Université Bordeaux 1	ANNÉE : 2007/2008 **PARCOURS :** Master 1 Informatique **UE :** INF 471, Conceptions Formelles **Module :** Modélisation et Vérification **Date :** Mercredi 16 avril 2008 **Heure :** 14 heures **Durée conseillée :** 1 heure 30 **Documents :** autorisés Épreuve de M. Alain GRIFFAULT	U. F. R. Mathématique Informatique

Code d'anonymat :

Avertissement

- La plupart des questions sont indépendantes.
- Le barème est (approximativement) proportionnel à la difficulté de la question.
- L'espace laissé pour les réponses est suffisant (sauf si vous utilisez ces feuilles comme brouillon, ce qui est fortement déconseillé).

Le sujet : la sûreté par redondance

L'objectif est la modélisation et la spécification d'architectures de redondances afin de vérifier que ces techniques engendrent ni perte ni gain de fonctionnalités.
La démarche est la suivante :

1. On se donne un système parfait ayant un certain nombre de fonctionnalités.
2. On ajoute du dysfonctionnement dans le système sous forme de pannes et de réparations.
3. On construit un système basé sur de la redondance pour palier les pannes possibles, afin d'avoir une meilleure disponibilité.
4. On spécifie le respect (ou le non respect) des fonctionnalités.
5. On vérifie les architectures vis à vis de ces spécifications.

Les propriétés

Elles spécifient que la redondance ne doit pas changer les fonctionnalités.

Prop1 : En présence de moins d'une panne, le système avec redondance se comporte comme le système parfait.

Prop2 : En présence de moins de deux pannes, le système avec redondance se comporte comme le système parfait.

Prop3 : En présence de moins de trois pannes, le système avec redondance se comporte comme le système parfait, avec éventuellement des pertes de disponibilité temporaires.

1 Le système parfait

1.1 Le nœud Machine

1.1.1 Le source AltaRica

```
node Machine
  state mode : [0,3] : parent;
  event a, b, c
  trans
    // les fonctionnalites
    mode = 0 |- a -> mode := 1;
    mode = 1 |- b -> mode := 2;
    mode = 2 |- a -> mode := 1;
    mode = 2 |- c -> mode := 3;
    mode = 3 |- b -> mode := 0;
    mode = 3 |- b -> mode := 1;
  init
    mode := 0;
edon
```

1.1.2 La sémantique

Question 1 (1 point) *Dessinez le graphe de la sémantique.*

1.2 La réalité

Malheureusement, dans les systèmes réels, les machines ne sont pas parfaites.

2 Le nœud MachineV1

Cette machine possède les mêmes fonctionnalités que la machine parfaite, mais elle peut malheureusement tomber en panne à tout moment en cours d'utilisation.
La réparation peut se faire sur place, et dès qu'elle est réparée, elle reprends son travail à partir de n'importe quelle situation qui lui sera indiquée.

2.1 Le source AltaRica

```
node MachineV1
  state mode : [0,3] : parent;
        ok   : bool  : parent;    // pour coder la panne
  flow  modeReprise : [0,3];
  event a, b, c, panne, reparationStart
  trans
    // les fonctionnalites ne sont possibles qu'en etat de marche
    ok & mode = 0 |- a -> mode := 1;
    ok & mode = 1 |- b -> mode := 2;
    ok & mode = 2 |- a -> mode := 1;
    ok & mode = 2 |- c -> mode := 3;
    ok & mode = 3 |- b -> mode := 0;
    ok & mode = 3 |- b -> mode := 1;
    // la panne est possible lorsque la machine fonctionne
    ok            |- panne -> ok := false;
    // la reparation demarre la machine dans le mode passe en parametre
    ~ok           |- reparationStart -> ok := true, mode := modeReprise;
  init
    mode := 0, ok := true;
edon
```

2.2 La sémantique

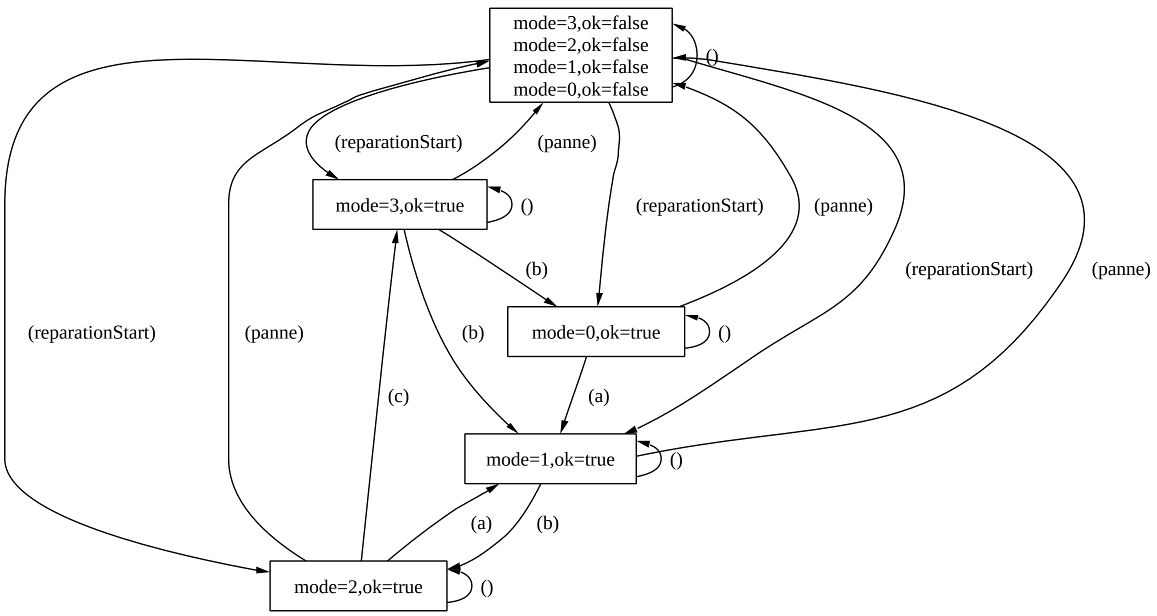

Fig. 1 – Le nœud MachineV1sans tenir compte de la variable `modeReprise`

3 Le nœud MachineV2

Cette machine possède les mêmes fonctionnalités que la machine parfaite, mais elle peut malheureusement tomber en panne à tout moment en cours d'utilisation.
Contrairement à la machine précédente qui redémarre dès qu'elle est réparée, cette machine attend une sollicitation pour démarrer.
Une fois réparée, elle est prête à être utilisée dès que nécessaire. Elle peut alors reprendre son travail à partir de n'importe quelle situation qui lui sera indiquée.

3.1 La sémantique

Question 2 (1 point) *Dessinez le graphe de la sémantique sans tenir compte de la variable* `modeReprise`

3.2 Le source AltaRica

Question 3 (3 points) *Complétez le source AltaRica de votre machine.*

```
node MachineV2
  state mode : [0,3] : parent;


  flow  modeReprise : [0,3];
  event a, b, c, panne, reparation, start
  trans
    // les fonctionnalites ne sont possibles qu'en etat de marche
    mode = 0                      |- a
      -> mode := 1                                          ;
    mode = 1                      |- b
      -> mode := 2                                          ;
    mode = 2                      |- a
      -> mode := 1                                          ;
    mode = 2                      |- c
      -> mode := 3                                          ;
    mode = 3                      |- b
      -> mode := 0                                          ;
    mode = 3                      |- b
      -> mode := 1                                          ;
    // la panne est possible lorsque la machine fonctionne
                                  |- panne
      ->                                                    ;
    // la reparation
                                  |- reparation
      ->                                                    ;
    // la machine demarre dans le mode passe en parametre
                                  |- start
      ->                                                    ;
  init
    mode := 0,
edon
```

4 La technique de redondance chaude

La redondance dite *chaude* est un technique qui consiste à utiliser plusieurs machines de type `MachineV1` simultanément de sorte que si l'une tombe en panne, l'autre est immédiatement opérationnelle.
Dans cette technique, tant que les machines ne sont pas en pannes, les deux machines fonctionnent. Lorsqu'une d'entre elles tombe en panne, la seconde continue seule. Enfin, dès qu'une machine est réparée, elle est paramétrée pour repartir du mode dans lequel se trouve l'autre machine.
Une difficulté de cette architecture apparaît lorsque les deux machines sont en pannes. Il faut que la première réparée reparte de la situation dans laquelle se trouvait la dernière tombée en panne. Cela afin de respecter le fonctionnement de la machine parfaite.

4.1 Le source AltaRica du nœud UsineV1

Question 4 (3,5 points) *Complétez le source AltaRica de votre système utilisant la redondance chaude.*

```
node UsineV1
  sub   M1, M2 : MachineV1
  state dernierEnPanne : [1,2];
  flow  modeCourant : [0,3];
  event panneM1, panneM2;
  trans
    // Memorisation de la derniere machine tombee en panne



  sync
    // Les fonctionnalites




    // Les pannes
    <panneM1, M1.panne>;
    <panneM2, M2.panne>;
  assert
    // Les machines non en panne sont dans le même mode
    (M1.mode = M2.mode) | ~M1.ok | ~M2.ok;
    // La definition du mode courant






    // La propagation des modes de reprise pour les machines



edon
```

5 La technique de redondance froide

La redondance dite *froide* est un technique qui consiste à utiliser plusieurs machines de type `MachineV2` simultanément de sorte que si l'une tombe en panne, l'autre est immédiatement opérationnelle.
Dans cette technique, au maximum une seule machine fonctionne. Lorsqu'elle tombe en panne, la seconde prend le relais. Enfin, dès qu'une machine est réparée, elle se tient prête à un démarrage paramétré pour repartir du mode dans lequel se trouve l'autre machine.
Une difficulté de cette architecture apparaît lorsque les deux machines sont en pannes. Il faut que la première réparée reparte de la situation dans laquelle se trouvait la dernière tombée en panne. Cela afin de respecter le fonctionnement de la machine parfaite.

5.1 Le source AltaRica du nœud UsineV2

Question 5 (3,5 points) *Complétez le source AltaRica de votre système utilisant la redondance froide.*

```
node UsineV2
  sub   M1, M2 : MachineV2
  state dernierEnPanne : [1,2];
  flow  modeCourant : [0,3];
  event panneM1, panneM2;
  trans
    // Memorisation de la derniere machine tombee en panne



  sync
    // Les demarrages des machines



    // Les pannes engendrent si possible les demarrages




  assert
    // Jamais deux machines en marche simultanement

    // La definition du mode courant






    // La propagation des modes de reprise pour les machines




edon
```

6 Validité des redondances

Nous souhaitons montrer que les techniques de redondance sont correctes au sens où elles ne modifient pas les fonctionnalités. Les machines en redondance se comportent exactement comme une machine parfaite si l'on *n'observe pas* les pannes et les réparations.
Pour cela nous utilisons le model-checker `acheck`.

6.1 Les propriétés

6.1.1 Les spécifications

Question 6 (7 points) *Complétez le source au format acheck afin de vérifier la technique de redondance* **chaude** *du modèle* `UsineV1`*.*

```
with UsineV1 do
 // les proprietes pour la validation generale du modele
 quot()      >  '$NODENAME.dot';
 dead        := any_s - src(any_t - self_epsilon);
 notCFC      := any_t - loop(any_t,any_t);
 // Partition des configurations en fonction du nombre de pannes
 nbPanne0   :=
 nbPanne1   :=
 nbPanne2   :=
 // Les fonctionnalites
 a           := label M1.a | label M2.a;
 b           := label M1.b | label M2.b;
 c           := label M1.c | label M2.c;
 // Les dysfonctionnalites
 panne           := label M1.panne | label M2.panne;
 reparationStart := label M1.reparationStart |
                    label M2.reparationStart;
 /* P1 : sans panne, les deux machines sont dans le meme mode
  * ATTENTION, ON CALCULE LES CONTRES-EXEMPLES
  */
 Not_P1          :=

 /* P2 : maximum une panne conserve les fonctionnalites
  * La propriete P2 est decomposee par rapport aux fonctionnalites
  * P2 = P2a & P2b & P2c
  * P2a : maximum une panne converse la fonctionnalite "a"
  * P2a est decomposee en deux proprietes
  * P2a = P2aPanne & P2aRepare
  * P2aPanne : une premiere panne conserve la fonctionnalite "a"
  *  1/ "a" possible avant "panne" => "a" possible apres "panne"
  *  2/ "a" possible apres "panne" => "a" possible avant "panne"
  * P2aRepare : la reparation conserve la fonctionnalite "a"
```

```
 *  1/ "a" possible avant "repare" => "a" possible apres "repare"
 *  2/ "a" possible apres "repare" => "a" possible avant "repare"
 *
 * ATTENTION, ON CALCULE LES CONTRES-EXEMPLES
 */
Not_P2aPanne1  := nbPanne0 &
                  src(a) &
                  src(panne & rtgt(any_s - src(a)));
/* Question : Faire un schema expliquant le calcul de Not_P2aPanne1
 *
 *
 *
 *
 *
 *
 *
 *
 *
 */
Not_P2aPanne2  :=


Not_P2aPanne   :=
Not_P2aRepare1 :=


Not_P2aRepare2 :=


Not_P2aRepare  :=
Not_P2a        :=
/* P3 : maximum deux pannes conserve les fonctionnalites
 * La propriete P3 est decomposee par rapport aux fonctionnalites
 * P3 = P3a & P3b & P3c
 * P3a : maximum deux pannes converse la fonctionnalite "a"
 * P3a est decomposee en deux proprietes
 *  1/ "a" possible avant seconde "panne" => "a" possible apres "repare"
 *  2/ "a" possible apres "repare" => "a" possible avant seconde "panne"
 *
 * ATTENTION, ON CALCULE LES CONTRES-EXEMPLES
 */
```

```
 Not_P3a1   := nbPanne1 &
               src(a) &
               src(panne & rtgt(src(reparationStart &
                                    rtgt(any_s - src(a)))));
 /* Question : Faire un schema expliquant le calcul de Not_P3a1
  *
  *
  *
  *
  *
  *
  *
  *
  */
 Not_P3a2   :=



 Not_P3a    :=
 // Sortie des resultats
 test(dead,0)   >  '$NODENAME.res';
 test(notCFC,0) >> '$NODENAME.res';
 test(Not_P1,0) >> '$NODENAME.res';
 test(Not_P2a,0)>> '$NODENAME.res';
 test(Not_P3a,0)>> '$NODENAME.res';
done
```

Question 7 (1 point) *Remplissez le tableau en précisant, lorsque vous répondez* `[FAILED]`*, si vous pensez que cela est du à votre modèle ou bien à votre spécification.*

Propriété	[PASSED]	[FAILED]	Explication
`TEST(dead=0)`			
`TEST(notCFC=0)`			
`TEST(Not_P1=0)`			
`TEST(Not_P2a=0)`			
`TEST(Not_P3a=0)`			