Spécification d'un carrefour à deux feux tricolores

- **Le système de feux du carrefour peut être :**
  - **en service :**
    - la couleur de chacun des feux change suivant le cycle orange puis rouge puis vert puis orange, etc ...
    - les feux doivent être réglés de façon à ce que deux véhicules venant de voies différentes ne se trouvent pas en même temps sur le carrefour
  - **hors service** : les deux feux sont oranges

Spécification d'un carrefour à deux feux tricolores

- **Identification des attributs ou états de la machine Carrefour**:
  - Etat du carrefour: en service ou non -> **Etat**
    Un élément de l'ensemble Etats= {hs,es}
  - Les deux feux ou plutôt leurs couleurs: **feuA, feuB**
    Deux éléments de l'ensemble Couleurs = {vert, orange, rouge}

- **Définition des relations entre les couleurs**: une fonction **succ: Couleurs -> Couleurs**
  telle que succ(Orange) = rouge, succ(rouge)= vert, succ(vert)= orange

Spécification d'un carrefour à deux feux tricolores

- **Invariants:**
  - Lorsque le système est hors service, les deux feux sont oranges

    $(etat = hs)\ ) \Rightarrow (feuA = orange \wedge feuB = orange)$

  - Lorsque le système est en service, les feux doivent être réglés de façon a ce que deux véhicules venant de voies différentes ne se trouvent pas en même temps sur le carrefour

$(etat=es) \Rightarrow ((feuA=rouge \vee feuB=rouge) \wedge feuA/=feuB)$

Spécification d'un carrefour à deux feux tricolores

Machine *CARREFOUR*
sets *Etats = {hs,es} ; Couleurs = {vert,orange,rouge}*
constants *Succ*

Constraints *(Succ ϵ Couleurs -> Couleurs) Λ (Succ(orange) = rouge) Λ (Succ(rouge) = vert) Λ (Succ(vert) = orange)*
variables *etat, feuA, feuB*
Invariant
(etat ϵ Etats Λ feuA ϵ Couleurs Λ feuB ϵ Couleurs) Λ
(etat = hs)) => (feuA = orange Λ feuB = orange) Λ
(etat=es) => ((feuA=rouge v feuB=rouge) Λ feuA /= feuB)

Initialization *etat, feuA, feuB* := *hs, orange, orange*

Operations
MiseEnService =
pre *etat = hs* **then**
any *f1, f2* **where** (*f1 ϵ Couleurs Λ f2 ϵ Couleurs) Λ* (*f1 = rouge V f2 = rouge*) *Λ f1 /= f2*
then *etat, feuA, feuB* := *es, f1, f2* **End**

Changement =
pre *etat = es* **then**
any *f1; f2* **where** (*f1 ϵ Couleurs Λ f2 ϵ Couleurs) Λ* (*f1 = rouge V f2 = rouge*) *Λ f1 /= f2 Λ*
(*f1 = Succ*(*feuA*)) *V* (*f2 = Succ*(*feuB*)) propriété de vivacité: au moins un des deux feux change
then *feuA, feuB* := *f1, f2* **End**

Raffinement

- C’est la caractéristique principale de la méthode B: passer d’une spécification abstraite à une autre qui l’est moins tout en conservant les propriétés.

- Le raffinement d’une machine peut se faire en considérant:
  - Les données: ajouter d’autres attributs,
  - Les opérations: affaiblissement des préconditions ou levée de l’indéterminisme

- Un raffinement n’est un raffinement correct que s’il conserve la validité des propriétés de la machine d’origine

Machine *CARREFOUR*
sets *Etats = {hs,es}; Couleurs = {vert,orange,rouge}*
constants *Succ*

Constraints *(Succ ϵ Couleurs -> Couleurs) Λ (Succ(orange) = rouge) Λ (Succ(rouge) = vert) Λ (Succ(vert) = orange)*
variables *etat, feuA, feuB*
Invariant
(etat ϵ Etats Λ feuA ϵ Couleurs Λ feuB ϵ Couleurs) Λ
(etat = hs)) => (feuA = orange Λ feuB = orange) Λ
(etat=es) => ((feuA=rouge v feuB=rouge) Λ feuA /= feuB)

Initialization *etat, feuA, feuB* := *hs, orange, orange*
Operations
MiseEnService =
pre *etat = hs* **then**
any *f*1, *f*2 **where** (*f*1 *ϵ Couleurs Λ f*2 *ϵ Couleurs) Λ*

(*f*1 = *rouge V f*2 = *rouge*) *Λ f*1 */= f*2
then *etat, feuA, feuB* := *es, f*1, *f*2 **End**

Changement =
pre *etat = es* **then**
any *f*1*; f*2 **where** (*f*1 *ϵ Couleurs Λ f*2 *ϵ Couleurs) Λ* (*f*1 = *rouge V f*2 = *rouge*) *Λ f*1 */= f*2 *Λ*
(*f*1 = *Succ*(*feuA*)) *V* (*f*2 = *Succ*(*feuB*))
then *feuA, feuB* := *f*1, *f*2 **End**

Refinement *CARREFOUR*1 **refines** *CARREFOUR*
sets *Etats = {hs,es}; Couleurs = {vert,orange,rouge}*

variables *etat1, feuA1, feuB1*
Invariant **de liaison**

etat1=etat *Λ feuA1=feuA Λ feuB1=feuB*

Initialization *etat1, feuA1, feuB1* := *hs, orange, orange*

Operations

MiseEnService = **on fixe un au rouge et l'autre au vert**
pre *etat*1 = *hs* then
*etat*1, *feuA*1, *feuB*1 := *es, rouge,* vert
End

Changement = **on décrit la fonction succ**
pre *etat*1 = *es* **then**
choice
when *feuA*1 = *vert* **then** *feuA*1 := *orange* **or**
when *feuA*1 = *orange* **then** *feuA*1*; feuB*1 := *rouge; vert* **or**
when *feuA*1 = *rouge ^ feuB*1 = *vert* **then** *feuB*1 := *orange* **or**
when *feuA*1 = *rouge ^ feuB*1 = *orange* **then** *feuA*1*; feuB*1 := *vert; rouge*
End

Correction du raffinement

- **Intuitivement:** *R* est un raffinement correct de *M*, s'il vérifie encore les propriétés que vérifie M. En particulier, *R* doit satisfaire, en quelque sorte, *l'invariant.*

- **Formellement:** *des obligations de preuves sont engendrées pour vérifier l'initialisation et les opérations. Soit* ***J*** *l'invariant de liaison*
  - [init-raffinement] non([init]) non(J)
  - Pre Λ inv Λ J => Pre-raffinement Λ [s-raffinement] non([S]) non(J)

- Bibliographie:

J.-Y. Chauvet, *1st Conference on the B method, Proceedings, ed. Henri Habrias, Nantes*