Environnement de conception d'automatismes discrets basé sur le langage Signal

Patricia Bournai, Michel Le Borgne, Hervé Marchand

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Environnement de conception d'automatismes discrets basé sur le langage Signal

P. Bournai, M. Le Borgne, H. Marchand

N 3254

Septembre 1997

THÈME 1

ISSN 0249-6399

RENNES

Environnement de conception d'automatismes discrets basé sur le langage Signal

P. Bournai, M. Le Borgne, H. Marchand

Thème 1 — Réseaux et systèmes
Projet EP-ATR

Rapport de recherche n° 3254 — Septembre 1997 — 50 pages

Résumé : Nous présentons l'intégration des techniques de vérification et de synthèse de contrôleurs dans l'environnement de programmation Signal à travers la description d'un prototype pour la conception d'automatismes et de logiciels sûrs les implémentant. Ce prototype est validé à travers divers exemples tirés du monde académique.

Mots-clé : Théorie du contrôle, Systèmes dynamiques polynomiaux, Méthodologie synchrone, Signal, Simulation

(Abstract: pto)

This work is supported by Électricité de France (EdF)

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Design environment of discrete controllers based on the Signal language

Abstract: In this report, we present the integration of verification and controller synthesis techniques in the Signal environment through the description of a prototype dedicated to the safe construction of reactive system controllers. This prototype is validated by different academic examples.

Key-words: Control Theory, Polynomial Dynamical Systems, Synchronous Methodology, Signal, Simulation

Introduction

Le langage SIGNAL[25, 26] est destiné à la conception et à la mise en œuvre sûre de systèmes réactifs temps réel. Certains domaines d'application exigent une grande fiabilité et une sûreté de fonctionnement. Ces exigences sont vérifiées *a posteriori* par des techniques de simulation et de vérification de propriétés. La théorie du contrôle [35, 21, 4, 28, 29] sur des systèmes à événements discrets permet de développer peu à peu le système en utilisant des méthodes de construction garantissant *a priori* les propriétés attendues du système global. Ces méthodes permettent de restreindre la phase de validation aux seules propriétés du système non garanties par construction.

La théorie du contrôle repose sur une représentation équationnelle de la partie logique d'un programme SIGNAL sous la forme d'un système dynamique polynomial dans $\mathbb{Z}/3\mathbb{Z}$. À partir d'un tel système, il est possible de synthétiser un contrôleur de manière à ce que le système contrôlé vérifie *a posteriori* un objectif de commande logique.

Parmi les obstacles s'opposant à la diffusion des méthodes formelles de synthèse d'automatismes logiques, un des plus important concerne le caractère abstrait des contrôleurs obtenus. Il n'existe pas de grandeurs simplement appréhendables, comme les constantes de temps ou l'amortissement dans le domaine continu, permettant de les qualifier. Dans notre théorie du contrôle des systèmes à événements discrets[22, 14, 32], les contrôleurs (ou superviseurs suivant la terminologie) se présentent le plus souvent sous forme de BDD/TDD (binary decision diagrams/ternary decision diagrams)[13, 31], ou d'automates impossibles à visualiser de manière concrète; le nombre de noeuds pour les uns et le nombre d'états pour les autres étant trop importants. C'est pourquoi, il apparaît utile de développer un logiciel permettant de visualiser le résultat de la synthèse par simulation interactive du système contrôlé. Ce logiciel met également en œuvre une interface intégrée de SIGNAL et SIGALI[13, 15], le logiciel de calcul formel dédié à SIGNAL.

L'intégration de SIGALI dans l'environnement SIGNAL se fait à deux niveaux. Tout d'abord, dans un souci d'expressivité, il semblait intéressant de faciliter l'expression des propriétés en s'abstrayant du modèle des systèmes dynamiques polynomiaux. Un ensemble de directives, permettant d'exprimer des propriétés sur les programmes spécifiés, a donc été intégré dans le langage SIGNAL. Ces propriétés peuvent ensuite être vérifiées ou synthétisées à l'aide du logiciel de calcul formel SIGALI. De plus, une fois un contrôleur obtenu satisfaisant à la fois des objectifs logiques et des objectifs d'optimalité, il est possible que les équations (éventuellement dynamiques) de contraintes supplémentaires ne soient pas suffisantes pour déterminer de façon unique les entrées du processus à contrôler. Nous avons donc développé un résolveur automatique d'équations algébriques permettant de résoudre, pas à pas, l'indéterminisme de manière à pouvoir simuler le système contrôlé.

Ce rapport s'organise comme suit. Dans un premier temps, nous expliquerons ce que nous entendons par synthèse sur un programme SIGNAL dans le but d'obtenir un contrôleur implémentable, ou du moins un exécutable permettant de simuler le comportement d'un système réel. Nous présenterons dans un deuxième temps les différentes directives qui ont été rajoutées au langage SIGNAL. Une description du prototype de simulation et de son implémentation en SIGNAL sera ensuite présentée. Finalement, diverses exemples académiques viendront illustrer ce prototype.

1 Présentation des différents outils existants

1.1 Le Langage Signal

SIGNAL est un langage de programmation et de spécification de systèmes réactifs[1] temps-réel. Tout comme LUSTRE[17, 18] ou ESTEREL[7, 11], il fait partie de la famille des langages synchrones[6][5][16]: on fait l'hypothèse que toutes les actions d'un programme (calculs et communications) sont de durée nulle. Bien entendu cette hypothèse n'est qu'une approximation de la réalité, mais elle est valable dès que l'on peut assurer que le programme agit à une vitesse plus rapide que celle de son environnement. Ainsi, pour garantir cette condition, il est nécessaire que le temps de calculs et de communications soit borné. De plus, l'hypothèse synchrone permet aussi de simplifier les raisonnements sur le temps. Les instants considérés dans de tels systèmes sont des instants logiques. Le comportement d'un système dans le temps est alors considéré comme une suite ordonnée d'instants logiques. Pour chacun de ces instants logiques, la valeur d'un signal se rapportant au système peut être présente ou non. La manipulation du temps est naturelle et exacte, et les propriétés temporelles du système sont vérifiables, toutes les parties du programme communiquant par diffusion instantanée de l'information.

Comme LUSTRE, SIGNAL [25] [26] est un langage de type déclaratif et flots de données. Les objets de base manipulés par ce langage sont les signaux. Les opérateurs du langage permettent de spécifier dans un style équationnel les relations entre les signaux, c'est-à-dire entre leur valeur et leur horloge. Un système d'équations est alors construit en réalisant la composition des différentes équations. Le compilateur [2, 8] se livre à une analyse de consistance du système d'équations ainsi défini et détermine si les contraintes de synchronisation sont vérifiées. Si c'est le cas et si le programme est contraint de façon à calculer une solution unique, alors un code exécutable est produit.

1.1.1 Signaux et processus

Un signal est une suite non bornée de valeurs typées à laquelle est associée une horloge qui détermine l'ensemble des instants où le signal est présent. Par exemple, un signal `X` dénote la séquence $(\mathtt{x}_t)_{t\in T}$ de données indexées par le temps t dans un domaine T. Des signaux d'un type particulier appelés `event` sont caractérisés seulement par leur horloge, c'est-à-dire leur présence (ils ont la valeur booléenne `true` à chaque occurrence). Étant donné un signal `X`, son horloge est donnée par l'expression `event X`, qui donne l'événement présent simultanément à `X`.

1.1.2 Le langage noyau

SIGNAL est construit autour d'un petit nombre d'opérateurs de base qui permettent de spécifier dans un style équationnel les relations entre les signaux. Chaque équation issue d'un programme SIGNAL peut être vue comme un processus élémentaire. Ces processus SIGNAL décrivent donc à la fois les relations fonctionnelles et temporelles entre les signaux. Ils peuvent communiquer, par l'intermédiaire de signaux constituant leurs ports d'entrée et

1. Une analyse complète des système réactifs est donnée dans [19]

de sortie, avec le monde extérieur ou avec d'autres processus. Enfin, le programme SIGNAL est obtenu en composant l'ensemble de processus.

En SIGNAL, les opérateurs de base définissent des processus élémentaires, chacun correspondant à une équation :

- **Les opérateurs fonctionnels.** Ils sont définis sur les types du langage. Par exemple la négation booléenne du signal `E` est écrite `not E` et le signal (Y_t), défini par la fonction f dans $Y_t = f(X_{1_t}, X_{2_t}, ..., X_{n_t})$, est écrit:

  `Y := f{ X1, X2, ... , Xn}`

  Les expressions fonctionnelles sont monochrones, ce qui signifie que les signaux `Y`, `X1`, ..., `Xn` sont dits synchrones : ils partagent la même horloge. En d'autres termes, pour calculer la valeur de Y_t, tous les X_i doivent être disponibles à l'instant t. Pour cette raison, ils sont contraints à avoir la même horloge : celle de Y.

  L'opérateur `event` qui permet d'accéder à l'horloge d'un signal peut être considéré comme une fonction particulière. Le noyau comporte aussi un opérateur relationnel qui spécifie que tous les signaux $\mathtt{X}_1$, ..., $\mathtt{X}_n$ sont synchrones. Il s'écrit `synchro{` $\mathtt{X}_1$`, ...,` $\mathtt{X}_n$ `}`.

- **Le filtre.** Le sous-échantillonnage d'un signal `X` selon une condition `C` est écrit :

  `Y := X when C`

  Cet opérateur est polychrone : les opérandes et le résultat n'ont pas la même horloge. Le signal `Y` est présent si et seulement si `X` et `C` sont présents au même instant et `C` a la valeur `true`. Ainsi, `Y` est moins fréquent que `X` et que `C` à la fois : l'intersection des horloges de `X` et de `C` (c'est-à-dire les instants où l'expression peut être évaluée) inclut l'horloge de `Y` (qui ne comporte que les instants où `C` s'évalue à `true`). Quand `Y` est présent, sa valeur est celle de `X`.

- **La fusion.** La fusion entre deux signaux `X` et `Y` de même type s'écrit.

  `Z := X default Y`

  La valeur de `Z` est celle de `X` quand il est présent, sinon celle de `Y` quand il est présent. Cet opérateur est également polychrone : l'horloge de `Z` est l'union de celles de `X` et `Y`, elle est donc plus fréquente que chacune d'elles.

- **Le retard** donne la valeur passée d'un signal. Il décrit donc des comportements dynamiques et est généralement noté $ZX_t = X_{t-d}$, avec la valeur initiale $ZX_i = V_i$, pour $0 < i \leq d$; en SIGNAL, pour le cas simple où $d = 1$, on écrit:

  `ZX := X$1` avec l'initialisation `ZX init V0`

  Le retard est un opérateur monochrone, c'est-à-dire que `X` et `ZX` ont la même horloge.

– **La composition de processus.** Les processus élémentaires peuvent être composés par l'opérateur commutatif et associatif " | " qui dénote l'union des systèmes d'équations. En SIGNAL, pour des processus P_1 et P_2, on écrit :

$$(|\ P_1\ |\ P_2\ |)$$

Une description complète du langage SIGNAL et des différents opérateurs est donnée dans [9]. Si le langage est construit autour de ce noyau élementaire, il comporte cependant des opérateurs dérivés pour les tableaux ou les variables par exemple, ainsi que des macro-instructions. Parmi ces nouvelles instructions, on retrouve notamment, en plus de l'opérateur `event` et `synchro` dont nous avons déjà parlé, le compteur : `C := # E`, dont l'extension est décrite par le processus `COUNT` :

```
process COUNT= {? E ! C}
  (| C := ZC+1
   | ZC := C$1
   | synchro{C,E}
   |)
  where ZC init 0
end
```

FIG. 1 – *Extension de* `C := # E`

Ce processus a un signal d'entrée `E` et un signal de sortie `C`. La valeur du compteur est `C`, et est définie comme la précédente valeur `ZC` incrémentée d'une unité à chaque occurrence du signal `E`. Le signal `ZC` est déclaré localement, et est défini en utilisant l'opérateur retard (`$`) sur le signal `C`. Sa valeur intiale est donnée par le paramètre `V0`. Finalement, ce processus compte les occurrences de l'événement d'entrée `E` : Ceci est obtenu en synchronisant la présence du compteur (signal `C`), avec les ocurrences de `E`.

1.1.3 L'éditeur graphique

De par sa modularité, un programme SIGNAL se prête facilement à une représentation graphique directe en bloc-diagramme. C'est pourquoi, de manière à faciliter l'écriture d'un programme SIGNAL, un éditeur [10] est disponible dans l'environnement SIGNAL. Celui-ci permet la construction sous forme mixte (mélange entre textes et graphiques) des programmes SIGNAL. Une représentation graphique du processus `COUNT` est donnée par la figure (2).

Les signaux d'entrée et de sortie sont symbolisés par des ports de connexion triangulaires situés sur les bords de la boîte corps de processus. Chaque boîte interne au corps représente une expression de processus pouvant être soit sous forme textuelle, soit construite récursivement à partir d'autres boîtes. Les signaux non masqués de chacune des sous-expressions sont représentés par des ports de connexion. Les liens entre les différents ports

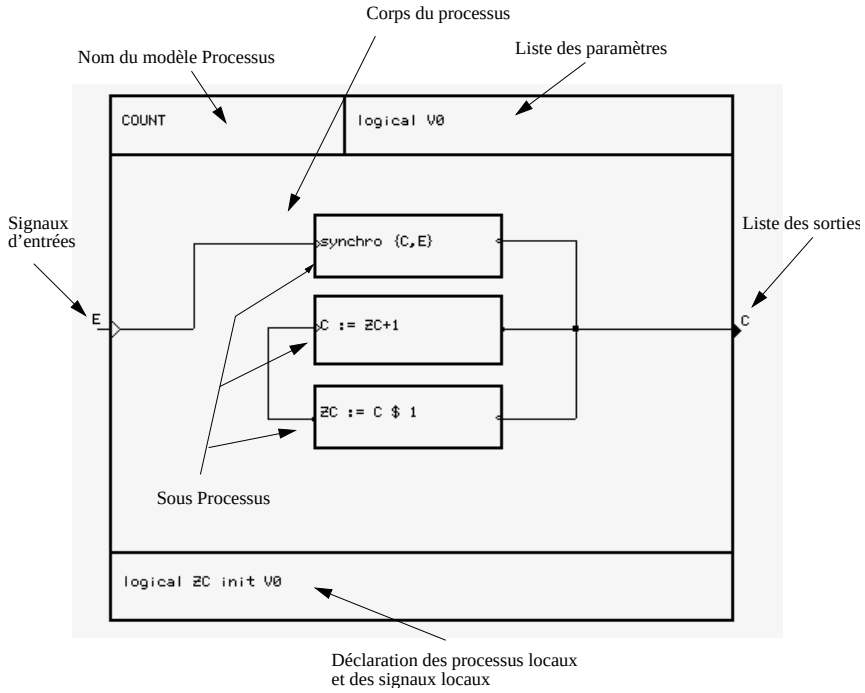

FIG. 2 – *processus* SIGNAL **COUNT** *sous sa forme graphique*

indiquent les voies de communication entre les divers sous-processus : deux ports reliés entre eux par un même lien représentent le même signal. Finalement, un décompilateur permet d'engendrer automatiquement un programme sous forme textuelle en vue de la compilation.

1.2 Sigali : un logiciel de calcul formel

SIGALI [13, 15] est l'outil de calcul formel dédié à l'analyse de programme SIGNAL. Cet outil travaille à partir du système dynamique polynomial résultant de la traduction de la partie booléenne du programme SIGNAL. L'interface entre SIGNAL et SIGALI est assurée par un traducteur, qui à partir du résultat d'un programme SIGNAL engendre le code $\mathbb{Z}/3\mathbb{Z}$ équivalent. À partir du système dynamique polynomial, il est possible d'analyser le comportement logique du programme SIGNAL (vérification de propriétés statiques ou portant sur les trajectoires du système), ainsi que de réaliser diverses opérations sur le système (synthèse de contrôleurs par exemple).

1.2.1 Interprétation Abstraite d'un programme SIGNAL

Un programme SIGNAL peut se voir comme un programme hybride, mélangeant des aspects logiques (calculs impliquant uniquement des booléens et les calculs d'horloges), et des aspects numériques (calculs sur des signaux entiers et réels). Dans le but de réaliser

des preuves sur ce programme, ou encore de la synthèse de contrôleur, il est nécessaire de s'abstraire de la partie numérique pour ne garder que la partie booléenne de celui-ci.

À ce niveau, plusieurs choix étaient possibles quant à la formalisation de cette abstraction du programme. Il était en effet possible de la traduire sous forme d'automates ou encore de réseaux de Petri. Cependant, le fait de traduire dans l'un de ces deux formalismes entraîne une perte de la hiérarchisation que l'on trouve dans un programme SIGNAL. De plus, la nature équationnelle de SIGNAL nous amène directement à une traduction de celui-ci sous forme d'un système d'équations.

Nous allons voir dans cette partie comment il est possible, après certaines simplifications, de traduire la partie booléenne d'un programme SIGNAL en un système dynamique polynomial dans $\mathbb{Z}/_{3\mathbb{Z}}$, *i.e.* les entiers modulo 3 : $\{-1, 0, 1\}$.

Les signaux. Un des aspects important de la programmation en SIGNAL est la hiérarchie d'horloges entre les signaux. Il convenait donc, lors de la traduction de conserver cette notion de présence/absence d'un signal comparativement à un autre. Ainsi, le principe est de coder les trois valeurs possibles d'un signal booléen A (*i.e.* *présent* et *vrai*, *présent* et *faux* et *absent* en une variable a de la manière suivante :

$$\begin{cases} Présent \wedge Vrai & \rightarrow & +1 \\ Présent \wedge Faux & \rightarrow & -1 \\ Absent & \rightarrow & 0 \end{cases}$$

Pour les signaux non-booléens dont la valeur ne peut pas être représentée, seule l'absence et la présence sont codées :

$$\begin{cases} Présent & \rightarrow & \pm 1 \\ Absent & \rightarrow & 0 \end{cases}$$

Par ce codage, un événement sur n signaux est représenté par un vecteur dans $(\mathbb{Z}/_{3\mathbb{Z}})^n$. De plus, la présence d'un signal est codée par la valeur 1. Par conséquent un signal `A` est synchrone avec un signal `B` si et seulement si ils satisfont l'équation $a^2 = b^2$.

Les processus primitifs. Chaque opérateur élémentaire du noyau SIGNAL est ainsi codé sous forme équationnelle. Par exemple `C := A when B`, signifiant *"if* $b = 1$ *then* $c = a$ *else* $c = 0$", est réécrit $c = a(-b - b^2)$. Le retard `$`, qui est l'opérateur dynamique, est différent, car il requiert la mémorisation de la dernière valeur dans une variable d'état. Ainsi pour coder `A := B$1 init A0`, il est nécessaire d'introduire les trois équations suivantes:

$$\begin{cases} x' & = & b + (1 - b^2)x & (1) \\ a & = & xb^2 & (2) \\ x_0 & = & a_0 & (3) \end{cases}$$

L'équation (1) permet de calculer la nouvelle valeur x' de la variable d'état. Si b est *présent*, x' est égal à b (parceque $(1 - b^2) = 0$); sinon x' est égal à la dernière valeur de b, mémorisé

Signaux booléens	
`B := not A`	$b = -a$
`C := A and B`	$c = ab(ab-a-b-1)$ $a^2 = b^2$
`C := A or B`	$c = ab(1-a-b-ab)$ $a^2 = b^2$
`C := A default B`	$c = a+(1-a^2)b$
`C := A when B`	$c = a(-b-b^2)$
`B := A $1 (init` b_0`)`	$x' = a+(1-a^2)x$ $b = a^2x$ $x_0 = \mathtt{b}_0$

Signaux non booléens	
`B := ` $f(A_1,\ldots,A_n)$	$b^2 = a_1^2 = \cdots = a_n^2$
`C := A default B`	$c^2 = a^2+b^2-a^2b^2$
`C := A when B`	$c^2 = a^2(-b-b^2)$
`B := A $1 (init` b_0`)`	$b^2 = a^2$

TAB. 1 – *Traduction des opérateurs de base* SIGNAL

par x. L'équation (2) donne à a la dernière valeur de b (*i.e.*la valeur de x) et contraint l'horloge de a à être égale à celle de b. En effet, on a $a^2 = x^2b^4$, et dans $\mathbb{Z}/3\mathbb{Z}$ nous avons $b^3 = b$, *i.e.*$b^4 = b^2$, ceci donne donc $a^2 = x^2b^2$; Comme de plus $x^2 = 1$ (car x est toujours présent), et finalement $a^2 = b^2$. L'équation (3) correspond à la valeur initiale de x, qui est la valeur initiale de a.

Certains opérateurs peuvent être codés de deux façons suivant le type des signaux. La table suivante donne le codage sous forme d'équations des processus élémentaires.

Les processus. En composant les différentes équations résultantes des différents processus élémentaires, tout programme SIGNAL peut être traduit en un ensemble d'équations, appelé système dynamique polynomial.

En utilisant ce codage, un processus est donc traduit en un système d'équations de la forme :

$$\left\{\begin{array}{rcl} Q(X,Y) & = & 0 \\ X' & = & P(X,Y) \\ Q_0(X) & = & 0 \end{array}\right.$$

Ce système est constitué

- d'un ensemble de n variables $X = \{X_1,\ldots,X_n\}$, appelées *variables d'états*;
- d'un ensemble de m variables $Y = \{Y_1,\ldots,Y_m\}$, appelées *variables d'événements*;
- d'un ensemble de *contraintes* $Q(X,Y) = 0$. Il est représenté par un vecteur $[Q_1,\ldots,Q_m]$, regroupant toutes les équations caractérisant l'aspect statique du système (invariant pour tout les instants t);

– d'une *fonction d'évolution* $P(X,Y)$ de $(\mathbb{Z}/_{3\mathbb{Z}})^{n+m}$ dans $(\mathbb{Z}/_{3\mathbb{Z}})^{n}$;

– d'un ensemble de n équations $[Q_{0_1},\ldots,Q_{0_m}]$, appelées *contraintes d'initialisation* notées $Q_0(X)=0$ et caractérisant l'initialisation des variables d'états du système.

Les X proviennent de la traduction des signaux retardés, les Y représentent les signaux booléens et les horloges du programme. Considérons, par exemple le programme SIGNAL

```
process altern = {? event A,B !}
   (| C := not ZC
    | ZC := C$1
    | synchro{A,when C}
    | synchro{B,when ZC}
    |)
  where
    logical C, ZC init false
  end
```

FIG. 3 – *Processus contraignant deux entrées à être alternées*

décrit par la figure (1.2.1) :

Le système dynamique résultant de la traduction de ce programme comprend les variables a, b, c et zc, correspondant respectivement aux événements A, B et aux signaux booléens C et ZC, et une variable d'état x, introduite par le retard. Le système obtenu est le suivant :

– une équation d'initialisation : $x=-1$,

– une équation d'évolution : $x'=c+(1-c^2)*x$

– un système d'équations de contraintes

$$c=-zc, zc=x*c^2, a^2=-c-c^2, b^2=-zc-zc^2$$

En partant d'un programme SIGNAL, il donc possible d'en extraire la partie contrôle. L'abstraction obtenue est alors un système dynamique polynomial dans $\mathbb{Z}/_{3\mathbb{Z}}$. En utilisant des opérations algébriques, il est alors possible de raisonner sur le système à événements discrets obtenu.

1.2.2 La preuve de programmes SIGNAL

Avant d'introduire la notion de système dynamique polynomial (SDP) contrôlable et de synthèse de contrôleur sur des SDP, il nous est nécessaire d'introduire le type de propriétés sur lesquels les objectifs de synthèse vont s'appuyer. Nous avons choisi de représenter ici des types génériques de propriétés traditionellement utilisées dans le monde industriel, pour qui l'aspect sureté de fontionnement est primordial.

Propriété de vivacité. On dit qu'un système est vivant si et seulement si il n'est pas jamais en "dead-lock". Un état x sera un état mort ou bloquant s'il n'existe pas d'événement admissible dans l'état x . Une trajectoire comportant un état mort est nécessairement finie. Un système vivace implique que toutes les trajectoires de celui-ci sont infinies. En terme de systèmes dynamiques polynomiaux, cette définition peut se décrire de la manière suivante:

Définition 1 *Un système est vivace, si et seulement si pour tous les couples (x,y) tq Q(x,y)=0, P(x,y) est un état qui admet encore des transitions admissibles.*

Propriété de sécurité Informellement,une propriété de sécurité décrit le fait que de "mauvaises choses" ne peuvent pas se produire. Dans notre cas, cette sorte de propriété couvre la classe des propriétés, décrivant un ensemble de bons états, qui restent invariants. La définition suivante rappelle la définition d'un ensemble d'états invariant.

Définition 2 *Un ensemble d'états E est donc invariant vis à vis d'un système dynamique polynomial, si et seulement si pour chaque état x de E et pour tout événement y admissible dans l'état x, l'état x'= P(x,y) appartient à E.*

Cette notion est illustrée par la figure (4(a)). Une notion plus souple d'invariance est donnée par la définition suivante :

Définition 3 *Un ensemble d'états E est invariant sous-contrôle vis à vis d'un système dynamique polynomial, si et seulement si pour chaque état x de E il existe un événement y admissible dans l'état x, tel que l'état x'= P(x,y) appartient à E.*

Informellement, un ensemble d'états est dit invariant sous-contrôle pour un système dynamique S si et seulement si il est possible de le "piloter" en réalisant un choix sur les événements, de manière à rester dans l'ensemble des "bons états". Cette notion est illustrée par la figure (4(b)).

D'autres classes de propriétés peuvent se déduire à partir des propriétés d'invariance et d'invariance sous contrôle.

Propriété d'atteignabilité (cf figure (4(c)))

Définition 4 *Un sous-ensemble F d'états est accessible pour un système dynamique, si et seulement si chaque état x de F peut être atteint à partir des états initiaux du système dynamique considéré (ie, il existe une trajectoire partant des états initiaux qui atteint x).*

Propriété d'attractivité (cf figure (4(d)))

Définition 5 *Un ensemble d'états F est attractif vis à vis d'un autre ensemble d'états E, si et seulement si, chaque trajectoire du système initialisée dans un état de E atteint l'ensemble F.*