PARALLEL ++

PRÁCTICA 4

REALIZADA POR

- Víctor Madrona Cisneros
- Francisco Javier Mota López
- Rubén Molina Lozano

[Q1] Sigue todos los pasos necesarios para utilizar el model checker de Maude para verificar la exclusión mútua del algoritmo de Dekker escrito en el lenguaje PARALLEL. (Todos: comprobación de que el espacio de búsqueda es finito, comprobaciones de confluencia, terminación, protección de booleanos, definición de proposiciones atómicas, . . .)

Verificación de exclusión mutua:

Comando search:

```
Maude> search in DEKKER : initial =>* {S | [0,crit0 ; R] | [1,crit1 ; P],M} .
search in DEKKER : initial =>* {S | [0,crit0 ; R] | [1,crit1 ; P],M} .

No solution.
states: 152  rewrites: 1090 in 0ms cpu (10ms real) (~ rewrites/second)
Maude>
```

Model checker:

```
reduce in DEKKER-CHECK : modelCheck(initial, 2Crit) .
rewrites: 22 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
```

Ausencia de bloqueos:

```
Maude> search in DEKKER : initial =>! MS:MachineState .
search in DEKKER : initial =>! MS:MachineState .

No solution.
states: 152  rewrites: 1090 in 0ms cpu (5ms real) (~ rewrites/second)
```

```
Maude> rew initial .
rewrite in DEKKER : initial .
```

Al hacer rewrite, este se ejecuta infinitamente, por lo que se puede deducir que las reglas no son terminantes.

[Q2] Define la semántica del lenguaje PARALLEL++. Se creará un fichero parallel++.maude con tantos módulos como se considere necesario, que extenderán los módulos en el parallel.maude.

Ver parallel++.maude

[Q3] Utiliza el comando search para verificar la exclusión mutua y la ausencia de bloqueos para el algoritmo de Dekker en su implementación original usando las definiciones de PARALLEL++. (El código está en el fichero dekker++.maude.)

```
Maude> search in DEKKER++ : initial =>* {S | [0,crit0 ; R] | [1,crit1 ; P],M} .
search in DEKKER++ : initial =>* {S | [0,crit0 ; R] | [1,crit1 ; P],M} .

No solution.
states: 20  rewrites: 258 in 0ms cpu (0ms real) (260606 rewrites/second)
```

```
Maude> search in DEKKER++ : initial =>! MS:MachineState .
search in DEKKER++ : initial =>! MS:MachineState .

No solution.
states: 20  rewrites: 258 in 0ms cpu (0ms real) (~ rewrites/second)
```

[Q4] Modifica la implementación del algoritmo de Dekker en el fichero dekker++.maude (crea una copia en un fichero dekker++-modificado.maude) de forma que tengamos un programa con el que podemos crear las hebras correspondientes. Tendremos un programa main:
op main : -> Program .
eq main = new(0, p(0)) ; new(1, p(1)) .
De forma que el estado inicial de nuestro sistema pueda ser simplemente:
eq initial = { [2, main], [wants-to-enter, false false] [turn, 0] } .

Ver archivo dekker++.maude.

[Q5] Comprueba utilizando el comando search la ausencia de bloqueo y la exclusión mutua de esa versión del algoritmo.

Comprobamos la exclusión mútua:

```
search in DEKKER++ : initial =>* {S | [0,crit0 ; R] | [1,crit1 ; P],M} .
```

```
Maude> search in DEKKER++ : initial =>* {S | [0,crit0 ; R] | [1,crit1 ; P],M} .
search in DEKKER++ : initial =>* {S | [0,crit0 ; R] | [1,crit1 ; P],M} .

No solution.
states: 1  rewrites: 6 in 0ms cpu (1ms real) (~ rewrites/second)
```

Verificamos la ausencia de bloqueo:

```
search in DEKKER++ : initial =>! MS:MachineState .
```

```
Maude> search in DEKKER++ : initial =>! MS:MachineState .
search in DEKKER++ : initial =>! MS:MachineState .

No solution.
states: 1  rewrites: 6 in 0ms cpu (1ms real) (~ rewrites/second)
```

[Q6] Escribe el algoritmo de la panadería de Lamport (el original) utilizando el lenguaje PARALLEL++ y utiliza su semántica de reescritura para comprobar la ausencia de bloqueo y la exclusión mutua.

Comprobamos la exclusión mútua:

```
search [1,10] initial =>* {S:Soup | [N1:Nat, crit ; P1:Program] | [N2:Nat,
crit ; P2:Program], M:Memory} .
```

```
Maude> search [1,10] initial =>* {S:Soup | [N1:Nat, crit ; P1:Program] | [N2:Nat, crit ; P2:Program], M:Memory} .
search [1, 10] in BAKERY++3 : initial =>* {S:Soup | [N1:Nat,crit ; P1:Program] | [N2:Nat,crit ; P2:Program],
    M:Memory} .

No solution.
states: 1  rewrites: 16 in 0ms cpu (1ms real) (~ rewrites/second)
```

Comprobamos la ausencia de bloqueo:

```
search in BAKERY++3 : initial =>! MS:MachineState .
```

```
Maude> search in BAKERY++3 : initial =>! MS:MachineState .
search in BAKERY++3 : initial =>! MS:MachineState .

No solution.
states: 1  rewrites: 16 in 0ms cpu (1ms real) (~ rewrites/second)
```

[Q7] Utiliza el model checker para probar la exclusión mutua y la viveza débil. Al tener un espacio de búsqueda infinito necesitaremos realizar una abstracción