Je suis pleinement conscient(e) que le plagiat de documents ou d'une partie de document constitue une fraude caractérisée.
Nom, date et signature :

Compilation and diagnosis for discrete controller synthesis

Seydou Coulibaly
.
Encadré par : Gwenaël Delaval.

Juin 2016

Résumé Cet article présente la compilation du langage Heptagon/BZR utilisant la synthèse des contrôleurs discrets, une méthode de conception qui génère des contrôleurs sur un système pour imposer certaines propriétés. Heptagon/BZR [8] [1] est un langage synchrone flots de données réprésentant les spécifications d'un système sous forme d'automates à état fini. Les résultats possibles de sa compilation sont variés, de l'échec de la synthèse de contrôleurs discrets à la géneration d'une logique de contrôle pour le système. Quand aux causes des échecs de la synthèse de contrôleurs discrets, ils sont multiples et généralement dues au manque de choix disponible des variables contrôlables pour résoudre les contrats imposés sur le système. En effet, la synthèse des contreurs discrets utilisent les variables ou entrées contrôlables pour contôler un système.
Par ailleurs des exemples sont illustrés dans ce document pour la compréhension du langage Heptagon/BZR et nécessaire à sa compilation. Le choix des exemples de systèmes à concevoir reste vastes mais cadré autour des systèmes pouvant être modéliser par des langages synchrones comme par exemple les systèmes automatiques, les systèmes industriels, les automates programmables.

Keywords langage synchrone · compilation · syntheses de controleurs discretes · BZR · Heptagon · Reax · système reactif · contrats · CTRL-A

Seydou coulibaly
Saint martin d'heres, 38400, France
E-mail: cseydou28@gmail.com

1 Introduction

On s'intéresse au diagnostic de la compilation d'un langage synchrone utilisant la synthèse des contrôleurs discrets, une méthode de conception et de validation permettant de contrôler un système. Des outils de synthèse de contrôleurs existent déja dont REAX [2] et SIGALI [5]. REAX est intégré dans un langage de programmation, spécifiquement un langage synchrone et tout au long de ce document, on utilisera le langage Heptagon/BZR qui est un exemple, pour nos analyses et interprétations.

Heptagon/BZR [8] [1] est un langage synchrone flots de données utilisé pour programmer des systèmes réactifs. Il décrit sous forme d'automate les comportements possible d'un système. Lors de sa compilation, différents résultats comme des problèmes de syntaxe, des échecs de la synthèse de contrôleurs discrets, des cas où tout fonctionne bien et des cas où la synthèse reussit bien à générer une logique de contrôle mais que le système controlé obtenu n'arrive pas à repondre à la spécification initiale du système au-delà des contraintes. Ce document décrit ainsi les éventuels résultats possibles de la compilation de la synthèse de contrôleurs discrets.

1.1 La synthèse de contrôleurs discrets

La synthèse de contrôleurs discrets consiste à réduire le comportement d'un système P par le biais d'un superviseur ou contrôleur C de manière à ce que le système ainsi contrôlé soit correct vis à vis d'un ensemble de propriétés D (ou d'objectifs de contrôle) que le système initial ne verifait pas. Plus spécifiquement, c'est une méthode qui genère un contrôleur (une logique de contrôle) sur un système de sorte que le comportement de ce dernier soit conforme à celui désiré [6]. La synthèse de contrôleurs discrets est appliquée sur des systèmes réprésentés par un ensemble de comportements comme sous la forme d'automates finis, de réseaux de pétri ou sous la forme de systèmes de transition symboliques. De tels systèmes sont constitués de sorties et d'entrées ; ces dernières étant classées dans deux catégories, les entrées contrôlables et les les entrées incontrôlables. La synthèse de contrôleurs n'a d'habilité d'agir que sur les entrées contrôlables et n'a aucun effet sur les entrées incontrôlables qui proviennent de l'environement (capteurs, des actions de l'utilisateur, évènements extérieurs au système) [3]. Peu connue des programmeurs dans les entreprises, la synthèse de contrôleurs est apparue dans les années 1980. C'est un domaine de recherche à part entière et reste une des méthodes de conception et de validation dont la conception est complexe. La figuire 1 suivante illustre un contrôleur sur un système dont I(u), I(c) et O correspondent respectivement aux entrrées incontrôlables, aux entrrées contrôlables et à la valeur de sortie.

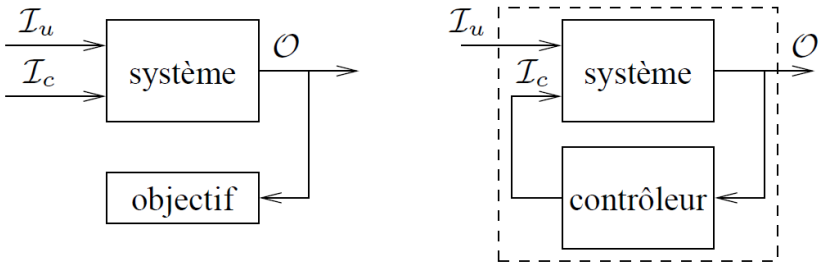


FIGURE 1 Système incontrôlé (à gauche) et contrôlé (à droite) *[4]*

1.2 Le langage Heptagon/BZR

Heptagon/BZR [8] [1] est un langage de programmation de la famille des langages synchrones, des langages développés dans les années 1980 pour programmer des système réactifs. Les systèmes réactifs sont des systèmes de contrôle automatique qui doivent réagir continuellement à leur environnement. Leur vitesse de réaction est imposée par l'environnement qui ne peut pas attendre (ne pas confondre avec les systèmes interactifs : Le rythme de l'interaction est déterminé par le système et non par l'environnement) [7]. Ces systèmes sont le plus souvent déterministes, critiques et exigent la notion de temps-réel.

Heptagon/BZR est un langage synchrone flots de données dont la fonctionnalité principale est l'intégration de la synthèse de contrôleurs discrets dans sa compilation. C'est grâce à un mécanisme de contrat qu'il décrit les contraintes rélatif aux propriétes devant être verifiées par le système contrôlé. Ce mécanisme permet la séparation de la partie modélisation du problème à la partie résolution ou contrat dont la particularité réside dans l'utilisation des contraintes pour réduire la taille de l'espace des solutions du système. La compilation de Heptagon/BZR permet de générer du code C ou JAVA (voir figure 2).

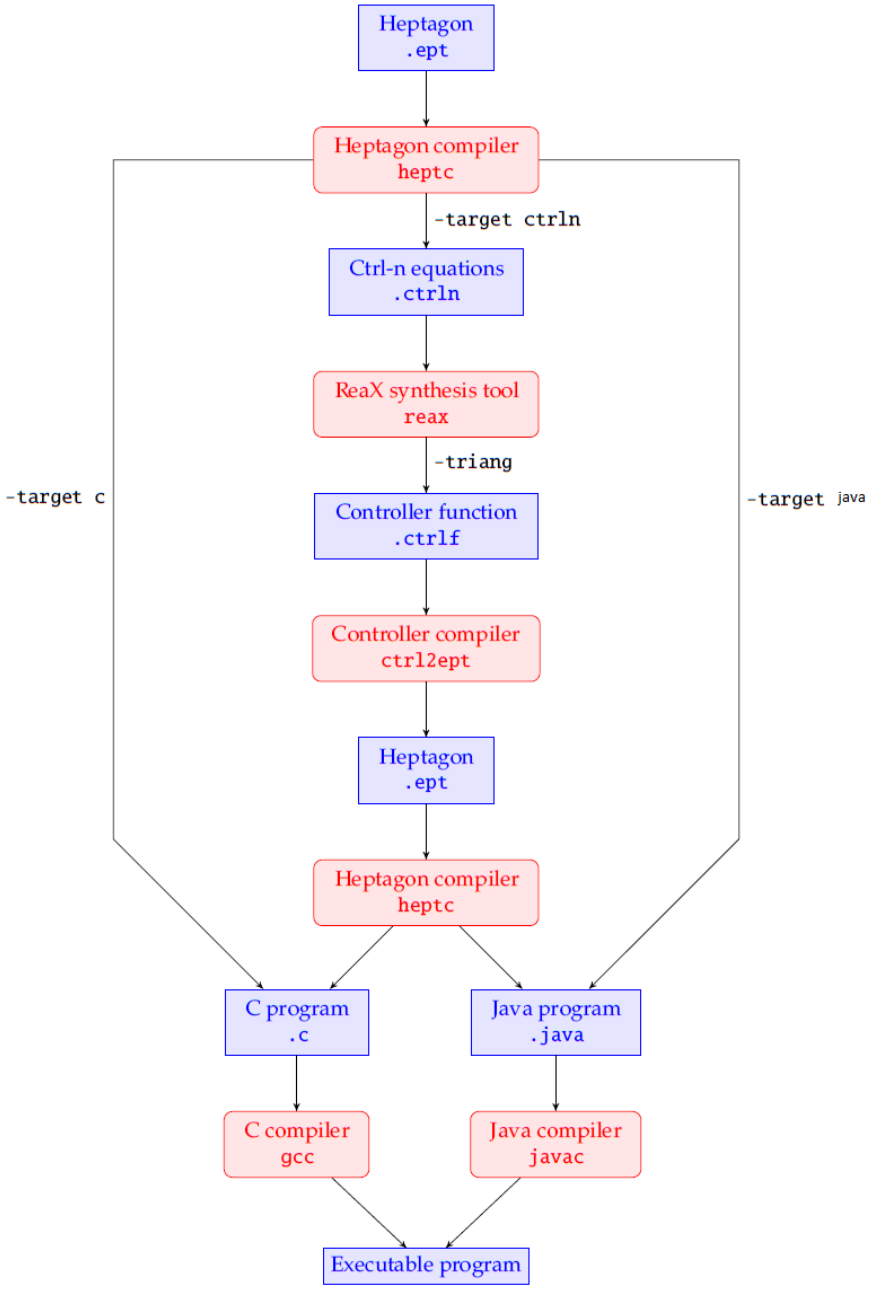


FIGURE 2 Schema de la chaine de la compilation de Heptagon/BZR [8] [1]

2 Diagnostic pour le langage Heptagon/BZR

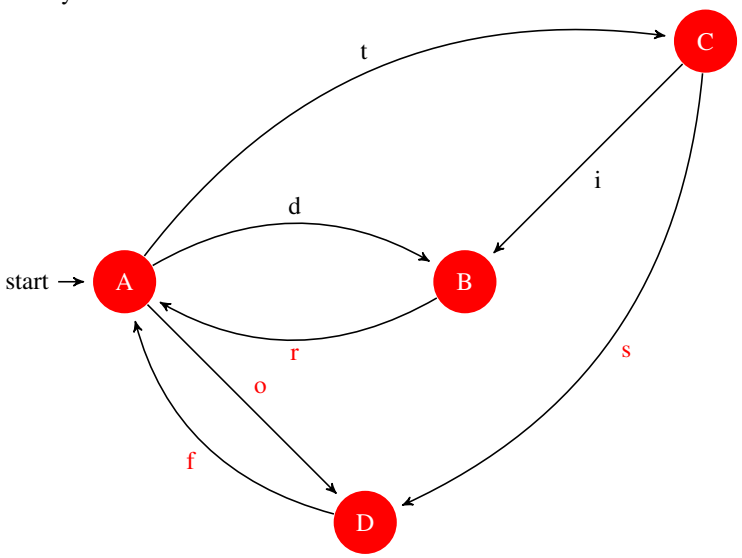
La synthèse de contrôleurs discrets est une méthode de conception et de validation dont l'objectif principal est d'interdire le fonctionnement non désiré d'un système donné.

Pour parvenir à contrôler un système d'une ou des conditions, la synthèse de contrôleurs utilise et n'a l'habilité d'agir que sur des événements ou variables déclarées contrôlables. C'est ainsi grâce à ces variables contôlables qu'il passe d'état en état pour respecter les contraintes ou conditions. Le choix d'activation ou de désactivation d'événements contrôlables et de choix d'états accepteurs à partir de l'état actuelle du système restent donc vastes et on peut souvent arriver à des cas où la synthèse n'a plus d'option pour contraindre une ou des conditions, c'est 'échec de la synthèse des contrôleurs discrets. Il y a plusieurs raisons causant l'échec de la synthèse de contrôleurs discrets et parmi lesquels, on a :

1. Présence d'une ou des propriétés non assurable par la synthèse de contrôleurs discrets ; ses entrées contôlables ne lui permettent pas le choix de décision des états à prendre pour respecter les propriétés en cause ;
2. Présence de conflit entre certaines propriétés, l'échec de la synthèse de contrôleurs discrets le plus difficile à détecter, qui survient lorsque la synthèse est obligée d'ignorer une propriété P2 pour qu'une autre propriété P1 soit validé ;

Exemple

Soit l'automate suivante dont les entrées contrôlables colorée en rouge et les entrées incontrôlables du Système en noir. On souhaite mettre en évidence les deux types d'échecs de la synthèse de contrôleurs discrets cité ci-dessus.



Premièrement, une propriété impossible d'être assuré par la synthèse de contrôleurs serait par exemple d'imposer au système de ne jamais y aller dans l'état à B (aucune variable contrôlable dans le sens A->B pour le contrôle de la proprété imposée, du coup la synthèse échoue).

Et le dernier cas serait par exemple d'imposer au Système les propriétés suivantes :

1. Jamais avoir l'évenement "t" dans l'état D ;
2. Jamais avoir l'évenement "i" dans l'état A ;

La synthèse de contrôleurs discrets produit une logique de contrôle pour chacune des deux propriétés alors qu'elle échoue lorsque les propriétés sont combinées en une seule contrainte. En effet la synthèse de contrôleurs utilise la variable contrôlable "o" dans l'état A pour résoudre la propriété (2) et passer à l'état D. Elle fait de même dans l'état D avec la variable contrôlable "f"pour résoudre la proprété (1) et passer à l'état A. On constate donc l'échec de la synthèse de contrôleur discrets dans la situation où toutes les variables i et t sont actives dans un des états A ou D.

Par ailleurs, il arrive que la synthèse de contrôleurs discrets marche mais qu'il n'arrive tout de même pas à respecter certaines spécifications du système initiale. Ces cas arrivent généralement lorsqu'il veut prevenir d'un éventuel échec. Par exemple lorsqu'un système dans un état A a la possibilité d'aller dans un autre état B, qui passe obligatoirement la main à un état C ne respectant pas une propriété donnée par absence d'événements contrôlables , alors il peut ainsi arriver que le contrôleur interdit au système d'aller de A à B.

Par conséquent, pour rémedier à ce problème d'échec de la synthèse de contrôleurs discrets, il faudra détécter la ou les mauvaises propriétés en les testant une à une, deux à deux jusqu'àu nombre de propriétés disponible. On prends donc, chacune des propriétes et la vérifiée par rapport aux différents états du sysème en imaginant tous les cas possibles c'est à dire qu'on donne différents valeurs aux évenements d'entrées du système pour voir la possibilité de produire une stratégie avec laquelle toutes les cas de tests ou d'imagination sont valides. La stratégie ainsi produite est la logique de contrôle ou le contrôleur pour le système.

3 Cas d'études et méthodes

3.1 Description d'un système de smarthome

On souhaite modéliser un système de gestion de maison, soit une maison intelligente [9], s'inspirant du domaine de la domotique. On a un ensemble de composants automatiques munis d'actionneurs qui sont :

- Des portes, des stores et des poubelles ayant chacun deux états, l'ouverture et la fermeture ;
- Des lampes qui s'allument ou s'éteignent
- Des ascenseurs dont les états possibles sont l'arrêt et le mouvement ;
- Une alarme qui protègent une habitation divisée en deux parties (maison et garage). On suppose que la maison est toujours sous surveillance dès que le dispositif de protection est mis sous tension. Le garage n'est sous protection qu'au bout d'un certain temps de vigilence à partir du moment où le dispositif est sous tension ; ce délai permet au propriétaire d'avoir le temps de sortir du garage après avoir armé son dispositif de protection. Son fonctionnement est de sorte qu'à partir du moment où elle sonne, elle le fait pendant un certain délai de reprise et à la fin de ce délai, selon la situation (présence ou non intrus dans la maison) sonne à nouveau ou non. Le dispositif ne peut être arrêté lorsqu'il est sous tension qu'en fournissant le code d'entrée. On dispose alors d'un délai de vigilence entre le moment où le code est fourni et la mise hors-circuit pendant lequel une présence dans le garage est tolérée. *On utilisera des constantes pour les différents délais.*

Les différents capteurs du système sont :

— Un détecteur de presence dans la maison ;
— Un détecteur de presence dans l'ascenseur ;
— Un détecteur de presence dans le garage ;
— Un détecteur de presence devant la poubelle ;
— Un détecteur de lumière, soit la quantité de soleil en dehors de la maison ;
— Un détecteur de vent, soit la quantité de vent en dehors de la maison ;
— Un détecteur du code d'alarme ;
— Un détecteur du code de la porte ;
— Un détecteur du code de la barrière pour le garage ;
— Un déctecteur de présence suivant le sens (entrée ou sortie) devant une porte ou une barrière.

Les actionneurs du système sont :

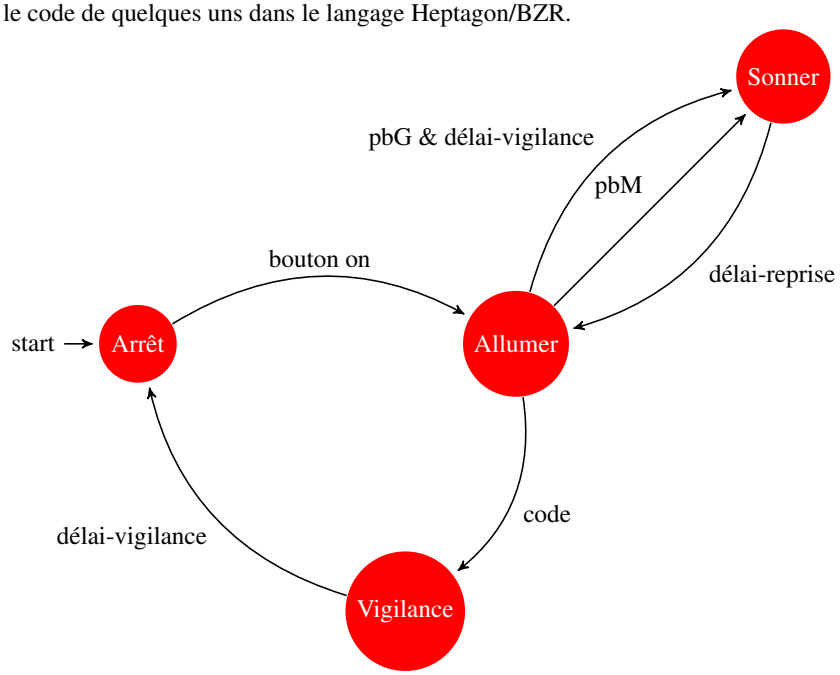
— L'interrupteur de la lampe ;
— Bouton d'ouverture et de fermeture de la stores, de la poubelle ;
— Bouton d'ouverture de la porte de maison et du garage ;
— bouton de demande d'étage pour l'ascenseur.

L'objectif est de décrire une modélisation du système envoyant en sortie l'état de ses composants tels que l'état de la maison, de la lampe, de la poubelle, du store, de la porte, de la barrière et l'alarme afin de pouvoir simuler quelques spécifications cités ci-dessus.

3.2 Modélisation

Suite à la description formelle, on décrit le système sous forme d'automate pour pouvoir le modélisé dans le langage Heptagon/BZR. L'automate ci-dessous décrit le composant d'alarme du système et ceux des autres composants du système figurent en annexe ainsi que

le code de quelques uns dans le langage Heptagon/BZR.




```
type lum = Rouge | Vert | Jaune
node alarme (marcheArret, code, pbGar, pbHab: bool; dReprise, dVigilence,
             dAlarme: int) returns (sonnerAlarme: bool; enMarche: lum)
var
last temps: int = 0;
last vigilence: int = 0;
last reprise: int = 0;
let
        automaton
                state Arret
                        do
                                sonnerAlarme = false;
                                enMarche = Rouge;
                        unless marcheArret then Allume
                state Allume
                        do
                                enMarche = Vert;
                                sonnerAlarme = false;
                                temps = 0 fby (temps +1);
                        until code then Vigilence
                        | pbHab then Sonner
                        | pbGar & (dVigilence <= temps) then Sonner
                state Sonner
                        do
                                enMarche = Vert;
```

```
                                sonnerAlarme = true;
                                reprise = 0 fby (reprise +1);
                        until code then Arret
                        |dReprise <= reprise then Allume
                state Vigilence
                        do
                                enMarche = Jaune;
                                sonnerAlarme = false;
                                vigilence = 0 fby (vigilence +1);
                        until pbHab then Sonner
                        | dVigilence <= vigilence then Arret
        end;
tel
```

3.3 Simulation et interprétation

On suppose ces propriétes ci-dessous :

1. si la maison est vide alors la lumière est éteinte ;
2. si la maison est vide alors de stores sont fermés ;
3. si la maison est non vide alors la lumière est allumée ou les stores sont fermés ;
4. s'il n'y a pas de vent les stores sont fermés ;
5. s'il y a du soleil alors les stores sont fermés ;
6. s'il y a du vent alors les stores sont fermés ;
7. s'il n'y a pas de soleil les stores sont fermés ;
8. s'il n'y a pas de stores alors la lumière doit être allumer ;
9. s'il n'y a pas de stores et que la maison n'est pas vide alors la lumière est allumée ;
10. s'il y a le store alors la lumière est éteinte ;
11. s'il n'y a personne devant la poubelle alors elle doit être fermée ;
12. s'il y a une personne devant la poubelle alors elle doit être ouverte ;
13. s'il n'y a pas de vent et qu'il y a du soleil alors le store est ouvert ;
14. s'il n'y a pas de vent et qu'il y a du soleil et une présence dans la maison alors le store est ouvert ;
15. s'il n'y a pas de vent et qu'il y a du soleil et personne dans la maison alors le store est fermé ;
16. s'il y a du vent et pas de soleil alors pas le store est fermé ;
17. s'il y a du vent et du soleil alors le store est fermé ;
18. s'il n'y a pas de vent et de soleil alors le store est fermé ;
19. s'il y a une personne derrière la porte alors l'ouvir ;
20. s'il n'y a personne près de la porte alors ne jamais l'ouvrir ;
21. s'il y a une personne devant la porte avec le bon code alors ouvrir la porte ;
22. s'il y a une voiture derrière la barrière alors l'ouvrir ;

23. s'il n'y aucune voiture près de la barrière alors ne jamais l'ouvrir ;
24. s'il y une voiture devant la barrière avec le bon code alors l'ouvir ;
25. si l'ascenseur n'est pas occupé alors pas de mouvement de sa part ;
26. si l'ascenseur est occupé alors il est en mouvement ; (choix de conception)
27. si la maison est non vide alors l'alarme doit être activée.

La compilation de toutes ces propriétés avec la synthèse de contrôleurs discrets échoue alors que chacune des propriétés à lui seul reussie. Grâce au diagnostic, on peut ainsi déduire que le problème (2 des causes) est la raison de l'échec de la synthèse de contrôleurs, qui stipule que plusieurs propriétés peuvent entrer en conflit.

On remarque ainsi, un conflit :
— Entre les propriétés (5) et (6)
— Entre les propriétés (4) et (7)
— Entre les propriétés (2) et (13)
— Entre les propriétés (1), (2) et (8)

On constate que les propriétés 1, 2, 3, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 24, 22, 20, 21, 25, 26 n'ont pas de conflits entre elles ; un contrôleur est générer mais semble avoir un problème avec la spécification initiale comme décrit au niveau de la Diagnostic de contrôleurs discrets ; il fait en sorte de ne jamais se situer dans un état donné (l'état de mise hors tension de l'alarme). Par contre les mêmes propriétes sans la 26 répond à la description décrit ci-dessus. La figure **??** réprend une capture d'écran de la simulation du système conçu.

Remarque : les propriétes 4, 5, 6, 7, 8 et 13 ont été remplacer par d'autres propriétes au cours du développement

4 Conclusion et perspectives

La synthèse de contôleurs discrets se revelent très éfficace dans les langages synchrones comme Heptagon/BZR, l'aidant à mieux concevoir et à contrôler des systèmes réactifs qui sont de nature détermiste, automatique et très critique mais le soucis majeur est que cette méthode peut fournir des résultats assez negatifs de spécification logiciel dans des cas où la synthèse de contrôleurs n'échoue pas mais que le système contrôlé obtenu n'arrive tout de même pas à faire ce que l'on souhaite.

D'autre parts, les causes probables d'échecs de la synthèse de contrôleurs discrets sont liés à des problèmes de conceptions et bornées par un nombre fini dont les plus frequents décrit par le diagnostics décrit ci-dessus. On peut ainsi chercher à implementer par le biais du diagnostic, des algorithmes de détection et de suggestion des différents problèmes de la synthèse de contrôleurs discrets et les intégrer dans la compilation du langage Heptagon/BZR.

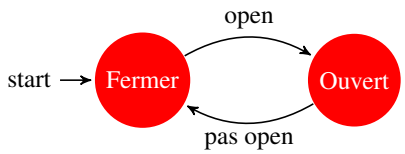
Acknowledgements *Je tiens à rémercier très sincèrement mon maitre de stage, Gwenaël Delaval pour son investissement dans ce travail, un vrai encadreur remarquable ainsi que toute ma famille, mes amis et tous ceux qui ont pu m'inspirer ou aider au cours de cet article de recherche.*

Références

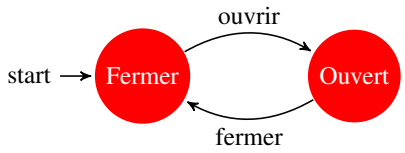
1. Heptagon/bzr manual (2013)
2. Berthier, N., Marchand, H. : Discrete controller synthesis for infinite state systems with reax. In : IEEE International Workshop on Discrete Event Systems, pp. 46–53. Cachan, France (2014)
3. Delaval, G., Rutten, E., Marchand, H. : Integrating discrete controller synthesis in a reactive programming language compiler. Discrete Event Dynamic Systems : Theory and Applications **23**(4), 385–418 (2013)
4. Dumitrescu, E., Girault, A., Marchand, H., Rutten, E. : Optimal discrete controller synthesis for the modeling of fault-tolerant distributed systems. In : First IFAC Workshop on Dependable Control of Discrete Systems (DCDS'07). Paris, France (2007)
5. Marchand, H., Bournai, P., Le Borgne, M., Le Guernic, P. : Synthesis of discrete-event controllers based on the signal environment. Discrete Event Dynamic System : Theory and Applications **10**(4), 325–346 (2000)
6. Michel, L., Gérard, V., Cédric, P., Guillaume, I. : Synthèse de contrôleur simplement valide dans le cadre de la programmation par contraintes (2010)
7. Naim, M.A. : Conception à base de model (2013)
8. Nicolas, B., Gwenaël, D. : Programming and controller synthesis with heptagon/bzr, manual (2013)
9. Sylla, A. : to be publish (2016)

5 Annexe

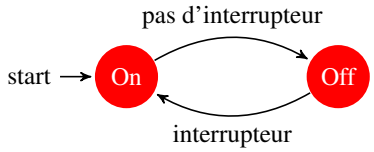
L'automate de la porte



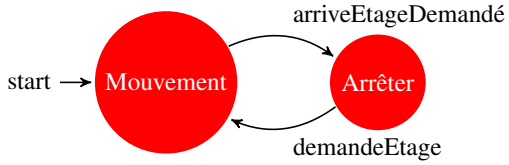
L'automate de la poubelle ou du stores



L'automate de la lampe



L'automate de l'ascenseur



Le code source du cas d'études

```
type position = Avant | Arriere | Null
type porte = Ouvert | Fermer
type lum = Rouge | Vert | Jaune
type stat = Arret | EnMouvement

node lampe(interrupteur:bool) returns (lumiere:bool)
let
```

```
 automaton
        state Off
        do
                lumiere = false;
        unless interrupteur then On

        state On
        do
                lumiere = true;
        unless not interrupteur then Off

 end
tel

node poubelle (con,coff:bool) returns (ouvrir: bool)
let
         automaton
                state Fermer
                do
                        ouvrir = false;
                unless con then Ouvrir

                state Ouvrir
                do
                        ouvrir = true;
                unless coff then Fermer
        end
tel

node porte (copen:bool) returns (ouvrir: porte)
let
         automaton
                state Fermer
                do
                        ouvrir = Ouvert;
                unless copen then Ouvrir

                state Ouvrir
                do
                        ouvrir = Fermer;
                unless not copen then Fermer
        end
tel


node stores (copen,close: bool) returns (store: bool)
let
         automaton
                state Fermer
```

```
                do
                        store = false;
                unless copen then Ouvrir

                state Ouvrir
                do
                        store = true;
                unless close then Fermer
        end
tel


node ascenseur(demandeEtage, arriveEtage:bool) returns (etat: bool)
let
        automaton
                state Stop
                do
                        etat = false;
                unless demandeEtage then Mouvement

                state Mouvement
                do
                        etat = true;
                unless not demandeEtage or arriveEtage then Stop
        end
tel

node smartHome (presenceMaison, presenceAscenseur, presenceGarage,
                presenceDevantPoubelle, vent, luminosite, marcheArretAlarme,
                codeAlarme, codePorte, codeBarriere, presenceEntreePorte,
                presenceSortiePorte, presenceEntreeBarriere,
                presenceSortieBarriere:bool; dReprise, dVigilence,
                dAlarme:int) returns (etatMaison, lumiere, poubelleOuvert,
                ouvertureStore, alarmeSonne, property: bool; porteStatus,
            barriere:porte; etatAlarme:lum; capteurPositionPorte,
            capteurPositionBarriere:position; ascenseur:bool)
contract
assume true
enforce property
with (interrupteur, copen, close, openPorte, openBarriere, con,
      coff, demandeEtage, arriveEtage:bool)

var
loccuperMaison, loccuperGarage, loccuperAscenseur, lpresencePoubelle,
lLumiere, louvrirPoubelle, lstores, lsonnerAlarme, lascenseur,
prop1, prop2, prop3, prop4, prop5, prop6, prop7, prop8, prop8a, prop9,
prop10, prop11, prop12, prop13, prop14, prop15, prop16, prop12a, prop12b,
prop17, prop18, prop19, prop20, prop21, prop22, prop23, prop24,
prop25, prop26:bool;
```

```
louvrirPorte , louvrirBarriere : porte ;
lenMarche : lum ;
lpositionDevantPorte , lpositionDevantBarriere : position ;

let
        loccuperMaison = inlined presence(presenceMaison);
        loccuperGarage = inlined presence(presenceGarage);
        loccuperAscenseur = inlined presence(presenceAscenseur);

        lLumiere = inlined lampe(interrupteur) ;
        lpresencePoubelle = inlined presence(presenceDevantPoubelle) ;
        louvrirPoubelle = inlined poubelle(con, coff);

        louvrirPorte = inlined porte (openPorte);
        lpositionDevantPorte = inlined presenceSens(presenceEntreePorte ,
                                presenceSortiePorte);

        louvrirBarriere = inlined porte (openBarriere);
        lpositionDevantBarriere = inlined presenceSens(presenceEntreeBarriere ,
                                  presenceSortieBarriere);

        lstores = inlined stores (copen , close);
        (lsonnerAlarme , lenMarche) = inlined alarme(marcheArretAlarme ,
                                      codeAlarme , loccuperGarage , loccuperMaison ,
                                      dReprise , dVigilence , dAlarme);
        lascenseur  = inlined ascenseur(demandeEtage , arriveEtage);

        prop1 = loccuperMaison or not lLumiere;
        prop2 = loccuperMaison or not lstores;
        prop3 = not loccuperMaison or (lstores or lLumiere);

        prop4 = vent or lstores;
        prop5 = not luminosite or lstores;
        prop6 = not vent or not lstores;
        prop7 = luminosite or not lstores;

        prop8 = lstores or lLumiere;
        prop9 = not (not lstores & loccuperMaison) or lLumiere;
        prop10 = not lstores or not lLumiere;

        prop11 = lpresencePoubelle or not louvrirPoubelle;
        prop12 = not lpresencePoubelle or louvrirPoubelle;

        prop13 = not (not vent & luminosite) or lstores;
        prop14 = not (not vent & luminosite & loccuperMaison) or lstores;
        prop15 = not (not vent & luminosite & not loccuperMaison)
                  or not lstores;
        prop16 = not (vent & not luminosite) or not lstores;
        prop17 = not (vent & luminosite) or not lstores;
```

```
prop18 = not (not vent & not luminosite) or not lstores;

prop19 = not (lpositionDevantPorte = Arriere) or
          (louvrirPorte = Ouvert);
prop20 = not (lpositionDevantPorte = Null) or
          (louvrirPorte <> Ouvert);
prop21 = not (codePorte & lpositionDevantPorte = Avant) or
          (louvrirPorte = Ouvert);

prop22 = not (lpositionDevantBarriere= Arriere) or
          (louvrirBarriere = Ouvert);
prop23 = not (lpositionDevantBarriere= Null) or
          (louvrirBarriere <> Ouvert);
prop24 =  not (codeBarriere & lpositionDevantBarriere = Avant)
          or (louvrirBarriere = Ouvert);

prop25 = loccuperAscenseur or  not lascenseur;
prop26 = not loccuperAscenseur or lascenseur;
prop27 = loccuperMaison or (lenMarche <> Rouge);
property = prop1 & prop2 & prop3 & prop9 & prop10 & prop111 &
            prop12 & prop14 & prop15 & prop16 & prop17 &
            prop18 & prop19 & prop24 & prop22 & prop20 & prop23 &
            prop21 & prop25 & prop26;

etatMaison = loccuperMaison;
lumiere = lLumiere;
poubelleOuvert = louvrirPoubelle;
porteStatus = louvrirPorte;
capteurPositionPorte = lpositionDevantPorte;
capteurPositionBarriere = lpositionDevantBarriere;
barriere = louvrirBarriere;
ouvertureStore = lstores;
etatAlarme = lenMarche;
alarmeSonne = lsonnerAlarme;
ascenseur = lascenseur;
```

tel