université de BORDEAUX

Année Universitaire 2017/2018
Session 2 2018

Parcours : Master 1 Informatique
Code UE : 4TIN801U
Épreuve : Conceptions Formelles
Date : Vendredi 29 juin 2018 **Heure :** 14 heures
Durée : 1 heure 30
Documents : autorisés
Épreuve de M. Alain Griffault

Collège Sciences et Technologies

Indiquez votre code d'anonymat :

Avertissement

— La plupart des questions sont indépendantes.
— L'espace laissé pour les réponses est suffisant (sauf si vous utilisez ces feuilles comme brouillon, ce qui est fortement déconseillé).

Question	Points	Score
Conception avec AltaRica	12	
Vérification de modèles avec AltaRica	8	
Total:	20	

L'objectif de l'étude est d'analyser deux techniques de redondances applicables à un serveur, qui peut défaillir, afin d'obtenir un serveur presque *parfait (perfect)*.

Un serveur est un simple automate qui délivre des services. D'un point de vue abstrait, chaque événement de l'automate peut être interprété comme un service, et l'automate décrit les dépendances entre les services.

Un serveur doit être installé sur un ordinateur, et du fait des défaillances possibles de l'ordinateur, le serveur peut de temps en temps être hors de service.

Afin d'obtenir une meilleure disponibilité du serveur, nous décrivons deux techniques de redondances et les comparons à un serveur *parfait (perfect)* qui ne tombe jamais en panne.

L'approche suivie pour la modélisation et la vérification du problème de redondance est la suivante :

1. Interprétation d'un serveur *réel*.
2. Modélisation et validation de la technique de redondance dite *chaude (hot)*.
3. Modélisation et validation de la technique de redondance dite *froide (cold)*.
4. Vérification d'exigences et de propriétés.
5. Analyse et interprétation des résultats.

Un serveur *parfait (perfect)* est un automate qui ne tombe jamais en panne. Pour cet exercice, nous choisissons un automate très simple.

```
domain modes = [0,1];

node ServerPerfect
  state
    mode : modes : public;
  init
    mode := 0;
  event
    a, b : public;
  trans
    // features or services.
    mode = 0 |- a -> mode := 1;
    mode = 1 |- b -> mode := 0;
edon
```

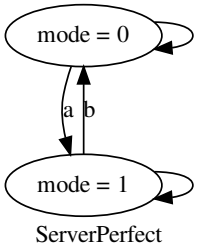


Figure 1 – Le graphe de la sémantique du serveur *parfait*

Exercice 1 : Conception avec AltaRica (12 points)

(a) Interprétation d'un serveur *réel (real)*.

Un serveur *réel (real)* est installé sur un ordinateur. Ainsi, à chaque instant, il est possible de le stopper, puis de le redémarrer dans n'importe lequel de ses "modes".

i. (2 points) Une description AltaRica d'un serveur *réel (real)* très simple est :

```
1  node ServerReal
2    state
3      mode : modes : public;
4    init
5      mode := 0;
6    flow
7      // on = true means the server is operational
8      on : bool;
9      // a variable to be able to "restart" from every modes.
10     modeForRestart : modes : public;
11   assert
12     on => (mode=modeForRestart);
13   event
14     a, b, restart : public;
15   trans
16     // features and services
17     on & mode = 0 |- a -> mode := 1;
18     on & mode = 1 |- b -> mode := 0;
19     // to restart the server from any state
20     ~on |- restart -> mode := modeForRestart;
21 edon
```

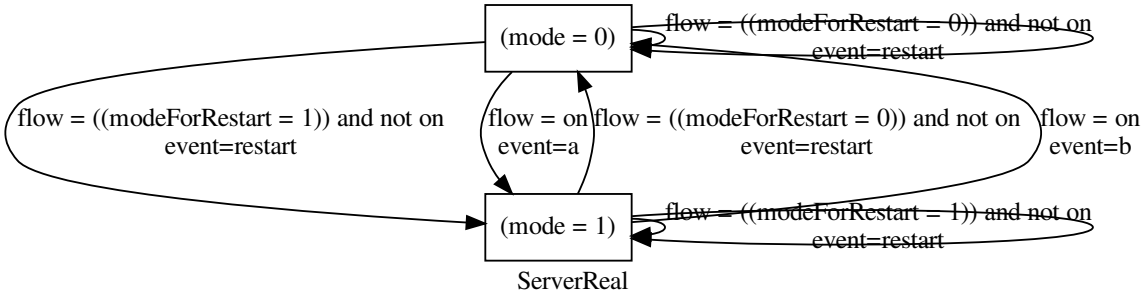


Figure 2 – Le graphe de la modes-sémantique du serveur *Réel*. Reachable (any_s : 6, any_t : 36)

Donnez les correspondances entre le code AltaRica et le graphe de la modes-sémantique.

(b) Modélisation et validation de la technique de redondance dite *chaude (hot)*.

La technique de la redondance *chaude (hot)* utilise deux serveurs en parallèle. Si l'un d'eux défaille, l'autre continue à délivrer les services. Deux difficultés de mise en oeuvre avec cette technique :

— Dès qu'un serveur *chaud (hot)* est réparé, il doit être actif et opérationnel.

— Dans le cas d'un système composé de deux serveurs *chauds (hot)*, si les deux sont en panne, le premier réparé doit redémarrer dans le "mode" correct.

i. (2 points) Un serveur *chaud (hot)*.

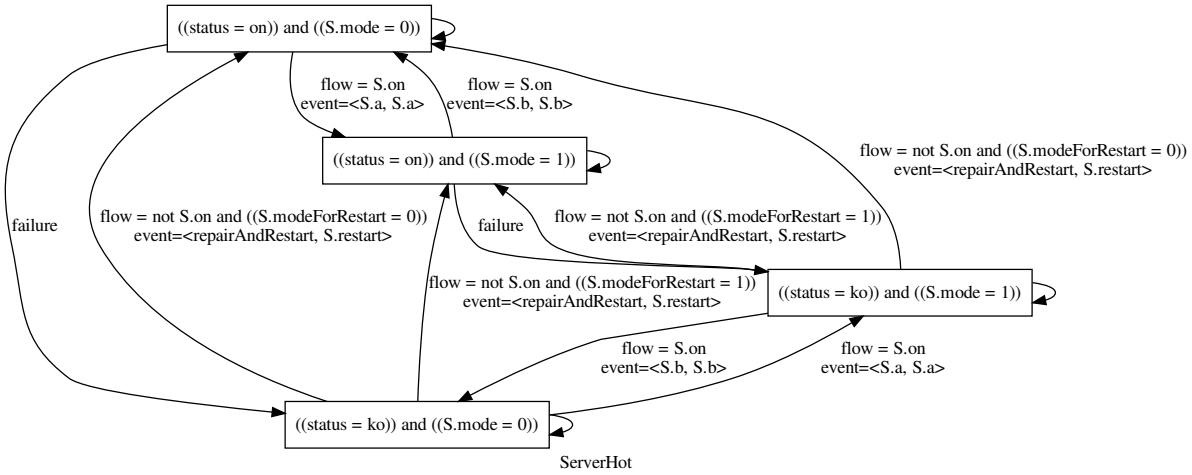


FIGURE 3 – Le graphe de la modes-sémantique d'un serveur *chaud*. Reachable (any_s : 6, any_t : 20)

Complétez le code ALTARICA de `ServerHot` afin d'obtenir la modes-sémantique de la figure 3.

```
1  node ServerHot
2    sub
3      S : ServerReal;
4    state
5      status : {on, ko} : parent;
6    init
7      status := on;
8    assert
9      // S server is operational if and only if this server is "on".
11   event
12     failure, repairAndRestart;
13   trans
14     status=on    |- failure -> status := ko;
15     status=ko    |- repairAndRestart -> status := on;
16   sync
17     // As soon as this server is repaired, S server must be
          operational.
19 edon
```

ii. (3 points) Un système *chaud (hot)*.

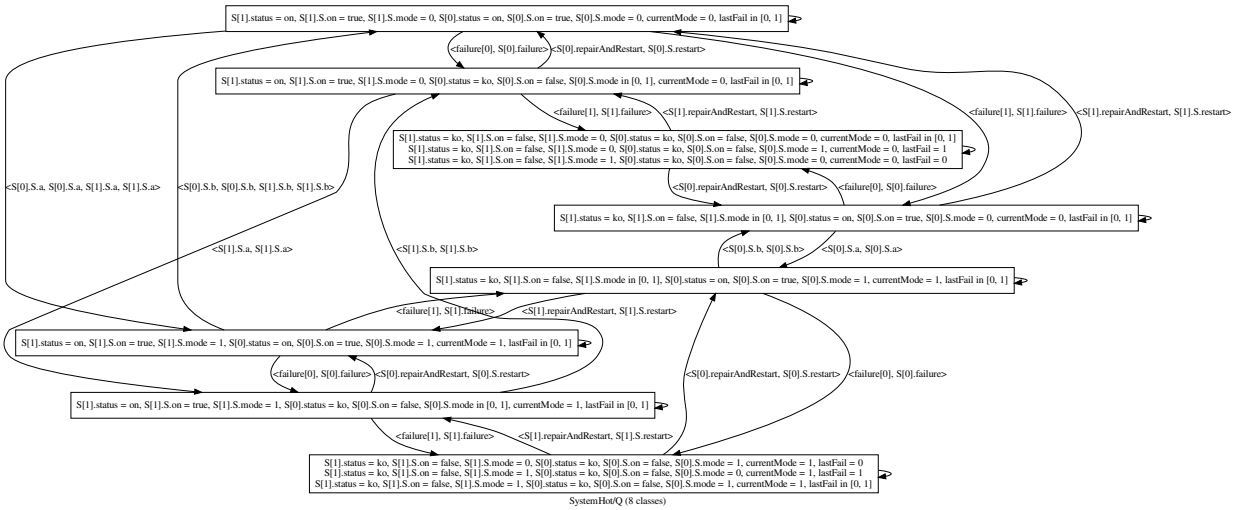


FIGURE 4 – Le graphe de la quot-sémantique du système *chaud (hot)*. Reachable (any_s : 28, any_t : 104)

Complétez le code ALTARICA de `SystemHot` afin d'obtenir la quot-sémantique de la figure 4.

```
1  node SystemHot
2    sub
3      S : ServerHot[2];
4    state
5      lastFail : [0,1];
6    flow
7      currentMode : modes;
8    assert
9      // The two Systems in use are in the same mode.
10     (S[0].S.mode = S[1].S.mode) | S[0].status!=on | S[1].status!=on;
11     // Definition of the current mode
12     (S[0].status=on) => (currentMode = S[0].S.mode);
13     (S[1].status=on) => (currentMode = S[1].S.mode);
14     (S[0].status!=on & S[1].status!=on & lastFail=0) => (currentMode =
           S[0].S.mode);
15     (S[0].status!=on & S[1].status!=on & lastFail=1) => (currentMode =
           S[1].S.mode);
16     // Definition of the two modeForRestart "parameters"
17     S[0].S.modeForRestart = currentMode;
18     S[1].S.modeForRestart = currentMode;
19   event
20     failure[2];
21   trans
22     // Memorization of the last server which fail




25   sync
26     // Maximum features and services in parallel if possible.
27     <S[0].S.a?, S[1].S.a?> >= 1;
28     <S[0].S.b?, S[1].S.b?> >= 1;
29     // Failures are failures of servers.



32 edon
```

(c) Modélisation et validation de la technique de redondance dite *froide (cold)*.

La technique de la redondance *froide (cold)* utilise deux serveurs en séquence. Si l'un d'eux défaille, l'autre le remplace pour délivrer les services. Deux difficultés de mise en oeuvre avec cette technique :

— Quand un serveur *froid (cold)* défaille, l'autre doit démarrer dès que possible.

— Dans le cas d'un système composé de deux serveurs *froids (cold)*, si les deux sont en panne, le premier réparé doit redémarrer dans le "mode" correct.

i. (2 points) Un serveur *froid (cold)*.

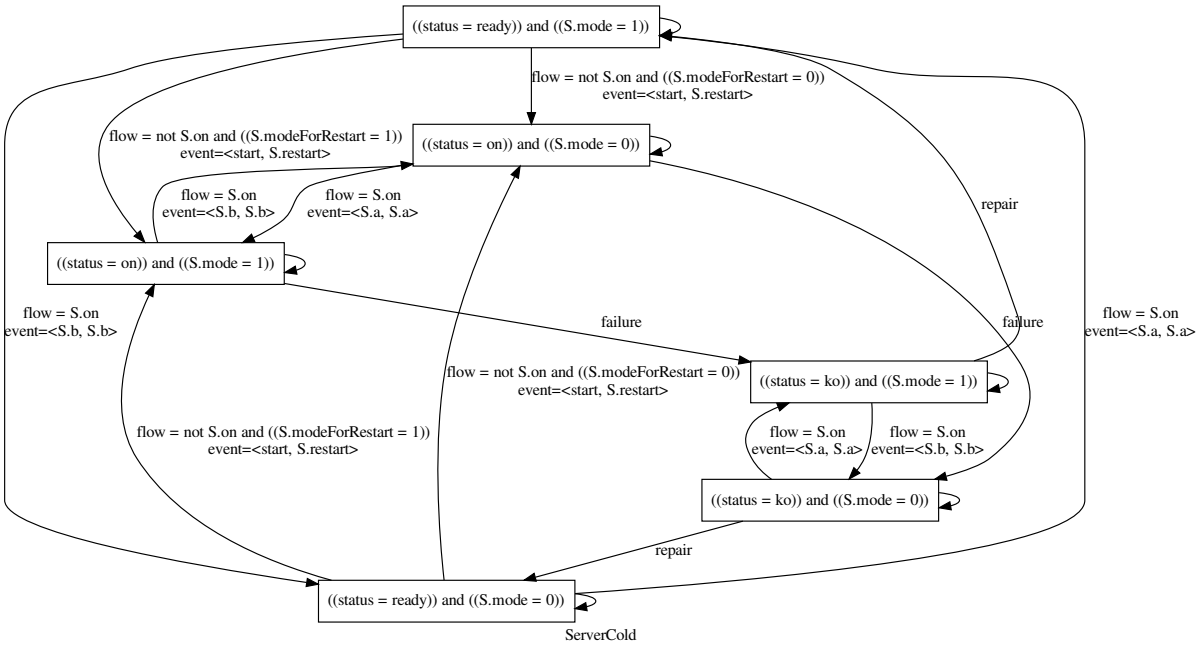


FIGURE 5 – Le graphe de la modes-sémantique du serveur *froid (Cold)*. Reachable (any_s : 10, any_t : 36)

Complétez le code ALTARICA de `ServerCold` afin d'obtenir la modes-sémantique de la figure 5.

```
1  node ServerCold
2    sub
3      S : ServerReal;
4    state
5      status : {on, ko, ready} : parent;
6    init
7      status := ready;
8    assert
9      // S server is operational if and only if this server is "on".


11   event
12     failure, repair, start;
13   trans




17   sync
18     // As soon as this server is required, S server must be
           operational.
19     <start, S.restart>;
20 edon
```

ii. (3 points) Un système *froid (cold)*.

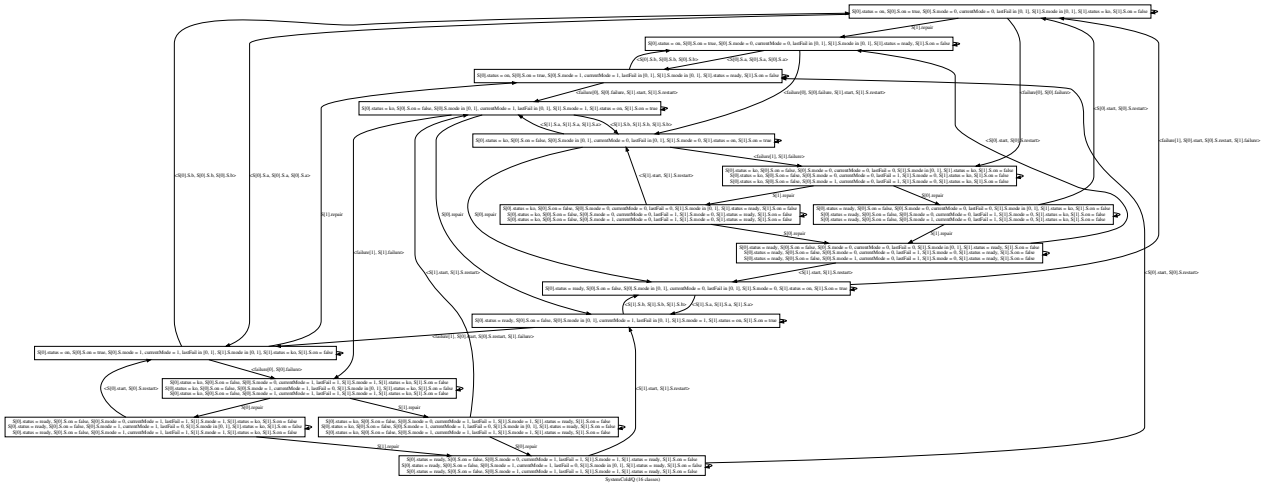


FIGURE 6 – Le graphe de la quot-sémantique du système *froid (cold)*. Reachable (any_s : 64, any_t : 208)

Complétez le code ALTARICA de `SystemCold` afin d'obtenir la quot-sémantique de la figure 6.

```
1  node SystemCold
2    sub
3      S : ServerCold[2];
4    state
5      lastFail : [0,1];
6    flow
7      currentMode : modes;
8    assert
9      // Never two ServerCold "on" at the same time.


11     // Definition of the current mode
12     (S[0].status=on) => (currentMode = S[0].S.mode);
13     (S[1].status=on) => (currentMode = S[1].S.mode);
14     (S[0].status!=on & S[1].status!=on & lastFail=0) => (currentMode =
           S[0].S.mode);
15     (S[0].status!=on & S[1].status!=on & lastFail=1) => (currentMode =
           S[1].S.mode);
16     // Definition of the two modeForRestart "parameters"
17     S[0].S.modeForRestart = currentMode;
18     S[1].S.modeForRestart = currentMode;
19   event
20     failure[2];
21   trans
22     // Memorization of the last ServerCold which fail



25   sync
26     // Independant starts of ServerCold
27     <S[0].start>;
28     <S[1].start>;
29     // Failure of a server starts the other server (if possible)



32 edon
```

Exercice 2 : Vérification de modèles avec AltaRica (8 points)

L'objectif de l'étude est de vérifier que les systèmes *chaud (hot)* et *froid (cold)* procurent les mêmes services que le serveur *parfait (perfect)*. Pour cela, nous devons vérifier que :

— Les services "a" and "b" alternent comme dans le serveur *parfait (perfect)*.
— "a" est le premier service réalisé.
— Le système est toujours opérationnel.

(a) Interprétations de formules.

i. (1½ points) Expliquez ce que calculent les expressions suivantes.

```
1  with ServerPerfect, ServerReal, SystemCold, SystemHot do
2    // Validation
3    dead := any_s - src(any_t -self_epsilon);
4    notResetable := any_s - coreach(initial,any_t);
5  done
```

```
6  with ServerPerfect, ServerReal do
7    // Usefull computations
8    Sa := any_t & label a;
9    Sb := any_t & label b;
10 done
```

```
11 with SystemCold, SystemHot do
12   // Usefull computations
13   Sa := any_t & (label S[0].S.a | label S[1].S.a);
14   Sb := any_t & (label S[0].S.b | label S[1].S.b);
15   fail0 := any_s & [S[0].status=ko];
16   fail1 := any_s & [S[1].status=ko];
17 done
```

ii. (1½ points) Expliquez, éventuellement avec un schéma, ce que l'on peut vérifier avec le calcul de `notP1`. Expliquez également la différence entre `notP1` and `notP1b`.

```
1  with ServerPerfect, ServerReal, SystemCold, SystemHot do
2    // P1 : between two Sa, there is at least one Sb.
3    notP1 := reach(tgt(Sa), any_t - Sb) & src(Sa);
4    notP1b := rsrc(reach(tgt(Sa), any_t - Sb)) & Sa;
```

```
5    // P2 : between two Sb, there is at least one Sa.
6    notP2 := reach(tgt(Sb), any_t - Sa) & src(Sb);
7    notP2b := rsrc(reach(tgt(Sb), any_t - Sa)) & Sb;
8    // P3 := System is always operational.
9    notP3 := any_s - (src(Sa)|src(Sb));
10 done
```

(b) (2 points) Complétez les calculs pour la vérification de la propriété suivante.

```
1  with ServerPerfect, ServerReal, SystemCold, SystemHot do
2    // Q1 : the first Sa is before the first Sb.
```

```
5  done
```

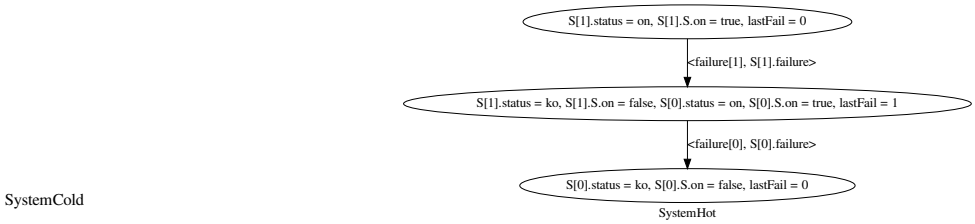
(c) Analyse des résultats.

Avec `AltaRica-Studio`, il est possible de connaître les cardinaux des propriétés calculées.
Si vous avez correctement répondu aux questions précédentes, vous obtiendrez les résultats présentés dans la table 1.

Propriétés	`ServerPerfect`	`Server`	`SystemCold`	`SystemHot`
`any_s`	2	6	64	28
`dead`	0	0	0	0
`notResetable`	0	0	0	0
`notP1`	0	1	0	0
`notP2`	0	1	0	0
`notP3`	0	4	32	8
`notQ1`	0	1	0	0
`any_t`	4	36	208	104
`notP1b`	0	3	0	0
`notP2b`	0	3	0	0
`notQ1b`	0	3	0	0

TABLE 1 – Les cardinaux des propriétés pour les quatre systèmes.

i. (1 point) Donnez une interprétation aux résultats de la table 1.



(a) `SystemCold` : un contre exemple à P3 (b) `SystemHot` : un contre exemple à P3

FIGURE 7 – Les techniques de redondance ne satisfont pas la propriété P3

ii. (1 point) Décrivez un moyen pour obtenir les deux graphes de la figure 7.

```
1  with ServerPerfect, ServerReal, SystemCold, SystemHot do
2    // A counter example for P3.
```

```
4  done
```

iii. (1 point) Expliquez pourquoi `SystemCold` et `SystemHot` ne satisfont pas `P3`.