# Implémentation de superviseurs générés par Synthèse de Contrôleurs Discrets

Emil Dumitrescu [1], Mingming Ren [1], Laurent Piétrac [1], Eric Niel [1]

[1]Laboratoire Ampère
INSA de Lyon, 20, Av. Albert Einstein, 69100 Villeurbanne Cedex, France

emil.dumitrescu@insa-lyon.fr, mingming.ren@insa-lyon.fr,
laurent.pietrac@insa-lyon.fr, eric.niel@insa-lyon.fr

*Résumé*— **Nous examinons l'implémentation de superviseurs générés par des techniques symboliques de Synthèse de Contrôleurs Discrets (SCD), basées sur l'utilisation des BDDs. La technique d'implémentation que nous proposons résout l'indéterminisme de contrôle ainsi que l'incompatibilité structurelle introduite par la SCD symbolique. Les propriétés structurelles du contrôleur implémenté démontrent l'intérêt de cette démarche. Notre technique est illustrée sur un exemple réel modélisant un composant d'un "Système sur une puce" : un convertisseur série-parallèle.**

*Mots-clés*— **Synthèse de Contrôleurs Discrets, parcours symbolique, Diagrammes de Décision Binaire (BDD), Systèmes à Evénements Discrets (SED)**

## I. Introduction

La problématique d'implémentation de superviseurs que nous examinons est spécifique au contexte de la Synthèse de Contrôleurs Discrets (SCD) [1–3]. Plus précisément, nous nous intéressons aux techniques de SCD *symbolique* [2, 3], car elles sont plus efficaces pour manipuler des espaces d'états de taille importante, qui caractérisent les systèmes réels.

Le problème de l'implémentation suscite plusieurs questions liées à efficacité, l'autonomie, la tolérance aux fautes, la distribution, la robustesse et la taille du superviseur implémenté. Pourtant, avant de pouvoir se pencher sur ces aspects, il faut préalablement résoudre deux problèmes principaux.

D'une part, le *non-déterminisme de contrôle*. La plupart des superviseurs synthétisés sont non-déterministes, à savoir que s'il existe plusieurs possibilités de contrôle à un instant donné, alors toutes sont disponibles. Un choix doit être fait, soit par l'opérateur humain, soit par des politiques de choix définies a priori afin de résoudre ce non-déterminisme. Les superviseurs déterministes sont rarement obtenus par application directe de la SCD.

D'autre part, l'*incompatibilité structurelle* vis-à-vis des exigences de la conception modulaire. Ce problème ne concerne que la SCD symbolique. En effet, les superviseurs symboliques ne disposent pas d'entrées/sorties . Ils sont représentés par une équation Booléenne encodant toutes les solutions de contrôle acceptables. Une approche directe d'implémentation serait de résoudre continuellement l'équation du superviseur durant l'exécution du système contrôlé. Ceci requiert un solveur d'équation Booléennes, exécuté physiquement sur un microprocesseur (dédié ou pas). L'architecture cible du superviseur implémenté serait ainsi une architecture logicielle, ce qui pose des limitations en termes d'efficacité pendant l'exécution.

*Contributions*

Notre objectif est de résoudre l'indéterminisme de contrôle et trouver une solution adéquate au problème de l'incompatibilité structurelle. La résolution de l'indéterminisme de contrôle garantit une implémentation autonome vis-à-vis du choix des solutions de contrôle. La résolution de l'incompatibilité structurelle revient à trouver une représentation du superviseur sous forme d'un ensemble de fonctions de contrôle. Cette solution présente un double avantage. Premièrement et le plus important, une implémentation matérielle peut être obtenue, sous forme de circuit électronique. Ceci est intéressant lorsque les performances d'exécution du système contrôlé ont de l'importance. Deuxièmement, l'implémentation d'un superviseur par un ensemble de fonctions de contrôle semble beaucoup plus compacte que le superviseur initial.

Par ailleurs, nous tentons d'adapter l'architecture en "boucle de contrôle" classique au contexte de la conception matérielle. Nous identifions une catégorie de problèmes de conception matérielle pour laquelle notre architecture de contrôle est appropriée.

*État de l'art*

L'implémentation de superviseurs a été examinée dans le passé, avec une attention particulière accordée à l'exécution de contrôleurs sur un automate programmable industriel [4–7].

La plupart des travaux de recherche à notre connaissance ont abordé le problème de l'implémentation de superviseurs à travers une démarche de raffinements successifs de l'objectif de contrôle aboutissant sur un superviseur déterministe (i.e. un contrôleur). D'autres approches préconisent la mise en place d'un choix aléatoire en cas de non-déterminisme. Le même problème peut aussi être traité en choisissant un mécanisme de priorité fixe ou dynamique.

Notre technique permet d'obtenir un contrôleur déterministe, à travers une étape de décomposition structurelle appliquée au superviseur, décomposition qui conserve l'ensemble des solutions de contrôle obtenues par synthèse. Des approches similaires ont été proposées et utilisées dans divers contextes, pas toujours en relation avec la synthèse

de contrôleurs. Dans [8], une technique de décomposition paramétrique est appliquée sur des prédicats Booléens dans le contexte de la vérification formelle de systèmes matériels. Plus proche de notre travail, dans [9] on présente une technique de mise en oeuvre de contrôleurs obtenus par synthèse optimale ; la technique est également symbolique, mais engendre seulement un sous-ensemble strict du superviseur. Dans [10] les auteurs étudient la triangulation d'une équation polynômiale, avec le même objectif d'atteindre une implémentation de superviseurs, en utilisant une représentation en logique ternaire. Le même objectif d'implémentation est examiné dans [11], où la technique de synthèse de contrôleurs opère directement depuis un ensemble de spécifications formelles (contrairement à la SCD, qui a besoin d'un modèle de procédé). Une autre approche d'implémentation matérielle de superviseurs, basée sur des réseaux de Pétri, est présentée dans [12].

L'organisation de ce papier est la suivante : La section II présente les définitions de concepts basiques et des notations. Notre méthode d'implémentation de superviseurs est présentée dans la section III. La section IV illustre l'application de notre méthode sur un système matériel. Enfin, la section V présente l'ensemble des outils que nous avons utilisés ou développés.

## II. Définitions

### A. Machines d'états finis

Cette section introduit les concepts basiques et les notations nécessaires par la suite. De plus amples détails peuvent être trouvés dans [2], [13]. Soit $\mathbb{B} = \{0, 1\}$ l'ensemble des valeurs Booléennes. Soit $P$ une machine à états finis définie par :

$$P = \langle I, S, \delta, s_0, O, \lambda \rangle$$

où :

- $I$ est l'ensemble des variables Booléennes d'entrée, tel que $I = U \cup C$, et $U \cap C = \emptyset$ ;
- $U = \{u_0, \cdots, u_{r-1}\}, r > 0$ est l'ensemble des variables incontrôlables d'entrée. Nous notons $\mathbf{u} = (\mathbf{u_0}, \cdots, \mathbf{u_{r-1}})$ un tuple des éléments de $U$ ;
- $C = \{c_0, \cdots, c_{p-1}\}, p > 0$ est l'ensemble des variables contrôlables d'entrée. Nous notons $\mathbf{c} = (\mathbf{c_0}, \cdots, \mathbf{c_{p-1}})$ un tuple des éléments de $C$ ;
- $S = \{s_0, \cdots, s_{n-1}\}, n > 0$ est l'ensemble des variables Booléennes d'états. Nous notons $\mathbf{s} = (\mathbf{s_0}, \cdots, \mathbf{s_{n-1}}), \mathbf{n > 0}$ un tuple des éléments de $S$ ;
- $\delta : \mathbb{B}^{p+r} \times \mathbb{B}^n \to \mathbb{B}^n$ est la fonction de transition de $P$ ;
- $s_0 \in \mathbb{B}^n$ est l'état initial de $P$ ;
- $O = \{o_0, \cdots, o_{m-1}\}, m > 0$ est l'ensemble des variables Booléennes de sortie ;
- $\lambda : \mathbb{B}^{p+r} \times \mathbb{B}^n \to \mathbb{B}^m$ est la fonction de sortie associée à $O$.

Dans la suite, nous appelons $P$ le *procédé* i.e. le système à contrôler par SCD. Les ensembles d'états de $P$ sont manipulés par l'intermédiaire de leur fonction caractéristique. Soit $E \subset \mathbb{B}^n$. La fonction caractéristique de $E$ est définie par

$$\mathcal{C}_E : \mathbb{B}^n \to \mathbb{B} \text{ et } \mathcal{C}_\mathcal{E}(e) = 1 \Leftrightarrow e \in E$$

Les opérations ensemblistes ordinaires ont des opérateurs Booléens correspondants : $''+''$ ("ou" logique) réalise l'union d'ensembles et $''.''$ ("et" logique) réalise l'intersection d'ensembles. La négation logique $\overline{\mathcal{C}_E}$ exprime le complément de $E$ par rapport à $\mathbb{B}^n$.

Soit $S' = \{s'_0, \cdots, s'_{n-1}\}$ un ensemble de variables dites de *"prochain état"*. La relation de transition $\mathcal{T}$ de $P$ est définie par l'ensemble de toutes les transitions possibles $\mathbf{s} \overset{\mathbf{u,c}}{\longmapsto} \mathbf{s'}$ tel que $\mathbf{s'} = \delta(\mathbf{s}, \mathbf{u}, \mathbf{c})$ :

$$\mathcal{T}(\mathbf{s}, \mathbf{u}, \mathbf{c}, \mathbf{s'}) = \prod_{\mathbf{i=0}}^{\mathbf{n-1}} \mathbf{s'_i} \Leftrightarrow \delta_\mathbf{i}(\mathbf{u}, \mathbf{c}, \mathbf{s})$$

où l'opérateur "produit" réalise un "et" logique sur $n$ opérandes.

### B. Synthèse de contrôleurs discrets

La SCD a été développée dans [1], utilisant la théorie des langages. Parallèlement, des travaux de recherche en conception de circuits ont démontré le grand intérêt de la représentation symbolique d'ensembles d'états utilisant les BDDs [14], pour traiter le problème de l'explosion combinatoire de l'espace d'états [13]. Des développements de la SCD basés sur des techniques symboliques ont été proposés dans [2, 3, 15].

L'approche symbolique de SCD manipule des ensembles d'états et/ou transitions, au lieu des langages. Pour un procédé donné $P$ et une spécification désirée $Q$, nommée un *objectif de contrôle*, la SCD symbolique calcule un superviseur $\mathcal{X}$ garantissant que $P$ satisfait toujours $Q$. L'architecture de contrôle est illustrée dans la Figure 1.

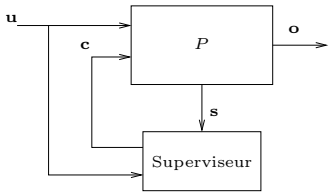


Fig. 1. Architecture de contrôle

Le calcul est composé par les deux étapes suivantes :

- calculer le sous-ensemble dit de l'*invariant sous contrôle* ($\mathcal{IUC}$) d'états de $P$. Tant que $P$ reste dans $\mathcal{IUC}$, le non respect de $Q$ peut être constamment évité. Les détails du calcul de $\mathcal{IUC}$ dépassent le cadre de ce papier. Ils peuvent être consultés dans [2, 15] ;
- calculer le superviseur $\mathcal{X}$ : l'ensemble de toutes les transitions $\mathbf{s} \overset{\mathbf{u,c}}{\longmapsto}$ de $P$ aboutissant à $\mathcal{IUC}$. L'expression précise du superviseur est :

$$\mathcal{X}(\mathbf{s}, \mathbf{u}, \mathbf{c}) = \exists \mathbf{s'} : \mathcal{T}(\mathbf{s}, \mathbf{u}, \mathbf{c}, \mathbf{s'}).\mathcal{IUC}(\mathbf{s'})$$

Une solution de contrôle existe ssi $s_0 \in \mathcal{IUC}$. D'un point de vue dynamique, on dit que $\mathcal{X}$ "joue" avec les entrées contrôlables $C$, contre l'environnement, "jouant" avec les entrées incontrôlables $U$ de $P$, avec l'objectif de ne jamais atteindre un état violant $Q$. Ainsi, pour tous les états $\mathbf{s} \in \mathcal{IUC}$ et pour tous les vecteurs d'entrées incontrôlables $\mathbf{u} \in \mathbb{B}^\mathbf{r}$, le superviseur $\mathcal{X}$ calcule les valeurs adéquates pour les entrées contrôlables $\mathbf{c}$ afin que la transition tirée par $P$ aille vers un état de $\mathcal{IUC}$.

Le superviseur symbolique $\mathcal{X}$ a deux propriétés importantes : (I) il est *maximalement permissif* i.e. toutes les transitions de $P$ allant vers $\mathcal{IUC}$ sont comprises dans $\mathcal{X}$ et (II) il est *non déterministe du point de vue du contrôle* i.e. pour un état donné et un valeur d'entrée incontrôlable, il peut exister plus d'un successeur possible dans $\mathcal{IUC}$. Notre objectif est d'*implémenter* $\mathcal{X}$, ce qui revient à résoudre l'indéterminisme de contrôle, tout en restant maximalement permissif. Cette démarche est développée dans la section suivante.

## III. Implémentation Symbolique du Superviseur

### A. Conditions pour la compatibilité structurelle

En général, l'implémentation de superviseurs se traduit par une résolution de l'indéterminisme de contrôle. La solution déterministe de contrôle obtenue s'appelle un *contrôleur* et est généralement un sous-ensemble du superviseur initialement synthétisé.

Cependant, en plus du non-déterminisme, la SCD symbolique introduit aussi un problème architectural : le superviseur $\mathcal{X}$ est représenté par une fonction caractéristique Booléenne encodant les transitions acceptables par rapport à l'objectif de contrôle. L'expression Booléenne de $\mathcal{X}$ est structurellement incompatible avec $P$ et ainsi avec l'architecture de contrôle représentée dans Figure 1. Cette incompatibilité structurelle est en générale éliminée en résolvant l'équation Booléenne :

$$\mathcal{X}(\mathbf{s}, \mathbf{u}, \mathbf{c}) = \mathbf{1} \tag{1}$$

soit "en-ligne" soit "hors-ligne".

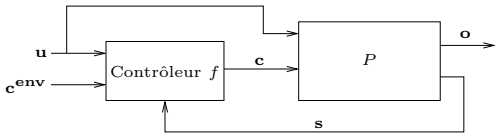


Fig. 2. Architecture de contrôle cible

La technique d'implémentation que nous présentons traite ces deux deux aspects, comme illustré dans la Figure 2. Ainsi, l'indéterminisme de contrôle est rendu explicite, en ajoutant les variables supplémentaires d'environnement $\mathbf{c^{env}}$. L'incompatibilité structurelle est résolue en calculant une expression individuelle pour chaque variable contrôlable $\mathbf{c}$ de $P$. Cela revient à résoudre l'équation (1) hors-ligne. A partir du superviseur $\mathcal{X}$, nous construisons un contrôleur sous la forme d'un vecteur de $p$ fonctions de contrôle :

$$f_i : \mathbb{B}^n \times \mathbb{B}^{p+r} \to \mathbb{B},\ i = 0 \ldots p-1,\ \text{tel que :}$$
$$c_i = f_i(s_0, \ldots, s_{n-1}, u_0, \ldots, u_{r-1}, c_0^{env}, \ldots, c_{p-1}^{env}) \tag{2}$$

Les variables auxiliaires $\mathbf{c^{env}}$ sont requises afin d'exprimer le choix non-déterministe parmi plusieurs valeurs contrôlables.

La relation entre $\mathcal{X}$ et $f$ est exprimée par l'équation suivante :

$$\mathcal{X}(\mathbf{s}, \mathbf{u}, \mathbf{c}) = \exists \mathbf{c_0^{env}}, \ldots, \mathbf{c_{p-1}^{env}} : \prod_{\mathbf{i=0}}^{\mathbf{p-1}} (\mathbf{c_i} \Leftrightarrow \mathbf{f_i}(\mathbf{s}, \mathbf{u}, \mathbf{c^{env}})) \tag{3}$$

TABLE I
Valeurs de $c_i$ lors que $\mathcal{X}$ est satisfaisant

| $\mathcal{X}\|_{c_i=1}$ | $\mathcal{X}\|_{c_i=0}$ | $c_i$ |
|---|---|---|
| 0 | 1 | **0** |
| 1 | 0 | **1** |
| 1 | 1 | **0 or 1** ($c_i^{env}$) |

Ainsi, notre objectif est de trouver une implémentation $f$ telle que (I) toutes les solutions de contrôle permises par $\mathcal{X}$ peuvent être reproduites par $f$ et (II) toutes les solutions de contrôle données par $f$ sont aussi acceptées par $\mathcal{X}$. De plus, trouver $f$ satisfaisant l'équation (3) revient également à rendre explicite l'indéterminisme de contrôle de $\mathcal{X}$ en utilisant les variables $\mathbf{c^{env}}$.

### B. Algorithme d'implémentation symbolique

Notons en préalable que pour pouvoir être implémenté, tout superviseur $\mathcal{X}$ doit être satisfaisable :

$$\forall \mathbf{s} \in \mathcal{IUC}, \forall \mathbf{u} \in \mathbb{B}^{\mathbf{r}}, \exists \mathbf{c} : \mathcal{X}(\mathbf{s}, \mathbf{u}, \mathbf{c}) \tag{4}$$

Notre algorithme d'implémentation utilise la décomposition de Boole appliquée à $\mathcal{X}$, qui est appliquée récursivement aux variables $c_i, i = 0 \ldots p-1$ :

$$\mathcal{X} = \overline{c_i}.\mathcal{X}|_{c_i=0} + c_i.\mathcal{X}|_{c_i=1}$$

Les valeurs de $c_i$ satisfaisant la décomposition de Shannon et telles que $\mathcal{X}$ reste satisfaisable sont résumées par la table de vérité présentée dans Table I.

Cette table résume les valeurs que la variable contrôlable $c_i$ peut prendre, afin que l'équation de superviseur (1) soit satisfaisable. Comme montré par cette table, la satisfaction simultanée des deux cofacteurs $\mathcal{X}|_{c_i=0}$ et $\mathcal{X}|_{c_i=1}$ a une signification particulière : quelle que soit la valeur de $c_i$, toutes les transitions permises par $\mathcal{X}$ vont vers $\mathcal{IUC}$. Ceci caractérise l'indéterminisme de contrôle par rapport à la variable contrôlable $c_i$. A chaque fois que $c_i$ n'a pas d'influence sur la satisfaisabilité de $\mathcal{X}$, sa valeur est déterminée par $c_i^{env}$. Structurellement, les variables $c^{env}$ sont des variables auxiliaires d'environnement (entrée) comme montré dans la Figure 2.

D'après Table I, l'expression Booléenne suivante calcule le valeur de $f_i$, associé à la variable contrôlable $c_i$ :

$$f_i = \overline{\mathcal{X}|_{c_i=0}}.\mathcal{X}|_{c_i=1} + c_i^{env}.\mathcal{X}|_{c_i=1}.\mathcal{X}|_{c_i=0} \tag{5}$$

L'expression (5) représente les étapes basiques de l'algorithme d'implémentation de contrôleur présenté dans Algorithme 1 : $f_0$ est calculé en supposant que $\mathcal{X}$ soit vrai. $c_0$ est substitué par $f_0$ dans $\mathcal{X}$, produisant $\mathcal{X}_1$. Ensuite l'algorithme calcule $f_1$ assumant que $\mathcal{X}_1$ soit vrai, etc.
L'algorithme 1 produit le vecteur de fonctions :

$$f = \begin{pmatrix} f_0(\mathbf{s}, \mathbf{u}, \mathbf{c_0^{env}}, \mathbf{f_1}, \ldots, \mathbf{f_{p-1}}) \\ f_1(\mathbf{s}, \mathbf{u}, \mathbf{c_1^{env}}, \mathbf{f_2}, \ldots, \mathbf{f_{p-1}}) \\ \ldots\ldots \\ f_{p-1}(\mathbf{s}, \mathbf{u}, \mathbf{c_{p-1}^{env}}) \end{pmatrix}$$

**Algorithm 1** Algorithme d'implémentation de contrôleur

**Require:** $\mathcal{X}$ un superviseur satisfaisant
1: {démarre avec $\mathcal{X}$ et calcule $f_0, \ldots, f_{p-1}$}
2: {résultats intermédiaire : $\mathcal{X}_0, \ldots \mathcal{X}_p$}
3: $\mathcal{X}_0 \leftarrow \mathcal{X}$
4: **for** $i = 0 \rightarrow p-1$ **do**
5: $f_i \leftarrow \overline{\mathcal{X}_i|_{c_i=0}}.\mathcal{X}_i|_{c_i=1} + c_i^{env}.\mathcal{X}_i|_{c_i=1}.\mathcal{X}_i|_{c_i=0}$
6: $\mathcal{X}_{i+1} \leftarrow$ substituer $c_i$ par $f_i$ dans $\mathcal{X}_i$
7: **end for**

ainsi qu'une expression résiduelle $\mathcal{X}_p(\mathbf{s}, \mathbf{u})$ obtenue par les substitutions successives de $c_i$ par $f_i$ dans $\mathcal{X}$. Afin de garantir que toutes les solutions de contrôle produites par $f$ sont acceptées par $\mathcal{X}$, nous devons montrer que $\mathcal{X}_p$ est une tautologie : $\forall \mathbf{s} \in \mathcal{IUC}, \forall \mathbf{u} \in \mathbb{B}^{\mathbf{r}} : \mathcal{X}_{\mathbf{p}}$

**Théorème 1** *$\mathcal{X}_p$ est une tautologie.*

*Eléments de preuve.* En appliquant les substitutions successives de $f_i$ dans $\mathcal{X}_i$, l'expression suivante est obtenue pour $\mathcal{X}_p$ :

$$\mathcal{X}_p = \exists c_{p-1}, \ldots, \exists c_0 : \mathcal{X}(\mathbf{s}, \mathbf{u}, \mathbf{c})$$

Ainsi, le prédicat $\forall \mathbf{s} \in \mathcal{IUC}, \forall \mathbf{u} \in \mathbb{B}^{\mathbf{r}} : \mathcal{X}_{\mathbf{p}}$ est identique à notre hypothèse initiale exprimée dans (4). □

Ce théorème montre qu'en substituant $f$ à la place des $\mathbf{c}$ dans $\mathcal{X}$, nous obtenons une tautologie et par conséquent toutes les solutions de contrôle produites par $f$ sont contenues dans $\mathcal{X}$. A présent il faut démontrer que :

**Théorème 2** *Toutes les solutions de contrôle comprises dans $\mathcal{X}$ peuvent être reproduites par $f$.*

*Eléments de preuve.* La déclaration ci-dessus est équivalente à :

$$\forall \mathbf{s}, \forall \mathbf{u}, \forall \mathbf{c} : (\mathcal{X}(\mathbf{s}, \mathbf{u}, \mathbf{c}) \implies \exists \mathbf{c^{env}} : \mathbf{c} = \mathbf{f}(\mathbf{s}, \mathbf{u}, \mathbf{c^{env}}))$$

En appliquant la quantification existentielle et en substituant l'expression de $f$ donnée dans (5), le terme à la droite de l'implication est équivalent à $\mathcal{X} + \overline{c_i}.\overline{\mathcal{X}|_{c_i=0}}.\overline{\mathcal{X}|_{c_i=1}}$. Ainsi, l'implication ci-dessus est une tautologie. □

Il est important de souligner que le contrôleur $f$ obtenu dépend de l'ordre des variables utilisé lors de l'application de la décomposition de Shannon. Cet ordre établit une priorité d'évaluation : $f_{p-1}$ est évalué d'abord, et son valeur est utilisé pour évaluer $f_{p-2} \ldots f_0$.

La complexité de notre algorithme dépend fortement de la taille de $\mathcal{X}$ en termes de noeuds de BDD. La plus coûteuse opération est le cofacteur généralisé, qui est utilisée pour substituer une expression pour une variable de BDD. Cette opération est exponentielle au nombre de variables de $\mathcal{X}$.

La génération de contrôleur est illustrée dans la prochaine section sur un composant matériel modélisant un convertisseur série-parallèle.

## IV. Exemple

### A. Le convertisseur série-parallèle

Nous illustrons l'application de notre technique de génération de contrôleurs dans un contexte de conception matérielle, avec le but de construire un convertisseur série-parallèle correct par construction.

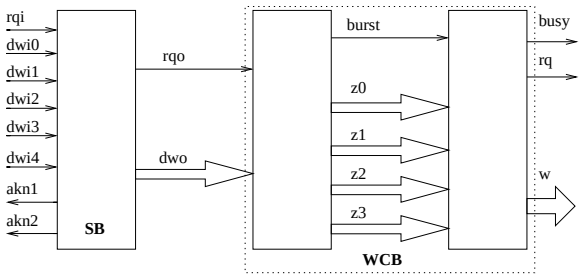


Fig. 3. Architecture du convertisseur série-parallèle

La figure 3 présente l'architecture d'un convertisseur série-parallèle. Il est composé par deux composants : un tampon série (SB) et un accumulateur (WCB) qui interagissent par l'intermédiaire d'un produit synchronisé. De plus amples détails de la modélisation peuvent être trouvés dans [16].

Le composant $SB$ reçoit des données série en 4-bits et ensuite fait un contrôle de parité. Selon le résultat, $SB$ envoie un acquittement ou un code d'erreur par *akn* à l'environnement.

Les données série arrivent selon une transaction bien établie : l'environnement envoie les données sur *dw* et positionne une requête d'entrée *rqi*, signifiant que des données d'entrée valides sont disponibles. Après une transition, correspondant à un moment de temps synchrone, $SB$ acquitte l'envoyeur en envoyant le résultat du contrôle de parité. Ainsi, transactions entrantes dure toujours deux moments de temps synchrone. A noter que le mécanisme de "poignée de main" n'est utilisé que pour garantir la correction des données entrantes par rapport au contrôle de parité. De plus, l'environnement est supposé maintenir sa requête et les données jusqu'à qu'il soit acquitté.

Le $SB$ renvoie toutes les données série correctes au $WCB$. Ce composant implémente deux comportements :
- il accumule les données série dans un tampon interne, en construisant des mots de 8 bits ;
- lorsque le tampon interne est plein, il est purgé en renvoyant en parallèle toutes les données accumulées. Pendant l'envoi, $WCB$ est *occupé* et il ne peut plus recevoir de données entrantes. Il devient disponible à nouveau dès que l'envoi est fini.

$SB$ implémente un comportement de "pipeline", i.e. à un moment donné $t$ il peut simultanément envoyer données séries correctes au $WCB$ et acquitter une donnée série entrante.

### B. Analyse de correction

Les composants $SB$ et $WCB$ sont des composants sur étagère, qui ont été vérifiés complètement et indépendamment l'un de l'autre. Ils sont corrects par rapport à leur spécifications fonctionnelles. Pourtant, ils n'ont pas été conçus pour fonctionner ensemble.

Nous souhaitons obtenir un convertisseur série-parallèle comme une nouvelle fonctionnalité obtenue en combinant deux composants existants et corrects. Pourtant, lorsque $SB$ et $WCB$ sont combinés, le composant résultant ne fonctionne pas comme attendu. Une propriété importante

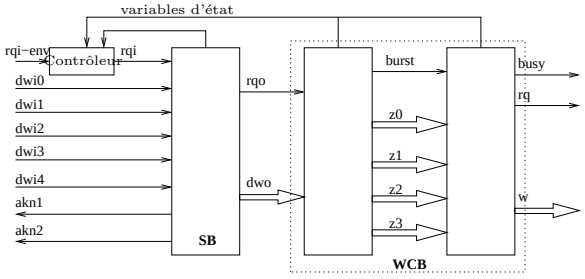


Fig. 4. Correction d'erreurs de conception par SCD

que le convertisseur doit avoir est que toutes les données série doivent être retransmises, i.e. une donnée série n'est jamais perdue. Cette propriété est exprimée par la formule temporelle $CTL$ [17] suivante :

$$\textit{Pas de perte : } AG\ \overline{busy \wedge rqo}$$

Ainsi, $SB$ n'enverra jamais une donnée au $WCB$ pendant un flush. Cette propriété est vraie sauf pour le scénario suivant : à certain moment $t$, $WCB$ a presque rempli tous ses tampons internes sauf un emplacement disponible pour une donnée série. En même temps, $SB$ renvoie une données série au $WCB$ et acquitte une donnée série simultanément. Au moment $t+1$, $WCB$ est plein et commence à purger les données accumulées. Pourtant, en même temps, $SB$ lui envoie la donnée série qu'il vient d'acquitter. Comme $WCB$ est occupé durant la purge, cette donnée est perdue. Ce scénario est confirmé par "model-checking" symbolique, qui démontre que la propriété *"Pas de perte"* est fausse.

Notre but est de garantir la satisfaction de la propriété *"Pas de perte"* sans réécrire ni $SB$ ni $WCB$. Nous utilisons la SCD symbolique afin de synthétiser un "patch" qui assure la satisfaction de la propriété ci-dessus.

### C. Correction de l'erreur par SCD

Afin d'appliquer la SCD au convertisseur, il faut préalablement déterminer les variables d'entrée contrôlables. Il n'y a pas d'indication a priori sur la ou les entrées à contrôler. Pourtant, on observe qu'en retardant judicieusement l'arrivée des données série, il est possible d'éviter la perte de données. Techniquement, retarder l'arrivée d'une donnée série entrante revient à retarder l'arrivée de la requête entrante de $rqi$. Ainsi, nous choisissons $rqi$ comme la variable contrôlable d'entrée. Les variables d'entrée restantes sont considérées comme étant incontrôlables.

Le superviseur a été synthétisé par l'outil de SCD *Sigali* [18]. Ensuite, nous avons implémenté le superviseur en générant un contrôleur grâce à l'Algorithme 1 présenté dans la Section III. L'architecture du convertisseur contrôlé est représentée dans la Figure 4, et correspond donc à notre objectif d'architecture représenté dans la Figure 2.

L'algorithme d'implémentation a introduit la variable auxiliaire de d'environnement $rqi - env$, correspondant à la variable contrôlable $rqi$. A noter que la requête entrante devient $rqi - env$, qui est pilotée par l'environnement et lue par le contrôleur synthétisé. Selon la valeur de $rqi - env$ et l'état interne du procédé (i.e. le convertisseur série-parallèle), le contrôleur affecte les valeurs adéquates à $rqi$.

Par conséquent, le contrôleur que nous obtenons fonctionne comme un filtre sur l'entrée $rqi - env$. Lorsqu'une donnée série arrive, soit le contrôleur la transmet au $SB$, soit il la retarde si la donnée peut être perdue. A tout instant $t$, si une série donnée arrive, $rqi - env$ est activé par l'environnement. Deux scénarios sont possibles :

- il ne reste qu'un emplacement mémoire libre dans $WCB$, et cet emplacement va être rempli par $SB$ avec une donnée acquise précédemment, à l'instant $t-1$. En ce moment, si $rqi - env$ est acquitté, la donnée sera perdue à l'instant $t+1$. Cependant, dès la réception de $rqi - env$ le contrôleur affecte $rqi = 0$. De cette manière, la requête entrante n'est par transmise au $SB$ et de ce fait, son acquittement sera différé ;
- il reste plus d'un emplacement mémoire libre dans $WCB$. Dans ce cas, lors de la réception de $rqi - env$ le contrôleur affecte $rqi = 1$. La requête entrante est transmise au $SB$ pour qu'elle soit acquittée.

Ainsi, la transaction d'entrée est retardée, dans le sens où $SB$ ne la reçoit pas, tant que la retransmission de la donnée série entrante n'est pas garantie sans perte. A noter qu'initialement, les transactions entrantes n'étaient utilisées que pour assurer la correction de parité ; en appliquant la SCD, ces transactions garantissent en plus que le convertisseur série-parallèle est réellement prêt pour le traitement complet d'une donnée série.

L'indéterminisme de contrôle induit par la synthèse du superviseur est "déplacé" vers l'environnement. Lorsqu'un choix non-déterministe existe sur la valeur de $rqi$, toute valeur affectée par l'environnement à $rqi - env$ se répercute sur $rqi$. C'est donc à l'environnement du procédé de lever ce non-déterminisme.

Le contrôleur que nous obtenons est un BDD comprenant les entrées et les variables d'état du convertisseur. Il comprend environ 200 noeuds, et ne peut de ce fait pas être représenté.

## V. Mise en œuvre

Le convertisseur série-parallèle a été modélisé en utilisant le langage des Automates de Mode et le compilateur Matou [19]. Le superviseur a été synthétisé par l'outil de SCD symbolique *Sigali* [18]. Les simulations intéractives ont été réalisées en utilisant l'outil *SigalSimu*. Le flot de conception est illustré dans la Figure 5.

Notre algorithme d'implémentation a été codé avec l'aide de la bibliothèque CUDD [20]. Il lit le superviseur produit par *Sigali* et produit un contrôleur qui est composé de plusieurs BDDs, chacun représentant une fonction de contrôle. Après l'étape d'implémentation, le procédé contrôlé par le contrôleur implémenté peut être vérifié en utilisant des outils de simulation ou de vérification formelle, et peut également être synthétisé vers une cible matérielle.

## VI. Conclusion

Nous avons présenté et illustré une technique d'implémentation de superviseurs obtenus par SCD symbolique. Cette technique résout le non-déterminisme de contrôle ainsi que l'incompatibilité structurelle du superviseur par rapport au procédé. L'avantage très important de notre méthode est la conservation de l'intégralité des solutions de contrôle. Cette propriété est très intéressante par rapport à une

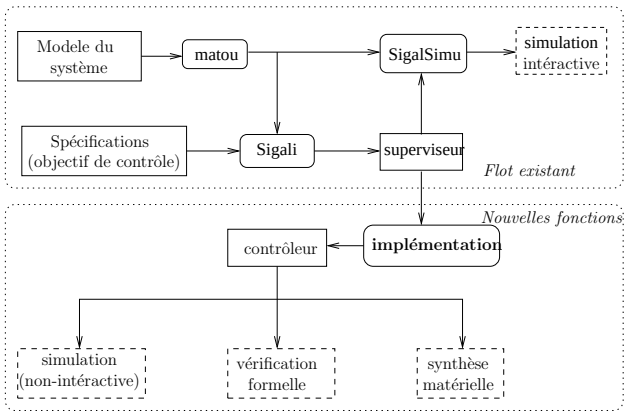


Fig. 5. Flot de conception obtenu grâce à l'implémentation de contrôleurs

déterminisation simpliste du superviseur synthétisé, qui produit presque toujours un sous-ensemble strict de $\mathcal{X}$. De plus, on note que le superviseur implémenté $f$ est généralement plus compact que $\mathcal{X}$, ce qui rend son utilisation plus efficace.

L'architecture de contrôle que nous proposons est une interprétation particulière de la boucle de contrôle traditionnelle, illustrée dans la Figure 1. Nous avons adapté cette architecture de contrôle au contexte de la conception matérielle. Dans ce contexte, la distinction entre variables contrôlables et incontrôlables n'est pas naturelle. Les variables d'entrée sont généralement dédiées à l'environnement du procédé. C'est pourquoi l'architecture de contrôle que nous proposons ne change pas l'interface d'entrée/sortie du procédé. Les entrées contrôlables sont en réalité *filtrées* par le contrôleur. Il faut noter cependant que le fait de filtrer les entrées d'un composant peut se révéler dangereux par rapport à son comportement recherché. Dans la Section IV nous avons identifié une classe d'applications où le filtrage des entrées peut être effectué sans risque.

Notons que l'implémentation des superviseurs produits par des techniques énumératives est algorithmiquement plus simple, car ils n'ont pas d'incompatibilité structurelle par rapport au procédé à contrôler. Implémenter un superviseur explicite revient à résoudre l'indéterminisme de contrôle. Pourtant, la SCD énumérative est coûteuse lors de son application à des systèmes de taille réelle. Notre choix pour les techniques de SCD symbolique est motivé par leur capacité de traiter des systèmes de taille réelle : bien que toujours sujettes à l'explosion combinatoire, elles restent bien plus efficaces que les techniques de SCD énumérative basées sur [1].

Les principales perspectives de recherche sont le développement d'algorithmes plus efficaces pour la décomposition structurelle du superviseur et l'étude de méthodes de synthèse compositionnelle.

*Remerciements*

Les auteurs souhaitent remercier Hervé Marchand pour ses conseils utiles concernant les techniques de triangulation d'équations polynômiales.

## Références

[1] P. Ramadge and W. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, Jan. 1989.

[2] H. Marchand, "Méthode de synthèse d'automatismes décrits par des systèmes à événements discrets finis," Ph.D. dissertation, Université* de Rennes I, 1997.

[3] A. Vahidi, M. Fabian, and B. Lennartson, "Efficient supervisory synthesis of large systems," *Control Engineering Practice*, vol. 14, no. 10, pp. 1157–1167, October 2006. [Online]. Available : http ://dx.doi.org/10.1016/j.conengprac.2006.02.013

[4] M. Fabian and A. Hellgren, "Plc-based implementation of supervisory control for discrete event systems," in *Proceedings of the 37th IEEE Conference on Decision and Control*, Tampa, Florida USA, 1998.

[5] M. H. Queiroz and J. E. R. Cury, "Synthesis and implementation of local modular supervisory control for a manufacturing cell," in *6th international Workshop On Discrete Event Systems (WODES)*, 2002, p. 6.

[6] H. Flordal, M. Fabian, K. Akesson, and D. Spensieri, "Automatic model generation and plc-code implementation for interlocking policies in industrial robot cells," *Control Engineering Practice*, vol. 15, no. 11, pp. 1416–1426, Nov. 2007. [Online]. Available : http ://www.sciencedirect.com/science/article/B6V2H-4N5KXR6-1/2/4117ec699af29c9e104dd62d359d7b09

[7] D. B. Silva, E. A. Santos, A. D. Vieira, and M. A. de Paula, "Application of the supervisory control theory to automated systems of multi-product manufacturing," in *12th IEEE international conference on Emerging Technologies and Factory Automation (ETFA)*. Patras, Greece : IEEE, september 25–28 2007, pp. 689–696.

[8] M. Aagaard, R. Jones, and C.-J. Seger, "Formal verification using parametric representations of boolean constraints," in *Design Automation Conference, 1999. Proceedings. 36th*, 21-25 June 1999, pp. 402–407.

[9] E. Tronci, "Automatic synthesis of controllers from formal specifications," in *ICFEM*, 1998, pp. 134–143. [Online]. Available : citeseer.ist.psu.edu/tronci98automatic.html

[10] H. Marchand and M. Le Borgne, "Note sur la triangulation d'une équation polynomiale," IRISA, Tech. Rep., March 2004.

[11] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer, "Interactive presentation : Automatic hardware synthesis from specifications : a case study," in *DATE '07 : Proceedings of the conference on Design, automation and test in Europe*. New York, NY, USA : ACM Press, 2007, pp. 1188–1193. [Online]. Available : http ://portal.acm.org/citation.cfm ?id=1266366.1266622

[12] S. Bulach, A. Brauchle, H.-J. Pfleiderer, and Z. Kucerovsky, "Design and implementation of discrete event control systems : a petri net based hardware approach," *Discrete Event Dynamic Systems : Theory and Applications*, no. 12, pp. 287–309, 2002.

[13] K. L. McMillan, "Symbolic model checking - an approach to the state explosion problem," Ph.D. dissertation, Carnegie Mellon University, 1992.

[14] R. Bryant, "Graph-based algorithms for boolean function manipulation." *IEEE Transactions on Computers*, August 1986.

[15] E. Asarin, O. Maler, and A. Pnueli, "Symbolic controller synthesis for discrete and timed systems," in *Hybrid Systems*, 1994, pp. 1–20.

[16] E. Dumitrescu and M. Ren, "Automatic error correction based on discrete controller synthesis," in *The 4th International Federation of Automatic Control Conference on Management and Control of Production and Logistics, IFAC MCPL'07*, 2007.

[17] E. Clarke and E. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," in *Logic of Programs*, ser. LNCS, vol. 131. Springer-Verlag, 1981.

[18] H. Marchand, P. Bournai, M. L. Borgne, and P. L. Guernic, "Synthesis of discrete-event controllers based on the Signal environment," *Discrete Event Dynamic System : Theory and Applications*, vol. 10, no. 4, pp. 325–346, Oct. 2000.

[19] F. Maraninchi and Y. Rémond, "Mode-automata : a new domain-specific construct for the development of safe critical systems," *Science of Computer Programming*, vol. 46, no. 3, pp. 219–254, 2003.

[20] F. Somenzi, "CUDD : CU decision diagram package release," 1998. [Online]. Available : citeseer.ist.psu.edu/somenzi98cudd.html