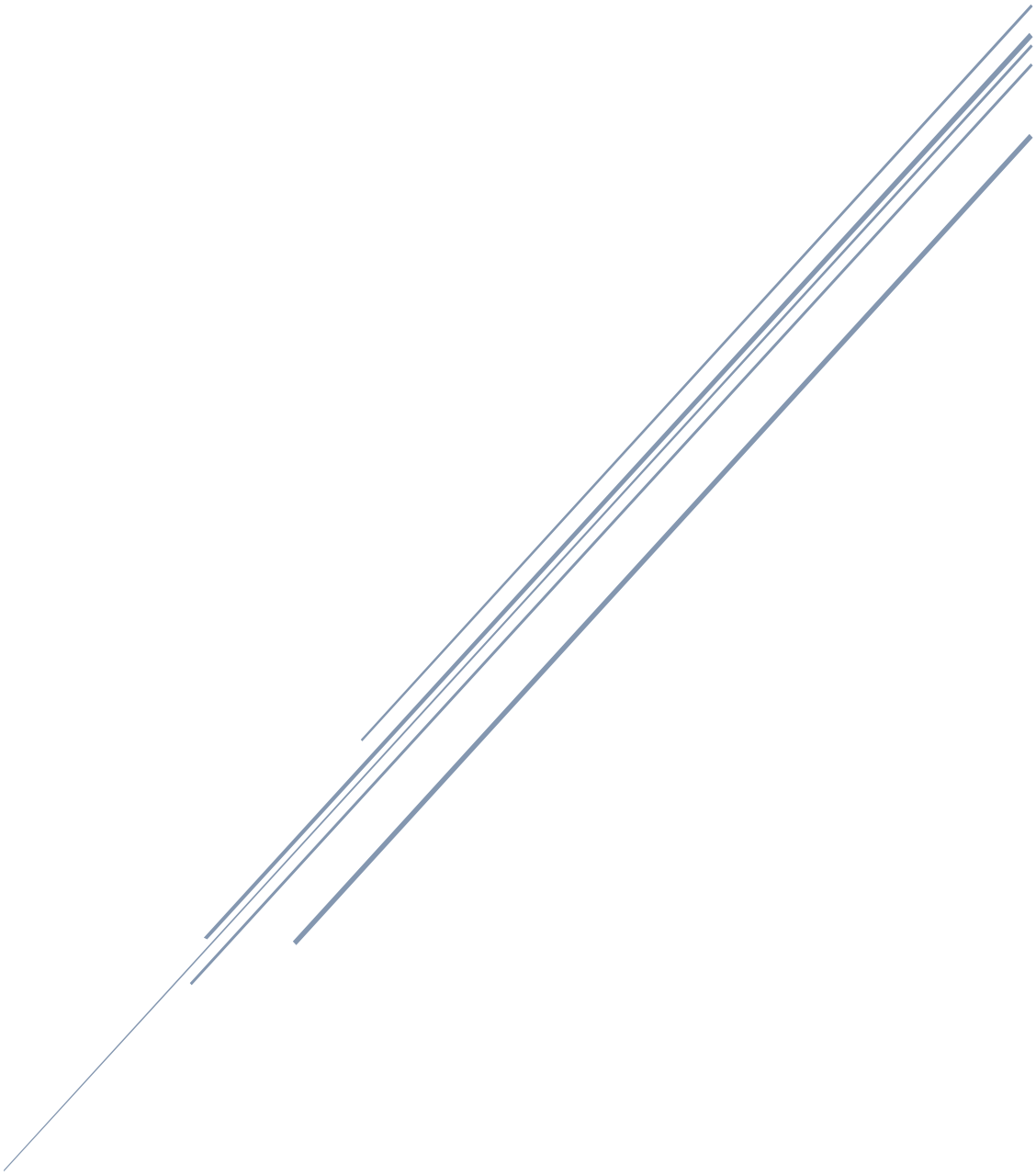
PRACTICA 4

LPM - Laboratorio de programación multiparadigma



MITSS
Sergi Sanz Carreres

Índice:

3.1. VECTORES....2

1.- ¿QUÉ DEVUELVE LA SIGUIENTE DEFINICIÓN ?....2
2.- ¿QUÉ DEVUELVE LA SIGUIENTE DEFINICIÓN ?....3

3.2. ARITMÉTICA....5

1.- ¿QUÉ DEVUELVE LA SIGUIENTE DEFINICIÓN?....5
2.- ¿QUÉ DEVUELVE LA SIGUIENTE DEFINICIÓN?....6
3.- ¿QUÉ DEVUELVE LA SIGUIENTE DEFINICIÓN?....7
4.- ¿QUÉ DEVUELVE LA SIGUIENTE DEFINICIÓN?....8

Antes de empezar en la realización de esta práctica es importante poner en contexto el tema sobre el tema sobre el que vamos a utilizar.

La satisfacción módulo teorias (SMT) generaliza la satisfacción booleana (SAT) al agregar razonamiento de igualdad, aritmética, vectores de bits de tamaño fijo,matrices, cuantificadores y otras teorías útiles de primer orden.

Ambos SAT y SMT se emplean para demostrar la satisfacción de las fórmulas a distintos niveles de satisfacción. A menudo se da el caso de que no solo existen una sino varias soluciones satisfactorias para una fórmula dada y cada una de ellas tiene un coste que se refiere a algún parámetro relacionado con el problema en sí.

Por lo tanto, es posible convertir el problema de satisfacción en uno de optimización al establecer una función objetivo donde no solo se quiere encontrar una solución satisfactoria, sino la óptima con respecto al coste que asignamos a las variables.

Un SMT Solver es una herramienta para decidir la satisfacción (o validez) de las fórmulas en estas teorías. Los SMT Solver son claves para el desarrollo de aplicaciones como la verificación automática, la abstracción de predicados, la generación de casos de prueba y la verificación de modelos acotados sobre dominios infinitos, por mencionar algunos.

Z3 es un SMT Solver propiedad de Microsoft Research y desarrollado en el lenguaje de programación C++ que destaca por ser eficiente con algoritmos especializados para resolver teorías de fondo. Z3 utiliza algoritmos novedosos para la instanciación de cuantificadores y la combinación de teorías siendo uno de los más completos.

Uno puede interactuar con Z3 a través de scripts de SMT-LIB suministrados como un archivo de texto, o usando llamadas a la API desde un lenguaje de programación de alto nivel (para la realización de esta practica vamos a utilizar SMT-LIB v2).

3.1. Vectores

1.- ¿Qué devuelve la siguiente definición ?

```
(declare-const x Int)
(declare-const i Int)
(declare-const a1 (Array Int Int))
(assert (not (= (select a1 i) x)))
(check-sat)
(get-model)
```

En este caso, la definición declarada hace lo siguiente:

- Se declara una constante de tipo entero x e y, al mismo tiempo se declara una constante de tipo *Array.*

- El comando *assert* se encarga de añadir la fórmula a la pila interna de Z3 , donde en este caso, añadirá que no se debe de cumplir que el valor almacenado en el array a1 en la posición i sea igual a x.

- El comando check-sat determina si las fórmulas actuales en la pila Z3 son satisfactorias o no. En caso de ser satisfacibles devolverá sat, mientras que en caso contario devolverá unsat.

- Por otra parte, el comando get-model se usa para cuando el comando check-sat devuelve sat, proporciona una interpretación donde hace que todas las fórmulas en la pila interna de Z3 sean verdaderas.

Al ejecutar la definición obtenemos el siguiente modelo:

```
sat
(model
  (define-fun i () Int
    0)
  (define-fun a1 () (Array Int Int)
    (_ as-array k!0))
  (define-fun x () Int
    2)
  (define-fun k!0 ((x!1 Int)) Int
    (ite (= x!1 0) 1
      1))
)
```

Donde se observa como en este caso *i* ha adquirido el valor *0*, mientras que *x* ha adquirido el valor *2*, de forma que el array *a1* en la posición de *i*, tendrá el valor *1*. Por tanto, se cumple la restricción.

2.- ¿Qué devuelve la siguiente definición?

```
(declare-const x Int)
(declare-const a1 (Array Int Int))
(assert (= (store a1 1 x) a1))
(assert (= (store a1 2 x) a1))
(assert (= (store a1 3 x) a1))
(assert (= (store a1 4 x) a1))
(assert (= (store a1 5 x) a1))
(assert (= (store a1 6 x) a1))
(assert (= (store a1 7 x) a1))
(assert (= (store a1 8 x) a1))
(assert (= (store a1 9 x) a1))
(assert (= (store a1 10 x) a1))
(check-sat)
(get-model)
```

En este caso, la definición declarada hace lo siguiente:

- Se declara una constante de tipo entero x, al mismo tiempo se declara una constante de tipo *Array a1.*

- El comando *assert* se encarga de añadir la fórmula a la pila interna de Z3 , donde en este caso, añadirá que en cada posición del array desde 1 hasta 10 el valor que tenga x (el cual desconocemos) sea el valor almacenado.

- El comando check-sat determina si las fórmulas actuales en la pila Z3 son satisfactorias o no. En caso de ser satisfacibles devolverá sat, mientras que en caso contario devolverá unsat.

- Por otra parte, el comando get-model se usa para cuando el comando check-sat devuelve sat, proporciona una interpretación donde hace que todas las fórmulas en la pila interna de Z3 sean verdaderas.

Al ejecutar la definición obtenemos el siguiente modelo:

```
sat
(model
  (define-fun a1 () (Array Int Int)
    (_ as-array k!0))
  (define-fun x () Int
    11)
  (define-fun k!0 ((x!1 Int)) Int
    (ite (= x!1 3) 11
    (ite (= x!1 5) 11
    (ite (= x!1 4) 11
    (ite (= x!1 2) 11
    (ite (= x!1 8) 11
    (ite (= x!1 10) 11
    (ite (= x!1 1) 11
    (ite (= x!1 9) 11
    (ite (= x!1 7) 11
    (ite (= x!1 6) 11
      0))))))))))
)
```

Donde se observa como en este caso *x* ha adquirido el valor *11*, de forma que todos los elementos del array en las posiciones 1 a la 10 tendrán el valor *11*. Por tanto, se cumple la restricción.

3.2. Aritmética

1.- ¿Qué devuelve la siguiente definición?

```
(declare-const x Int)
(declare-const y Int)
(assert (<= (+ x y) 1 ))
(check-sat)
(get-model)
```

En este caso, la definición declarada hace lo siguiente:

- Se declara una constante de tipo entero x e y.
- El comando *assert* se encarga de añadir la fórmula a la pila interna de Z3 , donde en este caso, comprobará si existe algún valor en el que *(x+y) <= 1.*
- El comando check-sat determina si las fórmulas actuales en la pila Z3 son satisfactorias o no. En caso de ser satisfacibles devolverá sat, mientras que en caso contario devolverá unsat.
- Por otra parte, el comando get-model se usa para cuando el comando check-sat devuelve sat, proporciona una interpretación donde hace que todas las fórmulas en la pila interna de Z3 sean verdaderas.

Al ejecutar la definición obtenemos el siguiente modelo:

```
sat
(model
  (define-fun y () Int
    0)
  (define-fun x () Int
    1)
)
```

Donde se observa como en este caso *x* ha adquirido el valor *1*, mientras que *y* ha adquirido el valor *0*, cumpliendo la restricción de que *(x+y) <=1* .

2.- ¿Qué devuelve la siguiente definición?

```
(declare-const y Int)
(assert (<= (+ 0 0) (+ y 1) ))
(check-sat)
(get-model)
```

En este caso, la definición declarada hace lo siguiente:

- Se declara una constante de tipo entero y.
- El comando *assert* se encarga de añadir la fórmula a la pila interna de Z3, donde en este caso, comprobará si existe algún valor en el que se cumpla *(0+0) <= (y+1).*
- El comando check-sat determina si las fórmulas actuales en la pila Z3 son satisfactorias o no. En caso de ser satisfacibles devolverá sat, mientras que en caso contario devolverá unsat.
- Por otra parte, el comando get-model se usa para cuando el comando check-sat devuelve sat, proporciona una interpretación donde hace que todas las fórmulas en la pila interna de Z3 sean verdaderas.

Al ejecutar la definición obtenemos el siguiente modelo:

```
sat
(model
  (define-fun y () Int
    (- 1))
)
```

Donde se observa como en este caso *y* ha adquirido el valor *-1*, cumpliendo la restricción de que *(0+0) <= (y+1).*

3.- ¿Qué devuelve la siguiente definición?

```
(declare-const x Int)
(declare-const y Int)
(assert (<= (+ x y) (+ x 1) ))
(check-sat)
(get-model)
```

En este caso, la definición declarada hace lo siguiente:

- Se declara una constante de tipo entero *x* e *y*.
- El comando *assert* se encarga de añadir la fórmula a la pila interna de Z3 , donde en este caso, comprobará si existe algún valor en el que se cumpla *(x+y) <= (x+1).*
- El comando check-sat determina si las fórmulas actuales en la pila Z3 son satisfactorias o no. En caso de ser satisfacibles devolverá sat, mientras que en caso contario devolverá unsat.
- Por otra parte, el comando get-model se usa para cuando el comando check-sat devuelve sat, proporciona una interpretación donde hace que todas las fórmulas en la pila interna de Z3 sean verdaderas.

Al ejecutar la definición obtenemos el siguiente modelo:

```
sat
(model
  (define-fun y () Int
    1)
)
```

Donde se observa como en este caso *y* ha adquirido el valor *1*, cumpliendo la restricción de que *(x+y) <= (x+1).* Se hace destacar el hecho de que x no tenga ningún valor, ya que como se encuentra en ambos lados de la condición no es influyente.

4.- ¿Qué devuelve la siguiente definición?

```
(declare-const x Int)
(declare-const y Int)
(assert (= (+ x y) 1))
(assert (<= x y))
(check-sat)
(get-model)
```

En este caso, la definición declarada hace lo siguiente:

- Se declara una constante de tipo entero x e y.
- El comando *assert* se encarga de añadir la fórmula a la pila interna de Z3, donde en este caso, comprobará si existe algún valor en el que se cumpla *(x+y) = 1* y *x <= y*.
- El comando check-sat determina si las fórmulas actuales en la pila Z3 son satisfactorias o no. En caso de ser satisfacibles devolverá sat, mientras que en caso contario devolverá unsat.
- Por otra parte, el comando get-model se usa para cuando el comando check-sat devuelve sat, proporciona una interpretación donde hace que todas las fórmulas en la pila interna de Z3 sean verdaderas.

Al ejecutar la definición obtenemos el siguiente modelo:

```
sat
(model
  (define-fun y () Int
    1)
  (define-fun x () Int
    0)
)
```

Donde se observa como en este caso *y* ha adquirido el valor *1*, mientras que *x* ha adquirido el valor *0*, cumpliendo la restricción de que *(x+y) = 1* y *x <= y*.