université de BORDEAUX

Année Universitaire 2013/2014
Session de Printemps 2014

Parcours : Master 1 Informatique
Code UE : J1IN8W14
Épreuve : Conceptions Formelles
Date : Lundi 28 avril 2014 **Heure :** 8 heures 30
Durée : 1 heure 30
Documents : autorisés
Épreuve de M. Alain Griffault

Collège Sciences et Technologies

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
Modélisation en AltaRica	13	
Vérification de modèles avec `ARC`	7	
Total:	20	

Le but de l'exercice est la conception et la vérification de quelques propriétés de circuits électriques utilisant différents types d'interrupteurs.

Exercice 1 : Modélisation en AltaRica (13 points)

Les circuits électriques utilisent des générateurs identiques à ceux utilisés en cours, et dont la sémantique est donnée par la figure 1.

```
node Generator
  state  ok : bool;
  init   ok := true;
  flow   plus, minus : [0,1];
  event  failure, repair;
  trans  ok |- failure -> ok := false;
         ~ok |- repair -> ok := true;
  assert plus = 1;
         ok = (minus=0);
edon
```

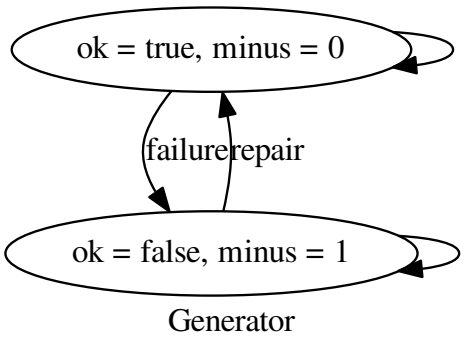


Figure 1 – La sémantique du nœud `Generator`

(a) (2 points) Les lampes utilisées sont différentes de celles du cours. Elles s'éteignent et s'allument *instantanément*, et réagissent en cas de court circuit en *grillant*.
Dessiner la sémantique du code AltaRica suivant.

```
node Lamplight
  state  ok : bool;
  init   ok := true;
  flow   light : bool;
         f1, f2 : [0,1];
  event  reaction, repair;
  trans  ok & (f1=1) & (f2=1) |- reaction -> ok := false;
         ~ok |- repair -> ok := true;
  assert light = (ok & (f1+f2=1));
edon
```

(b) Le premier circuit.

i. (2 points) Un interrupteur *va-et-vient* possède trois fils. Suivant les appuis sur l'interrupteur, il connecte deux d'entre eux et laisse le troisième *libre*.

Compléter la rubrique `assert` du code AltaRica du nœud `Switch` afin de respecter les indications du schéma, et d'obtenir pour sémantique la figure 2

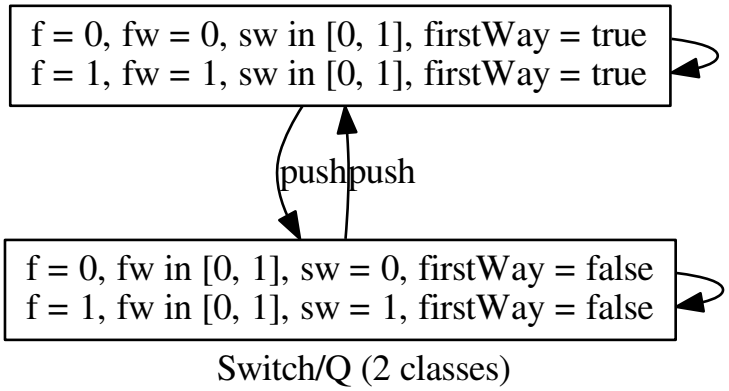
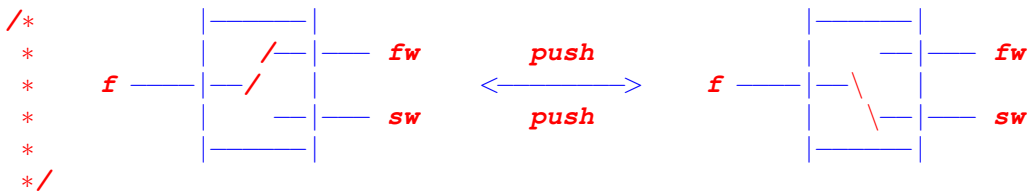


FIGURE 2 – Sémantique du nœud `Switch`




```
node Switch
  state  firstWay : bool;
  init   firstWay := true;
  flow   f, fw, sw : [0,1];
  event  push;
  trans  true |- push -> firstWay := ~firstWay;
  assert




edon
```

ii. (2 points) Le code AltaRica du premier circuit utilise deux interrupteurs dits *va-et-vient*, placés entre le générateur et la lampe.

```
node Circuit1
  sub G : Generator;
      L : Lamplight;
      S : Switch[2];
  assert
      // wires
      G.plus = S[0].f;
      G.minus = L.f2;
      S[0].fw = S[1].fw;
      S[0].sw = S[1].sw;
      S[1].f = L.f1;
      // directed flow from generator to switch
      (S[0].fw + S[0].sw)<=1;
edon
```

Dessiner le schéma du premier circuit électrique, en faisant apparaître les différents composants, et les noms des variables de flux.

(c) (3 points) Le second circuit.

Si votre modèle d'interrupteur est correct, la sémantique obtenue avec la commande `quot()` du nœud `Circuit1` correspond à la figure 3.

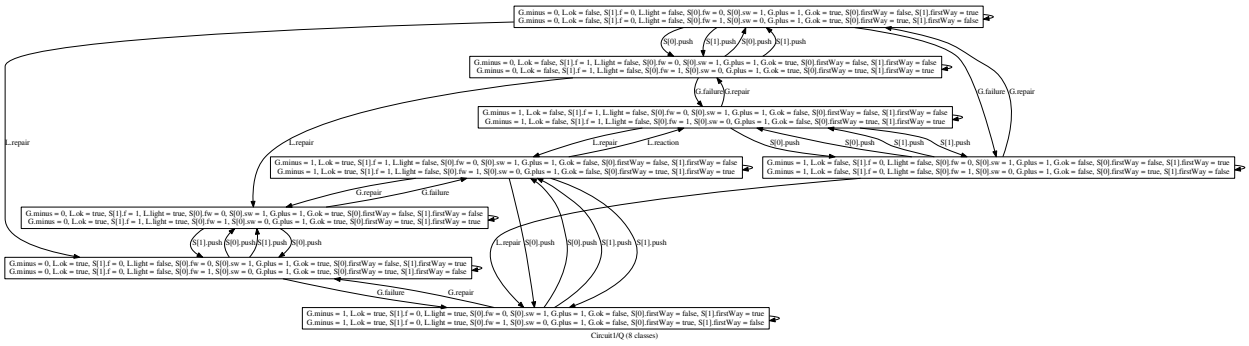


FIGURE 3 – Sémantique *quotient* du nœud `Circuit1`

D'après la sémantique, avec le modèle `Circuit1`, si le générateur tombe en panne, il est possible qu'un utilisateur appuie sur l'interrupteur ou bien que la réparation soit faite, avant que la lampe n'ait eue le temps de réagir.

Compléter les rubriques `event`, `trans` et `sync` du code du nœud AltaRica `Circuit2` afin de modifier certains comportements :

– En cas de panne du générateur, s'il est possible, l'événement `reaction` de la lampe est toujours *tiré* avant qu'un nouvel événement `push` ou `repair` soit possible.

– Le technicien répare toujours en une seule fois (simultanément) le plus possible de composants en panne, et au moins un à chaque réparation.

```
node Circuit2
  sub G : Generator;
      L : Lamplight;
      S : Switch[2];
  assert
      // wires
      G.plus = S[0].f;
      G.minus = L.f2;
      S[0].fw = S[1].fw;
      S[0].sw = S[1].sw;
      S[1].f = L.f1;
      // directed flow from generator to switch
      (S[0].fw + S[0].sw)<=1;
  event

  trans

  sync

edon
```

Si votre modèle `Circuit2` est correct, la sémantique obtenue avec la commande `quot()` correspond à la figure 4.

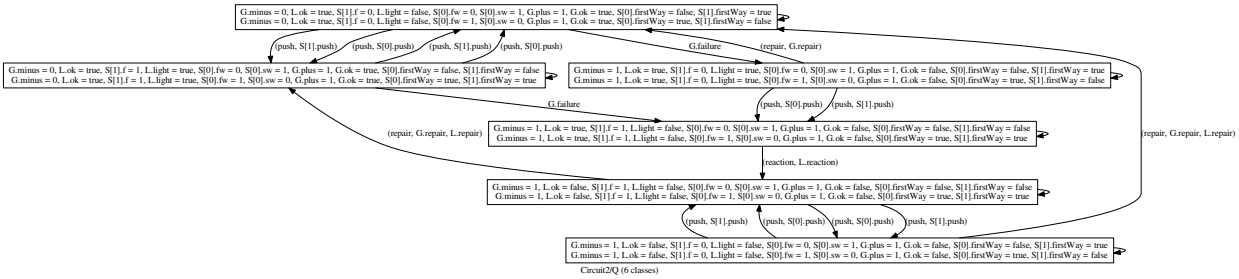


FIGURE 4 – Sémantique *quotient* du nœud `Circuit2`

(d) Le troisième circuit.

i. (2 points) Un interrupteur *croisé* possède quatre fils. Suivant les appuis sur l'interrupteur, il connecte parallèlement, ou bien en croix deux couples de fils.

Compléter la rubrique `assert` du code AltaRica du nœud `SwitchCross` afin de respecter les indications du schéma, et d'obtenir pour sémantique la figure 5

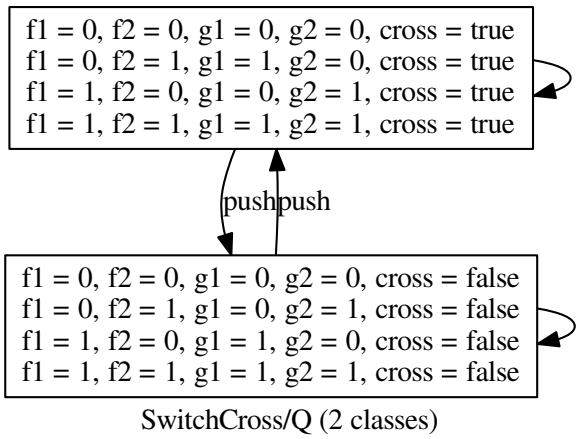
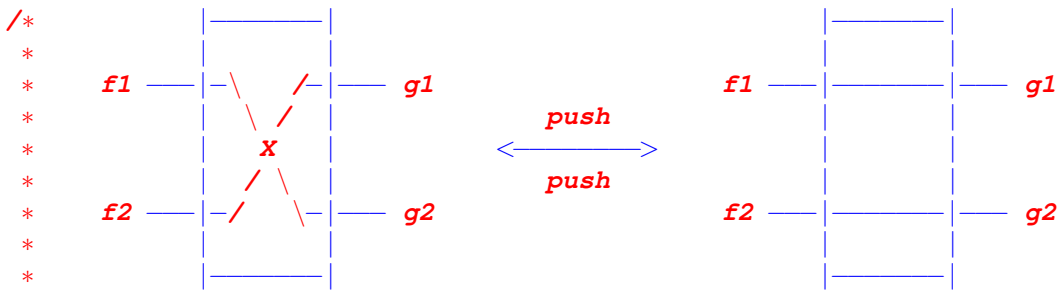


FIGURE 5 – Sémantique du nœud `SwitchCross`




```
node SwitchCross
  state  cross : bool;
  init   cross := true;
  flow   f1, f2, g1, g2 : [0,1];
  event  push;
  trans  true |- push -> cross := ~cross;
  assert
```

```
edon
```

ii. (2 points) Le troisième circuit ajoute au second circuit un interrupteur *croisé* entre les deux interrupteurs *va-et-vient*.
Compléter les rubriques `event`, `trans` et `sync` du code AltaRica du nœud `Circuit3` afin que sa sémantique corresponde à la figure 6, figure très similaire à la figure 4.

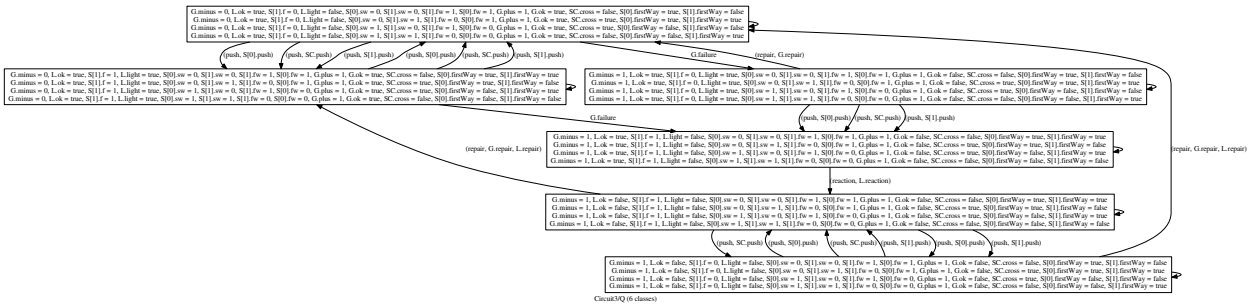


FIGURE 6 – Sémantique *quotient* du nœud `Circuit3`

```
node Circuit3
  sub G : Generator;
      L : Lamplight;
      S : Switch[2];
      SC : SwitchCross;
  assert
      // wires
      G.plus = S[0].f;
      G.minus = L.f2;
      S[0].fw = SC.f1;
      S[0].sw = SC.f2;
      S[1].fw = SC.g1;
      S[1].sw = SC.g2;
      S[1].f = L.f1;
      // directed flow from generator to switch
      (S[0].fw + S[0].sw)<=1;
  event


  trans



  sync









edon
```

Exercice 2 : Vérification de modèles avec ARC (7 points)

L'objectif de ces propriétés est de montrer pour ces circuits :
- qu'ils sont tous sans blocage ;
- qu'ils sont tous réinitialisables ;
- que le premier peut être distingué des deux autres par deux propriétés.

(a) (1 point) Écrire une propriété permettant de savoir si les circuits sont sans blocage.

```
with Circuit1, Circuit2, Circuit3  do
  deadlock :=




done
```

(b) (1 point) Écrire une propriété permettant de savoir si les circuits sont réinitialisables.

```
with Circuit1, Circuit2, Circuit3  do
  notResetable :=




done
```

(c) (2 points) Écrire une propriété permettant de savoir s'il existe des situations à partir desquelles il est possible d'une part d'appuyer sur un interrupteur ou bien de faire des réparations ; d'autre part que la lampe réagisse. Cela revient à savoir si des conflits sont possibles entre ces deux types d'événements.

```
with Circuit1, Circuit2, Circuit3  do
  reactions := (any_t - self_epsilon) & (epsilon | label L.reaction);
  failures  := (any_t - self_epsilon) & label G.failure;
  actions   := (any_t - self_epsilon) - (reactions | failures);
  conflicts :=
```

(d) (3 points) Écrire une propriété permettant de savoir si, lorsque la lampe est allumée, dès que le générateur tombe en panne, la lampe réagit avant que toute action humaine soit possible. Cela revient à savoir si la réaction de la lampe est prioritaire par rapport aux actions.

```
with Circuit1,Circuit2, Circuit3  do
  notLampReacts :=




done
```