Preprints of WODES 2012
October 3-5, 2012
Guadalajara, México

Incremental Discrete Controller Synthesis for communicating systems based on modular decomposition

Mingming REN * **Emil DUMITRESCU** * **Laurent PIETRAC** * **Eric NIEL** *

* *Laboratoire AMPERE, INSA-Lyon, Villeurbanne, F-69621 France (e-mail: first.last@insa-lyon.fr)*

Abstract: The symbolic Discrete Controller Synthesis (DCS) is applied incrementally on successive abstractions of the system to be controlled, which is composed of two or more concurrent communicating components. We keep one component while abstract away all others. DCS is applied on the resulting abstract system and produces an intermediate approximate control solution. We refine the abstract model incrementally by adding concrete model of the abstracted components one by one. At each refinement, the previous intermediate solution is used as a starting point synthesizing a more precise solution until the precise supervisor is reached. The efficiency of the incremental technique is illustrated with performance assessments on several models.

Keywords: Discrete controller synthesis, BDDs, incremental synthesis, symbolic DCS.

1. INTRODUCTION

The Discrete Controller Synthesis (DCS) technique [Ramadge and Wonham (1989)] is a promising approach for generating correct by construction behaviors of a discrete event system. DCS has been traditionally used in the automatic control area: the supervisor is meant to operate with elements of a manufacturing system, such as sensors and actuators. More recently, interesting DCS applications have been made in the area of electronic design automation [Bloem et al. (2007)]. Technical characteristics for these new communicating applications handle additional considerations such as Input/Output statements. The Ramadge-Wonham framework has a language-based foundation which fits perfectly well to the event-driven systems, where events from an alphabet occur in a strictly sequential order. On the contrary, in electronic systems, time-driven or sample-driven is more suitable, since sequential events are replaced by concurrent input/output variables. These variables as well as state variables evolve simultaneously at a same pace/clock. Thus it is more natural to use a synchronous modeling framework on such systems. As a result, we use the Binary Decision Diagram (BDD)-based symbolic DCS technique presented in [Marchand (1997)].

Regardless of the application domain, efforts made to apply DCS to larger and larger real-life designs came across the same difficulty: the size of the system leads to time and/or memory blow-up during the computation of the supervisor. The method extensively used to overcome this problem is decomposition. In particular, modular decomposition is a natural and intuitive approach used in discrete event system design, and exploited in several research contributions on DCS.

The incremental DCS technique we present exploits system modularity: given a system composed of two or more communicating modules, a supervisor is constructed incrementally, by (1) keeping one module while abstracting away all others, (2) computing an approximate intermediate solution at a lower cost, and (3) refine the abstract system model by adding one abstracted module, using the previous intermediate solution as a starting point for the computation of a more approaching solution, (4) repeating the refinement until the original system and the exact control solution are reached.

The outline of this paper is the following. Section 2 presents the running example used to illustrate our technique. The definitions and notations required follow in section 3. Section 4 formalizes and illustrates the incremental DCS technique. Section 5 discusses the particularities of our technique compared to other research contributions.

2. EXAMPLE : A 3-WAY ARBITER WITH TOKEN

As an example, we present the model of an arbiter managing exclusive access to a shared resource for three independent clients. The model only focuses on the access management feature.

The access management is modeled by three synchronous concurrent state machines, $M_i, (i = \{1, 2, 3\})$, which share identical function and structure. Each cell M_i receives a request req_i from its client and issues a corresponding access grant ack_i to that client. The access control is enforced by a token mechanism. A cell may acknowledge its client only if it holds the token. Each cell passes the token to its successor via its output $tout_i$. Since $tout_i$ and ack_i share the same value, we will only use ack_i in the following.

Cells M_i are modeled by two automata: M_i^1 receives the token. M_i^2 implements the access grant to the shared resource according to the availability of the token. It also forwards the token, once it has been effectively used. The automata of a single cell are shown in Figure 1. M_i^1 and M_i^2 communicate via the signal go_i, which is set to 1 whenever a token has been received and 0 otherwise. The state N_i means "no token received". State T_i means "a token has been received". State I_i is an idle state, waiting for a client request. State G_i is an active state, where a client has just been granted access and the token is forwarded. In the sequel, for reading commodity, compound states are referred to by concatenating state names of distinct components. For instance, M_i is said to be in the compound state N_iI_i iff M_i^1 is in state N_i and M_i^2 is in state I_i. Thus, N_iI_i is an abbreviation of the cartesian product $\{N_i\} \times \{I_i\}$.

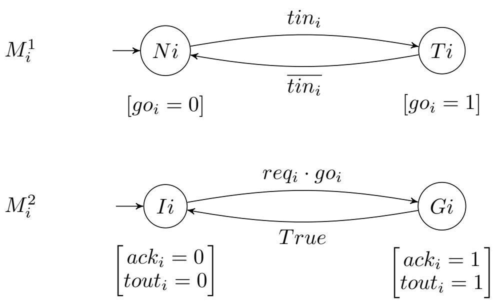

Fig. 1. Model of a single cell M_i

The 3-way arbiter is constructed by instantiating the three cells $M_i, i = 1, 2, 3$ and by connecting their interface and output variables, as shown in Figure 2. Communication is assumed by the pairs of $(tout_i, tin_{i+1}), i = 1, 2$. Tokens are fed to M by the environment through input tin_1.

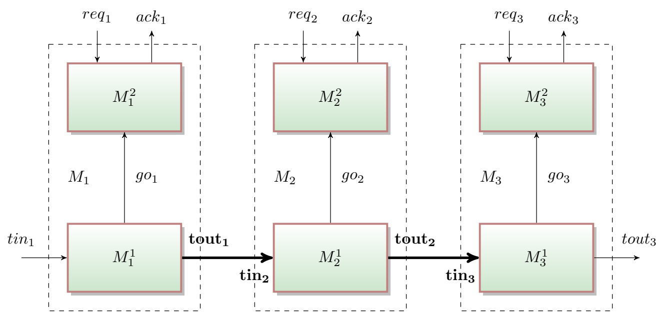

Fig. 2. Communication assumed by inter-connection

The access grant policy we wish to achieve is *mutual exclusion* : machines M_i should never emit their ack_i signal at the same moment. This requirement can be expressed by the following proposition:

$$spec : \overline{ack_1 \wedge ack_2 \vee ack_2 \wedge ack_3 \vee ack_1 \wedge ack_3}$$

The proposition *spec* must be made invariant over the arbiter model. This is formally specified in Computation Tree Logic (CTL) as $enforce: \ AG(spec)$.

3. DEFINITIONS

This section recalls the definition of communicating modules and of their synchronous composition. The principles of the classical DCS technique are also recalled. Each definition is illustrated with the arbiter example.

3.1 Controllable finite state machines (CFSM)

Let $\mathbb{B} = \{0, 1\}$ be the set of Boolean values. Given a set E, we note 2^E, the set of all subsets of E. We define a controllable modular Boolean FSM M as a tuple:

$$M = (Q, I, L, \delta, q_0, PROP, \lambda)$$

such that:

- Q : a finite set of n states $\{q_0, q_1, \ldots, q_{n-1}\}$;
- I : a set of Boolean input variables, such that $I = U \cup C$ and $U \cap C = \emptyset$:
  - · $U = \{u_0, \cdots, u_{m-1}\}, m > 0$: the set of uncontrollable input variables;
  - · $C = \{c_0, \cdots, c_{p-1}\}, p > 0$: the set of controllable input variables;
- $L = \{l_0, \cdots, l_{r-1}\}, r > 0$: a set of interface inputs;
- $\delta : Q \times \mathbb{B}^{m+p+r} \to Q$ is the transition function of M;
- $PROP = \{p_0, \cdots, p_{k-1}\}, k > 0$ is a set of k Boolean propositions;
- $\lambda : Q \to \mathbb{B}^{|PROP|}$ is a labeling function associating a set of atomic properties to each state $q \in Q$. The labeling function models the outputs of M;
- q_0 is the initial state of M.

Note that λ is not bijective. We define $\lambda^{-1} : PROP \to 2^Q$, the association of a Boolean proposition with the subset of states in Q where it holds. The function λ^{-1} has the following properties:

- $\lambda^{-1}(p_1 \wedge p_2) = \lambda^{-1}(p_1) \cap \lambda^{-1}(p_2)$;
- $\lambda^{-1}(p_1 \vee p_2) = \lambda^{-1}(p_1) \cup \lambda^{-1}(p_2)$;
- $\lambda^{-1}(\overline{p}) = Q \setminus \lambda^{-1}(p)$.

In the following, we note $\mathbf{u}, \mathbf{c}, \mathbf{l}$, the vectors containing the variables of U, C, L.

Example: Modules M_i are building blocks for designing a correct 3-way arbiter. They are assembled according to Figure 2. On the resulting model, the token mechanism is only partially defined. A token can be inserted from the environment, via input tin_1 which is set to 1. Once inserted, the token is passed from a cell to the following as soon as it is used. However, in order to satisfy *spec*, there should never be more than one token present inside the arbiter. Hence, the value of tin_1 cannot be chosen at random. It should not be set to 1 until no token is present inside M.

As tin_1 seems crucial for ensuring token unicity inside M we chose it as *controllable*. The incoming requests $req_i, i = 1, 2, 3$ can arrive at any moment, and so they are considered *uncontrollable*.

So the resulting supervisor should control M via its token tin_1 input, according to the uncontrollable inputs $\{req_1, req_2, req_3\}$ and the resulting controlled arbiter should satisfy the CTL [Clarke and Emerson (1981)] property:

$$enforce : \ AG(spec).$$

According to the synchronous composition definition for communicating CFSMs, modules M_i have both environment inputs and local inputs. Environment inputs are dedicated to communication with components situated outside M. Local inputs are used for modeling communication between the modules M_i of M. Here we have $L_1 = \emptyset$, $L_2 = \{tin_2\}$, $L_3 = \{tin_3\}$.

The sets of Boolean propositions associated to $M_i, i = 1, 2, 3$ are $PROP_i = \{ack_i, \overline{ack_i}\}$. The labeling functions λ_i associate these atomic propositions to the states of M_i: $\lambda_i(N_iI_i) = \{\overline{ack_i}\}, \ldots, \lambda_i(T_iG_i) = \{ack_i\}$.

Conversely, the set of states of M_i satisfying proposition ack_i is given by : $\lambda_i^{-1}(ack_i) = \{N_iG_i, T_iG_i\}$.

A supervisor needs to be built by DCS in order to implement the access grant policy *spec* by adequately driving the controllable variables.

3.2 Synchronous composition

We compose CFSMs according to the synchronous paradigm defined in [Milner (1983)]. Let us note $M = ||_{i=1}^{K} M_i$ the synchronous composition between K communicating modules $M_i = (Q_i, I_i, L_i, \delta_i, q_{0i}, PROP_i, \lambda_i), i = 1, \cdots, K$. In the following, we assume that an output cannot be driven by more than one CFSM inside a synchronous product:

$$\forall i, j = 1, \cdots, K, i \neq j : PROP_i \cap PROP_j = \emptyset.$$

We also assume that the sets of controllable and uncontrollable variables are coherent with respect to the synchronous composition: if an input is shared between M_i and M_j, it cannot be controllable in M_i and uncontrollable in M_j.

According to the actual hierarchical structure of M, the set of interface inputs of module M_i can be partitioned into $K-1$ disjoint subsets $\{L_i^j | j = 1, \ldots, K, j \neq i\}$ with

$$L_i^k \cap L_i^j = \emptyset, (\forall k, j = 1, \cdots, K, j \neq k)$$

and

$$\bigcup_{j=1,\cdots,K, j\neq i} L_i^j = L_i$$

and

$$0 \leq |L_i^j| \leq |L_i|.$$

The set of interface inputs L_i^j belongs to M_i and is connected to outputs of M_j.

Let $PROP_i^{out_j} \subseteq PROP_i$ be the subset of output propositions of M_i which are connected to module M_j through L_j^i. Similarly, the output functions λ_i of M_i are partitioned according to their connectivity. Let $(\lambda_i^{j,k}), 0 \leq k \leq |L_j^i|$ be the outputs of M_i connected to M_j.

The synchronous composition of CFSMs $M = ||_{i=1}^{K} M_i$ is defined as follows:

- $Q = Q_1 \times \cdots \times Q_K$;
- $I = I_1 \cup \cdots \cup I_K$, such that $I_i = U_i \cup C_i$;
- $L = \emptyset$;
- $\delta : Q \times \mathbb{B}^{m_1+p_1+r_1} \times \cdots \times \mathbb{B}^{m_K+p_K+r_K} \to Q$ is defined as:

$$\begin{aligned}\delta = \Big(&\delta_1\big(q_1, \mathbf{i_1}, \quad \lambda_2^{1,1}(q_2), \cdots, \lambda_2^{1,|L_1^2|}(q_2),\\ &\qquad \lambda_3^{1,1}(q_3), \cdots, \lambda_3^{1,|L_1^3|}(q_3), \cdots\big),\\ &\cdots,\\ &\delta_K\big(q_K, \mathbf{i_K}, \quad \lambda_1^{K,1}(q_1), \cdots, \lambda_1^{K,|L_K^1|}(q_1),\\ &\qquad \lambda_2^{K,1}(q_2), \cdots, \lambda_2^{K,|L_K^2|}(q_2), \cdots\big)\\ \Big)&;\end{aligned}$$

- $q_0 = (q_{01}, \cdots, q_{0K})$;
- $PROP = PROP_1 \cup \cdots \cup PROP_K$;
- $\lambda : Q \to \mathbb{B}^{\sum_{i=1}^{K} |PROP_i|}$ is defined as:

$$\lambda\big((q_1, \cdots, q_K)\big) = \big(\lambda_1(q_1), \cdots, \lambda_K(q_K)\big).$$

The inverse function $\lambda^{-1}(p) : PROP \to 2^Q$ is defined as:

$$\lambda^{-1}(p) = \bigcup_{i=1,\cdots,K} Q_1 \times \cdots \times \lambda_i^{-1}(p) \times Q_{i+1} \times \cdots \times Q_K.$$

Example: Consider the arbiter model; we illustrate the association of Boolean propositions to global states through the synchronous composition. The synchronous composition labels each global state (q_1, q_2, q_3) with all the atomic propositions associated to q_1, q_2 or q_3.

To illustrate the reverse labeling function λ^{-1}, let us apply it to the Boolean proposition *spec* we wish to enforce gives the following result:

$$\begin{aligned}\lambda^{-1}(\overline{ack_1 \wedge ack_2 \vee ack_2 \wedge ack_3 \vee ack_1 \wedge ack_3}) &=\\ \lambda^{-1}(\overline{ack_1 \wedge ack_2} \wedge \overline{ack_2 \wedge ack_3} \wedge \overline{ack_1 \wedge ack_3}) &=\\ \lambda^{-1}(\overline{ack_1 \wedge ack_2}) \cap \lambda^{-1}(\overline{ack_2 \wedge ack_3}) \cap \lambda^{-1}(\overline{ack_1 \wedge ack_3})&.\end{aligned}$$

By successively applying the properties cited in section 3.1 and the definition of λ we obtain:

$$\begin{aligned}\lambda^{-1}(\overline{ack_1 \wedge ack_2}) &=\\ Q \setminus \lambda^{-1}(ack_1 \wedge ack_2) &=\\ Q \setminus (Q_1^1 \times \{G_1\} \times Q_2 \times Q_3 \cap Q_1 \times Q_2^1 \times \{G_2\} \times Q_3)&,\end{aligned}$$

where Q represents the entire global state set.

A similar application of the same calculus leads to the global result :

$$\begin{aligned}&\lambda^{-1}(\overline{ack_1 \wedge ack_2 \vee ack_2 \wedge ack_3 \vee ack_1 \wedge ack_3}) =\\ &Q \setminus (Q_1^1 \times \{G_1\} \times Q_2 \times Q_3 \cap Q_1 \times Q_2^1 \times \{G_2\} \times Q_3\\ &\cup Q_1 \times Q_2^1 \times \{G_2\} \times Q_3 \cap Q_1 \times Q_2 \times Q_3^1 \times \{G_3\}\\ &\cup Q_1^1 \times \{G_1\} \times Q_2 \times Q_3 \cap Q_1 \times Q_2 \times Q_3^1 \times \{G_3\}).\end{aligned}$$

In the following we recall the basics of the Discrete Controller Synthesis technique on top of which we develop the incremental DCS technique.

3.3 Discrete controller synthesis : the global approach

The DCS technique we use is presented in details in [Marchand (1997)]. The synthesis algorithm starts with an initial set of states $\mathcal{I}^0$ of the system M and attempts to make it invariant. The set $\mathcal{I}^0$ is called a control objective, and corresponds to the set of states satisfying a given Boolean proposition, the specification.

First, DCS computes the set of states $\mathcal{I}^* \subseteq \mathcal{I}^0$ inside which it is always possible to remain by choosing the right values for the controllable inputs, despite any value taken by the uncontrollable inputs. The set $\mathcal{I}^*$ is called an invariant under control ($\mathcal{IUC}$) for M and the specification $\mathcal{I}^0$. The set $\mathcal{I}^*$ is built by successive computations of the controllable predecessors states of $\mathcal{I}^0$, until a fixed point is reached. The controllable predecessors $CPred$ of a given set of states $\mathcal{I}$ is defined as the set of states from which $\mathcal{I}$ can be reached in one transition, whatever the values of the uncontrollable inputs, by picking adequate values for the controllable inputs:

$$\mathrm{CPred}(\mathcal{I}, M) = \{q | \forall u, \exists c : \delta(q, u, c) \in \mathcal{I}\}.$$

The invariant under control function $\mathcal{I}^*(M, \mathcal{I}^0)$ computes the greatest fix-point of the following equation:

$$\mathcal{I}^{k+1} = \mathrm{CPred}(\mathcal{I}^k, M) \cap \mathcal{I}^k, \mathcal{I}^0 = \lambda^{-1}(spec).$$

If the resulting $\mathcal{IUC}$ is not void and if it contains the initial state q_0, a supervisor can be built. the resulting supervisor is defined as the set of all transitions of M leading to $\mathcal{I}^*$: $\mathcal{S} = \{(q, u, c) | \delta(q, u, c) \in \mathcal{I}^*\}$.

3.4 Example : Applying global DCS to the arbiter model

According to the specification stated in section 2, the global state $\lambda^{-1}(\overline{spec})$ should be prohibited by control. The DCS algorithm starts with the set of states satisfying $spec$: $\mathcal{I}^0 = \lambda^{-1}(spec)$, and it further prunes states $T_1I_1T_2I_2 \times Q_3 \cup Q_1 \times T_2I_2T_3I_3 \cup T_1I_1 \times Q_2 \times T_3I_3$.

The invariant under control set $\mathcal{I}^*$ with respect to $spec$ contains all the states of $M_1||M_2||M_3$ such that for any uncontrollable tuple $(req_1, req_2, req_3) \in \mathbb{B}^3$ there always exist $(tin_1) \in \mathbb{B}$ such that the transition obtained reaches $\mathcal{I}^*$. The corresponding supervisor is partially represented below :

$$(T_1G_1T_2I_2N_3I_3), (req_1 \vee \overline{tin}) \vee$$
$$(T_1I_1T_2G_2N_3I_3), (req_1 \vee \overline{tin}) \vee \dots$$

It represents all $M_1||M_2||M_3$ transitions leading to $\mathcal{I}^*$. For instance, in state $(T_1I_1T_2G_2N_3I_3)$, if req_1 are not active, the supervisor enforces $tin = 0$, otherwise, the supervisor does not constrain tin.

4. THE INCREMENTAL DCS TECHNIQUE

As in classical DCS, the incremental (IDCS) algorithm enforces the assertion $AG(spec)$, where $spec$ is a Boolean proposition expressed over the set $PROP_1 \cup \dots \cup PROP_K$ of $M_1|| \cdots ||M_K$ by using the classical Boolean connectors $\wedge, \vee, \overline{p}$. The incremental algorithm is based on an abstraction and an incremental procedure of refinement. Different abstraction techniques have already been applied in a DCS context. They are briefly reminded in section 5. The abstraction criterion we propose aims at hiding from M_1 all behaviors of $M_2, \cdots, M_K$ except those elements of $M_2, \cdots, M_K$ having an influence either on the behavior of M_1 or on the satisfaction of $spec$.

The IDCS acts as follows: First, it builds an abstract model $M_1||M_2^{abs}|| \cdots ||M_K^{abs}$. The abstraction replaces $M_2, \cdots, M_K$ by a most permissive abstract FSM model, which models all the possible behaviors for those outputs initially driven by $M_2, \cdots, M_K$, and having an influence either on the behavior of M_1 or on the satisfaction of $spec$. Such an abstract FSM model for a given variable x is a two-state non-deterministic automaton, having all transitions enabled and asserting $x = 0$ or $x = 1$, according to its current state. Modules $M_2^{abs}, \cdots, M_K^{abs}$ are said to be an *loose*, i.e. nonrestrictive environment for M_1, in the sense that it allows more behaviors of their outputs than $M_2, \cdots, M_K$ actually do.

The number of abstract FSMs obtained depends on the number of output variables shared with M_1 and $spec$. For m shared variables, the loose environment $M_i^{abs}, i = 2, \cdots, K$ models all possible values of these variables, and thus has 2^m states. Thus, the gain achieved through abstraction strongly depends on the connectivity between M_i with M_1 and $spec$, and the concrete model of M_i itself. The abstraction operation is formalized in the next section.

Second, an intermediate, approximate control solution $\mathcal{IUC}_1^{abs}$ is synthesized for $spec$ on the abstract system model $M_1||M_2^{abs} \cdots ||M_K^{abs}$. This is achieved by applying DCS and has a lower computation cost, because it operates on a smaller model, depending on the size of the abstraction previously achieved.

Finally, the abstraction is partially refined: M_2^{abs} is replaced by M_2. The intermediate result $\mathcal{IUC}_1^{abs}$ is used as the starting point for synthesizing a new control solution for $spec$ and $M_1||M_2||M_3^{abs} \cdots ||M_K^{abs}$. Modules $M_3 \cdots M_K$ are added progressively, and successive intermediate results are computed.

The last step operates on the whole system model. It is expected to benefit from the computations achieved at the previous steps on the abstract model. The IDCS technique is illustrated on the right side of Figure 3 for $K = 3$, by comparison to the direct application of DCS which is shown on the left side, and has been presented in section 3.

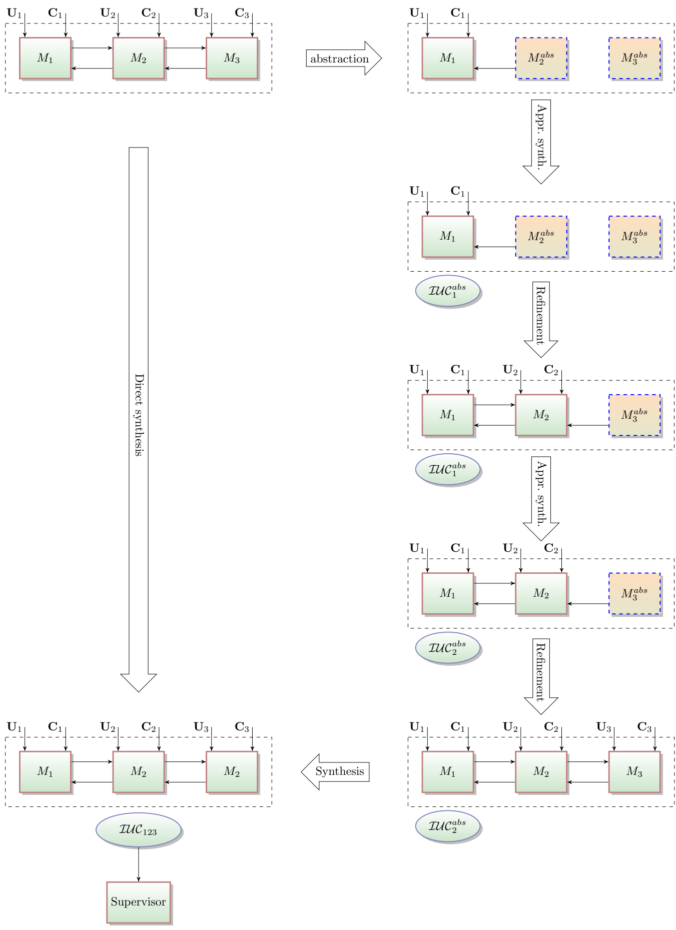

Fig. 3. Incremental synthesis on $M_1||M_2||M_3$

4.1 Abstraction

Let $p \in PROP$ a Boolean proposition of machine M. We define the abstraction of M with respect to p, as the set of configurations satisfying or not p. It is modeled as a non-deterministic FSM :

$$abs(p) = (Q^{abs}, \mathcal{I}^{abs}, \mathcal{L}^{abs}, \delta^{abs}, Q^{abs}, PROP^{abs}, \lambda^{abs})$$

where

- $Q^{abs} = \{q_p, \tilde{q_p}\}$, $\mathcal{I}^{abs} = \emptyset$ and $\mathcal{L}^{abs} = \emptyset$;
- $\delta^{abs} : Q^{abs} \rightarrow 2^{Q^{abs}}$ is defined as: $\delta^{abs}(q_p) = \delta^{abs}(\tilde{q_p}) = Q^{abs}$;
- the initial state can be any among Q^{abs};
- $PROP^{abs} = \{p, \overline{p}\}$;
- $\lambda^{abs} : Q^{abs} \rightarrow PROP^{abs}$ is defined as: $\lambda^{abs}(q_p) = p$ and $\lambda^{abs}(\tilde{q_p}) = \overline{p}$.

Thus, the abstraction of M with respect to p is a non-deterministic FSM where p can be either true or false, and where all transitions are possible. The abstraction of M with respect to a subset $PROP_k \subset PROP$ is defined as the synchronous composition of the individual abstractions defined on $p_i \in PROP_k$:

$$Abs(PROP_k) = ||_{j=1,\cdots,k}\ abs(p_i).$$

4.2 Abstract set refinement

The abstract set refinement operation projects an abstract set of states back on the original states of the global system. The projection is performed through the set of Boolean proposition mappings λ^{-1abs} and λ Let Q^{abs} be a subset of states of an abstract model $M_1||M_2^{abs}$ and Q the set of states of $M_1||M_2$. The refinement of Q^{abs} builds the subset of Q labelled with the same Boolean propositions as the states of Q^{abs}. We define $ref : Q^{abs} \rightarrow 2^Q$ as:

$$ref(q^{abs}) = \{q \in Q | q \in \bigcap \lambda^{-1}(p), p \in \lambda^{abs}(q^{abs})\}.$$

The refinement of an abstract set of states is defined as :

$$Ref(Q^{abs}) = \bigcup \{ref(q^{abs}) | q^{abs} \in Q^{abs}\}.$$

4.3 The Incremental DCS algorithm

The IDCS algorithm starts with the description of M and the specification *spec* to be enforced. It also requires an ordering between the modules of $M = M_1||M_2||\cdots||M_K$, which is to be applied during the successive refinement steps. At each iteration j, an abstraction $\mathcal{IUC}(M, j, spec)$ is computed with respect to:

- the outputs of $M_{j+1}\cdots M_K$ which are connected to local inputs of $M_1\cdots M_j$;
- the set of atomic propositions of $M_{j+1}\cdots M_K$ used to express *spec*.

The performance and advantages of the IDCS technique are commented in detail in section 4.5. In the following, we establish that the IDCS algorithm also produces a maximally permissive supervisor.

Theorem 1

Algorithms DCS and IDCS produce the same result.

Proof The above statement is true iff the incremental and direct synthesis algorithms produce the same invariant under control sets.

Algorithm 1 IDCS algorithm

{inputs:
- $M = M_1||M_2||\cdots||M_K, (K \geq 2)$, the system to be controlled;
- *spec* the specification to enforce;

output: $\mathcal{IUC}^{inc}$, the invariant under control set for M and *spec* }
$M^{abs} \leftarrow \mathcal{ABS}(M, 2, spec)$
$\mathcal{I} \leftarrow \lambda^{-1abs}(spec)$
$\mathcal{I} \leftarrow \mathcal{IUC}(M^{abs}, \mathcal{I})$
$j \leftarrow 3$
while $\mathcal{I} \neq \emptyset$ **and** $j \leq K$ **do**
  $M^{abs} \leftarrow \mathcal{ABS}(M, j, spec)$
  $\mathcal{I} \leftarrow Ref(\mathcal{I}, \{spec\}, Q^{abs})$
  $\mathcal{I} \leftarrow \mathcal{IUC}(M^{abs}, \mathcal{I})$
  $j \leftarrow j + 1$
end while
if $\mathcal{I} \neq \emptyset$ **then**
  {The last invariant under control is projected back on the states Q of M:}
  $\mathcal{I} \leftarrow Ref(\mathcal{I}, \{spec\}, Q)$
  $\mathcal{IUC}^{inc} \leftarrow \mathcal{IUC}(M, \mathcal{I})$
end if

Let us recall that for any DCS problem, we have $\mathcal{IUC} \subseteq \lambda^{-1}(spec)$.

Let $\mathcal{IUC}$ be the result produced by direct DCS and $\mathcal{IUC}^{inc}$ be the result produced by the IDCS algorithm. It can be observed that $\mathcal{IUC}_{inc} \subseteq \mathcal{IUC}$. Indeed, the last step of IDCS operates on the whole system M; it attempts to make invariant the set $\mathcal{I}$, refined at iteration $j = K$ with respect to *spec*. By construction, all these states are included in Q^M and satisfy *spec*. Thus, $\mathcal{I} \subseteq \lambda^{-1}(spec)$. The last DCS application computes $\mathcal{IUC}^{inc} \subseteq \mathcal{I}$ by making invariant the set $\mathcal{I}$.

So, on the one hand, direct DCS makes invariant the set $\lambda^{-1}(spec)$ on M and produces the set $\mathcal{IUC} \subseteq \lambda^{-1}(spec)$. On the other hand, the last step of IDCS starts with a subset of $\lambda^{-1}(spec)$: it makes invariant the set $\mathcal{I} \subseteq \lambda^{-1}(spec)$ on M and produces $\mathcal{IUC}^{inc}$.

Now, let us consider the state $q \in \mathcal{IUC}^{inc}$; q is contained in $\lambda^{-1}(spec)$. Assume that q is not an element of $\mathcal{IUC}$. This means that direct DCS has pruned the state q on M. The last step of IDCS performs an ordinary direct DCS operation on M, making invariant the set of states $\mathcal{I}$, containing q. Thus, state q should also be pruned by IDCS and should not be included in $\mathcal{IUC}^{inc}$. So it is true that $\forall q \in Q^M : q \in \mathcal{IUC}^{inc} \implies q \in \mathcal{IUC}$. We can conclude that $\mathcal{IUC}^{inc} \subseteq \mathcal{IUC}$.

Let us now assume that the above inclusion is strict. This means that there exists at least one state $q \in Q^M$ such that $q \in \mathcal{IUC}$ and $q \notin \mathcal{IUC}^{inc}$. State q has been pruned by the incremental DCS algorithm: there exists a transition leaving q and leading out of $\mathcal{IUC}^{inc}$ for a given uncontrollable value and for any value taken by the controllable variables. However, if such a transition exists, state q would also be pruned by DCS from $\mathcal{IUC}$. Hence, we conclude that $\mathcal{IUC}^{inc} = \mathcal{IUC}$. □

4.4 Example : applying IDCS to the arbiter model

In the following, $\{*_i\}$ abbreviates the set of all states of a given model M_i. We apply IDCS on the example model: M_1 and M_2 are abstracted by M_1^{abs} and M_2^{abs} respectively. Then they are incrementally refined: M_2 followed by M_1.

Abstraction. Let us abstract M_1 and M_2. The abstraction rule applied to M_1, M_2 concerns its output variables shared with M_3 and *spec*. We have $PROP_1^{out} = SPEC_1 = \{ack_1, \overline{ack_1}\}$, $PROP_2^{out} = SPEC_2 = \{ack_2, \overline{ack_2}\}$. Figure 4 represents the abstract model obtained. For readability reasons, abstract states are directly labeled with their corresponding Boolean proposition.

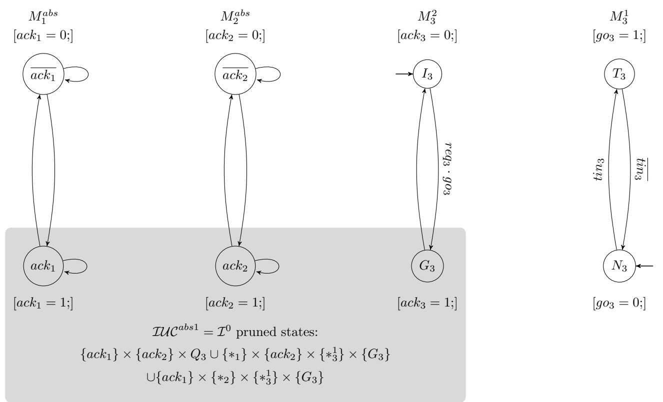

Fig. 4. Abstract model $M_1^{abs}||M_2^{abs}||M_3$ and the approximate computation of $\mathcal{IUC}^{abs1}$

Note that model M_1^{abs} and M_2^{abs} are entirely non-deterministic: at any moment, all transitions are enabled.

Approximate $\mathcal{IUC}$ computation. The approximate invariant under control for property *spec* is built upon the abstract model $M_1^{abs}||M_2^{abs}||M_3$. As shown in figure 4, $\mathcal{IUC}^{abs1}$ prunes the following abstract states : $\{ack_1\} \times \{ack_2\} \times \{*_3\} \cup \{*_1\} \times \{ack_2\} \times \{*_3^1\} \times \{G_3\} \cup \{ack_1\} \times \{*_2\} \times \{*_3^1\} \times \{G_3\}$.

Refinement We then refine the abstract model by replacing M_2^{abs} with M_2, as illustrated in figure 5.

The pruned state set is refined with respect to M_2 and *spec*. Thus the $\mathcal{IUC}^{abs1}$ prunes states of $\{ack_1\} \times \{*_2^1\} \times \{G_2\} \times \{*_3\} \cup \{ack_1\} \times \{*_2\} \times \{*_3^1\} \times \{G_3\} \cup \{*_1^{abs}\} \times \{*_2^1\} \times \{G_2\} \times \{*_3^1\} \times \{G_3\}$.

From the refined $\mathcal{IUC}^{abs1}$, the DCS algorithm then further finds another set of states to be pruned: $\{*_1^{abs}\} \times \{T_2 I_2 T_3 I_3\}$.

Final synthesis. To achieve the last step, we replace M_1^{abs} by M_1 and refine the states pruned from $\mathcal{IUC}^{abs2}$ on the states of $M_1||M_2||M_3$.

The computation of the final supervisor tries to make invariant the set $Ref(\mathcal{IUC}^{abs2})$. This last step finds no more states to be pruned. Thus we have an exact final solution $\mathcal{IUC}$, as shown in figure 6.

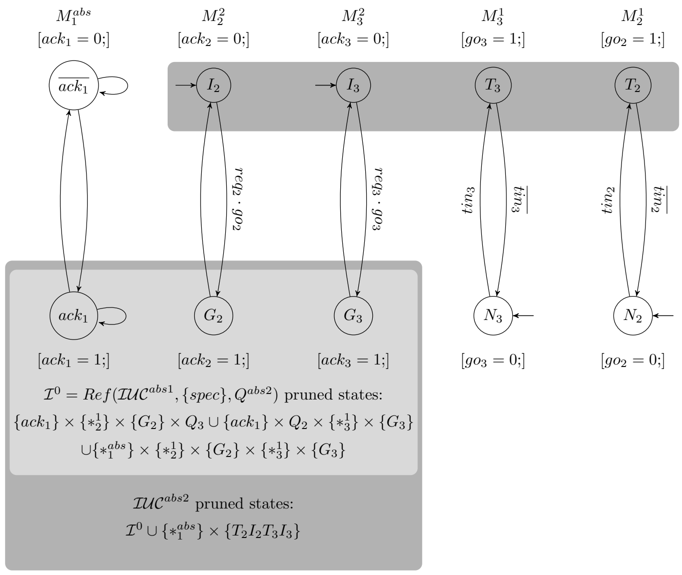

Fig. 5. Abstract model $M_1^{abs}||M_2||M_3$ and the approximate computation of $\mathcal{IUC}^{abs2}$

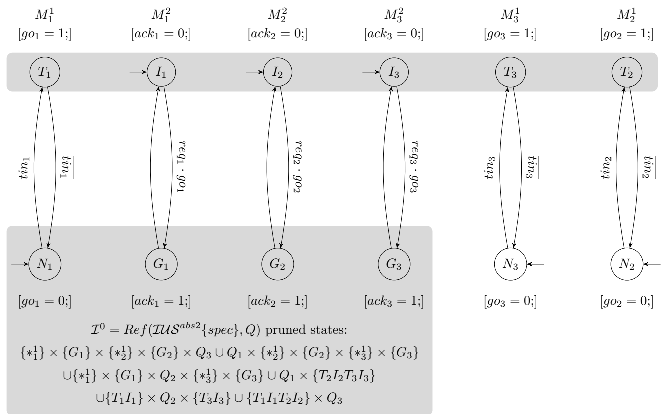

Fig. 6. Final DCS step ensuring *spec* on $M_1||M_2||M_3$

4.5 Implementation and performance issues

The performance of the IDCS algorithm strongly depends on the actual implementation of the underlying DCS technique. The technical possibilities available are *enumerative DCS* and *symbolic BDD-based DCS*.

Enumerative DCS techniques represent explicitly the set of states of the system. The complexity of the DCS is $O(n|\Sigma|)$ where n is the total number of states of the system composed to its specification, and $|\Sigma|$ represents the size of the input alphabet.

Symbolic BDD-based DCS manipulates sets of states, rather than individual states, and uses binary decision diagrams (BDDs) [Bryant (1986)] to represent them. The performance of this technique is promising, but remains bound by the spatial complexity for constructing a BDD, which is exponential in the number of Boolean variables representing the system. Hence, memory is a critical computing resource for symbolic DCS.

Regardless of the underlying DCS technique used, the computation of an abstract $\mathcal{IUC}$ (step 2 of IDCS algorithm) is definitely faster, as it operates on a reduced model. The speedup mainly depends on the structural decomposition achieved and thus on the final size of the abstract model obtained. However, the second application of the global DCS starting with an intermediate solution shall still feature the same complexity as classical DCS, plus the overhead generated by the computation of the abstract solution.

The ability of symbolic DCS to efficiently manipulate state sets instead of individual states is an important advantage. We choose to build the IDCS algorithm on top of the symbolic DCS technique developed in [Marchand (1997)].

We expect an *average* performance improvement of symbolic IDCS over symbolic DCS for the following reasons. First, the computation of the approximate $\mathcal{IUC}$ operates on the reduced model. As abstraction removes most states from M, the impact on the size of the BDDs built for the symbolic traversal of the abstract state space is indubitable. Second, the computation of $\mathcal{I}$ produces an *intermediate, approximate* solution of the DCS problem. It relies on a more compact BDD representation, as it is built over an abstract system containing less variables. Moreover, the set $\mathcal{I}$ is a subset of $\lambda^{-1}(spec)$. Thus, a number of states of the abstract model are pruned at a lower computation cost and need not be reconsidered anymore during subsequent DCS applications. Thus, we expect the final DCS step to converge faster (with less fix-point iterations) towards the final solution. Besides, if a DCS problem does not have a solution, this can be detected on the abstract system at a much lower cost.

It should be noted that, our IDCS technique requires a supplementary user-specified indication: an ordering between the modules which constitute the global system. IDCS works iteratively: for a system containing $K \geq 2$ modules, it requires K steps, one for each module, with $K!$ possible application orders. It is easy to observe that deciding which module order is optimal with respect to the global performance of IDCS faces exponential complexity problems, and thus is not feasible. We argue that an ordering between modules can be user specified, and determined according to the structure and the connectivity of the global system.

The performance figures of symbolic IDCS over symbolic DCS are presented in section 4.6.

4.6 Experimental results

We realized both global and incremental synthesis in a symbolic supervisor synthesis tool SIGALI [Marchand et al. (2000)]. The experimental figures are validated by a systematic formal proof that the two supervisors obtained by DCS and IDCS are the same. The quantitative figures obtained show that IDCS improves both computation time and memory usage. Results are shown in table 1, where memory usage is measured in megabytes, and numbers of BDD nodes are shown in millions except for example MA. All experiments are performed on a computer with Intel Core 2 T7100 and 2Gb memory.

Among these examples, PB stands for Pi-BUS, a bus controller that manages shared resources for several devices; BA is a distributed arbiter with 4 cells synchronized by a token ring; MA is the example illustrated in this paper; TA models the fault-tolerant scheduling of 2 tasks executing on 3 processors; CM models a “cat and mouse” problem with 2 mice and 1 cat in 5 rooms; examples PH1 and PH2 model the 3 philosophers dining problem. They only differ in the fact that PH2 performs a supplementary abstraction/refinement step, as it contains three modules.

Table 1. Experiment results

		PB	BA	MA	TA	CM	PH1	PH2
	mem	3.8	4.2	1.3	6.1	18	114	114
glob.	bdd	0.05	0.05	7728	0.08	0.3	2.4	2.4
	time	5.3m	7s	-	100s	27s	12m	12m
	mem	2.0	1.9	1.1	3.3	11	95	21
incr.	bdd	0.02	0.01	5785	0.04	0.2	1.9	0.4
	time	0.16s	2s	-	37s	3s	11m	42s
	mem	47%	54%	15%	46%	39%	17%	82%
gain	bdd	60%	80%	25%	50%	33%	21%	83%
	time		71%	-	63%	89%	8.3%	94%

The figures obtained show that the IDCS technique achieves very interesting performance improvements over classical DCS. Besides, the figures obtained for the example PH2 strongly suggest that generalization of IDCS to n modules can bring important improvements.

5. RELATED WORKS

Modular supervisory control is first studied in [Wonham and Ramadge (1988)]. Since, a number of methods have been proposed to reduce computation and/or memory efforts. In [Su and Wonham (2004)], redundant information such as transition constraints which are already enforced by the system are reduced from the supervisors. This technique can be applied to modular supervisors synthesis to improve efficiency, but it needs to build modular supervisors first. In [Schmidt et al. (2006)] authors apply abstraction techniques for decentralized control. Supervisors are computed for each reduced subsystem system alphabet, and the abstract behavior is reduced to this alphabet. Language projections are used in [Feng and Wonham (2006), Feng (2007)] to simplify and to construct modular supervisors. An abstraction based on automata rather than on language projections was proposed in [Su et al. (2008)] in order to preserve nonblocking properties. In [Flordal and Malik (2006), Flordal et al. (2007)], the authors present a framework for compositional synthesis, using abstractions based on a process equivalence called supervision equivalence. Using non-deterministic automata, the method supports a wide range of simplifications and can hide both controllable and uncontrollable events, while still ensuring a least restrictive result. In [Malik and Flordal (2008)] an equivalence of non-deterministic abstract processes, called synthesis equivalence, is proposed. In [Hill et al. (2008)] modular supervisors are built; potential conflicts between modular supervisors are solved by a set of coordinating filters. A modular technique based on concurrent automata decomposition is presented in [Gaudin and Marchand (2004), Gaudin (2004)]. The global supervisor is obtained by treating each automaton of a modular composition separately. The automata are supposed to share input events, but they do not communicate, i.e. no outputs of one automaton are connected to the inputs of another automaton. In [Hill (2006)] an incremental synthesis approach with abstraction is proposed. The abstraction is applied to the modular sub-controlled components by projecting away strictly private events. Supervisory synthesis is achieved in an incremental down-up manner until all specifications are satisfied.

The techniques enumerated above mostly exploit qualitative properties of the system, the specification or the supervisor: modular composition, locality of input events, locality of the specifications, behavioral equivalence, modularity of the supervisor, etc. The IDCS algorithm does not perform any qualitative analysis on the system. It abstracts away the environment of a module M, communicating with other modules inside the same system. The approximate solution is built upon the exact definition of M, and an *optimistic* assumption of its *internal* environment. This assumption is refined when the abstract environment of M is replaced by the original one, and the final DCS step is performed. The key advantage of IDCS is its ability to exploit modularity *with communication* between the different modules. Besides, IDCS can benefit from most techniques enumerated above, in order to make a finer usage of the qualitative properties of the system.

6. CONCLUSION

An Incremental Discrete Controller Synthesis (IDCS) algorithm was presented. It alternates automatic abstraction, based on the modular structure of the system, and classical DCS application steps, to build an exact supervisor. The IDCS technique improves the performance of the classical BDD-based DCS, for systems featuring concurrent communicating modules. The time/memory efficiency of IDCS is illustrated with quantitative figures which show interesting enhancements for both memory usage and execution time. However, the order in which IDCS should consider each module of a system must be user-specified. This choice is important for the performance of IDCS. Finding a good order for abstraction and refinement is left as a future research direction for this work. In fact, in the arbiter example, there exists a dependency chain among the components. M_2 influences M_3 directly, M_1 influences M_2 directly and M_3 indirectly. This dependency structure could be an indication for finding a good order.

REFERENCES

Bloem, R., Galler, S., Jobstmann, B., Piterman, N., Pnueli, A., and Weiglhofer, M. (2007). Automatic hardware synthesis from specifications: A case study. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 1188–1193.

Bryant, R. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*.

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

Feng, L. (2007). *Computationally Efficient Supervisor Design For Discrete-Event Systems*. Ph.D. thesis, University of Toronto.

Feng, L. and Wonham, M. (2006). Computationally efficient supervisor design: Abstraction and modularity. In *Proceedings of the 8th International Workshop on Discrete Event Systems, WODES\\,'06*, 8, 3.

Flordal, H. and Malik, R. (2006). Supervision equivalence [supervisor synthesis]. In *Discrete Event Systems, 2006 8th International Workshop on*, 155 –160.

Flordal, H., Malik, R., Fabian, M., and Akesson, K. (2007). Compositional synthesis of maximally permissive supervisors using supervision equivalence. *Discrete Event Dynamic Systems*, 17(4), 475–504.

Gaudin, B. and Marchand, H. (2004). Modular supervisory control of a class of concurrent discrete event systems. In *Workshop on Discrete Event Systems, WODES'04*, 181–186.

Gaudin, B. (2004). *Synthèse de contrôleurs sur des systèmes à événements discrets structurés*. Ph.D. thesis, Université de Rennes I.

Hill, R.C. Tilbury, D. (2006). Modular supervisory control of discrete-event systems with abstraction and incremental hierarchical construction. In *Discrete Event Systems, 2006 8th International Workshop on*, 399–406. Ann Arbor, MI.

Hill, R., Tilbury, D., and Lafortune, S. (2008). Modular supervisory control with equivalence-based conflict resolution. In *American Control Conference, 2008*, 491 –498.

Malik, R. and Flordal, H. (2008). Yet another approach to compositional synthesis of discrete event systems. In *Discrete Event Systems, 2008 9th International Workshop on*, 16–21. Goteborg, Sweden.

Marchand, H., Bournai, P., Borgne, M.L., and Guernic, P.L. (2000). Synthesis of discrete-event controllers based on the signal environment. *Discrete Event Dynamic System: Theory and Applications*, 10(4), 325–346.

Marchand, H. (1997). *Méthode de Synthèse d'automatismes décrits par des systèmes à événements discrets finis*. Ph.D. thesis, Université* de Rennes I.

Milner, R. (1983). Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25(3), 267–310.

Ramadge, P. and Wonham, W. (1989). The control of discrete event systems. *Proceedings of the IEEE*, 77(1), 81–98. doi:10.1109/5.21072.

Schmidt, K., Marchand, H., and Gaudin, B. (2006). Modular and Decentralized Supervisory Control of Concurrent Discrete Event Systems Using Reduced System Models. In *Workshop on Discrete Event Systems, WODES'06*, 149–154. IEEE Computer society, Ann-Arbor United States. doi: 10.1109/WODES.2006.1678423.

Su, R. and Wonham, W.M. (2004). Supervisor reduction for discrete-event systems. *Discrete Event Dynamic Systems*, 14(1), 31–53.

Su, R., van Schuppen, J.H., and Rooda, J.E. (2008). Supervisor synthesis based on abstractions of nondeterministic automata. In *Discrete Event Systems, 2008 9th International Workshop on*, 412–418. Goteborg, Sweden.

Wonham, W. and Ramadge, P. (1988). Modular supervisory control of discrete-event systems. *Mathematics of Control, Signals, and Systems (MCSS)*, 1(1), 13–30.