



Méthodes formelles

Synthèse des TPs de logique temporelle

Jean-Marie Le Yaouanc

Brest, 15 décembre 2004

Enseignant responsable : Mr Lionel Marcé

Département informatique

UFR Sciences et Techniques 6 avenue Le Gorgeu 29200 Brest

Master 1 informatique

Table des matières

1	Introduction à <i>Kronos</i>	1
2	Synthèse du TP1	3
2.1	Exercice 1	3
2.1.1	Modéliser l'automate de la figure 2.1.1	3
2.1.2	Vérifiez les formules	4
2.2	Exercice 2	4
2.2.1	Modéliser l'automate de la figure 2.2.1	4
2.2.2	A l'aide de l'outil <i>Kronos</i> , modélisez et vérifiez les propriétés suivantes :	6
2.2.3	Modéliser et vérifier la propriété suivante : si le système est à l'arrêt alors on aura obligatoirement une mise en route dans le futur.	7
3	Synthèse du TP2 : modélisation d'un passage à niveau	9
3.1	Modéliser les automates du train et de la barrière en <i>Kronos</i> puis effectuer leur produit synchronisé	9
3.2	Modéliser et vérifier les formules	11
4	Synthèse du TP3 : modélisation et vérification complète	13
4.1	Modéliser le système en utilisant le produit synchronisé	13
4.2	Modéliser et vérifier des formules	15
A	Annexes	19
A.1	Mise en oeuvre du passage à niveau	19
A.2	Mise en oeuvre de la porte à ouverture automatique	20

Table des figures

2.1	Automate non temporisé	4
2.2	Automate temporisé	5
3.1	Automate représentant le passage à niveau complet	11
4.1	Automate de la porte	14
4.2	Automate de l'individu franchissant la porte	15
4.3	Automate du système général	16

Chapitre 1

Introduction à *Kronos*

Kronos est un outil de model-checking. il permet de vérifier les systèmes temps réel à l'aide de formules de la logique TCTL. TCTL est une extension de la logique CTL vue en cours.

Le principe de la vérification de système est que l'on modélise le système sous forme d'automate puis on vérifie des propriétés modélisées sous forme de formules logiques.

Nous allons donc appliquer des formules CTL sur des structures représentées par des automates.

Chapitre 2

Synthèse du TP1

2.1 Exercice 1

2.1.1 Modéliser l'automate de la figure 2.1.1

On applique les règles de traduction en *Kronos* et on obtient le code suivant :

```
/*Nb d'états, de transitions et d'horloges*/
```

```
#states 3  
#trans 5  
#clocks 1 h
```

```
/*Description des etats*/
```

```
state: 0  
prop: Chaud Ok  
invar: h<1  
trans:  
true => ; reset{h}; goto 1
```

```
state: 1  
prop: Ok  
invar: h<1  
trans:  
true => ; reset{h}; goto 0  
true => ; reset{h}; goto 2
```

```
state: 2  
prop: Erreur
```

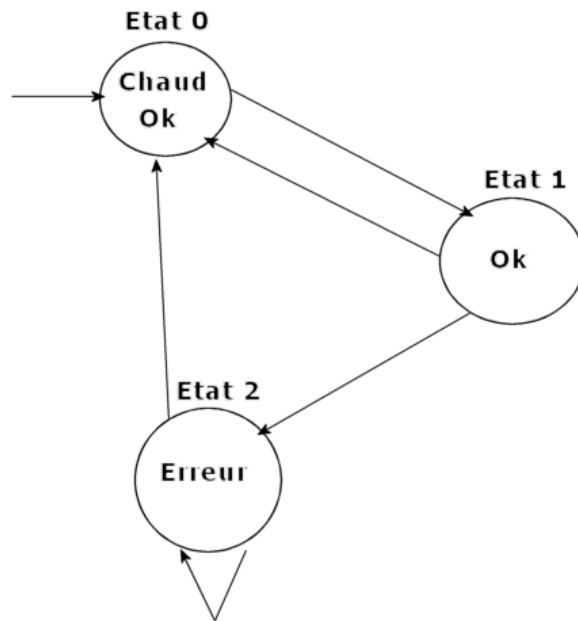


FIG. 2.1 – Automate non temporisé

```

invar: h<1
trans:
true => ; reset{h}; goto 0
true => ; reset{h}; goto 2
  
```

2.1.2 Vérifiez les formules

- $AG(\text{Chaud} \Rightarrow AF(\text{Ok}))$
- $AG(\text{Chaud} \Rightarrow AG(\text{Ok}))$
- $\text{Chaud} \Rightarrow EF(\text{Erreur})$
- $\text{Chaud} \Rightarrow EF(\text{Ok} \wedge \text{Chaud})$
- $(\text{Ok} \vee \text{Chaud}) \Rightarrow EG(\text{Ok})$
- $\text{Ok} \Rightarrow AF(\text{Chaud} \wedge \text{Erreur})$

L'ensemble de ces formules hormi la seconde retournent vrai.

2.2 Exercice 2

2.2.1 Modéliser l'automate de la figure 2.2.1

On applique les règles de traduction en *Kronos* et on obtient le code suivant :

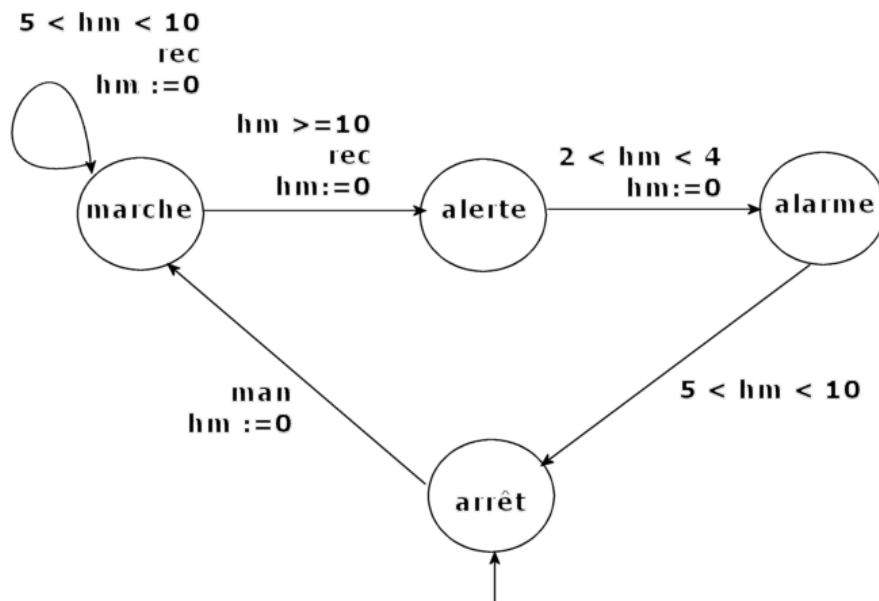


FIG. 2.2 – Automate temporisé

```
/*Nb d'états, de transitions et d'horloges*/
```

```
states 4
#trans 5
#clocks 1 hm
```

```
/*Description des etats*/
```

```
state: 0
prop: arret
invar: true
trans:
true => man; reset{hm}; goto 1
```

```
state: 1
prop: marche
invar: true
trans:
hm > 5 and hm < 10 => rec; reset{hm}; goto 1
hm >= 10 => rec; reset{hm}; goto 2
```

```
state: 2
prop: alerte
```

```

invar: hm<4
trans:
hm>2 and hm<4 =>; reset{hm}; goto 3

state: 3
prop: alarme
invar: hm<10
trans:
hm>5 and hm<10 =>; ; goto 0

```

2.2.2 A l'aide de l'outil *Kronos*, modélisez et vérifiez les propriétés suivantes :

Lorsque le système est à l'arrêt cela implique qu'il existe une exécution dans laquelle le système sera un jour en marche. Cela se modélise en logique TCTL par

```
arret => EF(marche)
```

et donc en *Kronos* par

```
arret impl ed(marche)
```

Dans toutes les exécutions et à tout moment, un état d'alerte implique qu'il y aura forcément une alarme dans le futur. Cela se modélise en logique TCTL par

```
AG(alerte => EF(alarme))
```

et donc en *Kronos* par

```
ab(alerte => ed(alarme))
```

Si le système est en marche, il est possible (il y a au moins une exécution) qu'il reste indéfiniment en marche. Cela se modélise en logique TCTL par :

```
marche => EG(marche)
```

et donc en *Kronos* par

```
marche impl eb(marche)
```

Le résultat pour ces formules est **true**.

2.2.3 Modéliser et vérifier la propriété suivante : si le système est à l'arrêt alors on aura obligatoirement une mise en route dans le futur.

Cela se modélise en logique TCTL par

```
arret => AF(marche)
```

et donc en *Kronos* par

```
arret impl ad(marche)
```

Le résultat obtenu est le suivant :

```
state: 1 and TRUE
or
state: 2 and hm < 4
or
state: 3 and hm < 10
```

Ce qui s'interprète : la formule est vraie dans les cas suivants :

- Soit on se trouve dans l'état *marche*
- Soit on se trouve dans l'état *alerte* et l'horloge est inférieure à 4 secondes.
- Soit on se trouve dans l'état *alarme* et l'horloge est inférieure à 10 secondes.

Chapitre 3

Synthèse du TP2 : modélisation d'un passage à niveau

3.1 Modéliser les automates du train et de la barrière en Kronos puis effectuer leur produit synchro- nisé

Comme pour le premier TP, nous formalisons en **Kronos** les deux automates données, en commençant par celui du train :

```
#states 3
#trans 3
#clocks 1 ht
#sync App Exit

/*Description des etats*/

state: 0
prop: Loin
invar: true
trans:
true => App; reset{ht}; goto 1

state: 1
prop: Avant
invar: ht<30
trans:
ht>20 and ht<30 =>; reset{ht}; goto 2
```

10 CHAPITRE 3. SYNTHÈSE DU TP2 : MODÉLISATION D'UN PASSAGE À NIVEAU

```
state: 2
prop: Sur
invar: ht<20
trans:
ht>10 and ht<20 =>Exit;; goto 0
```

Puis celui de la barrière :

```
#states 4
#trans 4
#clocks 1 hb
#sync App Exit

/*Description des etats*/

state: 0
prop: Ouverte
invar: true
trans:
true => App; reset{hb}; goto 1

state: 1
prop: Se_baisse
invar: hb<10
trans:
hb>8 and hb<10 =>;; goto 2

state: 2
prop: Baissee
invar: true
trans:
true => Exit; reset{hb}; goto 3

state: 3
prop: Se_leve
invar: hb<10
trans:
hb>8 and hb<10 =>;; goto 0
```

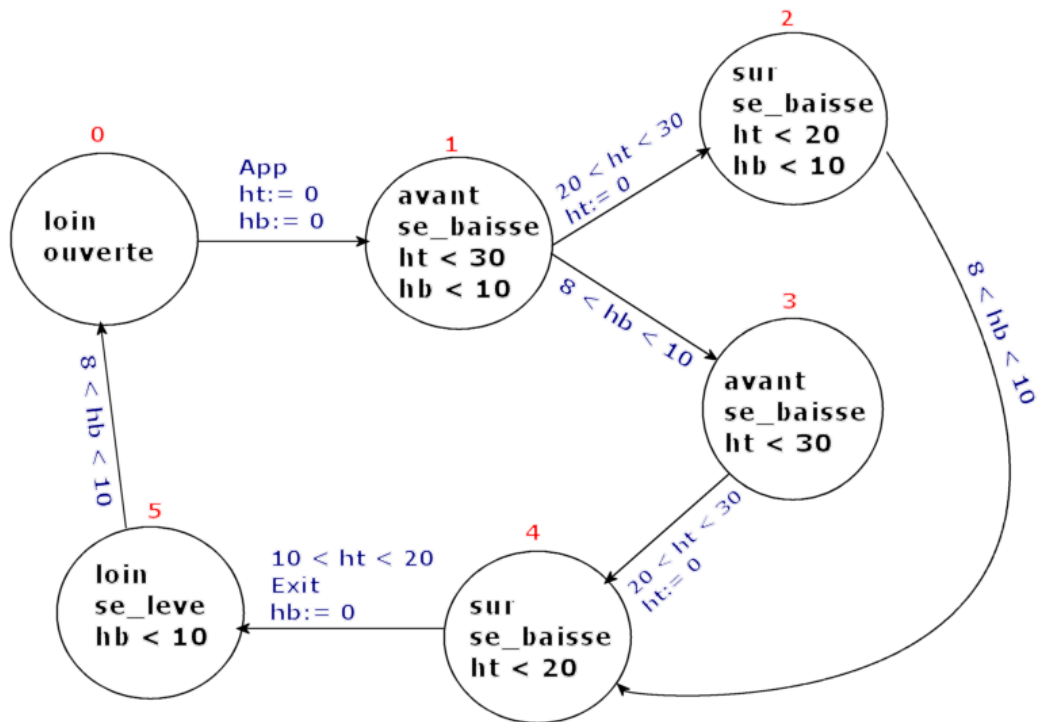


FIG. 3.1 – Automate représentant le passage à niveau complet

Les étiquettes de synchronisations sont placées dans chacun des fichiers. Reste à effectuer le produit synchronisé de ces deux automates grâce à la commande :

```
kronos -out passage.tg train.tg barriere.tg
```

Le contenu du fichier *passage.tg* est disponible en annexes [A.1](#) page 19.

3.2 Modéliser et vérifier les formules

Est-il possible d'avoir le train sur le passage et la barrière ouverte ?

Ceci se modélise par :

```
ed(Sur and Ouverte)
```

La réponse de **Kronos** est **false**

12 CHAPITRE 3. SYNTHÈSE DU TP2 : MODÉLISATION D'UN PASSAGE À NIVEAU

Si le train est avant la barrière , il passera obligatoirement sur la barrière

Ceci se modélise par :

```
ab(Avant impl ed(Sur))
```

La réponse de **Kronos** est **true**

Si le train passe, c'est à dire qu'il part de l'état initial, la barrière est fermée

Ceci se modélise par :

```
ab(Loin and Ouverte) impl ed(Baissee ans Sur)
```

La réponse de **Kronos** est **true**

S'il n'y a pas de train avant ou sur le passage, la barrière sera soit ouverte ou se levant

Ceci se modélise par :

```
ab((not Avant or Sur) impl ed(Ouverte or Se_leve))
```

La réponse de **Kronos** est **true**

Si le train est avant la barrière, cela implique que la barrière sera baissée dans un temps inférieur à 10 secondes

Ceci se modélise par :

```
ab(Avant impl ed{<10}Baissee)
```

La réponse de **Kronos** est **true**

S'il n'y a pas de train avant ou sur le passage, la barrière finira par s'ouvrir au bout d'un temps inférieur à 10 secondes

Ceci se modélise par :

```
ab((not Avant) or (not Sur) impl ed{<10}(Se_leve))
```

La réponse de **Kronos** est **false**

Chapitre 4

Synthèse du TP3 : modélisation et vérification complète

4.1 Modéliser le système en utilisant le produit synchronisé

Après avoir dessiné les différents automates, la définition en **Kronos** devient facile ; Pour la porte :

```
#states 4
#trans 5
#clocks 1 hb
#sync Ouv

/*Description des etats*/

state: 0
prop: Fermee
invar: true
trans:
true =>Cap; reset{hb}; goto 1

state: 1
prop: S_ouvre
invar: hb<5
trans:
hb>3 and hb<5 =>;reset{hb} ; goto 2

state: 2
```

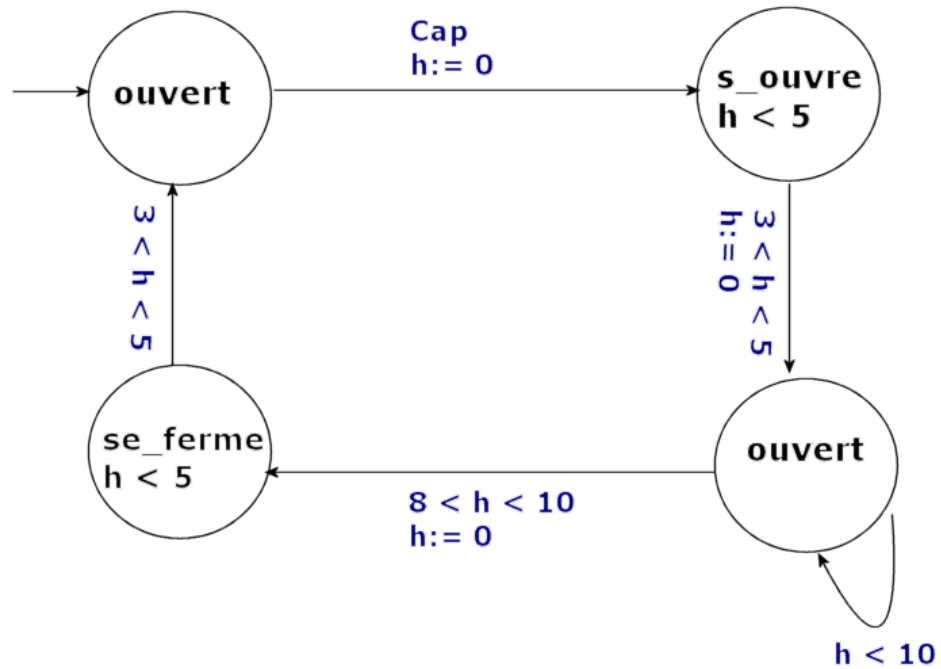


FIG. 4.1 – Automate de la porte

```

prop: Ouvert
invar: hb<10
trans:
true =>Ouv;; goto 2
hb>8 and hb<10 =>;reset{hb} ; goto 3

state: 3
prop: Se_ferme
invar: hb<5
trans:
hb>3 and hb<5 =>;; goto 0

```

Puis pour l'individu :

```

#states 2
#trans 2
#clocks 0
#sync Ouv

/*Description des etats*/

```

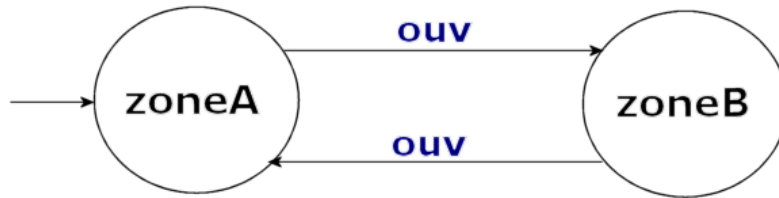


FIG. 4.2 – Automate de l'individu franchissant la porte

```

state: 0
prop: zoneA
invar: true
trans:
true =>ouv;; goto 1

state: 1
prop: zoneB
invar: true
trans:
true =>ouv;; goto 0
  
```

La seule étiquette de synchronisation est **ouv**. Reste à effectuer le produit synchronisé de ces deux automates grâce à la commande :

```
kronos -out systeme.tg man.tg porte.tg
```

Le contenu du fichier *systeme.tg* est disponible en annexes [A.2](#) page 20.

4.2 Modéliser et vérifier des formules

Si l'utilisateur est dans la zone A, il est possible qu'il atteigne la zone B

Ceci se modélise par :

```
ab(zoneA impl ed(zoneB))
```

La réponse de **Kronos** est **true**

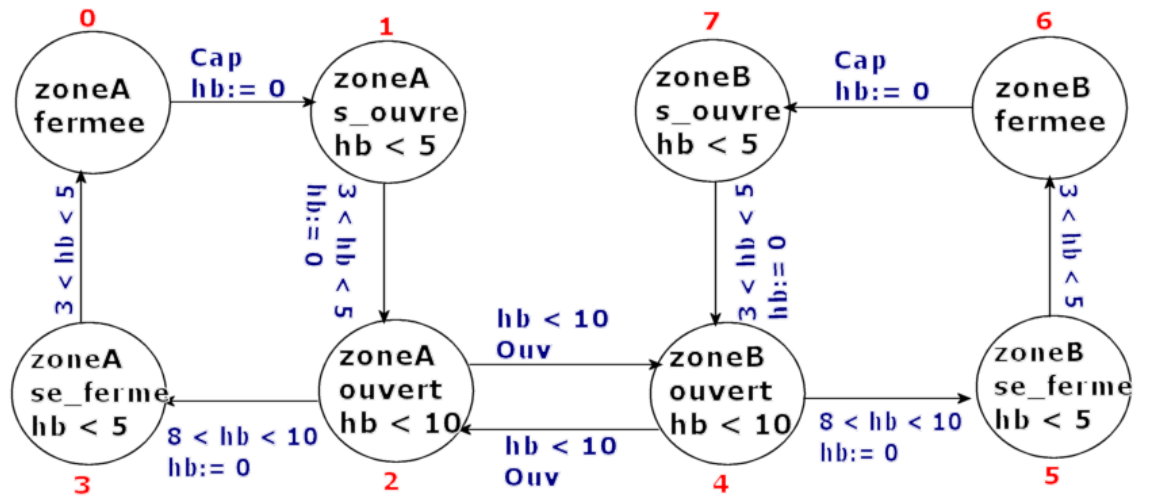


FIG. 4.3 – Automate du système général

Si l'utilisateur est en A, alors il n'existe pas d'exécutions qui permettent de passer en B sans que la porte soit ouverte.

Ceci se modélise par :

```
zoneA impl not((zoneA and not ouverte) impl zoneB)
```

La réponse de **Kronos** est **true**

Si l'utilisateur est en A et que la porte est ouverte, il peut aller en B puis revenir en A avant que la porte ne se ferme

Ceci se modélise par :

```
zoneA and ouverte impl ((zoneB impl zoneA) eu fermee)
```

La réponse de **Kronos** est **true**

La porte s'ouvre toujours en moins de 5 secondes

Ceci se modélise par :

```
ab{<5}ouverte
```

La réponse de **Kronos** est **true**

Si la porte est ouverte, elle restera ouverte au plus 10 secondes avant de se refermer

Ceci se modélise par :

`ab(>10)ouverte impl fermee)`

La réponse de **Kronos** est **true**

Si la porte est ouverte, elle sera fermée dans au plus 15 secondes

Ceci se modélise par :

`ouverte impl ab(<15) fermee)`

La réponse de **Kronos** est **true**

Annexe A

Annexes

A.1 Mise en oeuvre du passage à niveau

```
/* Timed graph generated for the parallel composition of:
automaton 0: train.tg
automaton 1: barriere.tg
*/
#states 6
#trans 7
#clocks 2
HT /* train */
HB /* barriere */

#sync
APP /* train barriere */
EXIT /* train barriere */

state: 0 /* vector state: < 0, 0 > */
prop: LOIN OUVERTE
invar: TRUE
trans:
L: TRUE =C> APP ; RESET{ HT HB }; goto 1

state: 1 /* vector state: < 1, 1 > */
prop: AVANT SE_BAISSE
invar: HT<30 and HB<10
trans:
L: 20<HT and HT<30 =C> ; RESET{ HT }; goto 2
L: 8<HB and HB<10 =C> ; reset{}; goto 3
```

```

state: 2 /* vector state: < 2, 1 > */
prop: SUR SE_BAISSE
invar: HT<20 and HB<10
trans:
L: 8<HB and HB<10 =C> ; reset{}; goto 4

state: 3 /* vector state: < 1, 2 > */
prop: AVANT BAISSÉE
invar: HT<30
trans:
L: 20<HT and HT<30 =C> ; RESET{ HT }; goto 4

state: 4 /* vector state: < 2, 2 > */
prop: SUR BAISSÉE
invar: HT<20
trans:
L: 10<HT and HT<20 =C> EXIT ; RESET{ HB }; goto 5

state: 5 /* vector state: < 0, 3 > */
prop: LOIN SE_LEVE
invar: HB<10
trans:
L: 8<HB and HB<10 =C> ; reset{}; goto 0

```

A.2 Mise en oeuvre de la porte à ouverture automatique

```

/* Timed graph generated for the parallel composition of:
automaton 0: man.tg
automaton 1: porte.tg
*/
#states 8
#trans 10
#clocks 1
HB /* porte */

#sync
OUV /* man porte */

state: 0 /* vector state: < 0, 0 > */

```

A.2. MISE EN OEUVRE DE LA PORTE À OUVERTURE AUTOMATIQUE21

```
prop: ZONEA FERMEE
invar: TRUE
trans:
L: TRUE =C> CAP ; RESET{ HB }; goto 1

state: 1 /* vector state: < 0, 1 > */
prop: ZONEA S_OUVRE
invar: HB<5
trans:
L: 3<HB and HB<5 =C> ; RESET{ HB }; goto 2

state: 2 /* vector state: < 0, 2 > */
prop: ZONEA OUVERT
invar: HB<10
trans:
L: 8<HB and HB<10 =C> ; RESET{ HB }; goto 3
L: HB<10 =C> OUV ; reset{}; goto 4

state: 3 /* vector state: < 0, 3 > */
prop: ZONEA SE_FERME
invar: HB<5
trans:
L: 3<HB and HB<5 =C> ; reset{}; goto 0

state: 4 /* vector state: < 1, 2 > */
prop: ZONEB OUVERT
invar: HB<10
trans:
L: 8<HB and HB<10 =C> ; RESET{ HB }; goto 5
L: HB<10 =C> OUV ; reset{}; goto 2

state: 5 /* vector state: < 1, 3 > */
prop: ZONEB SE_FERME
invar: HB<5
trans:
L: 3<HB and HB<5 =C> ; reset{}; goto 6

state: 6 /* vector state: < 1, 0 > */
prop: ZONEB FERMEE
invar: TRUE
trans:
L: TRUE =C> CAP ; RESET{ HB }; goto 7

state: 7 /* vector state: < 1, 1 > */
```

```
prop: ZONEB S_OUVRE
invar: HB<5
trans:
L: 3<HB and HB<5 =C> ; RESET{ HB }; goto 4
```