Práctica 3: Bakery

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

[Q1] Analiza la confluencia, terminación y coherencia del sistema definido. (Como siempre, realizamos el análisis de terminación y confluencia por separado para la parte ecuacional y la de reescritura).

La confluencia es una propiedad que garantiza que cualquier término se puede reescribir a cualquier otro término de manera única. Estas reglas son deterministas y no hay ambigüedad en la forma en que se aplican. Por lo tanto, se puede afirmar que el sistema de reescritura es confluyente.

La terminación es una propiedad que garantiza que cualquier proceso de reescritura termina después de un número finito de pasos. En este caso, nuestro sistema no es terminante ya que no se ve reducido en ninguna regla de reescritura. Por lo tanto, se puede afirmar que el sistema de reescritura es terminante.

La coherencia es una propiedad que garantiza que las definiciones de los símbolos en el sistema de reescritura son consistentes y no conducen a contradicciones. En este caso, las definiciones de los símbolos son coherentes y no hay contradicciones en las reglas de reescritura. Por lo tanto, se puede afirmar que el sistema de reescritura es coherente.

[Q2] ¿Es el espacio de búsqueda alcanzable a partir de estados definidos con el operador initial finito? Utiliza el comando search para comprobar la exclusión mutua del sistema con 5 procesos.

```
Maude> search [1,500] in BAKERY : initial(5) =>* [[ < O:Oid : BProcess | mode : crit, number : N:Nat > < O':Oid : BProcess | mod
e : crit, number  : M:Nat > C:Configuration ]] .
search [1, 500] in BAKERY : initial(5) =>* [[C:Configuration < O : BProcess | mode : crit, number : N > < O':Oid :
    BProcess | mode : crit, number : M >]] .

No solution.
states: 107977  rewrites: 214527 in 2656ms cpu (3138ms real) (80763 rewrites/second)
```

El espacio no es finito ya que siempre se van incrementando los tickets de cada proceso, y el next y el last del dispensador, dando lugar a nuevos estados.
El comando search no termina nunca, ya que el espacio de estados es infinito, pero podemos observar también que no va devolviendo ninguna solución porque se va cumpliendo la exclusión mutua, aunque sin la abstracción no podríamos estar seguros aún de que siempre se vaya a cumplir la propiedad.

[Q3] Utiliza el comando search para comprobar si hay estados de bloqueo.

```
Maude> search[1,500] in BAKERY : initial(5) =>! GB:GBState .
search [1, 500] in BAKERY : initial(5) =>! GB:GBState .

No solution.
states: 108178  rewrites: 214792 in 2640ms cpu (2902ms real) (81341 rewrites/second)
```

[Q4] Justifica la validez de la abstracción (protección de los booleanos, confluencia y terminación de la parte ecuacional, coherencia de ecuaciones y reglas).

Al incluir el módulo NAT, estamos incluyendo también el módulo BOOL.

Al igual que antes, existe confluencia, pues se pueden aplicar a términos distintos

Protección de los booleanos
En todos los módulos se usan los booleanos de forma protegida (protecting), ya que tenemos acceso al módulo BOOL porque protegemos el módulo NAT, que a su vez protege el módulo BOOL .

Confluencia y terminación de la parte ecuacional
Las ecuaciones son confluentes porque las nuevas (la de abstracción y las de decremento) solo se pueden aplicar a términos o subtérminos distintos.

Las ecuaciones son terminantes, las de decremento siempre llegan a la ecuación del caso base porque se va aplicando de forma recursiva sobre la configuración inicial entera y cada vez sobre una configuración con un elemento menos hasta llegar a none, donde se aplicaría la ecuación para el caso base. La ecuación de la abstracción, al utilizar decremento, que ya sabemos que es terminante, y al ir decrementando de 1 en 1, siempre acaba llegando a un término que no cumple la condición para que no pueda aplicarse más.

Coherencia
Las ecuaciones son coherentes con las reglas. Las partes izquierdas de las reglas están en forma normal y, tal y como están planteadas las reescrituras, es equivalente aplicar las reglas (ya sean wait, enter o exit) y después aplicar la ecuación de la abstracción para disminuir los valores de los números del dispensador y de los procesos que no están en sleep que disminuir primero los valores y luego aplicar cualquiera de las reglas.

[Q5] En el módulo ABSTRACT-BAKERY, ¿es finito el espacio de búsqueda alcanzable a partir de estados definidos con el operador initial?

Con la abstracción se puede reducir el espacio de estados infinito a un espacio finito al reducir los estados infinitos del módulo BAKERY a un conjunto finito con la nueva ecuación.

[Q6] Define, en un módulo ABSTRACT-BAKERY-PREDS proposiciones atómicas
op mode : Nat Mode -> Prop .
op 2-crit : -> Prop .
de forma que mode(N, M) se satisfaga si el proceso N está en modo M y 2-crit si hay dos procesos cualesquiera en la sección crítica.

[Q7] Comprueba que la abstracción es consistente con las proposiciones atómicas. La abstracción añade la ecuación

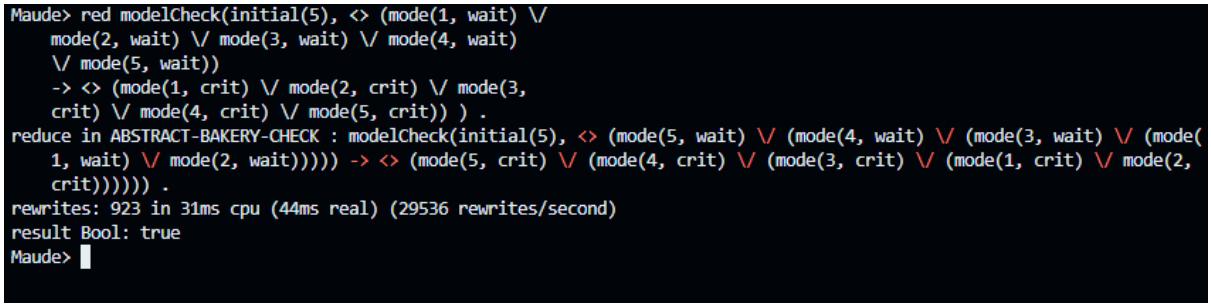
```
mod ABSTRACT-BAKERY is
    extending BAKERY .
    var BPid : Oid .
    var N : Nat .
    var M : Mode .
    ceq < BPid : BProcess | mode : M, number : s(N) > = < BPid : BProcess | mode : M, number : N > if M =/= sleep .
endm
```

La abstracción extiende el módulo Bakery, cuyos términos son de la clase BProcess y Dispenser. La ecuación de la abstracción coge como parámetro un objeto de la clase BProcess y devuelve también un objeto de la misma clase decrementando en uno el número. Esto no provoca ni junk ni confusion, ya que no se están agregando nuevos elementos ni se están identificando elementos previamente distintos.

[Q8] Utiliza el comprobador de modelos de Maude para comprobar la exclusión mutua del sistema con 5 procesos.

```
Maude> red modelCheck(initial(5),
      ( ([] ~(mode(1, crit) /\ mode(2, crit)))  /\
      ([] ~(mode(1, crit) /\ mode(3, crit)))  /\
      ([] ~(mode(1, crit) /\ mode(4, crit)))  /\
      ([] ~(mode(1, crit) /\ mode(5, crit)))  /\
      ([] ~(mode(2, crit) /\ mode(3, crit)))  /\
      ([] ~(mode(2, crit) /\ mode(4, crit)))  /\
      ([] ~(mode(2, crit) /\ mode(5, crit)))  /\
      ([] ~(mode(3, crit) /\ mode(4, crit)))  /\
      ([] ~(mode(3, crit) /\ mode(5, crit)))  /\
      ([] ~(mode(4, crit) /\ mode(5, crit))) )) .
reduce in ABSTRACT-BAKERY-CHECK : modelCheck(initial(5), []~ (mode(4, crit) /\ mode(5, crit)) /\ ([]~ (mode(3, crit) /\
    mode(5, crit)) /\ ([]~ (mode(3, crit) /\ mode(4, crit)) /\ ([]~ (mode(2, crit) /\ mode(5, crit)) /\ ([]~ (mode(2, crit) /\
    mode(4, crit)) /\ ([]~ (mode(2, crit) /\ mode(3, crit)) /\ ([]~ (mode(1, crit) /\ mode(5, crit)) /\ ([]~ (mode(1, crit) /\
    mode(4, crit)) /\ ([]~ (mode(1, crit) /\ mode(2, crit)) /\ []~ (mode(1, crit) /\ mode(3, crit))))))))))) .
rewrites: 29120 in 375ms cpu (412ms real) (77653 rewrites/second)
result Bool: true
Maude>
```

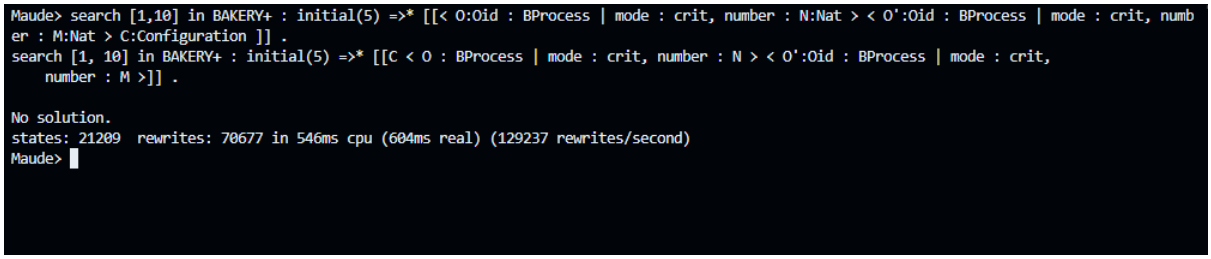
[Q9] Utiliza el comprobador de modelos de Maude para comprobar la propiedad de viveza débil y fuerte.



[Q10] Analiza la confluencia, terminación y coherencia del sistema definido.

El modelo mantiene la confluencia, sigue sin ser terminante y mantiene la coherencia (todo esto está explicado en Q1).

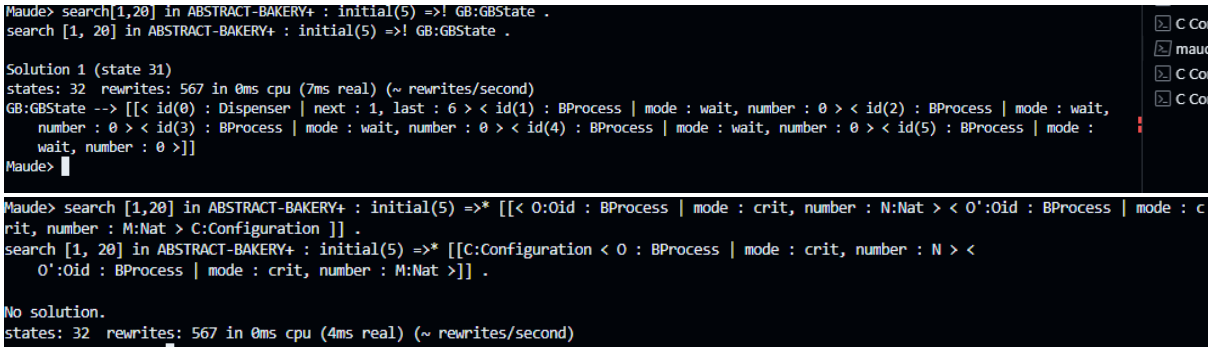
[Q11] ¿Es finito el espacio de búsqueda alcanzable a partir de estados definidos con el operador initial? Utiliza el comando search para comprobar la exclusión mutua del sistema con 5 procesos.



[Q12] La abstracción proporcionada por el módulo ABSTRACT-BAKERY no es suficiente para este sistema modificado, ¿por qué? Especifica una abstracción válida para este nuevo sistema en una módulo ABSTRACT-BAKERY+.

Al incorporar las nuevas reglas, es posible que surjan vacíos en la cola. Por lo tanto, simplemente reducir en una unidad el número de cada proceso ya no es suficiente, dado que pueden ingresar y salir procesos de manera indefinida. Esto podría hacer que el último proceso seleccionado (y, por ende, su número asociado) aumente indefinidamente.
Para abordar este problema, podemos implementar una solución donde, si un proceso detecta un espacio vacío delante de él en la cola, lo ocupa al reducir su propio número. Esto evita la formación de huecos en la cola y completa la abstracción de este sistema modificado.

[Q13] Utiliza la abstracción anterior para comprobar la no existencia de bloqueos y la exclusión mutua utilizando el comando search.



Podemos garantizar que en ningún momento acceden dos procesos a la sección crítica

(modo “crit) y que, por tanto, se cumple la exclusión mutua

[Q14] Utiliza el comprobador de modelos de Maude para comprobar la exclusión mutua del sistema con 5 procesos.

```
Maude> red modelCheck(initial(5),
      ( ([] ~(mode(1, crit) /\ mode(2, crit)))  /\
      ([] ~(mode(1, crit) /\ mode(3, crit)))  /\
      ([] ~(mode(1, crit) /\ mode(4, crit)))  /\
      ([] ~(mode(1, crit) /\ mode(5, crit)))  /\
      ([] ~(mode(2, crit) /\ mode(3, crit)))  /\
      ([] ~(mode(2, crit) /\ mode(4, crit)))  /\
      ([] ~(mode(2, crit) /\ mode(5, crit)))  /\
      ([] ~(mode(3, crit) /\ mode(4, crit)))  /\
      ([] ~(mode(3, crit) /\ mode(5, crit)))  /\
      ([] ~(mode(4, crit) /\ mode(5, crit))) )) .
reduce in ABSTRACT-BAKERY-CHECK+ : modelCheck(initial(5), []~ (mode(4, crit) /\ mode(5, crit)) /\ ([]~ (mode(3, crit) /\ mode(
   5, crit)) /\ ([]~ (mode(3, crit) /\ mode(4, crit)) /\ ([]~ (mode(2, crit) /\ mode(5, crit)) /\ ([]~ (mode(2, crit) /\ mode(4, crit)) /\
   ([]~ (mode(2, crit) /\ mode(3, crit)) /\ ([]~ (mode(1, crit) /\ mode(5, crit)) /\ ([]~ (mode(1, crit) /\ mode(4, crit)) /\ ([]~ (mode(1,
   crit) /\ mode(2, crit)) /\ []~ (mode(1, crit) /\ mode(3, crit))))))))))) .
rewrites: 29120 in 375ms cpu (387ms real) (77653 rewrites/second)
result Bool: true
```

Se cumple la propiedad de exclusión mutua.

[Q15] Utiliza el comprobador de modelos de Maude para comprobar la propiedad de viveza débil y fuerte.

Cumple la propiedad de viveza débil.

```
Maude> red modelCheck(initial(5), [] (mode(1, wait) \/
         mode(2, wait) \/ mode(3, wait) \/ mode(4, wait)
         \/ mode(5, wait))
         -> <> (mode(1, crit) \/ mode(2, crit) \/ mode(3,
         crit) \/ mode(4, crit) \/ mode(5, crit)) ) .
reduce in ABSTRACT-BAKERY-CHECK+ : modelCheck(initial(5), [](mode(5, wait) \/ (mode(4, wait) \/ (mode(3, wait) \/ (mode(1, wait) \/
    mode(2, wait))))) -> <> (mode(5, crit) \/ (mode(4, crit) \/ (mode(3, crit) \/ (mode(1, crit) \/ mode(2, crit)))))) .
rewrites: 346 in 15ms cpu (60ms real) (22144 rewrites/second)
result Bool: true
```

Cumple la propiedad de viveza fuerte.

```
Maude> red modelCheck(initial(5), []<> (mode(1, wait) \/
        mode(2, wait) \/ mode(3, wait) \/ mode(4, wait)
        \/ mode(5, wait))
        -> []<> (mode(1, crit) \/ mode(2, crit) \/ mode(3,
        crit) \/ mode(4, crit) \/ mode(5, crit)) ) .
reduce in ABSTRACT-BAKERY-CHECK+ : modelCheck(initial(5), []<> (mode(5, wait) \/ (mode(4, wait) \/ (mode(3, wait) \/ (mode(1, wait)
    \/ mode(2, wait))))) -> []<> (mode(5, crit) \/ (mode(4, crit) \/ (mode(3, crit) \/ (mode(1, crit) \/ mode(2, crit)))))) .
rewrites: 956 in 46ms cpu (58ms real) (20394 rewrites/second)
result Bool: true
Maude>
```