A supervisor implementation approach in Discrete Controller Synthesis

Emil Dumitrescu Mingming Ren Laurent Pietrac Eric Niel

Laboratoire AMPERE
INSA-Lyon
F-69621
FRANCE

Abstract

We investigate the implementation of supervisors generated by symbolic BDD-based Discrete Controller Synthesis (DCS). The implementation technique proposed is able to solve both control non-determinism and the structural incompatibility introduced by symbolic DCS. We highlight and illustrate interesting structural properties of the supervisor implementation. Our technique is illustrated on a real-life example modeling a System-on-chip component: a serial to parallel converter.

1 Introduction

The supervisor implementation problem we investigate is specific to the context of Discrete Controller Synthesis (DCS) [11, 19, 23]. More precisely, we only focus on *symbolic* DCS techniques [11, 23] as they are more efficient in handling large state spaces which characterize real-life designs.

The implementation issue raises multiple questions, mostly related to efficiency, autonomy, fault-tolerance, distribution, robustness and compactness requirements of the implemented supervisor. However, prior to adressing any of these aspects, two main problems need to be solved.

On the one hand, the *control non-determinism*. Most synthesized supervisors are control non-deterministic, in the sense that if several control possibilities exist at the same time, all of them are available. A choice must be made either by a human operator or by some heuristic method able to automatically solve this non-determinism. Deterministic supervisors are rarely obtained directly after applying DCS.

On the other hand, the *structural incompatibility* with respect to the modular design patterns. This problem only concerns symbolic DCS. Indeed, symbolic supervisors do not feature input/output signals. They are represented by a Boolean equation encoding all acceptable control solutions. A straightforward supervisor implementation would involve performing on-line resolution of the supervisor at the run-time. This requires a Boolean solver software tool, physically running on a microprocessor (dedicated or not). The target architecture of the implemented supervisor is a software architecture, which has limitations in terms of run-time efficiency.

Contributions
Our objective is to solve the control non-determinism and find a adequate solution to the structural incompatibility problem. Solving the control non-determinism guarantees an autonomous implementation. Solving the structural incompatibility problem amounts to finding a representation of the supervisor as a set of control functions. This has multiple advantages. First and most important, a hardware implementation can be obtained, which is sometimes interesting when execution speed is seeked. Second, the supervisor implementation represented as a set of control functions appears to be much more compact than the initial supervisor.

Besides, we attempt to translate the classical control loop architecture into the hardware design context. We identify a class of hardware design problems for which our control architecture is suitable.

State of the art
Supervisor implementation has been investigated in the past, with a special concern about execution on a PLC [8, 9, 17, 20].

Most research we are aware of have achieved supervisor implementation by developing hints in order to successively refine the control objective and finally yield a deterministic supervisor (i.e. a controller).

The implementation aproach proposed in [16] is based on a latency minimization criterion. Non-determinism is solved such that the progress speed towards a target set is maximized. This is a constraint specific to minimizing protocol latency. However, once this optimality criterion is satisfied, any remaining non-determinism is solved by making arbitrary choices. This work has been generalized in [15, 18].

Other implementation approaches implement the non-deterministic choice by a random choice. The same problem can also be addressed by choosing either fixed or dynamic priority mechanisms over the controllable input set.

Our technique achieves deterministic supervisor implementation by applying a decomposition principle to the supervisor. Similar approaches have been proposed and used in different contexts. In [1], a parametric decomposition technique is applied over Boolean predicates in the context of formal verification of hardware designs. This technique applies a decomposition principle quite close to ours. In [22], a controller implementation technique is presented. Controllers are obtained by optimal synthesis (with quantitative criteria); the technique is also symbolic, but it produces a strict subset of the supervisor. In [13] the authors study the triangulation of a polynomial equation over ternary values, with the same objective of achieving a supervisor implementation. The same objective is also investigated in [3], where the control synthesis technique operates directly from specifications (by contrast to DCS, which needs a plant). A supervisor hardware implementation approach starting from Petri nets is presented in [5].

The outline of this paper is the following: Section II presents basic concept definitions and notations. Section III presents our supervisor implementation method. Section IV illustrates the application of our method on a hardware design. Section V gives information about the implementation framework that has been used.

2 Definitions

2.1 Controllable finite state machines

Let P be a controllable finite state machine defined as follows:

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

such that:

- I is the set of Boolean input variables, such that $I = U \cup C$, and $U \cap C = \emptyset$;
- $U = \{u_0, \cdots, u_{r-1}\}, r > 0$ is the set of uncontrollable input variables. We note $\mathbf{u} = (u_0, \cdots, u_{r-1})$ a tuple of elements of U;
- $C = \{c_0, \cdots, c_{p-1}\}, p > 0$ is the set of controllable input variables. We note $\mathbf{c} = (c_0, \cdots, c_{p-1})$ a tuple of elements of C;
- $S = \{s_0, \cdots, s_{n-1}\}, n > 0$ is the set of state variables. We note $\mathbf{s} = (s_0, \cdots, s_{n-1}), n > 0$ a tuple of elements of S;
- $\delta : \mathbb{B}^{p+r} \times \mathbb{B}^n \to \mathbb{B}^n$ is the transition function of P;
- $s_0 \in \mathbb{B}^n$ is the initial state of P;
- $O = \{o_0, \cdots, o_{m-1}\}, m > 0$ is the set of output variables;
- $\lambda : \mathbb{B}^{p+r} \times \mathbb{B}^n \to \mathbb{B}^m$ is the output function associated to O.

In the sequel, we refer to P as the *plant* i.e. the system to be controlled by DCS. The sets of states of P are handled by their characteristic function. Let $E \subset \mathbb{B}^n$. The characteristic function of E is defined as

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

The usual set operations have corresponding Boolean operators : $''+''$ (logical "or") performs the set union and $''.''$ (logical "and") performs the set intersection. The logical negation $\overline{\mathcal{C}_E}$ expresses the complement of E with respect to $\mathbb{B}^n$.

Let $S' = \{s'_0, \cdots, s'_{n-1}\}$ be a set of *next state variables*. The transition relation $\mathcal{T}$ of P is defined as the set of all legal transitions $\mathbf{s} \xmapsto{\mathbf{u},\mathbf{c}} \mathbf{s}'$ such that $\mathbf{s}' = \delta(\mathbf{s}, \mathbf{u}, \mathbf{c})$:

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

2.2 Symbolic Discrete Controller Synthesis

DCS has been developed in [19], using language theory. In parallel, research in design automation has shown the great interest of symbolic representation of sets of states using BDDs [4], in order to tackle complexity issues due to exponential state space explosion [14]. DCS developments on the top of symbolic techniques have been proposed in [2, 11, 23].

We assume in the sequel that our system is fully observable. The symbolic DCS approach handles sets of states and/or transitions, instead of languages. For a given plant P and a desired specification Q, called a *control objective*, symbolic DCS computes a supervisor $\mathcal{X}$ guaranteeing that P always satisfies Q. The control architecture is illustrated in Figure 1.

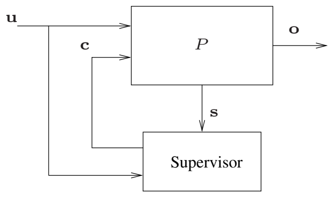

Figure 1. Control Architecture

The computation proceeds following two steps:

- compute the *invariant under control* ($\mathcal{IUC}$) subset of states of P. As long as P remains inside $\mathcal{IUC}$ the violation of Q can be indefinitely postponed. The computation details of $\mathcal{IUC}$ is beyond the scope of this paper. Details are available in [2, 11];

- compute the supervisor $\mathcal{X}$: the set of all transitions $\mathbf{s} \overset{\mathbf{u},\mathbf{c}}{\longmapsto}$ of P leading to $\mathcal{IUC}$. The exact expression of the supervisor is:

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

A control solution exists iff $s_0 \in \mathcal{IUC}$. From a dynamic point of view, $\mathcal{X}$ is said to "play" with the controllable inputs C, against the environment, "playing" with the uncontrollable inputs U of P, with the objective of never reaching a state violating Q. Thus, for any state $\mathbf{s} \in \mathcal{IUC}$ and for any uncontrollable input vector $\mathbf{u} \in \mathbb{B}^r$, the supervisor $\mathcal{X}$ computes adequate values for the controllable inputs $\mathbf{c}$ so that the transition fired by P leads into $\mathcal{IUC}$.

The symbolic supervisor $\mathcal{X}$ has two important properties: (1) it is *maximally permissive* i.e. all transitions of P leading to $\mathcal{IUC}$ are contained in $\mathcal{X}$ and (2) it is *control non-deterministic* i.e. for a given current state and a given uncontrollable input value, there may exist more than one possible successor states in $\mathcal{IUC}$. Our objective is to *implement* $\mathcal{X}$, which amounts to solving the control non-determinism, while attempting to conserve maximal permissivity. This is developed in the next section.

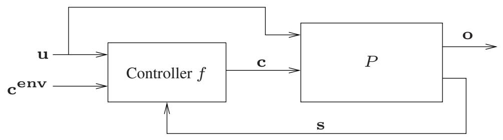

Figure 2. Target control architecture

3 Symbolic Supervisor Implementation

3.1 Requirements for structural compatibility

The supervisor implementation generally amounts to solving the control non-determinism. The deterministic control solution obtained is known as the *controller* and is generally a subset of the synthesized supervisor.

However, besides non-determinism, symbolic DCS also raises an architectural problem: the supervisor $\mathcal{X}$ is represented by a Boolean characteristic function encoding acceptable transitions with respect to the control objective. The Boolean expression of $\mathcal{X}$ is structurally incompatible with P and thus with the control architecture represented in Figure 1. This structural incompatibility is usually eliminated by solving the Boolean equation:

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

either "on-line" or "off-line".

The implementation technique we present addresses both of these issues, as depicted in Figure 2. The control non-determinism is made explicit by adding supplementary environment variables $\mathbf{c}^{env}$, each $c_i^{env}, i = 0 \ldots p-1$ corresponding to c_i. The structural incompatibility is solved by computing individual expressions for each controllable variable c_i of P. This amounts to solving equation (1) off-line. Starting from the the supervisor $\mathcal{X}$ we construct a controller as a tuple of p control functions:

$$f_i : \mathbb{B}^n \times \mathbb{B}^{p+r} \to \mathbb{B},\ i = 0 \ldots p-1, \text{ such that:}$$

$$c_i = f_i(s_0, \ldots, s_{n-1}, u_0, \ldots, u_{r-1}, c_0^{env}, \ldots, c_{p-1}^{env}) \tag{2}$$

Table 1. Thuth table of the Boole decomposition of $\mathcal{X}$ with respect to c_i

$\mathcal{X}\|_{c_i=1}$	$\mathcal{X}\|_{c_i=0}$	c_i	$\mathcal{X}$
0	0	0	**0**
0	0	1	**0**
0	1	0	**1**
0	1	1	**0**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

The relationship between $\mathcal{X}$ and f is expressed by the following equation:

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

In other terms, we wish that (1) any control solution allowed by $\mathcal{X}$ can be reproduced by f and (2) any control solution computed by f is also accepted by $\mathcal{X}$. Finding f satisfying equation (3) also amounts to making explicit the control non-determinism of $\mathcal{X}$ by introducing the auxiliary environment variables $\mathbf{c^{env}}$ as displayed in Figure 2.

3.2 Symbolic implementation algorithm

A major condition prior to implementing $\mathcal{X}$ is that it must be satisfiable:

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

Our implementation algorithm relies on the Boole decomposition of $\mathcal{X}$, which is applied to the controllable variables $c_i, i = 0 \dots p-1$:

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

The impact of variable c_i on the satisfiability of $\mathcal{X}$ is summarized by the truth Table 1. Our objective is to find an expression yielding the values of c_i such that $\mathcal{X}$ is satisfied. The values of c_i such that $\mathcal{X}$ is satisfied are summarized by Table 2. This table summarizes the values that a controllable variable c_i may take, so that the supervisor equation (1) is satisfied. As pointed out by this table, the simultaneous satisfaction of both cofactors $\mathcal{X}|_{c_i=0}$ and $\mathcal{X}|_{c_i=1}$ has a special meaning: regardless of the value of c_i, all transitions allowed by $\mathcal{X}$ lead to $\mathcal{IUC}$. This characterizes the control non-determinism with respect to the controllable variable c_i. Each time c_i has no impact on the satisfaction of $\mathcal{X}$, its value is driven by c_i^{env}. Structurally, variables c^{env} are auxiliary environment (input) variables as pointed out in Figure 2.

Table 2. Values of c_i when $\mathcal{X}$ is satisfiable

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

According to Table 2, the following Boolean expression computes the value of f_i, associated to the controllable variable c_i:

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

The expression (5) represents the basic step of the controller implementation algorithm presented in Algorithm 1: f_0 is computed by assuming that $\mathcal{X}$ holds. f_0 is substituted for c_0 inside $\mathcal{X}$, yielding $\mathcal{X}_1$. Then the algorithm computes f_1 assuming that $\mathcal{X}_1$ holds, and so on.

Algorithm 1 Controller implementation algorithm

```
Require: $\mathcal{X}$, a satisfiable supervisor
1: {starts with $\mathcal{X}$ and computes $f_0,\dots,f_{p-1}$}
2: {intermediate results: $\mathcal{X}_0,\dots\mathcal{X}_p$}
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:     $\mathcal{X}_{i+1} \leftarrow$ substitute $f_i$ for $c_i$ in $\mathcal{X}_i$
7: end for
```

Algorithm 1 produces the tuple of functions:

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

as well as a residual expression $\mathcal{X}_p(\mathbf{s},\mathbf{u})$ obtained by successive substitutions of f_i for c_i into $\mathcal{X}$. In order to guarantee that all control solutions generated by f are accepted by $\mathcal{X}$, we must show that $\mathcal{X}_p$ is a tautology: $\forall \mathbf{s} \in \mathcal{IUC}, \forall \mathbf{u} \in \mathbb{B}^r : \mathcal{X}_p$

Theorem 1 *$\mathcal{X}_p$ is a tautology.*

Proof By applying the successive substitutions of f_i into $\mathcal{X}_i$ the following expression is obtained for $\mathcal{X}_p$:

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

Thus, the predicate $\forall \mathbf{s} \in \mathcal{IUC}, \forall \mathbf{u} \in \mathbb{B}^r : \mathcal{X}_p$ is identical to our initial assumption expressed in (4).

This theorem shows that by substituting f for $\mathbf{c}$ in $\mathcal{X}$ we obtain a tautology and hence that all control solutions generated by f are contained in $\mathcal{X}$. Conversely, we need to prove that:

Theorem 2 *All control solutions contained in $\mathcal{X}$ can be reproduced by f.*

Proof The above statement is equivalent to:

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

By applying the existential quantification and by substituting the expression of f given in (5), the right term of the implication is equivalent to $\overline{\mathcal{X} + \prod_{i=0}^{p-1} \overline{c_i}.\overline{\mathcal{X}|_{c_i=0}}.\overline{\mathcal{X}|_{c_i=1}}}$. Thus, the above implication is a tautology.

It is important to note that the controller f we obtain depends on the variable order used when applying the Shannon decomposition. This order establishes an evaluation priority: f_{p-1} is evaluated first, and its value is used to evaluate $f_{p-2} \dots f_0$.

The complexity of our generation algorithm strongly depends on the size $\mathcal{X}$ in terms of BDD nodes. The most expensive operation we perform is the generalized co-factor, which is used to substitute an expression for a BDD variable. This operation is exponential in the number of variables of $\mathcal{X}$.

The controller generation is illustrated in the next section on a hardware component modeling a serial-parallel converter.

4 Example

4.1 The serial-parallel converter

We illustrate the application of our controller generation technique in a hardware design context, with the aim of constructing a correct-by-construction serial-parallel converter.

Figure 3 presents the architecture of the serial-parallel converter. It is composed of two components: a Serial Buffer (SB) and a Word Construction Buffer (WCB) which are composed together by a

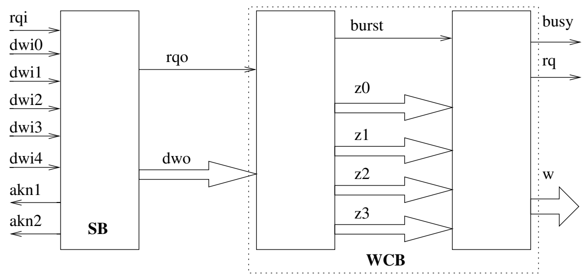

Figure 3. Architecture of the serial-parallel converter

synchronous product. Modeling details can be found in [7].

Component SB receives serial pieces of 4-bits data and performs a parity correction check. Depending on the result, SB sends an acknowledge or an error code via akn to the sender of the data.

Serial data arrive according to a well-established transaction: the sender writes the data over dw and rises the input request *rqi*, meaning that valid input data is available. After one transition, corresponding to one synchronous time moment, SB acknowledges the sender by communicating the result of the parity check. Thus, incoming transactions always last two synchronous time moments. Note that the handshaking mechanism is only used to guarantee correct incoming data with respect to the parity check. Besides, the sender is assumed to maintain its request and data till it is acknowledged.

The SB sends all pieces of correct serial data to the WCB. This component implements two behaviors:

- it accumulates serial data inside an internal buffer, by constructing 8-bit words;
- when the internal buffer is full, it flushes it by sending all parallel pieces of data it has previously constructed. During the flush, WCB is $busy$ and it cannot receive any incoming data. It becomes available again once the flush is over.

SB implements a pipeline behavior, i.e. at a given time t it can simultaneously send correct serial data to the WCB and acknowledge an incoming piece of serial data to the sender.

4.2 Correction analysis

Components SB and WCB are off-the-shelf components, which have been thoroughly verified independently of each other. Both of them are correct with respect to their functional requirements. However, they were not initially designed to work together.

We wish to obtain the serial-parallel converter as a new functionality obtained by combining two existing correct components. However, when SB and WCB are combined together the resulting component does not behave as expected. One important property the converter should feature is that all pieces of serial data must be eventually retransmitted i.e. a piece of serial data is never lost. This property is expressed by the following temporal logic formula written in CTL [6]:

$$\textit{No data loss: } AG\ \overline{busy \wedge rqo}$$

Thus, SB should never send a piece of data to the WCB during a flush. This property holds except for the following scenario: at some time t, WCB has almost filled its internal buffer except for one free slot of serial data. At the same moment, SB sends a piece of serial data to WCB and acknowledges a pieces of input serial data simultaneously. At time $t{+}1$, WCB is full and starts a flush. However, at the same moment SB sends the piece of serial data it has previously acquired. As WCB is busy and flushing, this piece of data is lost. This scenario is confirmed by symbolic model-checking, which proves that property *"No data loss"* is false.

Our goal is to ensure that property *"No data loss"* holds without rewriting neither SB nor WCB. We use symbolic DCS in order to synthesize a correcting patch that ensures the satisfaction of the above property.

4.3 Discrete Controller Synthesis

In order to apply DCS to the converter the designer must indicate which input variables are controllable. There is no a priori indication about the input to be controlled. However, it can be observed that by delaying the arrival of the input serial data adequately, the data loss can always be avoided. Technically, delaying the arrival of an incoming serial data amounts to delaying the arrival of the incoming request rqi. Thus, we choose rqi to be the controllable input variable. The remaining input variables of the plant are considered uncontrollable.

The supervisor has been synthesized by the *Sigali* DCS tool [12]. Then, we have implemented the supervisor by generating a controller according to the Algorithm 1 presented in Section III. The controlled converter architecture is represented in Figure 4 and is is easy to check that it corresponds to the generic target architecture shown in Figure 2.

The implementation algorithm has introduced the auxiliary environment variable rqi-env, corresponding to the controllable variable rqi. Note that the incoming request is now rqi-env, which is driven by the environment and read by the controller. According to the value of rqi-env and the internal state of the plant (i.e. the serial-parallel converter), the controller assigns adequate values to rqi.

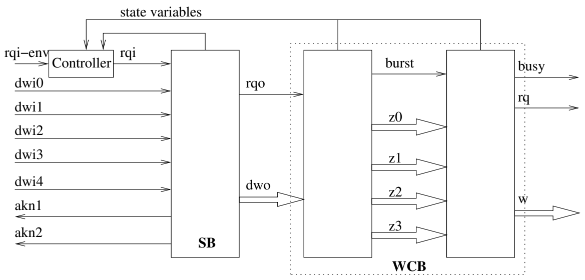

Figure 4. Error correction by DCS

Hence, the controller we obtain acts like a filter on the rqi-env input. When an incoming piece of serial data arrives, the controller either transmits it to the SB, or blocks it if that piece of data is likely to be lost. At any time t, if an incoming piece of serial data arrives, rqi-env is raised. Two scenarios are possible:

- there remains only one free memory slot inside WCB, and this slot is about to be filled by SB with a piece of previously acquired data. At this moment, if rqi-env is acknowledged, the data is lost. However, upon reception of rqi-env the controller sets $rqi = 0$. Thus, incoming requests are not transmitted to the SB;
- there remains more than one free memory slot inside WCB. In that case when receiving rqi-env the controller sets $rqi = 1$. The incoming requests are transmitted to the SB so that they can be acknowledged.

Thus, the input transaction is delayed, in the sense that SB has no knowing about it as long as the data cannot be safely stored and retransmitted. Initially, incoming transactions were only used to ensure parity correction; by applying DCS, these transactions ensure in addition that the serial-parallel converter is ready.

Our controller shifts the control non-determinism towards the environment. When a non-deterministic choice is left on the value of rqi, this non-determinism is solved by the environment through $rqi\text{-}env$. By construction of our controller, the value assigned to $rqi\text{-}env$ by the environment is directly fed to rqi.

According to our example, the input filtering mechanism established by addition of a bug correcting controller heaviliy relies on the assumption of non-determinism. Indeed, if the synthesized controller was completely deterministic then the functional decompostion would produce a real closed-loop system, which is of interest in automatic control, but less interesting with respect to our bug-correction objective.

The controller we obtain is a BDD over the input and state variables of the converter. It contains about 200 nodes, which is why we choose not to display it.

5 Implementation

The serial-parallel converter was modeled using the Mode Automata language and Matou compiler [10]. The supervisor was synthesized by the *Sigali* [12] symbolic DCS tool. Interactive simulations have been performed using the dedicated tool *SigalSimu*. The design flow is illustrated in Figure 5.

Our implementation algorithm has been coded on the top of the CUDD library [21]. It reads the supervisor generated by *Sigali* and produces a controller which is composed of several BDDs each representing a control function. After the implementation step, the plant together with the implemented controller can be fed to standard design tools such as batch simulation, hardware synthesis or formal verification.

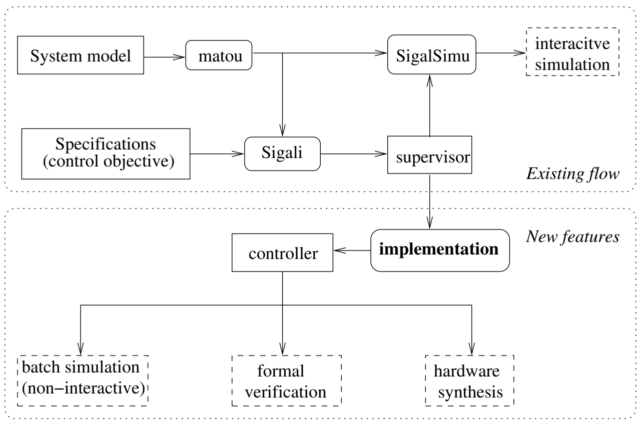

Figure 5. Existing and proposed synthesis framework

6 Conclusion

We presented and illustrated a technique implementing supervisors synthesized by symbolic DCS tools. This technique tackles the control non-determinism issue as well as the structural incompatibility of the supervisor with respect to the modular structure expected for it. A very important advantage is that our controller implementation conserves the integrality of the control solutions produced by DCS. This feature is very interesting with respect to straightforward determinization of a synthesized supervisor, which always produces a strict subsets of $\mathcal{X}$. Besides, it is to be noted that the implemented supervisor f is generally more compact than $\mathcal{X}$, which makes it easier to handle.

Still it may be argued about the introduction of the auxiliary variables c^{env} and about their physical correspondence in general. Actually, the control architecture we implement is a rather particular interpretation of the traditional control loop shown in Figure 1. The main difficulty comes from the translation of this control architecture to the hardware design context. The distinction between controllable/uncontrollable variables is not natural. Input variables are generally dedicated to the environment, and cannot be simply encapsulated inside the control loop. The control architecture we propose leaves the input/output interface unchanged. The controllable inputs are actually filtered by the controller. It should be noted that filtering the inputs of a design is generally not safe with respect to its expected behavior. In section IV we have identified a class of aplications in hardware design where input filtering can be safely achieved.

Our choice for symbolic DCS techniques was mainly motivated by their ability to treat real-life systems: they are less subject to state space explosion than state-enumerative DCS techniques directly based on [19].

It ought however be noted that implementing a supervisor generated with enumerative techniques is algorithmically easier, as they do not exhibit any structural incompatibility with respect to the plant they control. The explicit supervisor implementation merely amounts to solving control non-determinism. However, enumerative DCS is algorithmically expensive when applied on real-life designs.

On the other hand, while symbolic DCS is less subject to state space explosion than enumerative DCS, its complexity still remains exponential in the number of state variables encoding the state of the plant. Moreover, our symbolic implementation algorithm adds its own complexity, which is not negligible. This is why, future research directions include developing compositional synthesis techniques, as well as directly synthesizing a controller instead of implementing a supervisor. Besides, we shall extend our study with figures illustrating the time and memory performance of our technique.

Acknowledgements.
The authors would like to thank Hervé Marchand for useful discussions about polynomial triangulation techniques.

References

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

[2] E. Asarin, O. Maler, and A. Pnueli. Symbolic controller synthesis for discrete and timed systems. In *Hybrid Systems*, pages 1–20, 1994.

[3] 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*, pages 1188–1193, New York, NY, USA, 2007. ACM Press.

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

[5] 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*, (12):287–309, 2002.

[6] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logic of Programs*, volume 131 of *LNCS*. Springer-Verlag, 1981.

[7] 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.

[8] 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.

[9] 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*, 15(11):1416–1426, Nov. 2007.

[10] F. Maraninchi and Y. Rémond. Mode-automata: a new domain-specific construct for the development of safe critical systems. *Science of Computer Programming*, 46(3):219–254, 2003.

[11] H. Marchand. *Méthode de Synthèse d'automatismes décrits par des systèmes à événements discrets finis.* PhD thesis, Université* de Rennes I, 1997.

[12] 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*, 10(4):325–346, Oct. 2000.

[13] H. Marchand and M. Le Borgne. Note sur la triangulation d'une équation polynomiale. Technical report, IRISA, March 2004.

[14] K. L. McMillan. *Symbolic Model Checking - An approach to the state explosion problem.* PhD thesis, Carnegie Mellon University, 1992.

[15] R. Passerone, L. de Alfaro, and T. Henzinger. Convertibility verification and converter synthesis: Two faces of the same coin. In *International Conferenca on Computer Aided Design (ICCAD)*, 2002.

[16] R. Passerone, J. A. Rowson, and A. L. Sangiovanni-Vincentelli. Automatic synthesis of interfaces between incompatible protocols. In *Design Automation Conference*, pages 8–13, 1998.

[17] 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)*, page 6, 2002.

[18] J.-B. Raclet. Residual for component specifications. In *Proceedings of the 4th International Workshop on Formal Aspects of Component Software*, Sophia-Antipolis, France, September 2007.

[19] P. Ramadge and W. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, Jan. 1989.

[20] 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)*, pages 689–696, Patras, Greece, september 25–28 2007. IEEE.

[21] F. Somenzi. CUDD: CU decision diagram package release, 1998.

[22] E. Tronci. Automatic synthesis of controllers from formal specifications. In *ICFEM*, pages 134–143, 1998.

[23] A. Vahidi, M. Fabian, and B. Lennartson. Efficient supervisory synthesis of large systems. *Control Engineering Practice*, 14(10):1157–1167, October 2006.