Projet Conceptions Formelles 2006-2007
AltaRica
Synthèse (assistée) d'un contrôleur de trains

Prenom1 Nom1 and Prenom2 Nom2

10 novembre 2006

Table des matières

Chapitre 1

Cahier des charges

Le problème général est le contrôle de plusieurs trains circulant sur les mêmes voies, et devant transporter des marchandises d'un lieu à un autre.

1.1 Composition d'un système

Le système que l'on souhaite concevoir est composé :

de 11 cantons qui représentent les onze portions de voies ferrées sur lesquelles circulent les trains. Chaque canton est simplement représenté par un numéro.

de 3 trains circulant sur les cantons.
- le train T1 circule en boucle sur les cantons numérotés entre 0 et 5.
- le train T2 circule en boucle sur les cantons numérotés entre 5 et 10.
- le train T3 circule en boucle sur les cantons numérotés entre 0 et 10.

de 4 gares positionnées sur les cantons.
- la gare G1 sur le canton 0, qui permet au train T1 de décharger sa marchandise.
- la gare G2 sur le canton 3, qui permet aux train T1 et T3 de charger la marchandise.
- la gare G3 sur le canton 6, qui permet au train T2 de charger sa marchandise.
- la gare G4 sur le canton 9, qui permet aux train T2 et T3 de décharger la marchandise.

d'un contrôleur représenté dans le modèle par le système lui-même ayant à disposition : d'une part des commandes possibles ; d'autre part une observation totale du système.

1.2 Détails techniques

1.2.1 Les cantons

Ils représentent une portion de voie ferrée. C'est l'unite de discrétisation au sens où il est impossible que deux trains soient simultanément sur un même canton.

Il est possible de les abstraire par un simple numéro.

1.2.2 Les gares

Elles correspondent à un canton particulier. Un train peut charger et décharger de la marchandise dans les gares à condition qu'il soit à l'arrêt.

1.2.3 Les trains

Ils circulent de canton en canton. Ils possédent une seule vitesse non nulle de circulation. Ils peuvent s'arrêter sur un canton. Ils doivent obligatoirement charger la marchandise et décharger la marchandise dans les gares atitrées.

1.2.4 Le contrôleur

Il commande les chargements, les déchargements et les déplacements des trains avec la contrainte supplémentaire suivante :

1. Le déchargement de T1 est synchrone au chargement de T2.

1.3 Les spécifications

Le système devra vérifier la propriété suivante :

Propriété 1

1. *L'accident (deux trains sur le même canton) est interdit.*
2. *Le blocage du système est interdit.*

Propriété 2 *Le système cherchera également à être le plus productif possible.*

1.4 L'étude

L'étude doit au moins aboutir à un contrôleur satisfaisant la propriété 1 tout en respectant les contraintes du transport des marchandises.

Si possible, l'étude fournira aussi un contrôleur satisfaisant également la propriété 2.

1.5 Modalités

- Travail en binôme (trinôme refusé).
- Rapport (sources AltaRica et acheck, résultats des analyses et un minimum d'explications) à copier sous le nom `Nom1Nom2.tar.gz` dans le répertoire `~algriffa/Master2CF/` avant **vendredi 17 novembre à 21 heures**.

Chapitre 2

Les composants

2.1 La première version du nœud Train

2.1.1 Le source AltaRica

```
node Train1_V1
  state
    canton : [0,5] : public;
    charge : bool : public;
    vitesse : [0,1] : public;
  flow
    cc, cd : [0,5];
  event
    nop, inc, dec, put, get;
  trans
    true  |- nop -> canton := (canton+vitesse) mod 6;
    true  |- inc -> canton := (canton+vitesse) mod 6, vitesse := vitesse+1;
    true  |- dec -> canton := (canton+vitesse) mod 6, vitesse := vitesse-1;
    (canton = cc) & (vitesse = 0) & ~charge |- put -> charge := true;
    (canton = cd) & (vitesse = 0) & charge  |- get -> charge := false;
  assert
    cc = 3;
    cd = 0;
    // Il est obligatoire de charger
    (canton = (cc+1) mod 6) => charge;
    // Il est obligatoire de decharger
    (canton = (cd+1) mod 6) => ~charge;
  init
    vitesse :=0, canton := 2, charge := false;
edon

node Train2_V1
  state
    canton : [5,10] : public;
    charge : bool : public;
    vitesse : [0,1] : public;
  flow
    cc, cd : [5,10];
  event
```

```
    nop, inc, dec, put, get;
  trans
    true |- nop -> canton := 5+((canton-5+vitesse) mod 6);
    true |- inc -> canton := 5+((canton-5+vitesse) mod 6), vitesse := vitesse+1;
    true |- dec -> canton := 5+((canton-5+vitesse) mod 6), vitesse := vitesse-1;
    (canton = cc) & (vitesse = 0) & ~charge |- put -> charge := true;
    (canton = cd) & (vitesse = 0) & charge  |- get -> charge := false;
  assert
    cc = 6;
    cd = 9;
    // Il est obligatoire de charger
    (canton = 5+((cc-4) mod 6)) => charge;
    // Il est obligatoire de decharger
    (canton = 5+((cd-4) mod 6)) => ~charge;
  init
    vitesse :=0, canton := 10, charge := false;
edon

node Train3_V1
  state
    canton : [0,10] : public;
    charge : bool : public;
    vitesse : [0,1] : public;
  flow
    cc, cd : [0,10];
  event
    nop, inc, dec, put, get;
  trans
    true  |- nop -> canton := (canton+vitesse) mod 11;
    true  |- inc -> canton := (canton+vitesse) mod 11, vitesse := vitesse+1;
    true  |- dec -> canton := (canton+vitesse) mod 11, vitesse := vitesse-1;
    (canton = cc) & (vitesse = 0) & ~charge |- put -> charge := true;
    (canton = cd) & (vitesse = 0) & charge  |- get -> charge := false;
  assert
    cc = 3;
    cd = 9;
    // Il est obligatoire de charger
    (canton = (cc+1) mod 11) => charge;
    // Il est obligatoire de decharger
    (canton = (cd+1) mod 11) => ~charge;
  init
    vitesse :=0, canton := 1, charge := false;
edon
```

2.1.2 La sémantique

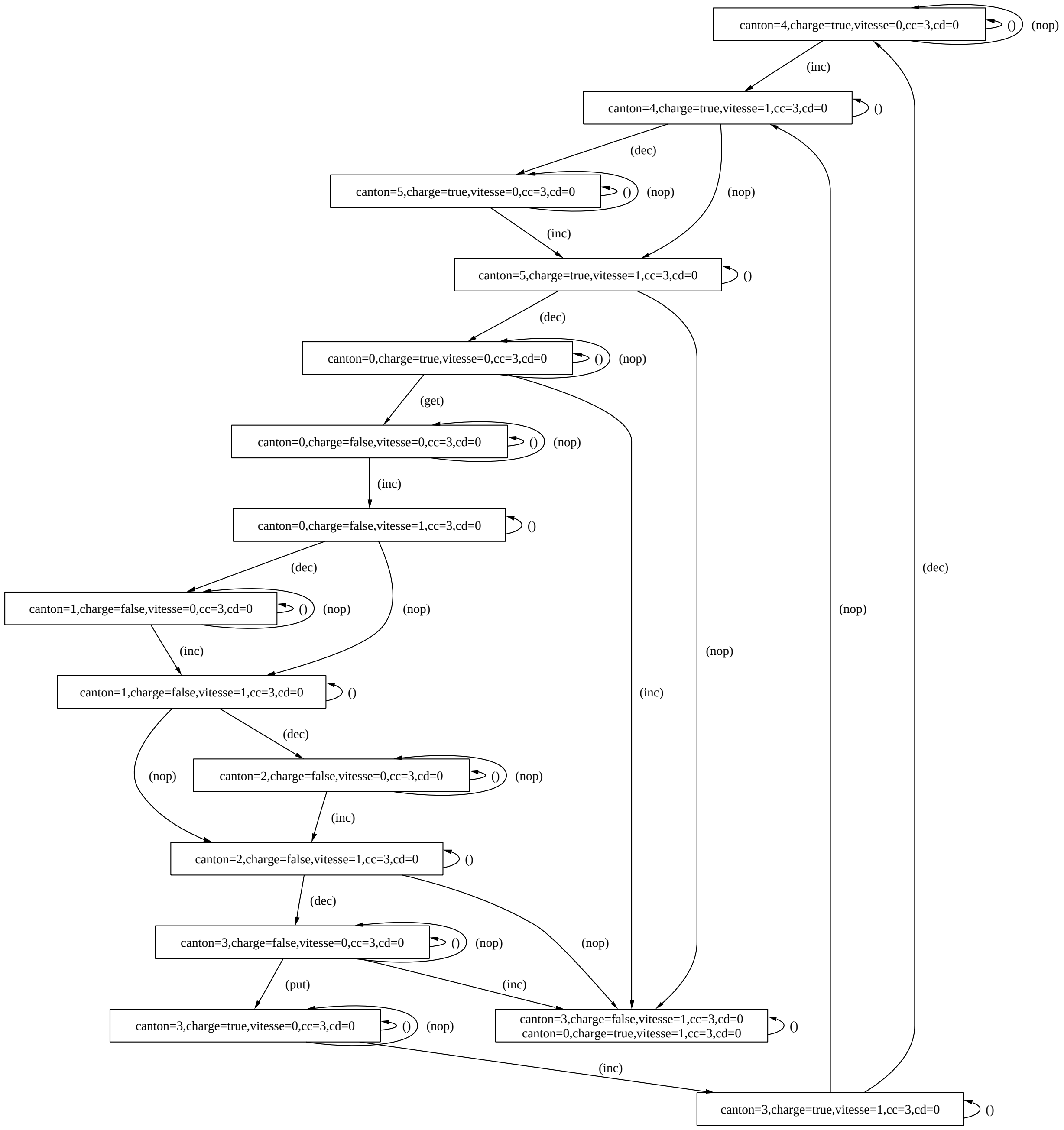

FIG. 2.1 – La première version du nœud Train

2.1.3 Les propriétés

Les spécifications

Les résultats

```
TEST(dead=0) [FAILED] actual size = 2
TEST(notCFC=0) [FAILED] actual size = 4
```

Interprétations

```
/*
 * # state properties : 3
 *
 * initial = 1
 * dead = 2
 * any_s = 16
 *
 * # transition properties : 6
 *
 * not_deterministic = 0
 * self = 24
 * notCFC = 4
 * any_t = 46
 * self_epsilon = 16
 * epsilon = 16
 */
```

Question 2.1 *Expliquez la modélisation de la première version du Train.*

Réponse à la question 2.1 :

2.2 La seconde version du nœud Train

2.2.1 Le source AltaRica

```
node Train1_V2
  state
    canton : [0,5] : public;
    charge : bool : public;
  flow
    vitesse : [0,1];
    cc, cd : [0,5];
  event
    top, put, get;
  trans
    true  |- top -> canton := (canton+vitesse) mod 6;
    (canton = cc) & (vitesse = 0) & ~charge |- put -> charge := true;
    (canton = cd) & (vitesse = 0) & charge  |- get -> charge := false;
  assert
    cc = 3;
    cd = 0;
    // Il est obligatoire de charger
    (canton = (cc+1) mod 6) => charge;
    // Il est obligatoire de decharger
    (canton = (cd+1) mod 6) => ~charge;
  init
    canton := 2, charge := false;
edon

node Train2_V2
  state
```

```
    canton : [5,10] : public;
    charge : bool : public;
  flow
    vitesse : [0,1];
    cc, cd : [5,10];
  event
    top, put, get;
  trans
    true  |- top -> canton := 5+((canton-5+vitesse) mod 6);
    (canton = cc) & (vitesse = 0) & ~charge |- put -> charge := true;
    (canton = cd) & (vitesse = 0) & charge  |- get -> charge := false;
  assert
    cc = 6;
    cd = 9;
    // Il est obligatoire de charger
    (canton = 5+((cc-4) mod 6)) => charge;
    // Il est obligatoire de decharger
    (canton = 5+((cd-4) mod 6)) => ~charge;
  init
    canton := 10, charge := false;
edon

node Train3_V2
  state
    canton : [0,10] : public;
    charge : bool : public;
  flow
    vitesse : [0,1];
    cc, cd : [0,10];
  event
    top, put, get;
  trans
    true  |- top -> canton := (canton+vitesse) mod 11;
    (canton = cc) & (vitesse = 0) & ~charge |- put -> charge := true;
    (canton = cd) & (vitesse = 0) & charge  |- get -> charge := false;
  assert
    cc = 3;
    cd = 9;
    // Il est obligatoire de charger
    (canton = (cc+1) mod 11) => charge;
    // Il est obligatoire de decharger
    (canton = (cd+1) mod 11) => ~charge;
  init
    canton := 1, charge := false;
edon
```

2.2.2 La sémantique

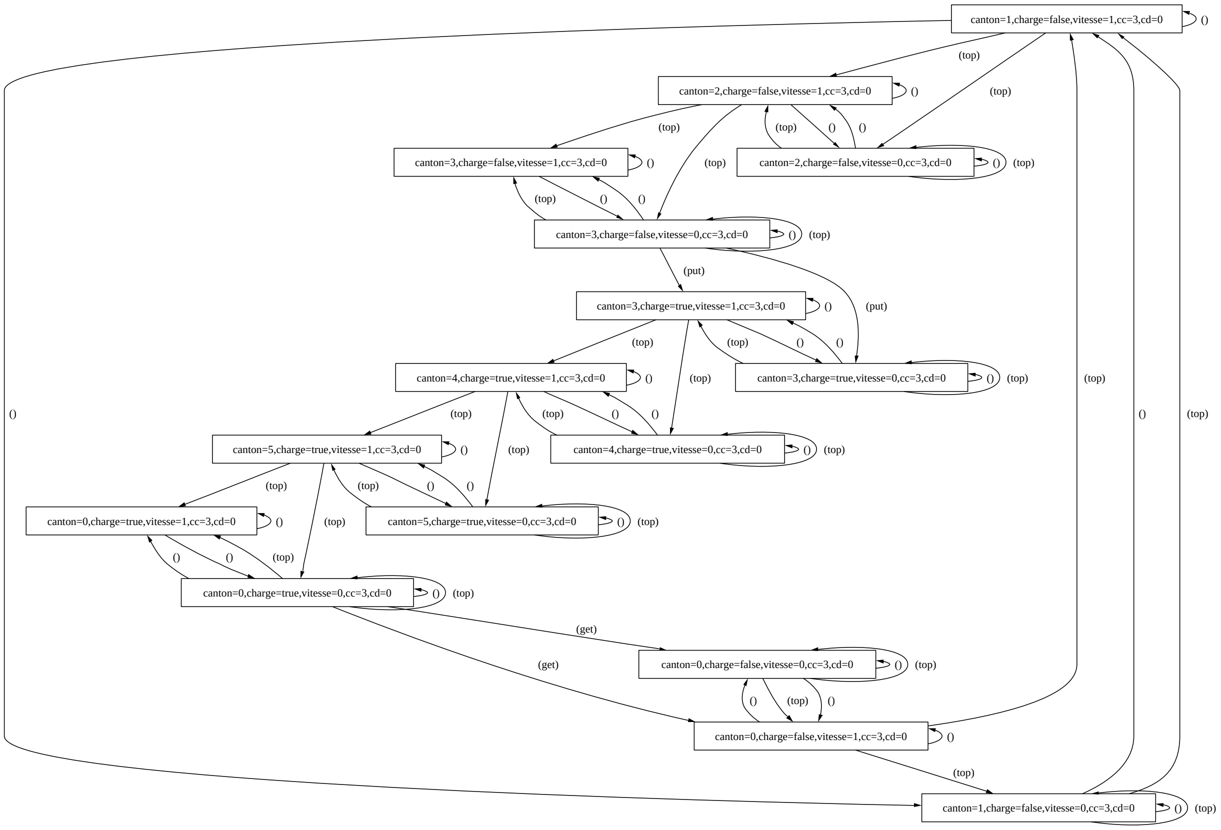

FIG. 2.2 – La seconde version du nœud Train

2.2.3 Les propriétés

Les spécifications

Les résultats

```
TEST(dead=0) [PASSED]
TEST(notCFC=0) [PASSED]
```

Interprétations

```
/*
 * # state properties : 3
 *
 * initial = 2
 * dead = 0
 * any_s = 16
 *
 * # transition properties : 6
 *
 * not_deterministic = 64
 * self = 24
 * notCFC = 0
```

```
 * any_t = 64
 * self_epsilon = 16
 * epsilon = 32
 */
```

Question 2.2 *Expliquez la modélisation de la seconde version du Train.*

Réponse à la question 2.2 :

Chapitre 3

Le système complet

3.1 Composition d'un système

Le systeme est composé :

de 11 cantons qui représentent les onze portions de voies ferrées sur lesquelles circulent les trains. Chaque canton est simplement représenté par un numéro.

de 3 trains circulant sur les cantons.
- le train T1 circule en boucle sur les cantons numérotés entre 0 et 5.
- le train T2 circule en boucle sur les cantons numérotés entre 5 et 10.
- le train T3 circule en boucle sur les cantons numérotés entre 0 et 10.

de 4 gares positionnées sur les cantons.
- la gare G1 sur le canton 0, qui permet au train T1 de décharger sa marchandise.
- la gare G2 sur le canton 3, qui permet aux train T1 et T3 de charger la marchandise.
- la gare G3 sur le canton 6, qui permet au train T2 de charger sa marchandise.
- la gare G4 sur le canton 9, qui permet aux train T2 et T3 de décharger la marchandise.

d'un contrôleur représenté dans le modèle par le système lui-même ayant à disposition : d'une part des commandes possibles ; d'autre part une observation totale du système.

Question 3.1 *Précisez et justifiez votre choix pour une version du modèle de Train.*

Réponse à la question 3.1 :

3.2 La première version du nœud Systeme

3.2.1 Le source AltaRica

```
node Systeme_V1
  sub
    T1 : Train1_V1;
    T2 : Train2_V1;
    T3 : Train3_V1;
  event
    listeEvenementsControles
  trans
    true |-
      listeEvenementsControles
      -> ;
```

```
  //sync
    // synchronisations correspondantes a "listeEvenementsControles"

  //assert
   // Des contraintes eventuellement

edon
```

3.3 La seconde version du nœud Systeme

3.3.1 Le source AltaRica

```
node Systeme_V2
  sub
    T1 : Train1_V2;
    T2 : Train2_V2;
    T3 : Train3_V2;
  state
    v1, v2, v3 : [0,1];
  init
    v1 := 0, v2 := 0, v3 := 0;
  event
    listeEvenementsControles
  trans
    true |-
      listeEvenementsControles
      -> ;

  //sync
    // synchronisations correspondantes a "listeEvenementsControles"

  //assert
    // v1, v2 et v3 sont les vitesses des trains
    // D'autres contraintes eventuellement
edon
```

3.4 Le calcul des contrôleurs

L'approche utilisée est la suivante

1. Pour l'ensemble des exigences, calculer le plus grand contrôleur global.
2. Tester si la configuration initiale appartient au contrôleur. Cela indique l'existence ou pas d'un contrôleur.
3. Tester si le contrôleur peut faire des erreurs. Cela indique si ce contrôleur respecte les exigences.
4. Projeter le contrôleur calculé sur le système global.
5. S'il existe, choisir le contrôleur qui satisfait le plus d'exigences, et le substituer au système global ; puis recommencer l'analyse.

3.4.1 Les spécifications des contrôleurs

```
with Systeme_V1, Systeme_V2 do
```

```
 quot()                    >  '$NODENAME.dot';
 dead                      := any_s - src(any_t - self_epsilon);
 notCFC                    := any_t - loop(any_t,any_t);
 nonControle               := epsilon;
 controle                  := any_t - nonControle;
 ER                        := dead;

/* Les equations classiques
 *  - de strategies gagnantes
 *  - et de synthese de controleurs
 *  coupA represente les actions du joueur A ou du controleur
 *  coupB represente les actions du joueur B ou du systeme
 *  Gagne represente des configurations de A
 *    - les configurations a atteindre lors d'un pppf
 *    - le complementaire des configurations a eviter lors d'un pgpf
 *  Perdu represente des configurations de B
 *    - les configurations a eviter lors d'un ppf
 *    - le complementaire des configurations a eviter lors d'un pgpf
 Gagnant                 = Gagne |& \\ | pour pppf et & pour pgpf
                           src(CoupGagnant);
 CoupGagnant             = coupA & rtgt(Perdant);
 Perdant                 = Perdu |& \\ | pour pppf et & pour pgpf
                           (src(CoupPerdant) - src(CoupNonPerdant));
 CoupPerdant             = coupB & rtgt(Gagnant);
 CoupNonPerdant          = coupB - rtgt(Gagnant);
*/

 // les actions de controle pour eviter ER : pgpf
 GagneER := src(controle) - ER;
 PerduER := src(nonControle) - ER;
 CtrlER  -= controle &
            rtgt(PerduER &
                 (src(nonControle & rtgt(GagneER & src(CtrlER)))-
                  src(nonControle - rtgt(GagneER & src(CtrlER)))));

 // Le systeme est-il controlable
 ControlableER             := initial & src(CtrlER);
 // les erreurs de commande possibles
 ErreurControleER          := controle - CtrlER;
 // Generation des controleurs
 project(any_s, CtrlER, '$NODENAME_Controleur', true)
        > 'Alt/$NODENAME_Controleur.alt';

 // les actions de controle pour optimiser la production et eviter ER
 //       A FAIRE

 show(all)                 >  '$NODENAME.prop';
 test(dead,0)              >  '$NODENAME.res';
 test(notCFC,0)            >> '$NODENAME.res';
 test(ControlableER,1)            >> '$NODENAME.res';
 test(ErreurControleER,0)            >> '$NODENAME.res';
done
```

3.4.2 La première version du nœud Systeme

3.4.3 Les propriétés

Les résultats

Interprétations

3.5 La seconde version du nœud Systeme

3.5.1 Les propriétés

Les résultats

Interprétations

Question 3.2 *Commentez les résultats.*

Réponse à la question 3.2 :

Chapitre 4

Le système contrôlé

4.1 La première version du nœud Systeme

4.1.1 Le source AltaRica

4.1.2 Les propriétés

Les résultats

Interprétations

4.2 La seconde version du nœud Systeme

4.2.1 Le source AltaRica

4.2.2 Les propriétés

Les résultats

Interprétations

Question 4.1 *Commentez les résultats.*

Réponse à la question 4.1 :

Chapitre 5

Le système contrôlé et optimisé

5.1 La première version du nœud Systeme

5.1.1 Le source AltaRica

5.1.2 Les propriétés

Les résultats

Interprétations

5.2 La seconde version du nœud Systeme

5.2.1 Le source AltaRica

5.2.2 Les propriétés

Les résultats

Interprétations

Question 5.1 *Commentez les résultats.*

Réponse à la question 5.1 :