 3



```
#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))))
> |
```

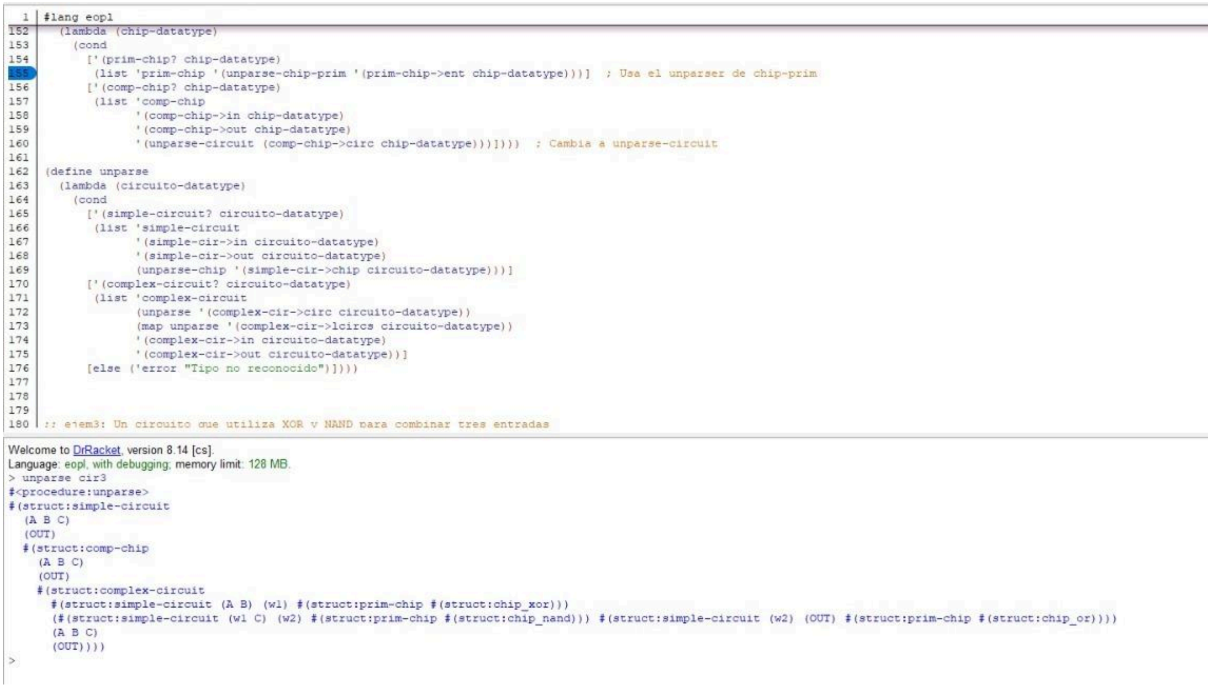
Unparse-chip ejemplo3



```
#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))))
>
```

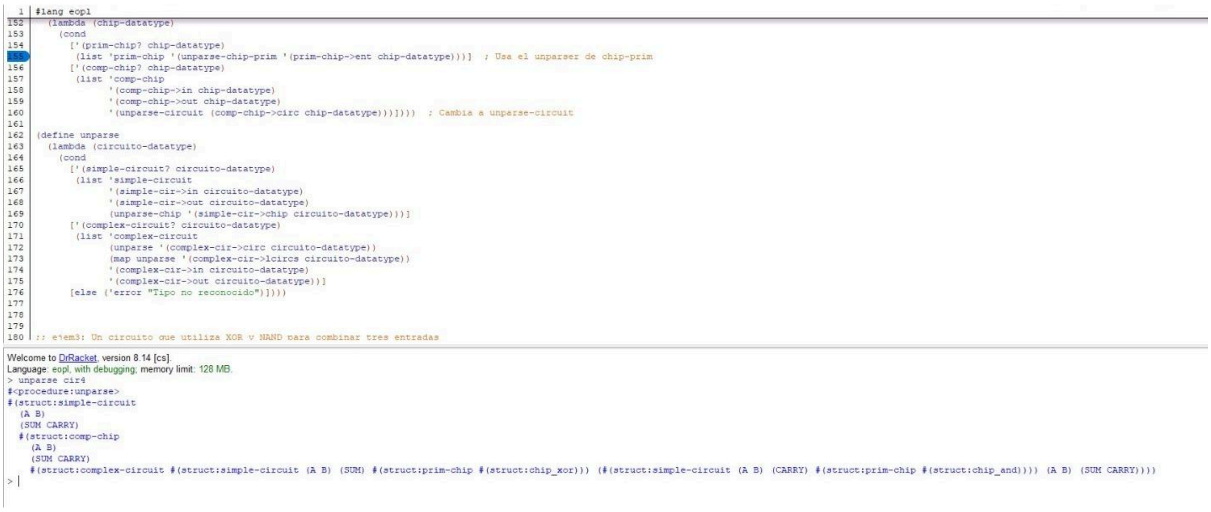
unparse ejemplo3



```
#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))))
>
```

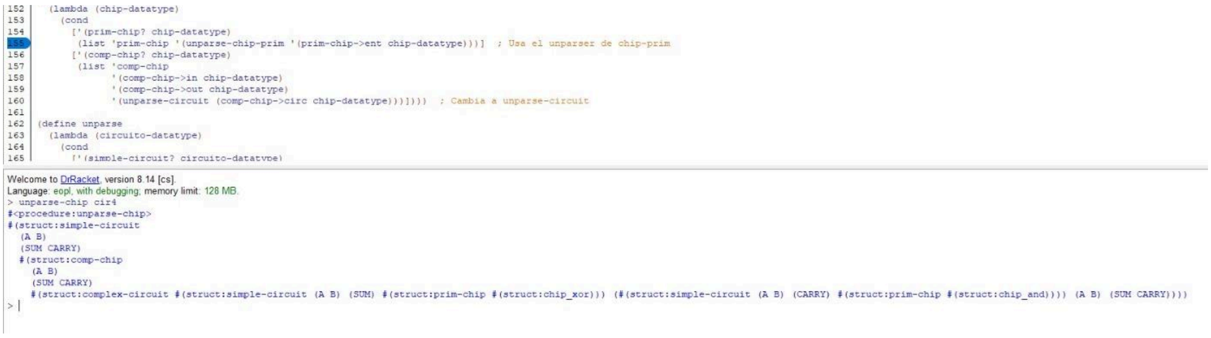
unparse ejemplo4



```
#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 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))))
> |
```

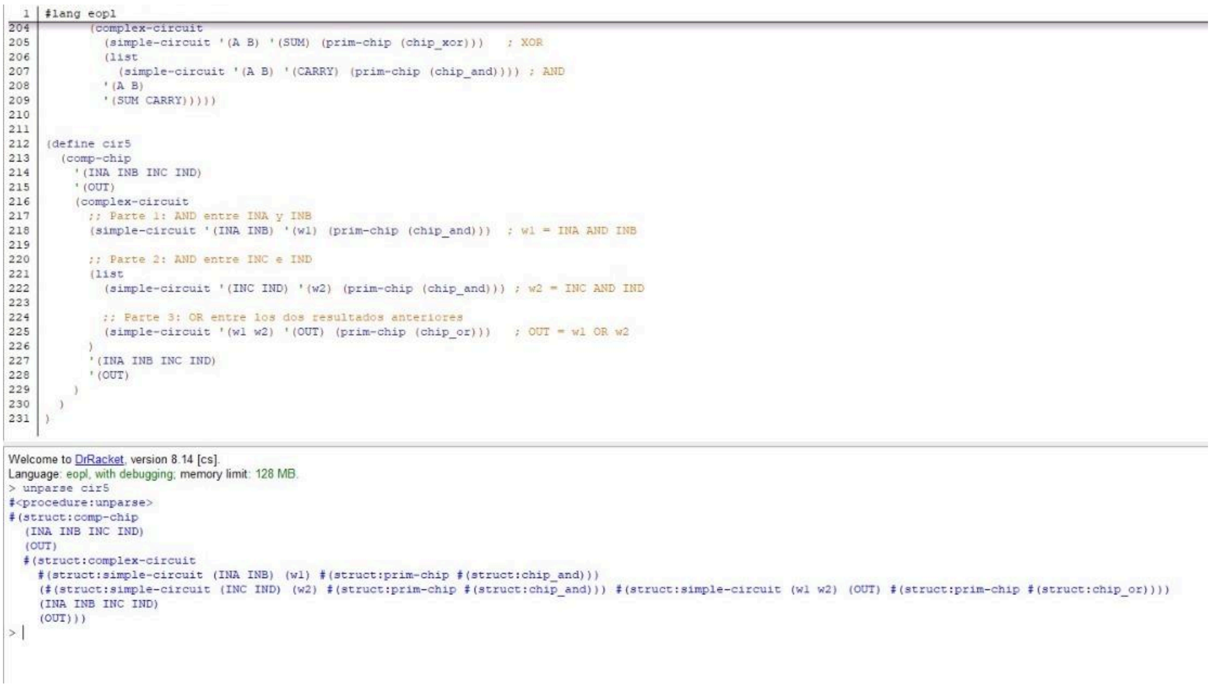
unparse-chip ejemplo4



```
  (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)
```

```
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))))
> |
```

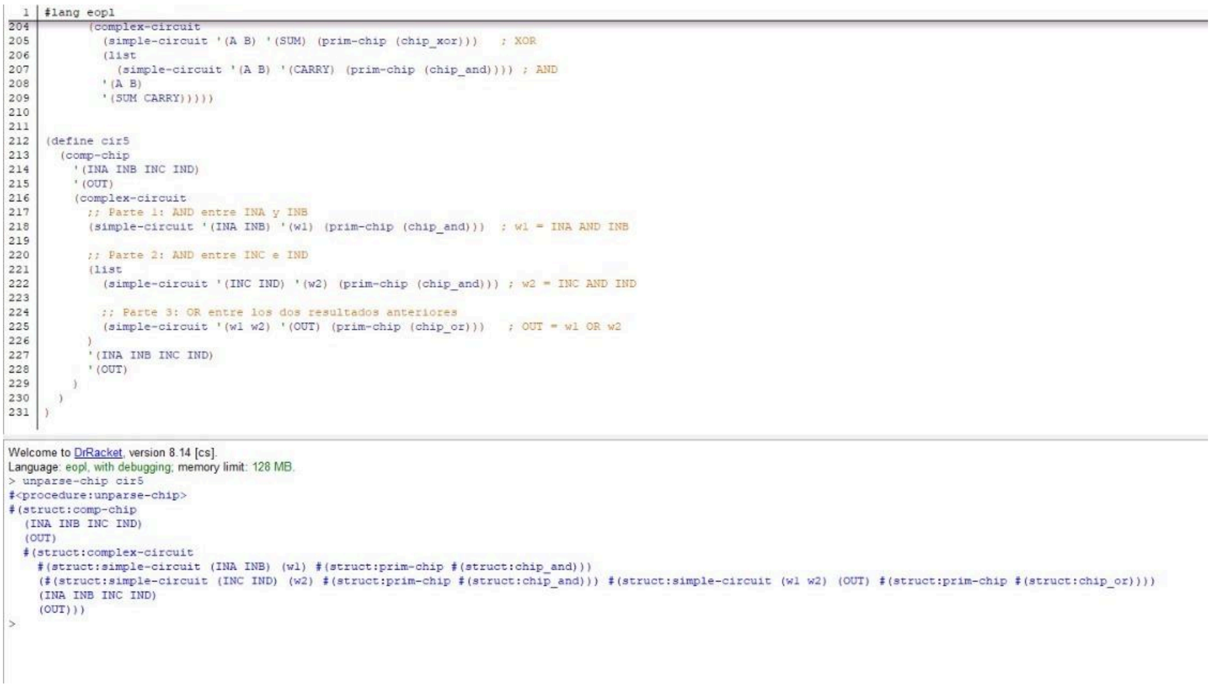
Unparse ejemplo 5



```
#lang eopl
        (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)))))


(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)
    )
  )
)
```

```
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



```
#lang eopl
        (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)))))


(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)
    )
  )
)
```

```
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)))
>
```