

# SPÉCIFICATION ET VÉRIFICATION DE SYSTÈMES RÉACTIFS : EXPÉRIMENTATION DE LA MÉTHODOLOGIE SYNCHRONE SIGNAL

## MODELING AND SPECIFYING REACTIVE SYSTEMS EXPERIMENTS WITH THE SIGNAL SYNCHRONOUS METHODOLOGY

Hervé Marchand, Éric Marchand, Éric Rutten

IRISA - INRIA Rennes - Université de Rennes I  
Campus de Beaulieu, F-35042 RENNES cedex  
Fax : (33) 99 84 71 71, e-Mail : {marchand, hmarchan, rutten}@irisa.fr

**Résumé** : SIGNAL est un langage synchrone flot de données, étendu par des tâches préemptives dans SIGNAL*GTi*. Il est ainsi adapté à la spécification de systèmes réactifs présentant une dualité continu (échantillonné) et discret (événementiel ou séquentiel). La vérification formelle de leur correction se fait dans un modèle équationnel. Deux exemples de systèmes réactifs serviront de support à cet article.

**Abstract** : We present the synchronous approach to reactive systems programming. We applied in an experiment in a significant problem in robot vision. This application consists of the specification of a system dealing with various domains such as robot control, computer vision and transitions between different modes of control.

### 1. Langages synchrones et systèmes réactifs

Les langages synchrones sont particulièrement adaptés à la conception des systèmes réactifs. Une manière d'interpréter l'hypothèse synchrone est de considérer que chaque calcul, au sens large du terme, se produit dans l'instant logique. Une famille de langages est basée sur cette hypothèse [Hal93]. Leur but est de permettre une conception sûre d'applications critiques, en particulier celles concernant le traitement du signal ou le contrôle de processus. Basés sur une abstraction du temps réel en un temps discret et logique, ces langages sont munis d'une forte sémantique. Cette sémantique leur permet d'offrir à l'utilisateur des environnements de programmation complets allant de la spécification à la génération automatique de code exécutable en passant par la vérification formelle du code généré. Parmi eux, SIGNAL est un langage temps réel, synchrone de caractère flot de donnée [LLGL91]. Son modèle de temps est basé sur les instants, et ses actions sont réalisées lors de ces instants. Nous verrons cependant qu'une extension de SIGNAL : SIGNAL*GTi* offre la possibilité de spécifier des tâches ayant une durée et autorise la préemption de tâches sur des intervalles de temps [RL94]. De plus, il existe un système autorisant la vérification formelle de code source et les preuves de propriétés : SIGALI. L'utilisation de SIGNAL offre donc un très haut niveau d'abstraction concernant la spécification ainsi qu'un modèle temporel cohérent et puissant.

### 2. Spécification et mise en œuvre de systèmes réactifs.

Nous présentons ici de façon succincte le langage synchrone SIGNAL ainsi qu'une extension récente de ce

langage aux intervalles de temps. La spécification d'un système réactif de vision robotique est ensuite présentée tant au niveau "continu" (flots de données) que discret (séquencement et préemption de tâches).

#### 2.a Le langage SIGNAL

SIGNAL [LLGL91] est un langage de programmation et de spécification de systèmes interactifs ou temps-réel, faisant partie de la famille des langages synchrones [Hal93]: on fait ainsi l'hypothèse que toutes les actions d'un programme sont instantanées. Le langage SIGNAL, est un langage de type flot de données et de style déclaratif. Il est construit autour d'un noyau restreint d'opérateurs de base. Ceux-ci manipulent des signaux, qui sont des suites non bornées de valeurs typées, dont une horloge associée détermine les instants auxquels la valeur est présente; par exemple, un signal  $X$  dénote la séquence  $(x_t)_{t \in T}$  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  $X$ . Les constructeurs du langage permettent de spécifier dans un style équationnel des relations entre les signaux, c'est-à-dire entre leurs valeurs et entre leurs horloges. Des systèmes d'équations sont construits en utilisant la composition.

Le compilateur se livre à une analyse de la consistance du système d'équations, et détermine si les contraintes de synchronisation sont bien 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 (en C ou en Fortran).

A. *Le noyau de SIGNAL.* 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  $E$  :  $\text{not } E$ ). Le signal  $(Y_t)$ , défini par la fonction  $f$  dans :  $Y_t = f(X_{1t}, X_{2t}, \dots, X_{nt})$  est écrit :

$$\boxed{Y := f\{X_1, X_2, \dots, X_n\}}$$

Les expressions fonctionnelles sont monochrones, ce qui signifie que les signaux  $Y, X_1, \dots, X_n$  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  $Y$ .

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

$$\boxed{ZX := X\$1} \text{ avec l'initialisation } \boxed{ZX \text{ init } V_0}$$

Le délai est monochrome lui aussi, c'est-à-dire que  $X$  et  $ZX$  ont la même horloge.

**Le filtre.** Le sous-échantillonnage d'un signal  $X$  selon une condition  $C$  est écrit :  $\boxed{Y := X \text{ when } C}$ . 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 que  $C$  a la valeur  $\text{true}$ . 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  $\text{true}$ ). Quand  $Y$  est présent, sa valeur est celle de  $X$ .

**La fusion.** On définit la fusion de deux signaux  $X$  et  $Y$  par :  $\boxed{Z := X \text{ default } Y}$ . Cet opérateur aussi est polychrone : l'horloge de  $Z$  est l'union de celles de  $X$  et  $Y$ , elle est donc plus fréquente que chacune d'elles. La valeur de  $Z$  est celle de  $X$  quand il est présent, sinon celle de  $Y$  quand il est présent.

**La composition.** 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 :  $\boxed{(| P_1 | P_2 |)}$ .

**Par exemple,** l'équation  $x_t = f(x_{t-1}) + 1$  peut aussi s'écrire :

$$\begin{cases} x_t = f(zx_t) + 1 \\ zx_t = x_{t-1} \end{cases}$$

ce qui s'écrit en SIGNAL :

$$\boxed{(| X := f(ZX) + 1 | ZX := X\$1 |)}$$

Le langage est construit autour de ce noyau et comporte des opérateurs dérivés pour les tableaux ou les variables par exemple. On peut définir des sortes de macro-instructions : des schémas de processus, qui ont un nom, des paramètres et des signaux d'entrée et de sortie typés, un corps et des

déclarations locales. Les instances de schémas de processus rencontrées dans un programme sont expansées par un préprocesseur du compilateur.

B. *Séquencement de tâches flot de données.* Une récente extension à SIGNAL (SIGNALGTi) introduit la notion d'*intervalle de temps*, défini par des événements de début et de fin et désignant un état alternant entre les valeurs *inside* et *outside*. Associer un intervalle de temps  $I$  et un processus flots de données  $P$  définit une tâche. Cette tâche est active lorsque l'intervalle  $I$  est *inside* et est inexistante (pas d'horloge), lorsque l'intervalle  $I$  est *outside*.

Il existe deux méthodes pour construire les tâches, qui sont aussi deux structure de préemption: dans  $P \text{ on } I$ , le comportement de  $P$  reprend dans l'état courant de ses variables d'états, alors que dans  $P \text{ each } I$ , il reprend depuis son état initial. On peut ainsi spécifier simplement des séquençements et des structures de préemption sur des tâches à flots de données [RL94]. Une version de cette extension de SIGNAL a été mise en œuvre sous la forme d'un préprocesseur au compilateur, qui traduit un programme SIGNALGTi en un programme SIGNAL.

## 2.b Application : un système de vision active

Dans un premier temps, nous prendrons comme exemple de système réactif, un système de vision robotique permettant la reconstruction de scènes tridimensionnelles. La tâche de ce robot est de créer un modèle géométrique 3D de son environnement à partir de la séquence d'images acquises en temps réel. Une méthode générale a été établie permettant de localiser toute primitive géométrique paramétrable. Elle est basée sur l'utilisation de la matrice d'interaction qui permet de relier les informations visuelles sur la primitive considérée au mouvement de la caméra. Lorsque les mouvements sont quelconques les erreurs de localisation sont très importantes. Il a été cependant montré qu'un déplacement adéquat de la caméra par rapport à la primitive à localiser permettra de minimiser les erreurs sur l'estimation des paramètres de la primitive. De tels mouvements seront réalisés en utilisant l'approche commande référencée-vision ou asservissement visuel. Cependant, cette approche ne permet de reconstruire qu'une seule primitive à la fois ; de plus une connaissance *a priori* sur la nature de la primitive est nécessaire afin de générer les mouvements optimaux de la caméra. Il faut donc s'abstraire de ces contraintes en définissant des stratégies de perception de la scène afin de permettre une représentation 3D précise et complète de la zone à reconstruire. Le schéma de base proposé consiste à focaliser successivement la caméra sur les différents objets de la scène et à les reconstruire. La stratégie de reconstruction de la scène 3D est donc constituée de phases d'asservissement visuel durant lesquelles la primitive 3D sur laquelle la caméra est focalisée sera reconstruite, et de phases de recherche d'informations permettant de sélectionner les futures primitives à reconstruire. Il est donc apparu clairement que l'une des difficultés de cette étude résidait, entre autres, dans la façon de gérer la dualité

*continu/événementiel* dans la gestion du mouvement de la caméra et des stratégies de perception. Dans un tel système, la réactivité intervient donc à deux niveaux distincts : dans le calcul de la commande en boucle fermée où le système doit réagir à chaque instant en fonction des informations transmises par le capteur, et au niveau du contrôle de tâche où le système doit réagir à des événements extérieurs par des changements d'état. Dans cette optique, SIGNAL met à notre disposition une méthodologie de programmation unifiée permettant d'intégrer l'aspect "continu" des algorithmes de commande et d'estimation avec l'aspect "événementiel" des stratégies de perception au sein d'un langage temps réel de haut niveau [MCR95]. Les méthodes de programmation classiques ne sont pas bien adaptées à la spécification et à l'implémentation de tels algorithmes. Un langage classique impératif requiert une prise en compte explicite des aspects bas niveau de la mise en œuvre (tel le séquencement des calculs imposé par les dépendances entre les données) ainsi que des problèmes temporels pour lesquels ils ne fournissent pas de support ou de modèle bien établis. Les intérêts de SIGNAL sont multiples pour cette application. D'une part, son caractère flot de données est parfaitement adapté aux tâches d'asservissement visuel et aux algorithmes de reconstruction 3D. Les opérateurs de SIGNAL permettent une programmation aisée des algorithmes utilisant des valeurs mesurées à des instants précédents (dans notre cas, mesure du déplacement de la caméra, mesure du déplacement des primitives entre plusieurs images successives, filtres moyennant, etc).

*A. L'asservissement visuel : algorithme flot de données.*  
L'asservissement visuel consiste à utiliser les informations fournies par une caméra mobile commandable afin de réaliser des tâches élémentaires (positionnement, suivi d'objet mobile, suivi de trajectoire) en contrôlant la situation ou le mouvement de la caméra par rapport à son environnement. L'approche consiste à spécifier le problème en termes de régulation dans l'image ce qui permet de compenser les imprécisions des modèles de la caméra et de l'effecteur.

La loi de commande en boucle fermée sur les informations capteurs utilisées en asservissement visuel consiste en la régulation d'une fonction de tâche qui peut, de manière schématique, s'écrire  $c = f(s)$  où  $c$  est la vitesse de l'effecteur exprimé en fonction des signaux capteurs  $s$ . Le contrôle de l'effecteur est une fonction continue  $f$  plus ou moins complexe. L'implémentation d'une telle loi de commande est réalisée en "discrétisant" le flot d'information  $s$  en provenance de la caméra (cadence vidéo) en un flot de valeurs  $s_t$ , qui est ensuite utilisé pour calculer un flot de commande  $c_t : \forall t, c_t = f(s_t)$ . Ce type de calcul numérique et flot de données est le champ d'application privilégié des langages flot de données et de SIGNAL en particulier. De plus, comme le montre l'indice  $t$  dans l'équation schématique présentée, la présence simultanée des signaux mis en jeu est parfaitement gérée par l'hypothèse synchrone.

Une description modulaire du programme SIGNAL met-

tant en œuvre l'asservissement visuel est présentée sur la figure 1. Au plus haut niveau de description on retrouve les 3 processus principaux synchrones entre eux :

- le processus CAMERA\_OUTPUT qui produit un flot d'information capteur à la cadence vidéo (horloge du système) ;
- Ces informations sont reçues par le processus de calcul de la commande qui est calculée en utilisant l'approche fonction de tâche ;
- La commande de la caméra ainsi calculée est transmise à la commande numérique du robot par le processus ROBOT\_CONTROL.

Le processus de calcul de la commande est lui-même décomposé en sous-processus permettant de calculer les erreurs entre les signaux capteurs courants et les signaux capteurs désirés, la valeur courante de la matrice d'interaction  $L$ , la tâche primaire (convergence vers un motif à atteindre dans l'image) et la tâche secondaire (suivi de trajectoire), etc.

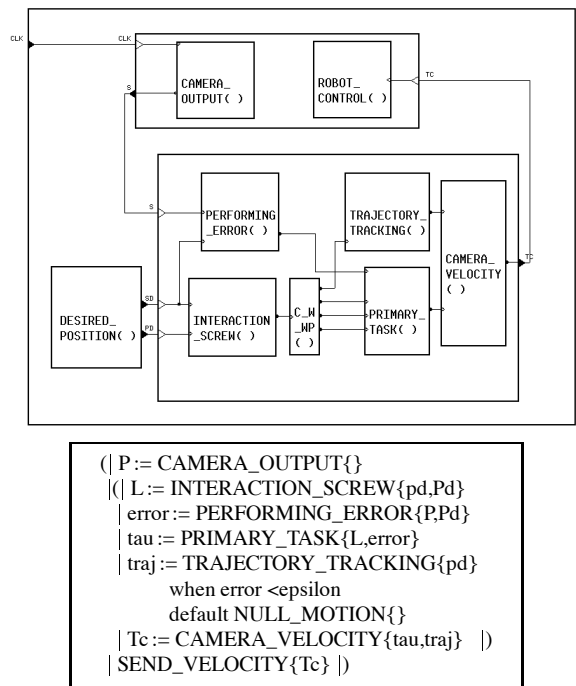


FIG. 1 – Programme SIGNAL pour l'asservissement visuel (graphique et texte).

*B. La reconstruction 3D : processus dynamique et parallélisme.* L'objectif de la reconstruction est de remonter à la structure 3D de l'objet observé en utilisant les informations extraites d'une séquence d'image [CBBJ94]. Le mouvement de la caméra nécessaire à une reconstruction optimale sera généré en utilisant la méthode de l'asservissement visuel décrite précédemment. En fait, l'estimation de la structure 3D de l'objet se fera même en parallèle du calcul de la loi de commande.

Le processus d'estimation des paramètres 3D d'un objet est basé sur l'utilisation de la valeur courante et des valeurs

passées de la position de la primitive dans l'image, ceci afin de mesurer la vitesse de celle-ci dans l'image. De plus, il est aussi nécessaire de connaître la vitesse de la caméra entre deux instants  $t$  et  $t - 1$ . Ces valeurs passées des signaux peuvent aisément être exprimées grâce à l'opérateur **décalage** de SIGNAL. Ainsi, si  $P$  est le signal portant la position de la primitive dans l'image et  $Tc$  celui portant la vitesse de la caméra l'estimation  $p$  des paramètres de la caméra pourra s'exprimer par ;

$$\left( \begin{array}{l} | p := \text{ESTIMATION}\{P, ZP, ZTc\} \\ | ZP := P\$1 \quad | ZTc := Tc\$1 \end{array} \right)$$

Le parallélisme entre processus peut être obtenu en utilisant l'opérateur de composition “|” qui est alors interprété comme un opérateur de parallélisme entre les processus communiquant à travers les signaux mis en jeux. On pourra alors ajouter en parallèle du processus d'estimation précédent, le processus de contrôle présenté à la section précédente. Schématiquement on aura ;

$$\left( \begin{array}{l} | Tc := \text{CONTROL}\{P, P_d, p\} \\ | p := \text{ESTIMATION}\{P, ZP, ZTc\} \end{array} \right)$$

**C. Séquencement de tâche** Finalement, l'introduction récente de la notion d'intervalle de temps dans SIGNAL offre la possibilité de gérer la hiérarchisation, le séquencement ainsi que la préemption de tâches. Ce niveau de programmation autorise la spécification et l'implémentation de manière aisée d'un automate hiérarchique parallèle qui assure le séquencement des différentes tâches de vision.

L'approche utilisée pour la reconstruction ne permet de reconstruire qu'une primitive à la fois, le capteur devient donc une ressource “rare”. Afin d'éviter des requêtes conflictuelles, il faut concevoir un séquenceur donnant alternativement à une tâche le contrôle du système de vision. Un séquencement de sous-tâches d'asservissement visuel est donc nécessaire pour passer de la reconstruction d'une primitive à une autre, et ainsi obtenir, par construction incrémentale, une représentation complète de la scène. Ce séquenceur peut souvent être réduit à un simple automate (c'est souvent le cas dans les systèmes réactifs). Une approche objet similaire la nôtre, utilisant le langage synchrone ESTEREL et le système ORCAD, a été proposée dans [SECK93]. Pour notre part, nous nous sommes intéressés à la conception d'un automate hiérarchique parallèle (voir Figure 2) capable de gérer l'ensemble du processus de reconstruction d'une scène complexe. Cet automate sélectionne et gère les actions à effectuer en fonction des événements 2D perçus dans l'image, des connaissances acquises et répertoriées au fur et à mesure de la reconstruction, ainsi que de la détection de la fin d'une tâche d'asservissement visuel [MCR95]. De manière plus précise, à chaque état de l'automate hiérarchique, on associe une tâche, par exemple une tâche d'asservissement visuel, et un intervalle de temps pendant lequel cette tâche sera active (voir Figure 3. À ce niveau, chaque action (tâche) peut être considérée comme atomique ce qui nous place dans le même formalisme que celui des systèmes à événements discrets et nous permet donc d'utiliser le même type d'outils. En contraignant

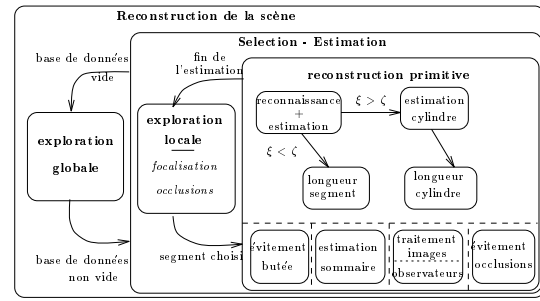


FIG. 2 – Automate hiérarchique parallèle

les tâches sur un intervalle de temps, l'utilisation de  $\text{SIGNALGT}_i$ , autorise par exemple la combinaison de comportements (parallélisme entre tâches), la préemption ainsi que le séquencement de tâches. Il permet d'autre part le traitement de la terminaison des tâches flot de données.

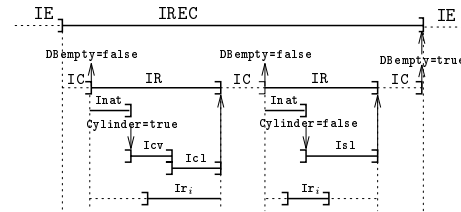


FIG. 3 – Spécification du séquencement en terme d'intervalle : une trace possible.

Un processus flot de données, comme nos tâche de vision, porte en lui la spécification complète de son comportement mais pas celle de sa terminaison. Cet aspect doit alors être défini séparément. Une façon de décider de la terminaison d'une tâche est de spécifier un critère dépendant des données capteurs ou des calculs effectués par le processus lui même. L'évaluation de ce critère doit se faire à chaque instant, de ce fait, cette évaluation devient alors un autre processus flot de donnée. L'instant où la condition est vérifiée peut être marqué par un événement discret, qui, causant la terminaison d'une tâche, peut aussi être la cause d'une transition vers une autre tâche au même niveau ou à un niveau plus élevé dans l'automate hiérarchique. Dans cette optique ce type d'événement peut être utilisé pour marquer la fin de l'exécution interne d'une tâche.

```
Primitive_estimation:
(| Optimal_estimation
 |(| Coarse_estimation_1 | ... | Coarse_estimation_n )
 | Occlusion_avoidance | Joint_limits_avoidance |)
-----
Coarse_estimation_i:
(| Ir_i := ] New_Segment, Segment_Lost ] init outside
 | Coarse_estimation each Ir_i |)
-----
Optimal_estimation:
(| Inat := ] close Isl default close Icl, Cyl ] init inside
 | Nature each Inat
 | Icv := ] when Cyl, when ( | prec < ε_cv ) init outside
 | Cylinder_vertex each Icv
 | Icl := ] close Icv, when ( | prec < ε_cl ) init outside
 | Cylinder_length each Icl
 | Isl := ] when not Cyl, when ( | prec < ε_sl ) init outside
 | Segment_length each Isl |)
```

En utilisant l'opérateur de composition, le parallélisme

entre deux tâches devient transparents. C'est le cas par exemple de l'estimation optimale et des estimations sommaires qui sont réalisées en parallèle. Les flots d'informations en entrées de ces processus provenant du même capteur sont synchrones, de fait, il peuvent donc être utilisés au même instant logique. En fait, nous avons ici, un parallélisme de spécification, le travail de synchronisation et de communication entre les différentes tâches est laissé au compilateur.

La spécification de l'application en SIGNALGTi est donnée ci dessus d'une manière simplifiée en gardant uniquement les aspects essentiels.

La figure 4 montre l'environnement d'exécution de l'application créée avec le langage Signal.

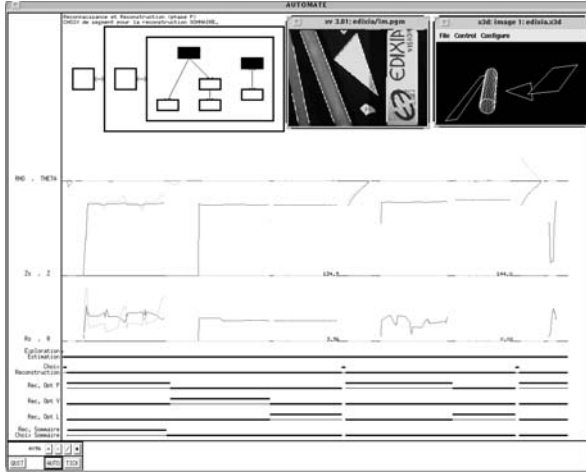


FIG. 4 – Environnement synchrone pour la reconstruction de scènes 3D.

### 3. Vérification formelle - Preuve de propriété.

Nous regardons dans cette partie à la vérification formelle du code source. SIGNAL étant basé sur une approche équationnelle, les programmes SIGNAL peuvent donc être considérés comme un ensemble d'équations de contraintes entre les différents signaux. Pour réaliser la preuve d'un programme SIGNAL, celui ci est, dans un premier temps, traduit en un système d'équations polynomiales.

#### 3.a Codage polynomial

Le codage polynomial décrit les aspects logiques et la synchronisation des programmes. Les informations représentées sont l'absence ou la présence des signaux ainsi que la valeur des signaux booléens.

Pour les booléens, le codage est :

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

Pour les signaux non-booléens dont la valeur ne peut pas être représentée, on code simplement l'absence ou la présence :

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

Chaque opérateur élémentaire du noyau SIGNAL est ainsi codé sous forme équationnelle.

Par exemple  $C := A \text{ when } B$ , signifiant *if*  $b = 1$  *then*  $c = a$  *else*  $c = 0$ , est réécrit  $c = a(-b - b^2)$ . Certains opé-

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

| Signaux booléens                 |                               |
|----------------------------------|-------------------------------|
| $Y := \text{not } X$             | $y = -x$                      |
| $Z := X \text{ and } Y$          | $z = xy(xy - x - y - 1)$      |
| $Z := X \text{ or } Y$           | $z^2 = y^2$                   |
| $Z := X \text{ default } Y$      | $z = xy(1 - x - y - xy)$      |
| $Z := X \text{ when } Y$         | $x^2 = y^2$                   |
| $Y := X \$1 \text{ (init } y_0)$ | $z = x + (1 - x^2)y$          |
|                                  | $z = x(-y - y^2)$             |
|                                  | $\xi' = x + (1 - x^2)\xi$     |
|                                  | $y = x^2\xi$                  |
|                                  | $\xi_0 = y_0$                 |
| Signaux non booléens             |                               |
| $Y := f(X_1, \dots, X_n)$        | $y^2 = x_1^2 = \dots = x_n^2$ |
| $Z := X \text{ default } Y$      | $z^2 = x^2 + y^2 - x^2y^2$    |
| $Z := X \text{ when } Y$         | $z^2 = x^2(-y - y^2)$         |
| $Y := X \$1 \text{ (init } y_0)$ | $y^2 = x^2$                   |

Un processus est donc traduit en un système d'équations de la forme :

$$\begin{cases} Q(X, Y) & = 0 \\ X' & = P(X, Y) \\ Q_0(X) & = 0 \end{cases}$$

Ce système est constitué

- d'un ensemble de  $n$  variables  $X = \{X_1, \dots, X_n\}$ , appelées *variables d'états*;
- d'un ensemble de  $m$  variables d'événements  $Y = \{Y_1, \dots, Y_m\}$ ,
- d'un ensemble de contraintes  $Q(X, Y) = 0$ . Il est représenté par un vecteur  $[Q_1, \dots, Q_m]$ , 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  $(\mathbb{Z}/3\mathbb{Z})^n$ . Cet ensemble regroupe toutes les équations concernant les variables d'état et caractérise l'aspect dynamique du système;
- d'une contrainte d'initialisation  $Q_0(X) = 0$ .

Les  $X$  proviennent de la traduction des signaux retardés, les  $Y$  représentent les signaux booléens et les horloges du programme. Le système dynamique décrit les suites de couples (état, événement) que peut produire le programme, c'est à dire les suites  $(x_i, y_i)_{i \in \mathbb{N}}$  telles que  $Q_0(x_0) = 0$  et pour tout  $i \in \mathbb{N}$ ,

$$\begin{aligned} Q(x_i, y_i) &= 0 \\ P(x_i, y_i) &= x_{i+1} \end{aligned}$$

Ces suites sont appelées les *trajectoires* du système.

Nous avons donc un modèle mathématique qui caractérise le comportement logique de notre modèle. Dans la perspective de l'analyse du comportement de ce système et de la validation de propriétés, nous avons besoin d'opérations sur les systèmes d'équations polynomiales

#### 3.b Opérations sur les systèmes dynamiques polynomiaux.

La théorie des systèmes dynamiques polynomiaux utilise les opérations élémentaires d'algèbre. L'idée est de

convertir des propriétés exprimées à l'aide d'ensembles d'états ou d'événements en propriétés équationnelles équivalentes. Par la suite, la méthode classique utilisée est de remplacer ces systèmes d'équations par des idéaux de polynômes. On peut alors traduire les propriétés des ensembles par des propriétés sur les idéaux. L'intérêt des équations est de représenter de manière compacte les ensembles manipulés avec un ensemble de méthodes bien établies [LBL91, Le 93].

*A. Description des objets de base et des opérations.* Soit l'anneau  $A[X, Y] = \mathbb{Z}/3\mathbb{Z}[X, Y]/\langle X^3 - X, Y^3 - Y \rangle$  quotient des fonctions polynomiales<sup>1</sup> : Cet anneau représente l'ensemble des polynômes dans  $\mathbb{Z}/3\mathbb{Z}$  pour lesquels le degré en chaque variable est inférieur ou égal à 2. Soit  $E$  un ensemble de variables événements et de variables d'états appartenant à  $(\mathbb{Z}/3\mathbb{Z})^{n+m}$ . Alors, l'ensemble suivant de polynômes :

$$I(E) = \{p \in A[X, Y] / \forall (x, y) \in E, p(x, y) = 0\}$$

est appelé l'*idéal* de  $E$  dans  $A[X, Y]$ . Cet ensemble représente tous les polynômes, pour lesquels l'ensemble  $E$  est solution.

Réciproquement, à chaque ensemble de polynômes  $G$ , on peut associer un ensemble de  $(\mathbb{Z}/3\mathbb{Z})^{n+m}$ , appelé la *variété* de  $G$ , définie de la manière suivante :

$$V(\langle G \rangle) = \{(x, y) \in (\mathbb{Z}/3\mathbb{Z})^{n+m} / \forall p \in G, p(x, y) = 0\}$$

où, pour un ensemble de polynômes  $G$ ,  $\langle G \rangle$  est l'ensemble de toutes les combinaisons linéaires entre les polynômes de  $G$ . L'ensemble  $V(\langle G \rangle)$  représente toutes les solutions de l'ensemble  $G$  de polynômes.

L'intérêt des idéaux réside dans le fait qu'il existe une correspondance directe entre un idéal et la variété associée. Ainsi, au lieu de manipuler l'ensemble des solutions  $E$  de l'équation de contraintes, représenté dans notre cas par une variété, on peut facilement la convertir en un idéal  $I(E)$ , représenté par un unique polynôme. De cette manière, il est possible de convertir des propriétés portant sur des ensembles d'états et événements en des propriétés équivalentes sur les idéaux de polynômes associés.

*B. Opérations portant sur le comportement dynamique.*

Pour traduire le comportement dynamique d'un système dynamique polynomial, on introduit la notion de morphisme et de comorphisme. Un *morphisme* est une fonction polynomiale  $P$  de  $(\mathbb{Z}/3\mathbb{Z})^{n+m}$  dans  $(\mathbb{Z}/3\mathbb{Z})^n$  (l'*équation d'évolution* du système  $X' = P(X, Y)$ , par exemple). À cette fonction polynomiale, ou morphisme  $P$ , on associe une fonction duale : le *comorphisme*  $P^*$  de  $\mathbb{Z}/3\mathbb{Z}[X]$  dans  $\mathbb{Z}/3\mathbb{Z}[X, Y]$  définie par : pour un polynôme  $p \in \mathbb{Z}/3\mathbb{Z}[X]$  :

$$\begin{aligned} P^*(p(X)) &= P^*(p(X_1, \dots, X_n)) \\ &= p(P_1(X, Y), \dots, P_n(X, Y)) \end{aligned}$$

où  $P_1, \dots, P_n$  sont les composants de  $P$ . En d'autres termes,  $P^*(p(X))$  est obtenu en substituant à chaque  $X_i$  dans  $p$ , le polynôme correspondant  $P_i(X, Y)$ ,  $i$ ème composante de  $P$ . En fait le comorphisme peut être vu

1.  $X^3 - X$  (resp.  $Y^3 - Y$ ) représente tous les polynômes  $X_i^3 - X_i$  (resp.  $Y_i^3 - Y_i$ ).

comme une application calculant l'ensemble des états à partir desquels, il est possible d'atteindre en un pas les états solutions de l'*équation d'évolution*.

Pour implémenter ces relations sur les idéaux et les variétés, il est nécessaire d'avoir un système de calcul symbolique performant : SIGALI. On utilise pour cela une représentation des fonctions polynomiales sous forme de graphes de décision ternaires [Le 93, Dut92], qui est une extension des graphes de décision binaires qui sont performants dans l'algèbre de Boole [Bry86, BRB90].

### 3.c Propriétés sur les systèmes dynamiques polynomiaux

Informellement, une propriété de sécurité décrit le fait que de "*mauvaises choses*" ne peuvent pas se produire [AS86]. Dans notre cas, cette sorte de propriété couvre la classe des propriétés, décrivant un ensemble de bon états, qui restent invariants. La définition suivante rappelle la définition d'un ensemble d'états invariant.

**Definition 1 Invariance.** *Un ensemble d'états  $E$  est invariant pour un système dynamique polynomial, si et seulement si pour chaque état  $x \in E$  et pour tout événement  $y$  admissible dans l'état  $x$ , l'état  $x' = P(x, y)$  appartient à  $E$ .*

De cette manière, si l'on décrit une propriété par un ensemble équivalent d'états, vérifiant la propriété, celle-ci est toujours vérifiée si et seulement si cet ensemble équivalent d'états est invariant pour le système dynamique. En partant de cette définition, on peut prouver, qu'une propriété, représentée par un ensemble d'états  $E$ , est invariante pour un système dynamique polynomial, si et seulement si :

$$\langle P^*(I(E)) \rangle \subseteq \langle Q \rangle + I(E)\mathbb{Z}/3\mathbb{Z}[X, Y]$$

Il peut cependant arriver qu'une propriété ne soit pas invariante. Dans ce cas, il est intéressant de calculer le *plus grand sous-ensemble invariant* contenu dans  $E$ . Ce calcul est réalisé à l'aide du calcul d'un point fixe. Un autre type de propriété appartient à la classe des propriétés de sécurité : l'invariance sous-contrôle d'un ensemble d'états.

**Definition 2 Invariance sous-contrôle.** *Un ensemble d'états  $E$  est invariant sous-contrôle pour un système dynamique polynomial, si et seulement si pour chaque état  $x \in E$ , il existe un événement  $y$  admissible dans l'état  $x$ , tel que l'état  $x' = P(x, y)$  appartienne à  $E$ .*

En partant de cette définition, on peut prouver, qu'une propriété, représentée par un ensemble d'états  $E$ , est invariante sous-contrôle pour un système dynamique polynomial, si et seulement si :

$$(\langle Q \rangle + P^*(I(E))) \cap \mathbb{Z}/3\mathbb{Z}[X] \subseteq I(E)$$

Il est également possible de calculer le *plus grand sous-ensemble invariant sous-contrôle* contenu dans un ensemble  $E$  donné.

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

**Definition 3 Propriété d'accessibilité.** *Un sous-ensemble  $F$  d'états est accessible pour un système dynamique, si et*

seulement si chaque état  $x \in F$  peut être atteint à partir des états initiaux du système dynamique considéré (i.e., il existe une trajectoire initialisée en  $E_0$  qui atteint  $x$ ).

La vérification de cette propriété se fait en utilisant le plus grand sous-ensemble invariant, décrit brièvement dans le paragraphe précédent. En effet, on peut prouver qu'un ensemble d'états  $F$  est accessible à partir des états initiaux d'un système dynamique si et seulement si les états initiaux du système dynamique considéré ne sont pas inclus dans le plus grand sous-ensemble invariant du complémentaire de  $F$  (Ceci revient à dire qu'il n'existe pas de trajectoires du système, initialisées en  $E_0$ , qui atteignent des états à partir desquels il est impossible d'atteindre des états vérifiant la propriété).

**Definition 4 Propriété d'attractivité.** *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  $F$ .*

On peut prouver qu'un ensemble d'états  $F$  est attractif vis-à-vis de  $E$  si et seulement si l'ensemble d'états  $E$  n'est pas inclus dans le plus grand sous-ensemble invariant sous-contrôle contenu dans le complémentaire de  $F$  (i.e., une trajectoire du système ne peut atteindre un ensemble invariant, qui ne contient pas l'ensemble  $F$ , et à partir duquel il est impossible d'atteindre  $F$ ).

Le lecteur intéressé par des études de vérification de propriétés sur divers systèmes pourra se reporter à [MRS95a] ou à [ALMR95].

### 3.d Application à la vérification du contrôleur d'un poste de transformation électrique

Cette méthodologie de preuve à été utilisée pour la vérification de la spécification d'un transformateur de tension du réseau EdF. Ce poste a pour but d'abaisser la tension du courant en vue de sa distribution dans les centres urbains. Durant l'exploitation d'un poste, plusieurs types de défauts peuvent apparaître (défaut de phase (PH), homopolaire (H), ou wattmétrique (W)). Pour protéger le matériel et l'environnement, plusieurs disjoncteurs ont été placés sur différentes parties du poste. Lors de l'apparition d'un défaut, des capteurs alertent les différents disjoncteurs, contrôlés par des contrôleurs locaux appelés *cellules* (cellule liaison, cellule arrivée et cellule départ).

*A. Description formelle d'une cellule départ.* Cette cellule a deux activités: la phase de confirmation du défaut, suivie de la phase de traitement du défaut.

La *phase de confirmation* a pour but de faire disparaître les défauts fugitifs. Pour chacun des types de défauts un délai est déclenché, permettant de tester si ce défaut est permanent ou non. Ils sont testés en séquence, jusqu'à ce qu'un d'entre eux soit confirmé (i.e., présent à la fin du délai correspondant). De plus cette séquence est interrompue dès que le défaut disparaît, ou quand un des défauts préalablement examinés apparaît.

La *phase de traitement* commence dès que le défaut a été confirmé. On alterne alors entre une ouverture du disjoncteur durant un délai variable et sa fermeture afin de vérifier

si le défaut n'a pas disparu. On ouvre le disjoncteur durant un délai donné, on le referme alors. Si le défaut est toujours présent, on répète cette opération pendant un certain nombre de cycles. Si le défaut persiste à la fin du dernier cycle, le disjoncteur est définitivement ouvert et son traitement est pris en charge par un intervenant extérieur.

La spécification de cette cellule (et des deux autres) a été réalisée en utilisant SIGNALGTi [MRS95b].

*B. Vérification de la cellule départ.* On rappelle brièvement le but de la phase de confirmation spécifié en SIGNAL. Le processus confirmation est activé lorsque le défaut apparaît (i.e., émission de l'événement `first_Defect`). Le processus émet l'événement de sortie `Def_Conf` lorsque le défaut est confirmé et le signal booléen `Defect`, donnant l'état de cette cellule. Ce dernier est *vrai* quand un défaut extérieur est détecté (réception de `Ext_Defect`), ou quand un défaut a été confirmé (`Def_Conf`), autrement il est *faux* lorsque le défaut n'est pas présent.

On veut donc analyser les propriétés suivantes :

(1) *La phase de confirmation et la phase de traitement ne sont jamais en cours aux mêmes instants*

Cette propriété peut être établie, en prouvant que l'ensemble des états correspondant à la situation où la phase de traitement et la phase de confirmation sont actives en même temps, n'est pas accessible depuis les états initiaux du système dynamique polynomial.

C'est ainsi, que l'on considère les deux intervalles `I_Treat` et `I_PH`, encodés par des booléens qui sont vrais quand le système est en phase de traitement, respectivement en phase de confirmation. Après la traduction du programme SIGNAL en système dynamique polynomial, on calcule l'ensemble des états `conf_and_treat`, où `I_Treat=1` et `I_PH=1`. La méthode consiste alors à vérifier que l'ensemble des états `conf_and_treat` n'est pas accessible depuis les états initiaux du système.

L'accessibilité de cet ensemble d'états peut être vérifiée en utilisant la fonction *reachable(prop)* qui rend *vrai* si les états vérifiant la propriété sont accessibles depuis les états initiaux et *faux* sinon. Dans notre cas le résultat est faux.

(2) *Quand un défaut apparaît, on aura nécessairement:*

(a) *La confirmation du défaut,*

(b) *ou la disparition du défaut,*

(c) *ou l'apparition d'un défaut extérieur.*

ces trois possibilités ont été spécifiées en SIGNAL par un unique booléen `DEFECT` qui est *présent* quand l'une des trois possibilités est présente<sup>2</sup>.

Cette propriété peut être prouvée en vérifiant l'attractivité de l'ensemble des états  $F$ , où `DEFECT` est *présent*, à partir de l'ensemble  $E$ , le défaut apparaît (i.e., il y a une occurrence de l'événement `First_Defect`). En appliquant la fonction *attractivity*, décrite par la définition 4, on peut prouver que  $F$  est attractif vis à vis de  $E$  (i.e., chaque fois qu'un défaut apparaît, toutes les trajectoires du système amènent dans des états où `DEFECT` est présent.

<sup>2</sup>. Ce signal est *présent* et *vrai* quand les conditions (a) et (c) sont vérifiées. il est *présent* et *faux* lorsque la condition (b) est vérifiée.

(3) Quand un défaut a été confirmé, on aura nécessairement :

- (a) Soit le défaut ne disparaît pas et le signal Def\_Break est émis,
- (b) soit le défaut disparaît avec le disjoncteur fermé

L'événement Def\_Break signale que le disjoncteur doit être ouvert définitivement (i.e., la phase de traitement n'a pu faire disparaître le défaut). la méthode utilisée pour prouver cette propriété est la même que celle utilisée pour prouver la propriété (2). On calcule l'ensemble des états  $E$ , correspondant aux états où le défaut a été confirmé, et l'ensemble  $F$  correspondant à l'union des états où la condition (a) et la condition (b) sont vérifiées. Ainsi on montre que l'ensemble  $F$  est attractif vis à vis de  $E$ . D'autres propriétés concernant cette cellule ont été prouvées. On pourra se reporter à [MRS95a] pour avoir plus de détails sur la spécification de cette cellule, ainsi que sur la preuve de propriétés.

#### 4. Une méthodologie unifiée.

Nous disposons donc avec SIGNAL (système échantillonné, flot de données), SIGNALGTi (système à événements sporadiques - préemption de tâche) et SIGALI (preuve de propriété) d'un environnement de programmation unifié permettant la conception aisée de systèmes réactifs sûrs. La spécification et la mise en œuvre d'une système robotique réactif de vision 3D. Des résultats expérimentaux sur une cellule robotique ont montré la validité de l'approche utilisée. Nous avons ensuite décrit la vérification formelle d'un contrôleur d'un poste de transformation électrique. Les perspectives autour de l'approche SIGNAL concernent aussi les systèmes hybrides, l'implémentation de systèmes distribués, etc. Nous travaillons également à la synthèse automatique de contrôleurs sur des systèmes à événements discrets [RW89] et plus précisément sur des systèmes dynamiques polynomiaux [DL94]. Une autre perspective concerne la possibilité de prouver des propriétés, dont le comportement dépend de variables numériques.

#### Remerciement

Ce travail a été en partie soutenu par Électricité de France pour la partie vérification et par le MESR à travers le projet CNRS inter PRC VIA (*Vision Intentionnelle et Action*) pour la partie vision robotique.

#### Références

- [ALMR95] Amagbegnon (T.), Le Guernic (P.), Marchand (H.) et Rutten (E.). – SIGNAL – the specification of a generic, verified production cell controller. In : *Froaml Development of Reactive Systems - Case Study Production Cell*, éd. par Lewerentz (C.) et Lindner (T.). – LNCS 891, Springer, 1995.
- [AS86] Alpern (B.) et Schneider (F. B.). – *Recognizing Safety and Liveness*. – Technical Report n86-727, Department of Computer Science Cornell University, Ithaca, New York, January 1986.
- [BRB90] Brace (K.S.), Rudell (R.L.) et Bryant (R.E.). – Efficient implementation of a BDD package. In : *27th*

ACM/IEEE design automation conference, pp. 40–45.

- [Bry86] Bryant (R.E.). – Graph-based algorithms for boolean function manipulations. *IEEE Transaction on Computers*, vol. C-45, n8, August 1986, pp. 677–691.
- [CBBJ94] Chaumette (F.), Boukir (S.), Bouthemy (P.) et Juvin (D.). – Optimal estimation of 3D structures using visual servoing. In : *Proc. of IEEE Int. Conf. on Computer Vision and Pattern Recognition*, pp. 347–354. – Seattle, USA, June 1994.
- [DL94] Dutertre (B.) et Le Borgne (M.). – *Control of Polynomial Dynamic Systems : an Example*. – Rapport technique, INRIA, No 2193, January 1994.
- [Dut92] Dutertre (B.). – *Spécification et preuves de systèmes dynamiques : Application à SIGNAL*. – IRISA, Thèse de PhD, Université de Rennes I, December 1992.
- [Hal93] Halbwachs (N.). – *Synchronous programming of reactive systems*. – Kluwer, 1993.
- [LBL91] Le Borgne (M.), Benveniste (A.) et Le Guernic (P.). – Dynamical systems over galois fields and dedcs control problems. In : *Proc. of 33<sup>th</sup> IEEE Conf. on Decision and Control*, pp. 1505–1509.
- [Le 93] Le Borgne (M.). – *Systèmes dynamiques sur des corps finis*. – IRISA, Thèse de PhD, Université de Rennes I, September 1993.
- [LLGL91] Le Guernic (P.), Le Borgne (M.), Gautier (T.) et Le Maire (C.). – Programming real time application with SIGNAL. *Proc. of the IEEE*, vol. 79, n9, September 1991, pp. 1321–1336.
- [MCR95] Marchand (E.), Chaumette (F.) et Rutten (E.). – Real time active visual reconstruction using the synchronous paradigm. In : *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'95*, pp. 96–102. – Pittsburgh, USA, August 1995.
- [MRS95a] Marchand (H.), Rutten (E.) et Samaan (M.). – *Specifying and verifying a transformer station in SIGNAL and SIGNALGTI*. – Rapport technique n2521, INRIA, 1995.
- [MRS95b] Marchand (H.), Rutten (E.) et Samaan (M.). – Synchronous design of a transformer station controller in SIGNAL. In : *Proc of 4th IEEE Conf. on Control Applications*. – Albany, New York, USA, September 1995.
- [RL94] Rutten (E.) et Le Guernic (P.). – The sequencing of data flow tasks in SIGNAL. In : *Proc. of the ACM SIGPLAN Workshop on Language, Compiler and Tool Support for Real-Time Systems*. – Orlando, Florida, June 1994.
- [RW89] Ramadge (P.J.) et Wonham (W.M.). – The control of discrete events systems. *Proc. of the IEEE*, vol. 77, n 1, January 1989, pp. 81–97.
- [SECK93] Simon (D.), Espiau (B.), Castillo (E.) et Kapellos (K.). – Computer-aided design of a generic robot controller handling reactivity and real-time controller issues. *IEEE Trans. on Control Systems Technology*, vol. 1, n4, December 1993, pp. 213–229.