TP Astre – Évaluation des Propriétés CTL

Exercice 1

Le modèle utilisé pour décrire la structure de Kripke est le suivant :

```
MODULE main
VAR state : {E0, E1, E2, E3, E4, E5, E6, E7, E8};
        p : boolean;
        q : boolean;
ASSIGN
        init(state) := E0;
        init(p) := FALSE;
        init(q) := FALSE;
        next(state) := case
                state = E0 : {E2};
                state = E1 : {E2, E4};
                state = E2 : {E3};
                state = E3 : {E5, E0, E1, E3};
                state = E4 : {E3, E5, E6};
                state = E5 : {E3};
                state = E6 : {E7};
                state = E7 : {E8};
                state = E8 : {E8};
        esac;
next(p) := (next(state) = E1) | (next(state) = E3) | (next(state) = E4) |
(next(state) = E6) | (next(state) = E7);
next(q) := (next(state) = E2) | (next(state) = E5) | (next(state) = E8) ;
```

Les propriétés énoncées sont les suivantes avec leur trace d'exécution :

-- specification AG (state = E1 -> EX p = TRUE) is true

-- specification AG ((p = TRUE & EX p = TRUE) -> (((state = E1 | state = E3) | state = E4) | state = E6)) is true

-- specification AG (state = E1 -> (p = TRUE & q = FALSE)) is true

-- specification AG (state = E6 -> E [p = TRUE U q = TRUE]) is true

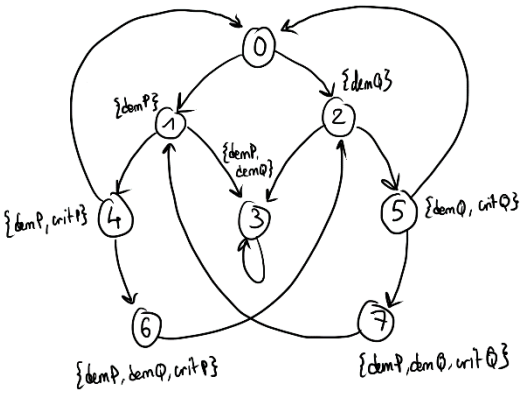
-- specification AG (!(state = E0) -> E [p = TRUE U q = TRUE]) is true

On voit donc que toutes ces propriétés sont bien vérifiées

Léo Check : 3671227
Quentin Micheletti : 3301777

Exercice 2

Voici la modélisation du problème associé à la structure de Kripke correspondante :




```
MODULE main

VAR state : {E0, E1, E2, E3, E4, E5, E6, E7};
        demP   : boolean;
        demQ   : boolean;
        critP  : boolean;
        critQ  : boolean;
ASSIGN
        init(state) := E0;
        init(demP) := FALSE;
        init(demQ) := FALSE;
        init(critP) := FALSE;
        init(critQ) := FALSE;
        next(state) := case
                state = E0 : {E1, E2};
                state = E1 : {E3, E4};
                state = E2 : {E3, E5};
                state = E3 : {E3};
                state = E4 : {E0, E6};
                state = E5 : {E0, E7};
                state = E6 : {E2};
                state = E7 : {E1};
                 esac;
next(demP) := (next(state) = E1) | (next(state) = E3) | (next(state) = E4) |      (next(state) = E6)
| (next(state) = E7);

next(demQ) := (next(state) = E2) | (next(state) = E3) | (next(state) = E5) | (next(state) = E6) |
(next(state) = E6);

next(critP)   := (next(state) = E4) | (next(state) = E6);

next(critQ)   := (next(state) = E5) | (next(state) = E7);
```

Voici la modélisation des propositions CTL données avec leur trace d'exécution :

P1. -- specification AG !(critP & critQ) is true

P2. -- specification AG ((demP -> AF critP) & (demQ -> AF critQ)) is false

-- as demonstrated by the following execution sequence

Trace Description: CTL Counterexample

Trace Type: Counterexample

-> State: 1.1 <-

state = E0

demP = FALSE

demQ = FALSE

critP = FALSE

critQ = FALSE

-> State: 1.2 <-

state = E1

demP = TRUE

P3. -- specification AG ((demP -> EF critP) & (demQ -> EF critQ)) is false

-- as demonstrated by the following execution sequence

Trace Description: CTL Counterexample

Trace Type: Counterexample

-> State: 2.1 <-

state = E0

demP = FALSE

demQ = FALSE

critP = FALSE

critQ = FALSE

-> State: 2.2 <-

state = E1

demP = TRUE

-> State: 2.3 <-

state = E3

demQ = TRUE

P4. -- specification AG (((demP & !demQ) -> A [!critQ U critP])
& ((demQ & !demP) -> A [!critP U critQ])) is false

-- as demonstrated by the following execution sequence

Trace Description: CTL Counterexample

Trace Type: Counterexample

-> State: 3.1 <-

state = E0

demP = FALSE

demQ = FALSE

critP = FALSE

critQ = FALSE

-> State: 3.2 <-

state = E1

demP = TRUE

On observe que la seule des propriété qui est vérifiée par le système est le fait que les deux processus ne peuvent pas entrer en section critique en même temps.

Pour les autres, elles sont toutes fausse, ce qui est normal au vu de l'interblocage possible lorsque les deux processus ont fait leur demande mais aucun n'a eu le temps d'entrer en section critique. Ce qui implique que chaque processus va entrer dans une boucle infinie et ainsi rester bloqué.

Pour empêcher ce cas d'arriver, on rajoute une variable « mutex » qui devra être mise à « VRAI » avant toute demande d'entrée en section critique ce qui permettra de prévenir cet interblocage

Après modification du code des processus, on obtient :

Processus P

Tant que VRAI faire

mutex <- vrai

demP <- VRAI

tant que mutex = VRAI faire

finTq

<< SECTION CRITIQUE >>

demP <- FAUX

mutex = FAUX

FinTq

Processus Q

Tant que VRAI faire

mutex <- vrai

demQ <- VRAI

tant que mutex = VRAI faire

finTq

<< SECTION CRITIQUE >>

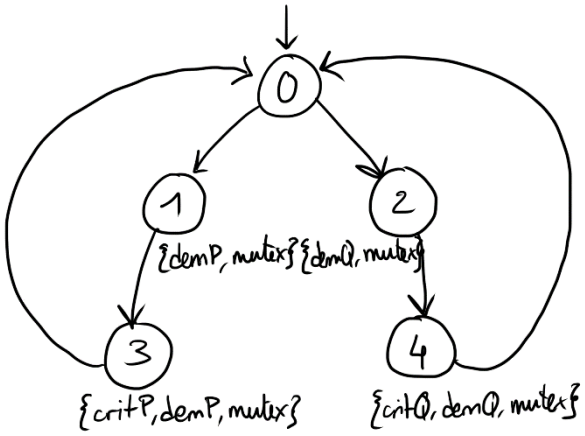
demQ <- FAUX

mutex = FAUX

FinTq

Nous avons alors un nouveau système modélisé par le code suivant et la structure de Kripke associée

```
MODULE main
VAR state : {E0, E1, E2, E3, E4};
        demP   : boolean;
        demQ   : boolean;
        critP  : boolean;
        critQ  : boolean;
        mutex  : boolean;
ASSIGN
 init(state)      := E0;
 init(demP)       := FALSE;
 init(demQ)       := FALSE;
 init(critP)      := FALSE;
 init(critQ)      := FALSE;
 init(mutex)      := FALSE;
 next(state)      := case
                     state = E0 : {E1, E2};
                     state = E1 : {E3};
                     state = E2 : {E4};
                     state = E3 : {E0};
                     state = E4 : {E0};
                      esac;
next(demP)        := (next(state) = E1) | (next(state) = E3);
next(demQ)        := (next(state) = E2) | (next(state) = E4);
next(critP)       := (next(state) = E3);
next(critQ)       := (next(state) = E4);
next(mutex)       := (next(state) = E1) | (next(state) = E2) | (next(state) = E3)
                     | (next(state) = E4);
```



Le choix d'introduire le mutex, nous évite d'avoir à modifier les propriétés écrites précédemment, on peut donc réutiliser les mêmes avec la nouvelle modélisation du système. Nous obtenons alors le résultat suivant :

-- specification AG !(critP & critQ) is true

-- specification AG ((demP -> AF critP) & (demQ -> AF critQ)) is true

-- specification AG ((demP -> EF critP) & (demQ -> EF critQ)) is true

-- specification AG (((demP & !demQ) -> A [!critQ U critP]) & ((demQ & !demP) -> A [!critP U critQ])) is true

Nous avons bien réussi à créer un système dans lequel les propriétés sont vérifiées et qui est bien conforme au système inital