LENGUAJES, TECNOLOGÍAS Y PARADIGMAS
DE PROGRAMACIÓN

TEMA 5:

TECNOLOGÍAS DE SOPORTE

(EJERCICIOS)

Salvador Lucas, Javier Piris, María José Ramírez,
María José Vicent, Germán Vidal y
Alicia Villanueva

1. Indica cuál de las siguientes afirmaciones es correcta

   [A] El *software testing* puede garantizar la ausencia de errores de un programa.

   [X] La tarea de diseñar casos de prueba de forma manual es la fase que consume más tiempo.

   [C] Si ejecutamos el conjunto de casos de prueba asociado a un programa y no se producen errores, podemos decir que el programa no tiene errores.

   [D] No existen herramientas que puedan servir de soporte al *software testing*

2. Indica cuál de las siguientes afirmaciones es **cierta**

   [A] El análisis estático se hace en tiempo de ejecución.

   [B] El *software testing* se hace en tiempo de compilación.

   [X] El análisis estático es una técnica para predecir en tiempo de compilación el comportamiento dinámico (en tiempo de ejecución) del programa.

   [D] El análisis estático, para ser correcto, tiene que estar basado en la sintaxis de los lenguajes.

3. Indica qué opción es **CIERTA** sobre la ejecución simbólica:

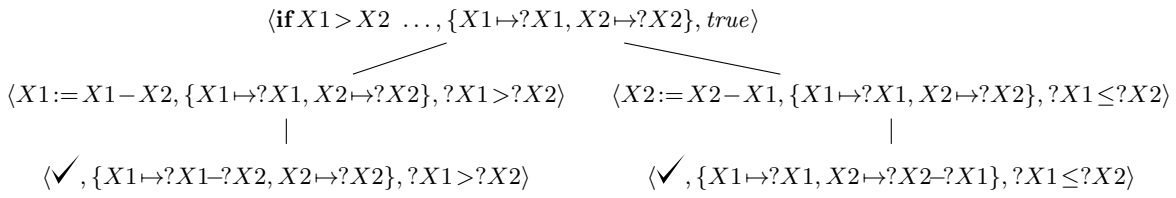
   [A] Recibe como entrada datos concretos.

   [B] Siempre devuelve como salida el resultado concreto de la ejecución.

   [X] Recibe como entrada valores simbólicos.

   [D] Devuelve como salida el resultado concreto de la ejecución sólo en el caso de que ésta termine.

4. Dado el siguiente árbol de ejecución simbólica de un programa cuyos argumentos de entrada son `X1` y `X2`



indica cuál podría ser un conjunto de casos de prueba (expresados como tuplas) asociados al árbol siguiendo la estrategia vista en clase:

X `{(X1=23,X2=12),(X1=22,X2=22)}`

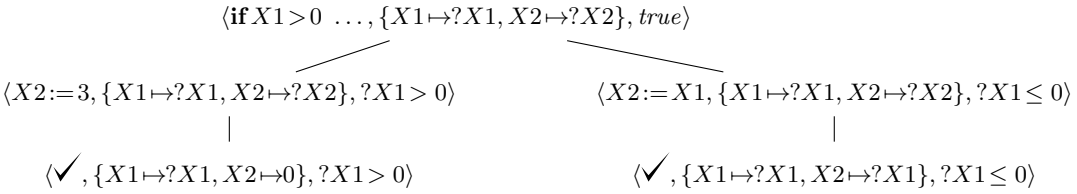
B `{(X1=10,X2=12),(X1=22,X2=22)}`

C `{(X1=23,X2=12),(X1=22,X2=12),(X1=22,X2=22)}`

D `{(X1=12,X2=12)}`

NOTA: Los casos de prueba correspondientes a un árbol de ejecución simbólica es el conjunto *más pequeño* que cubra todas las ramas del árbol.
Por este motivo, la opción C **no** es correcta.

5. Dado el siguiente árbol de ejecución simbólica correspondiente a un programa cuyos argumentos de entrada son `X1` y `X2`



indica cuál podría ser el conjunto de casos de prueba (expresados como tuplas) asociados al árbol siguiendo la estrategia vista en clase:

A `{(X1=1),(X1=0)}`

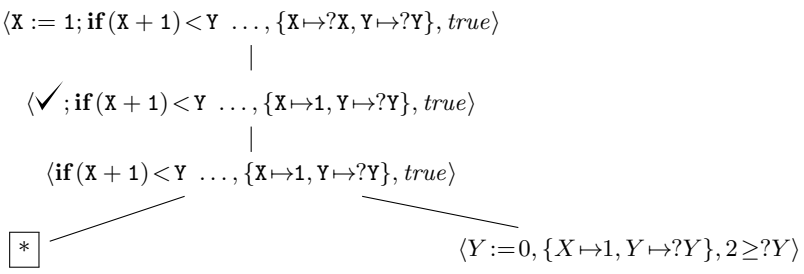
B `{(X1=10,X2=12),(X1=-1,X2=22),(X1=-1,X2=0)}`

X `{(X1=1,X2=12),(X1=0,X2=12)}`

D `{(X1=0,X2=0)}`

6. Dado el siguiente fragmento de código y de árbol de ejecución simbólica

```
X := 1;
if (X+1 < Y) then X:=0
             else Y:=0
```



cuál de las siguientes es la configuración a la que se llegaría en el nodo marcado con [*]?

[X] $\langle \mathtt{X} := 0, \{\mathtt{X}\mapsto 1, \mathtt{Y}\mapsto ?\mathtt{Y}\}, 2 < ?\mathtt{Y}\rangle$

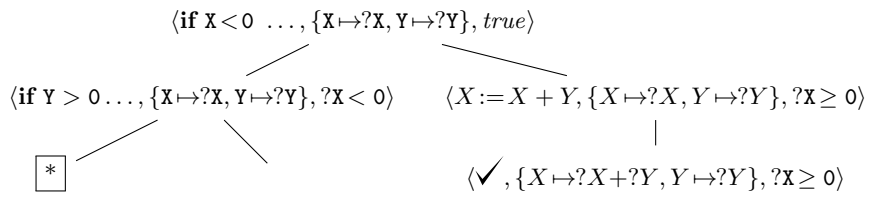
[B] $\langle \mathtt{X} := 0, \{\mathtt{X}\mapsto 1, \mathtt{Y}\mapsto ?\mathtt{Y}\}, \mathit{true}\rangle$

[C] $\langle \mathtt{X} := 0, \{\mathtt{X}\mapsto 1, \mathtt{Y}\mapsto ?\mathtt{Y}\}, ?\mathtt{X} + 1 < ?\mathtt{Y}\rangle$

[D] $\langle \mathtt{X} := 0, \{\mathtt{X}\mapsto 1, \mathtt{Y}\mapsto ?\mathtt{Y}\}, \mathtt{X} + 1 < \mathtt{Y}\rangle$

7. Dado el siguiente fragmento de código y de árbol de ejecución simbólica asociado

```
if (X < 0) then if (Y > 0) then X:=Y-X
                           else Y:=-X
           else X:=X+Y
```



cuál de las siguientes es la configuración a la que se llegaría en el nodo marcado con [*]?

[A] $\langle \mathtt{X} := \mathtt{Y} - \mathtt{X}, \{\mathtt{X}\mapsto ?\mathtt{X}, \mathtt{Y}\mapsto ?\mathtt{Y}\}, ?\mathtt{X} < 0\rangle$

[B] $\langle \mathtt{X} := \mathtt{Y} - \mathtt{X}, \{\mathtt{X}\mapsto ?\mathtt{X}, \mathtt{Y}\mapsto ?\mathtt{Y}\}, ?\mathtt{Y} > 0\rangle$

[X] $\langle \mathtt{X} := \mathtt{Y} - \mathtt{X}, \{\mathtt{X}\mapsto ?\mathtt{X}, \mathtt{Y}\mapsto ?\mathtt{Y}\}, ?\mathtt{X} < 0 \wedge ?\mathtt{Y} > 0\rangle$

[D] $\langle \mathtt{X} := \mathtt{Y} - \mathtt{X}, \{\mathtt{X}\mapsto ?\mathtt{Y} - ?\mathtt{X}, \mathtt{Y}\mapsto ?\mathtt{Y}\}, ?\mathtt{Y} < 0\rangle$

8. Dado el siguiente fragmento de código:

```
void testme (int x, int y) {
           int z = 2*y;
           if (z != x) {
             if (x > 0) z = 5           }
}
```

indica cuál de las siguientes restricciones **NO** se calcularía en un camino de ejecución simbólica del método main:

```
int main() {
           int x = sym_input();
           int y = sym_input();
           testme (x,y);
           return 1;
}
```

A `2*?y = ?x`

B `2*?y != ?x` $\wedge$ `?x <= 0`

X `2*?y != ?x` $\wedge$ `?y > 5`

D `2*?y != ?x` $\wedge$ `?x > 0`

9. Indica cuál de las siguientes afirmaciones es cierta:

X JML es una notación para incluir información adicional en programas Java.

B JML es una herramienta para hacer análisis estático de programas.

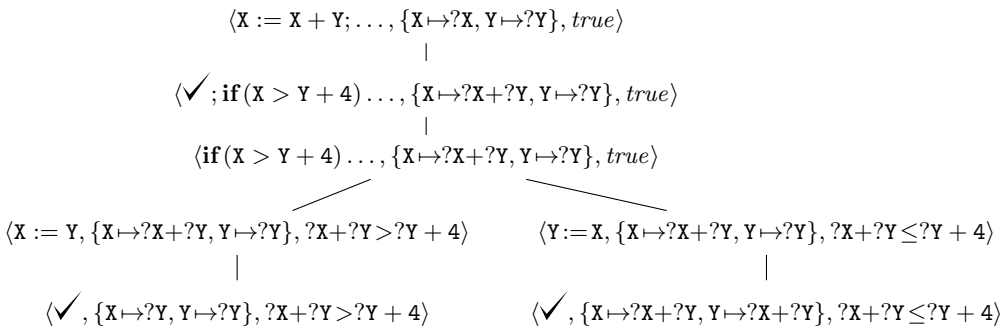
C Dafny usa anotaciones JML para hacer testing.

D En model checking usamos anotaciones JML para especificar la propiedad sobre el comportamiento concurrente del sistema.

CUESTIONES

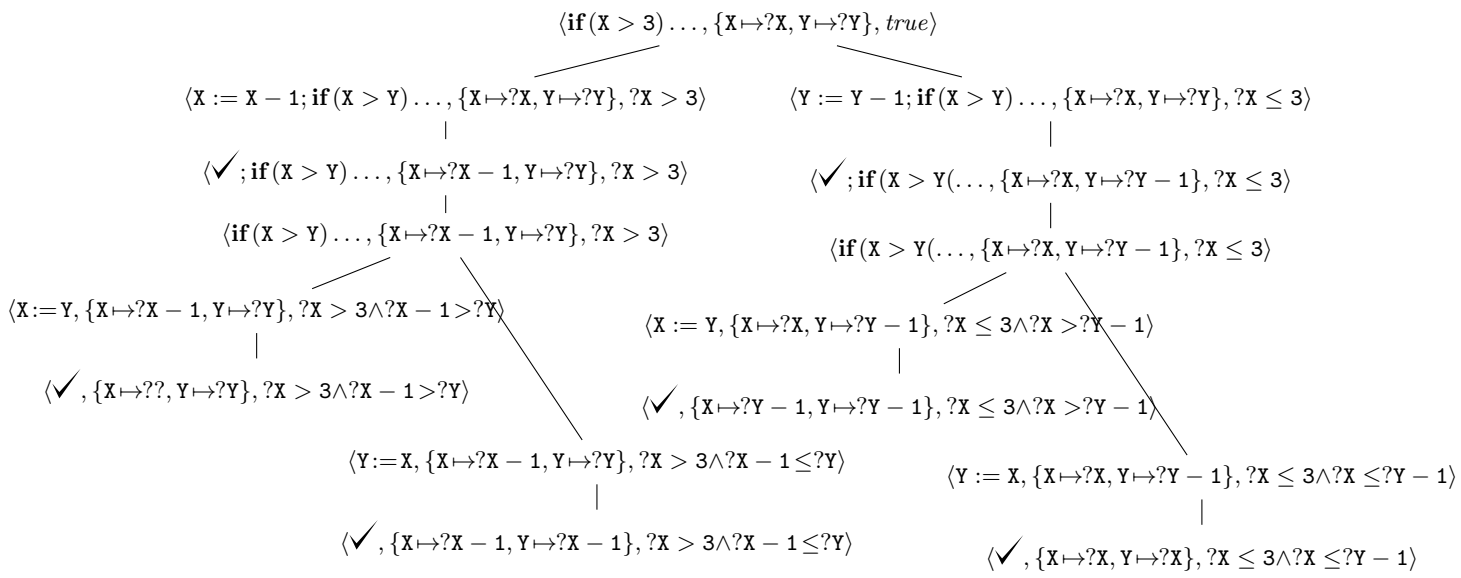
1. Construye el árbol de ejecución simbólica del siguiente fragmento de programa, asumiendo que tanto X como Y son parámetros de entrada:

```
X:=X+Y;
if (X>Y+4) then X:=Y
           else Y:=X
```



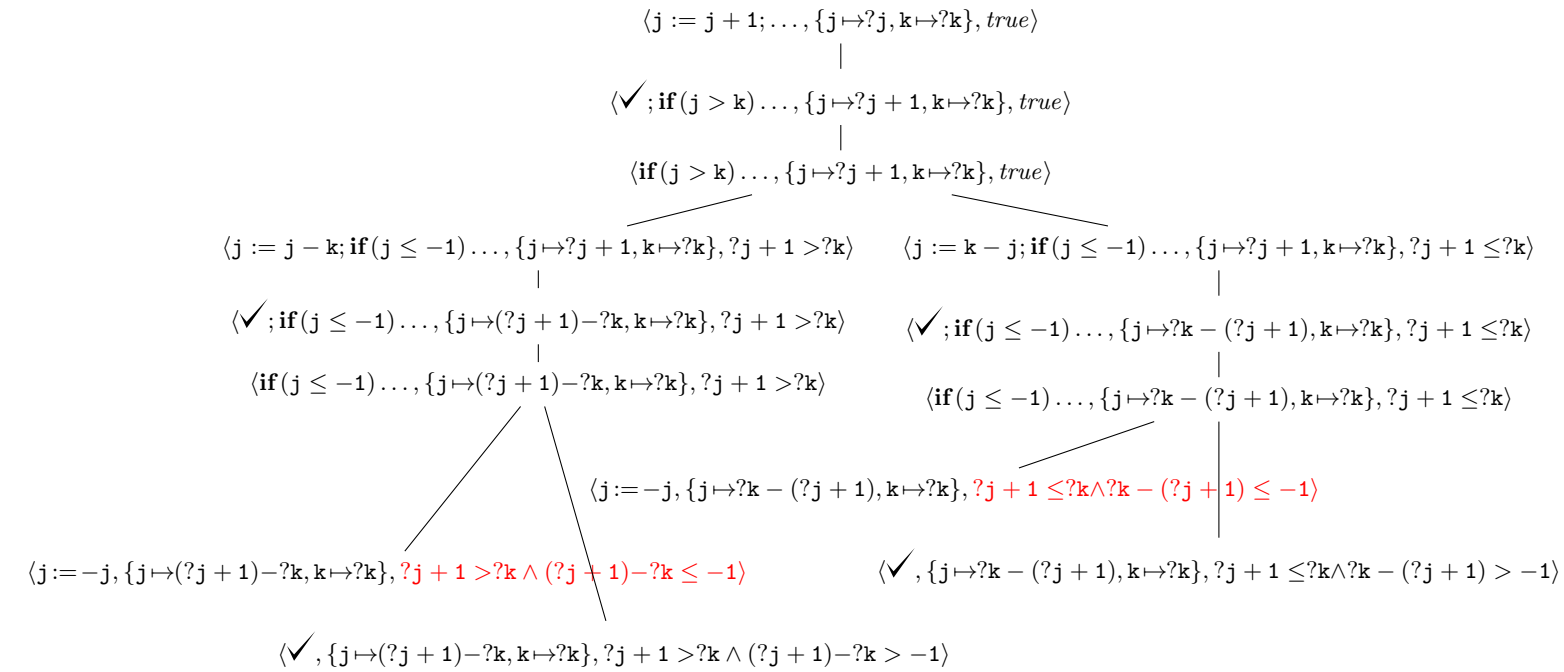
2. Construye el árbol de ejecución simbólica del siguiente fragmento de programa, asumiendo que tanto X como Y son parámetros de entrada:

```
if (X>3) then X:=X-1
         else Y:=Y-1;
if (X>Y) then X:=Y
         else Y:=X
```



3. Construye el árbol de ejecución simbólica del siguiente fragmento de programa, asumiendo que tanto `j` como `k` son parámetros de entrada:

```
j := j+1;
if (j>k) then j:=j-k
         else j:=k-j;
if (j<=-1) then j:=-j
           else skip
```



- Observa el árbol obtenido. ¿Hay ramas inalcanzables? ¿Crees que sería posible simplificar el programa a la vista del resultado?

  Las *path-conditions* señaladas en rojo (las hojas cuya instrucción no es ✓), son insatisfacibles. Es decir, no es posible encontrar valores para `?j` y `?k` que hagan que esas restricciones se satisfagan ya que están en contradicción.
  Esto quiere decir que la rama *then* del segundo condicional nunca va a ejecutarse (la guarda es siempre falsa), por lo que se trata de código muerto y podría eliminarse del programa todo el condicional.