Ejercicios de Abstraccción de datos
Fundamentos de Lenguajes Programación

Septiembre 2022

Para los siguientes casos elabore representación basada en listas y en procedimientos. Recuerde que la construcción de los datos y funciones solicitadas deben ser las mismas, lo que cambia es la INTERFAZ.

1. LCD: Lenguaje para programar Circuitos Digitales

El objetivo de este punto es construir un TAD para el lenguaje de programación de circuitos digitales LCD.

Un circuito digital consiste de un conjunto de dispositivos lógicos, llamados chips, interconectados por medio de cables. Todo chip se puede ver como una caja negra, con una funcionalidad especifica. Esta caja cuenta con un conjunto de puertos de entrada y puertos de salida, donde se conectan cables.

Los chips más sencillos son las compuertas lógicas o primitivas *AND,OR,XOR,NOR,NAND y XNOR* tienen dos puertos de entrada y uno de salida. La compuerta *not* tiene un puerto de entrada y uno de salida.

Para construir chips con funcionalidades más complejas se utilizan chips primitivos (compuertas) o chips complejos previamente construidos, los cuales se interconectan.

La funcionalidad de cada chip está definida por la tabla de valores de su tabla de verdad, como en el caso de *and,or* y *not*

x	**y**	**not(x)**	**or(x,y)**	**and(x,y)**
0	0	1	0	0
0	1	1	1	0
1	0	0	1	0
1	1	0	1	1

Tabla 1: Funcionalidades de las compuestas and,or y not

1.1. Implementando un TAD para LCD

Considere la siguiente gramática en formato BNF para el lenguaje:

```
<circuito> := {cable}* {cable}* <chip>
              simple-circuit (in out chip)

            := <circuito> {<circuito>}+ {cable}* {cable}*
            complex-circuit (circ lcircs in out)
```

El circuito es la estructura principal de este TAD, a partir de este construimos la especificación de un circuito simple o compuesto. Los chips son estructuras base para la construcción de circuitos:

```
<chip> := <chip_prim>
        prim-chip(chip-prim)

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

Así mismo, contamos con chips primitivos (por definición) que corresponden a las compuertas conocidas OR, AND, NOT, XOR, NAND, NOR y XNOR. Para que estas puedan ser funcionales deben estar integradas a un circuito simple.

```
<chip prim> := prim_or
               chip-or()

               := prim_and
               chip-and()

               := prim_not
               chip-not()

               := prim_xor
               chip-xor()

               := prim_nand
               chip-nand()

               := prim_nor
               chip-nor()

               := prim_xnor
               chip-xnor()
```

donde <port> , <cable> y <dchip> representan un identificador (de puerto, de cable y de chip respectivamente). Tenga en cuenta que los identificadores son símbolos. A partir de este lenguaje podemos construir elementos digitales como:

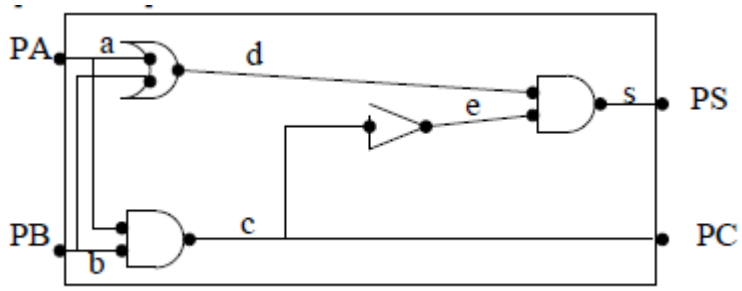

Figura 1: Sumador (Half adder)

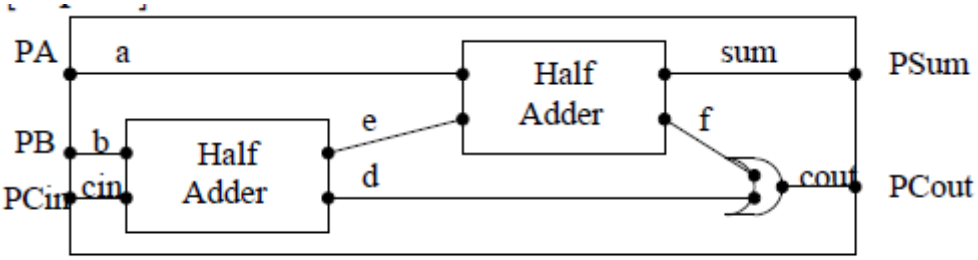

Figura 2: Sumador completo

1.2. Representación basada en listas (30 puntos)

1. **(15 puntos)** Defina los constructores y observadores para este lenguaje utilizando una representación basada en listas. Tome en cuenta que debe construir para cada gramática.

2. **(5 puntos)** Usando las funciones creadas implemente el sumador y el sumador completo de las figuras 1.1 y 2.

3. **(10 puntos)** Implemente:

   a) Defina una función que reciba un circuito y retorne la lista de símbolos con los identificadores de los cables.

   b) Defina una que reciba un circuito y retorne una lista de símbolos con los identificadores de los puertos.

1.3. Representación basada en procedimientos

4. Defina los constructores y observadores para este lenguaje para este lenguaje utilizando una representación basada en procedimientos. Tome en cuenta construir para cada gramática

5. Usando las funciones creadas implemente el sumador y el sumador completo de las figuras 1.1 y 2.

6. Implemente:

   a) Defina una función que reciba un circuito y retorne la lista de símbolos con los identificadores de los cables.

   b) Defina una que reciba un circuito y retorne una lista de símbolos con los identificadores de los puertos.

* Si usted ha realizado bien el proceso de construcción de procedimientos observadores, este punto es copiar y pegar lo realizado en listas.

1.4. Representación usando Datatypes

7. Defina los datos usando **define-datatype**. En este caso serían 3, uno para circuito, otro para chip y el ultimo para chip primitivo.

8. Usando las funciones creadas implemente el sumador y el sumador completo de las figuras 1.1 y 2.

9. El parseo y un-parseo permite pasar de un lenguaje escrito por un programador y su representación en sintaxis abstracta y viceversa. Para esto suponga que **su representación basada en listas** es lo que escribe el programador, implemente:

   a) La función **parser-lcd** que toma una lista y la transforma en un árbol de sintaxis abstracta.

   b) La función **unparser-lcd** que toma un árbol de sintaxis abstracta y lo transforma en la especificación basada en listas.

   En este punto haga al menos 3 ejemplos de funcionamiento de cada una de las funciones.

2. Ejemplo

A continuación se muestra un ejemplo de la construcción de un chip:

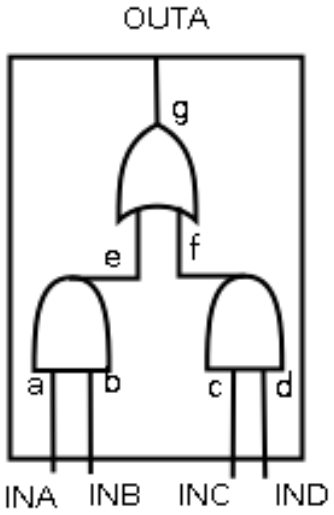

Figura 3: Chip de ejemplo

```
( 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)
    )
)
```

A continuación un circuito complejo

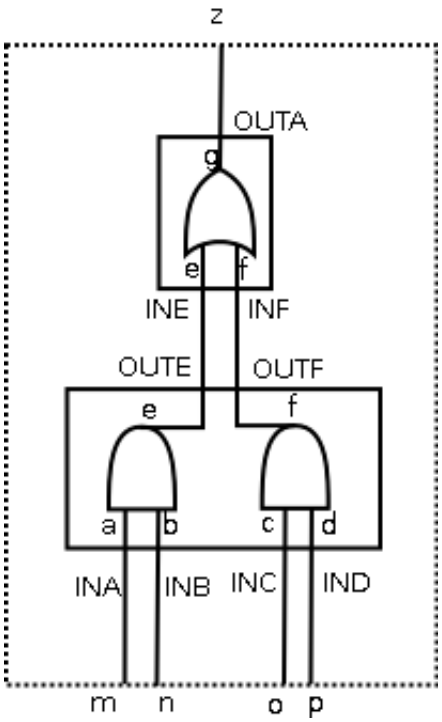

Figura 4: Circuito complejo

```
(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)
)
```