Generative Programming for Programmable Logic Controllers

Daniel Côté, Richard St-Denis, and Sylvain Kerjean
Département d'informatique
Université de Sherbrooke
Sherbrooke, Québec, CANADA J0B 2C0
{Daniel.R.Cote, Richard.St-Denis, Sylvain.Kerjean}@USherbrooke.ca

Abstract

Many attempts have been made to implement supervisors derived by synthesis procedures peculiar to the Supervisory Control Theory (SCT), most adopting the event-based supervisory control paradigm. However, when considering code generation schemata for programmable logic controllers (PLCs), hardware resources are limited and event tracking is hard to realize satisfactorily. Moreover, previous work has highlighted differences between the abstract model adopted by SCT and realistic process control situations. Inappropriate solutions to these issues may result in code generation schemata that produce unreliable PLC code. A generative programming approach for PLCs based on a dual paradigm, the state-based supervisory control paradigm, is investigated in this paper. Such an approach exhibits interesting properties. For instance, the maximum depth of the PLC stack as well as PLC cycle timing evaluations become possible. Furthermore, well-known code optimization techniques can be used to obtain more efficient code.

1. Introduction

Recently, the need for formal approaches in PLC programming has been fully justified, mainly because of the complexity of control programs and industrial manufacturing systems combined with the wide use of PLCs in industry [10, 16]. These formal approaches include mathematically-based specification and reasoning techniques to verify or synthesize sophisticated control policies. Verification refers to a variety of techniques used to prove the satisfaction of a given property by a program to be checked. Synthesis refers to a category of procedures used to derive a program satisfying a given property. Both methods have been applied to the realization of control programs for PLCs (e.g., see [5] for verification and [12] for synthesis). Despite the efforts done in this direction, the ideal solution identified by several organizations is automatic code generation by specialized tools, which offer formal representations of systems and control requirements. The dream of software engineers is to revolutionize software development in a similar way as automation and components revolutionized manufacturing.

To make this engineer's dream come true, research efforts must be focused on three related issues: generative programming, component engineering, and domain-specific languages. Generative programming consists in conceiving programs that synthesize others programs. Component engineering is concerned with the development and maintenance of software intensive systems by means of component customization, assembly, and deployment. Domain specific languages make it possible to enhance program specifications by giving more importance to compact domain-specific notations that are easier to write and maintain. Global solutions to these three issues will contribute to automating software development, particularly generation of programs controlling complex manufacturing processes in which failures are costly not only in terms of productivity, but also in terms of safety.

The *Supervisory Control Theory* (SCT), which provides a theoretical framework for solving various control problems specific to processes seen as discrete event systems (DESs) [15], includes various elements to tackle the aforementioned issues. First, SCT advocates a synthesis approach which can be considered as a particular case of generative programming. It is powerful enough to automatically generate correct supervisors. Second, SCT embraces various forms of supervision, in particular modular supervision which makes it possible to construct reusable local supervisors and combine them provided that non-conflicting properties are satisfied [14, 18, 8]. Third, SCT introduces fundamental concepts in a manner that is independent of any particular modeling formalism. Specifications can be written in a domain specific language and translated into a representation that is more appropriate for synthesis algorithms (e.g., linear temporal logic formula templates with an associated natural language interpretation [2] and supervisor synthesis from temporal logic specifications [3]).

Keeping in mind that a DES is characterized by a discrete state space and an event-driven dynamics, there are two dual approaches to modeling DES control problems in SCT. One in which the sequences of events generated by the process to be controlled play a central role in the de-

0-7803-9402-X/05/$20.00 © 2005 IEEE

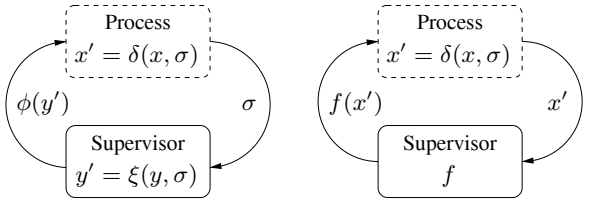
cisions taken by a supervisor based on its internal states (which represent sequences of events generated by the process). Another in which the states of the process to be controlled are used by a supervisor to restrain its behavior. In both approaches a supervisor prevents event occurrences so as to satisfy given safety and progress specifications while being maximally permissive and nonblocking. On one hand, the former approach is less appropriate for control, because a controller must generally force some events to occur in the process which is not the case of supervisors [1]. Even though some extensions of SCT, in which supervisors force events, have been proposed, the PLC code obtained from the underlying code generation procedures is hazardous, because of unrealistic assumptions or inconsistencies between the conceptual model and implementation model. Surprisingly, to the best of our knowledge, this approach is the only one that has been applied to generate PLC code from synthesized supervisors. On the other hand, the latter approach is very close to the more conventional viewpoint adopted in industry, but it has not been yet investigated in connection with PLC code generation. The generative programming method described in this paper adopts the state-based supervisory control approach to generate correct PLC code.

The rest of this paper is organized as follows. Section 2 recalls the main concepts of the dual approaches to modeling DES control problems. Section 3 presents existing PLC code generation schemata with their limitations. It also introduces a new code generation schema based on the state-based supervisory control approach. Section 4 gives an overview of the generative programming method and Section 5 illustrates the proposed method from an application realized on a modular production system by Festo. Finally, Section 6 ends the paper with some concluding remarks.

2. Dual Approaches to Supervisory Control

In the context of automaton-based models, a DES is represented by a deterministic finite automaton (DFA), also called a *generator*, $G := (X, \Sigma, \delta, x_0, X_m)$, where X is the finite set of states; Σ is the finite set of events divided into two disjoint subsets Σ_c and Σ_u of controllable and uncontrollable events, respectively; $\delta : X \times \Sigma \to X$ is the partial transition function; $x_0 \in X$ is the initial state; and X_m is the subset of marked states, which represents the completed tasks.

Given a generator G and control requirements, a supervisor S must be constructed such that the behavior of G under the control of S is restrained to the greatest possible number of admissible event sequences with respect to the control requirements. Thus, the process represented by G and the supervisor S are usually brought together in a *closed loop system* denoted by S/G. Several representations of a supervisor are possible. Figure 1 illustrates two of them.



Event−based supervisory control State−based supervisory control

Figure 1. Closed loop systems

2.1. Event-Based Supervisory Control

A convenient representation of a supervisor is a pair (C, ϕ), where $C := (Y, \Sigma, \xi, y_0, Y_m)$ is a DFA and $\phi : Y \to 2^{\Sigma_c}$ is a *feedback function* that maps each state of C to a subset of controllable events [15]. If $\sigma \in \phi(y)$, then σ is *disabled*; otherwise, it is *enabled*. An uncontrollable event $\sigma \in \Sigma_u$ cannot be disabled by a supervisor. The role of C is to mimic the behavior of the DES under constraints, while the role of ϕ is to restrict its behavior. The supervisor and DES evolve concurrently so that they interact with each other. Their interactions may be regarded as events that require their simultaneous participation, events being generated by G with respect to control inputs determined by ϕ. Let x and y be the current states of G and C, respectively. On the occurrence of event σ, $\sigma \notin \phi(y)$, the next states of G and C are $\delta(x, \sigma)$ and $\xi(y, \sigma)$, respectively. Furthermore, the controllable events that belong to $\phi(\xi(y, \sigma))$ are disabled.

2.2. State-Based Supervisory Control

An alternative representation of a supervisor is a *state feedback control* (SFBC) policy for G [17]. A SFBC is a total function $f : X \to 2^{\Sigma_c}$ that associates a set of disabled events to each state $x \in X$. For each $\sigma \in \Sigma$, the predicate $f_\sigma : X \to \{0, 1\}$ is defined by $f_\sigma(x) = 1 \Leftrightarrow \sigma \notin f(x) \vee \sigma \in \Sigma_u$. Thus a SFBC f may be described by a family of predicates $\{f_\sigma \mid \sigma \in \Sigma\}$ on X. A SFBC f observes the state of the DES and prohibits controllable transitions with the aim of avoiding bad states. Let x be the current state of G. On the occurrence of event σ, $\sigma \notin f(x)$, the next state of G is $\delta(x, \sigma)$ and the controllable events that belong to $f(\delta(x, \sigma))$ are disabled. Formally, the closed loop formed from G and f is defined as follows: $G^f := (X, \Sigma, \delta^f, x_0, X_m)$, where $\delta^f(x, \sigma) := \delta(x, \sigma)$ if $\delta(x, \sigma)$ is defined and $\sigma \in f_\sigma(x)$, and is undefined otherwise.

In both representations, the language defined by admissible event sequences is called the *supremal controllable sub-language* and the synthesized supervisor is termed *maximally permissive*. Under some assumptions such a supervisor always exists and it may be *nonblocking* in the sense that all subtasks in S/G can be completed.

3. Generative Programming with SCT

In order to lay out a code generation schema that produces PLC code from supervisors, two concepts from the theory must be mapped onto corresponding common concepts of PLC-based architectures: *event* generated by a process and *state* of a supervisor. Most of the literature on the subject adopts simple conventions.

Events that occur spontaneously in the SCT model correspond to time-consuming *commands* and *responses* according to the following mapping:

- significant changes in PLC outputs are classified as controllable events, in particular for discrete outputs, rising and falling edges of actuators result from commands;
- significant changes in PLC inputs are classified as uncontrollable events, in particular for discrete inputs, responses to commands coincide with rising and falling edges of sensors.

In the context of event-based supervisory control, establishing such a correspondence between the notion of state of an abstract artifact, such as a supervisor, and physical features of a PLC may be difficult. Such a correspondence must be addressed because a supervisor gives proper feedback based on its current state. One way to avoid this difficulty altogether is to directly implement the state transition structure and feedback function of a supervisor in PLC code. This solution does not require any further mapping between an abstract state space and hardware state space. However, for a PLC-based implementation, this strategy is fraught with difficulties.

Henceforth the DFA of a supervisor viewed as a passive language acceptor is no longer appropriate because controllable events correspond to commands. A hybrid machine with a distinctive semantics, accepting uncontrollable events and generating controllable events, must be considered in the implementation of a supervisor. This means that whenever there are more than one transition on controllable events originating from the same state of the supervisor, an ambiguity exists as to which of these transitions must be selected. This problem is not specific to PLC implementations and most of the literature on the subject acknowledges it. A reasonable solution is a procedure, called *controller extraction* [6], that must be performed on the synthesized supervisor in order to obtain an unambiguous hybrid machine.

3.1 Simple Implementation

Figure 2 gives the schematic Ladder diagrams corresponding to a translation schema for a simple implementation of a control achieved by a *synchronous composition based supervisor* (the feedback function ϕ is implicit and the closed loop system is given by $C \| G$) [11]. This schema has been experimented in some applications (e.g., [4]). The Ladder logic of the program is partitioned into blocks of Ladder instructions. Each block (corresponding to a state of the supervisor being implemented) is under the control of a special relay called a *stage relay*. Whenever a stage relay is closed (active), the instructions below this stage are executed during the current PLC cycle, otherwise this logic is not executed. A special instruction is provided (JMP $target$) to deactivate the stage relay corresponding to the currently executing logic, and activate the target stage relay. This instruction does not, however, behave as a *goto* statement; only the value of the stage relays are affected. A block continues its normal execution even after a JMP instruction has been issued. Within a block there are two kinds of statements corresponding to transitions on uncontrollable and controllable events. Each transition on an uncontrollable event is guarded by an expression matching the event (rising or falling edge of a sensor) and has no action associated with it. Each transition on a controllable event is systematic (the guard condition is always true), and the associated action is meant to issue the command that corresponds to the event.

A similar code generation schema is proposed in [13], but some modifications must be done on the synthesized supervisor before generating the PLC code. Close examination of these strategies reveals serious flaws, particularly those related to uncontrollable events.

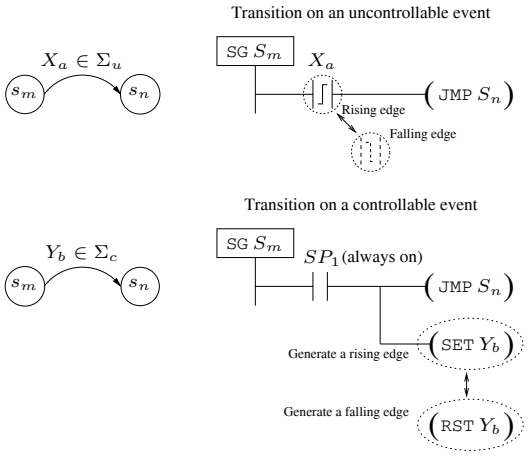


Figure 2. Simple code generation schema

3.1.1 Simultaneous Occurrence of Uncontrollable Events

The cyclical scan activity of a PLC induces a delay between I/O samplings. In this period, more than one sensor can change state. These changes are seen as the simultaneous occurrence of two or more uncontrollable events by the PLC.

3.1.2 Loss of Uncontrollable Events

Generally, an event that is visible in one PLC cycle will no longer be visible in the next. Suppose two uncontrollable events e_1 and e_2 are sensed at the onset of the current PLC cycle, and that e_1 is causing a transition from the current state to a state called s_2. Furthermore, suppose that the code for state s_2 (which has a transition on event e_2) is located upstream from the code for the current state. Then, the code for state s_2 will only be executed on the next PLC cycle, but in the next PLC cycle, event e_2 will no longer be visible (unless explicitly memorized by the PLC logic) so the transition from state s_2 on event e_2 will never occur and event e_2 will then be lost. That kind of situation will not systematically occur, but it cannot be ruled out.

3.2 Hierarchical Implementation

To mitigate the problems mentioned in the previous section and the state space explosion problem inherent in SCT [15], a three level hierarchical implementation has been proposed [9]. It is based on a strategy in which events are distributed one at a time to the supervisory logic expressed as a set of modular supervisors of smaller combined size than a centralized supervisor. As illustrated in Fig. 3, the high level implements the modular supervisors as DFAs running in parallel. Based on the sole occurrence of an event, the supervisors execute at most one transition and enable or disable controllable events according to their current state by resetting or setting binary flags. The intermediate level simulates the parallel execution of DFAs modeling the free behavior of the different components of the process. They trigger controllable events according to their current state and the state of the inhibition flags. They also generate uncontrollable events according to responses received from the low level, which transforms input signals into responses and commands into outputs signals. Unlike the first two levels, the low level is programmed in an ad hoc way.

Therefore, the intermediate level is responsible for distributing events one by one to the supervisory logic. At every occurrence of an event, the states of the modular supervisors are reevaluated before proceeding any further. This mechanism is implemented with some kind of *function call*, a capability that may not be easy to provide on many PLCs. Furthermore, the hierarchical implementation entails some risks. For instance, constraints that imply counting consecutive occurrences of the same uncontrollable event require a lot of attention. This particular case must be handled differently than the other cases. As a result the code generation schema suffers from a lack of homogeneity.

3.3 Composite Implementation

Composite implementations refer to strategies in which the specifications of logic controllers are given, generally by *grafcets* or *sequential function charts* (SFCs). Furthermore, supervisors are synthesized [19] or specified by grafcets [7]. In the latter case, all the grafcets, including those modeling the process, are transformed into DFAs with the aim of using standard SCT algorithms to verify the *controllability* property. In both cases, the grafcets modeling the logic controller and the DFA or grafcet corresponding to the supervisor are implemented on a PLC. No specific code generation schema is explicitly proposed.

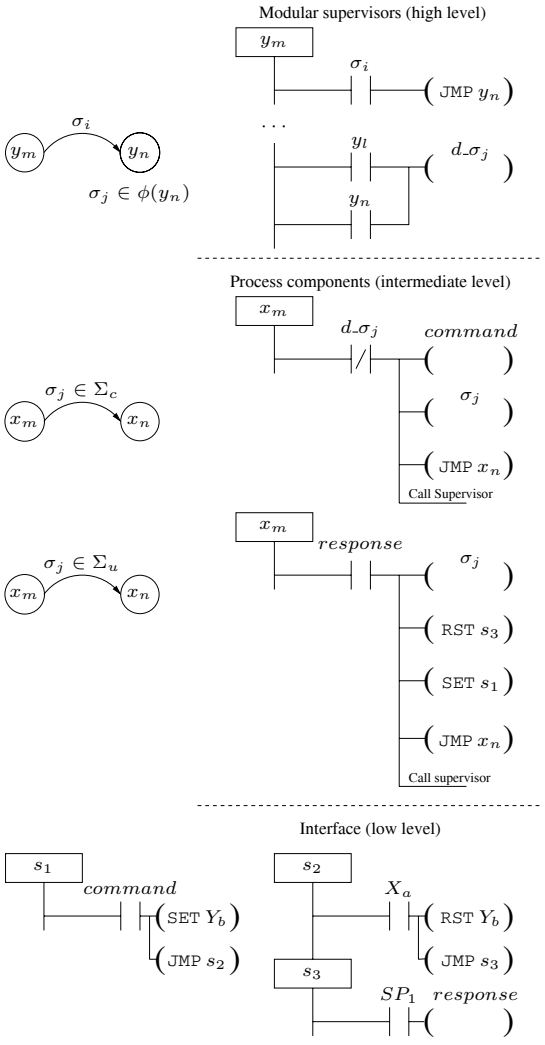


Figure 3. Hierarchical code generation schema

The method suggested in [7] is sound under an assumption that is unlikely to reveal to be true. Indeed it is impossible to guaranty that uncontrollable events occurring in the process will never be perceived as simultaneous due to the delay between I/O samplings induced by the PLC cyclical scan activity (particularly in processes with a high level of parallelism). Recall that the PLC memory I/O images are only updated at the beginning of each

scan cycle. Furthermore, the method seems to assume that the SFC logic used for the controller and supervisor will always reach a stable situation in one single iteration despite the feedback loops that can exist between the two components.

A similar approach proposed in [19] does take on account SFC stable situation analysis, but modeling the control as an SFC significantly adds to both, the space complexity of the solution (thus, high consumption of PLC resources), and the algorithmic complexity of its calculation with the use of theoretical artifacts and related synthesis algorithms going outside the range of SCT.

3.4 State-Based Implementation

The problems connected to the three previous implementations seem to suggest that an event-based PLC implementation for a synthesized supervisor is not very realistic. The main difficulties relate to the supervisor state space size and the ability of a PLC to track events in order to maintain the abstract state of S/G in synchronization with the state of the physical process. In other words, to remedy to those problems, it seems necessary to do away with event tracking and explicit supervisor state modeling.

A SFBC provides an alternate PLC implementation strategy that neither requires event tracking nor any explicit notion of abstract state. Considering the sub-family of predicates $\{f_\sigma \mid \sigma \in \Sigma_c\}$ on X introduced in Sec. 2.2, the idea is to express each f_σ of this sub-family under the form of a logical formula F_σ that depends only on the PLC I/Os.

Figure 4 presents the schematic Ladder diagram corresponding to this new code generation schema. The complete Ladder program is then made up of one Ladder rung per controllable event $\sigma \in \Sigma_c$. The condition part is a Boolean formula F_σ which implements the corresponding predicate f_σ based on disjunctions and conjunctions of literals, which are Boolean variables or the negation of Boolean variables associated to PLC I/Os. Each formula F_σ can be seen as the conjunction of two main conditions: a precondition defining a subset of states of the free behavior where σ is physically possible and an enabling condition reducing further that subset to states where $\delta^f(x, \sigma)$ is defined in G^f. The precondition usually translates into a small disjunction since it is rather uncommon that an actuator is shared among several components and there are generally few transitions labeled with σ. The action part is a sequence of instructions that trigger the controllable event σ. No other logic is necessary since the system no longer tracks uncontrollable events, and that controllable events are generated with respect to the current state of the process.

Rather than following the current state of the physical process through event tracking, the Ladder logic must determine this state from raw values of PLC I/Os. A discrete PLC I/O point is represented by a Boolean variable. The values of digitized analog PLC I/Os are divided into ranges of interest and each range is represented by an intermediate Boolean variable [1]. It is useful to observe that when all events (e.g., range entry, range exit, rising or falling edge of a sensor, and rising or falling edge of an actuator) are accounted for in the free behavior model of the process G, the value of each PLC I/O in a given free behavior state $x \in X$ is known, provided that initial conditions are also known. In this case, the ordered set of all PLC I/Os can be seen as a kind of state vector for the process. It is then always possible to generate a logical formula F_x for any given state $x \in X$ that returns `true` if and only if the process is in that specific state. This logical formula takes the form of a conjunction of elements of the state vector.

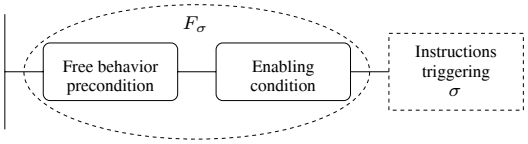


Figure 4. SFBC code generation schema

Let $\sigma \in \Sigma_c$ and $X_\sigma \subseteq X$ defined as follows: $X_\sigma := \{x \in X \mid f_\sigma(x) = 1\}$. The formula F_σ is then given by $\vee_{x \in X_\sigma} F_x$. It should be noted that there are as many disjuncts as there are elements in X_σ and there are at most as many conjuncts as there are discrete PLC I/Os or intermediate Boolean variables. Furthermore, since $X_\sigma \subseteq X$ and X is typically very big, then the Boolean formulas F_σ can grow out of proportion, but X_σ is usually small and the set of states to consider is limited to the reachable states under the SFBC f.

It is worth noting that by construction the logical formulas are in disjunctive normal form, and therefore all the known techniques of factorization are available to simplify them. If a formula F_σ is too big, it could be broken down in intermediate sub-formulas, provided that they are calculated every time that the triggering of a controllable event is susceptible to change their value, and prior to any further use of them during a PLC cycle. If the target PLC supports *function call*, this feature can be used to minimize the code generated from common sub-formulas.

It is interesting to note that the structure of the resulting Ladder program, a simple sequence of Ladder rungs, one per controllable event, allows better characterization of the control software. The PLC cycle timing is fixed (therefore stable) and can be calculated starting form the timing specifications of the target PLC. Also, if the functions F_σ in disjunctive normal form, for all $\sigma \in \Sigma_c$, are implemented without any function call, then the depth of the stack can be limited to three levels, which is adequate for most PLCs. The maximum depth of the stack can also be determined when applying optimization techniques to generate more efficient code. Finally, no special PLC capability is required since the program structure style is pure relay logic.

4. A Generative Programming Method

Without loss of generality, only discrete PLC I/Os are considered with the aim of simplifying explanations. To get a PLC implementation following the lines of the rationale described in Sec. 3.4, it is assumed that feedback only depends on the current state of the process, a precondition for using the SFBC approach. A typical framework is used to derive a SFBC f, but with specific ingredients in order to derive the formulas F_σ, for all $\sigma \in \Sigma_c$, from the elements of the problem model. Process modeling is performed as follows. For each physical component:

- define a state vector, which associates the component with its PLC I/Os;
- create a DFA that models its free behavior, taking into consideration all physically possible events related to its state vector;
- adjust the DFA to obtain a stable behavior;
- merge states to reduce the state space size after replacing controllable event sequences by an equivalent event representing a single command.

In a stable behavior, when a command is issued, no other command will be possible until the desired result has been achieved. This operation corresponds to the enforcement of local liveness constraints, but the constraints are expressed in the DFA of a component. Stable behavior precludes event traces of the form $(\uparrow Y_b \downarrow Y_b)^+ \uparrow Y_b$ for commands, which are equivalent to the more efficient form $\uparrow Y_b$, because it eliminates purposeless oscillations of a command.

The last step of this modeling phase consists in calculating the DFA G that models the free behavior of the process from the shuffle or synchronous product of all the DFAs of the participating components.

Constraint modeling is performed as follows for each constraint:

- for a component under a local constraint, take a copy of its DFA;
- for components under a non local constraint, take a copy of the product of their DFAs;
- add self-loops labeled by the missing events to each state;
- remove transitions that should be inhibited in a given state;
- merge equivalent states to reduce the state space size.

The results of this phase are DFAs that share the same state vectors and structure as the DFAs modeling the free behavior of components. The DFAs are combined by a synchronous composition in order to obtain a DFA E, which represents the conjunction of all the constraints. Therefore, it is always possible to establish a mapping between states of G and states of E, in order to determine what controllable events are enabled or disabled with respect to the current state of the process.

Finally, the last phase consists in synthesizing the behaviorally least restrictive SFBC f from G and E such that the closed loop f/G achieves the supremal controllable constraint stronger than the one represented by E.

5. Experimentation

To validate the new generative programming approach, a small scale experiment is presented. The target hardware is the first station of a modular production system (MPS) manufactured by Festo and controlled by Koyo PLCs. This station, called *distribution station*, comprises four components[1].

1. A feed magazine on top of a *barrel* that introduces workpieces into the process. The barrel is monitored by an optical sensor detecting the presence of a workpiece. Its state vector is $\langle X_6 \rangle$ (see Tab. 1 for its possible values).
2. A pneumatic ejecting cylinder, called the *injector*, coupled to the barrel that moves a workpiece from the barrel to a loading dock facing the barrel. This component has two stable positions: at the barrel and at the loading dock. Two magnetic sensors allow to detect its position. Its state vector is $\langle Y_2, X_0, X_1 \rangle$.
3. The *boom* of a handling device that moves a workpiece from the loading dock to an adjacent testing station. The boom can be stopped in any position, but only two valid positions can be detected by two mechanical sensors. Its state vector is $\langle Y_0, Y_1, X_2, X_3 \rangle$.
4. The *hook* of the handling device that is a vacuum suction cup which can grab and hold a workpiece while the vacuum pump is on. The vacuum pump is driven by two discrete output points. A pressure sensor can detect whether the suction cup is holding a workpiece or not. Its state vector is $\langle Y_3, Y_4, X_5 \rangle$.

Figure 5 shows the DFA modeling the stable behavior of the boom. All devices are modeled in this way. The DFA G obtained from a shuffle product has 128 states and 512 transitions.

The free behavior of the distribution station must be restrained in order to satisfy the following constraints.

1. When the injector is in position at the barrel, it may not inject on an empty barrel. Furthermore, it may not inject unless the boom is in position at the testing station. While holding a workpiece at the loading dock position, the injector may not reset unless the hook has grabbed the workpiece at the loading dock.

[1]The PLC discrete input points and output points are numbered and prefixed by the letter X and the letter Y, respectively.

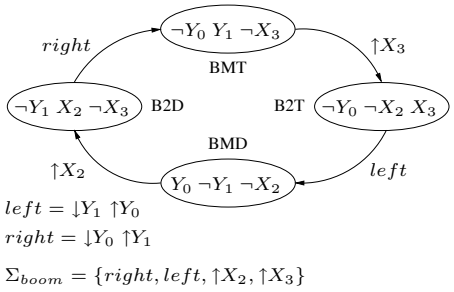


Figure 5. DFA modeling the stable behavior of the boom

2. When the boom is in position at the loading dock, it may not move towards the testing station unless either a workpiece has been grabbed by the hook or there is no workpiece under the hook and the injector is ready to inject. When the boom is in position at the testing station, it may not move towards the loading dock until the hook has finished releasing a workpiece and unless either the injector is in position at the loading dock or no workpiece is available at the barrel.

3. The hook must never try to grab a workpiece at the loading dock unless both the injector and the boom are in position at the loading dock. Once a workpiece is grabbed, the hook must keep holding it until the boom is in position at the testing station.

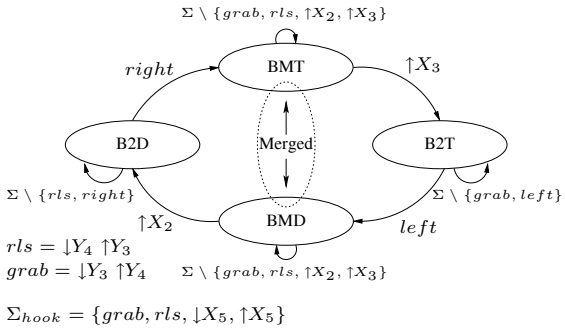


Figure 6. DFA modeling the constraint of the boom over the hook

These constraints are formally expressed by six DFAs. Figure 6 gives one of them, the constraint of the boom over the hook. It is worth noting that the DFA of Fig. 6 is derived from the one of Fig. 5. The DFA representing the conjunction of constraints has 302 states and 2940 transitions.

The SFBC f has been calculated by our own software environment MELODIES[2]. It has 52 entries. Some of them are presented in Tab. 1. They correspond to the boom being in position at the loading dock and ready to move right towards the testing station. In this table d_1 denotes the conjunction I2B $\wedge$ BE $\wedge$ B2D $\wedge$ HF and similarly for the other mnemonics.

Table 1. A part of the feedback function f

	State of G				Inhibitions
	Injector	**Barrel**	**Boom**	**Hook**	
d_1	I2B	BE	**B2D**	HF	inject right grab
d_2	I2B	BF	**B2D**	HF	inject grab
d_3	I2B	BF	**B2D**	HH	inject rls
d_4	I2B	BE	**B2D**	HH	inject right rls
d_5	I2D	BF	**B2D**	HF	reset right
d_6	I2D	BE	**B2D**	HF	reset right
d_7	I2D	BF	**B2D**	HG	reset right
d_8	I2D	BE	**B2D**	HG	reset right
d_9	I2D	BF	**B2D**	HH	right rls
d_{10}	I2D	BE	**B2D**	HH	right rls
d_{11}	IMB	BF	**B2D**	HF	grab
d_{12}	IMB	BE	**B2D**	HF	grab
d_{13}	IMB	BF	**B2D**	HH	rls
d_{14}	IMB	BE	**B2D**	HH	rls

Injector:
I2B : Injector to barrel = $\neg Y_2 \wedge X_0 \wedge \neg X_1$
IMD : Injector moving to dock = $Y_2 \wedge \neg X_1$
I2D : Injector to dock = $Y_2 \wedge \neg X_0 \wedge X_1$
IMB : Injector moving to barrel = $\neg Y_2 \wedge \neg X_0$

Barrel:
BE : Barrel empty = X_6
BF : Barrel full = $\neg X_6$

Boom:
B2D : Boom to loading dock = $\neg Y_1 \wedge X_2 \wedge \neg X_3$
BMT : Boom moving towards testing station = $\neg Y_0 \wedge Y_1 \wedge \neg X_3$
B2T : Boom to testing station = $\neg Y_0 \wedge \neg X_2 \wedge X_3$
BMD : Boom moving towards loading dock = $Y_0 \wedge \neg Y_1 \wedge \neg X_2$

Hook:
HF : Hook free = $\neg Y_4 \wedge \neg X_5$
HG : Hook grabbing = $\neg Y_3 \wedge Y_4 \wedge \neg X_5$
HH : Hook holding = $\neg Y_3 \wedge X_5$
HR : Hook releasing = $Y_3 \wedge \neg Y_4 \wedge X_5$

In order to generate the Ladder code that causes the controllable event $right$ to occur, the Boolean formula F_{right} must be constructed. First, only entries of f that do not disable the event $right$ must be selected. Therefore, $F_{right} = d_2 \vee d_3 \vee d_{11} \vee d_{12} \vee d_{13} \vee d_{14}$. Second, some factorization techniques are applied to simplify F_{right} resulting in the Ladder code of Fig. 7. The precondition clearly appears at the left in the rung (B2D). The enabling condition follows immediately at the right. The instructions of the rung are the commands causing the event $right$. The overall program consists of six rungs, one per controllable event, for a total of 137 PLC machine instructions.

6. Conclusion

This paper has introduced a new code generation schema for implementing a synthesized supervisor based

[2]MELODIES stands for Modeling Environment for LOgical DIscrete Event Systems.

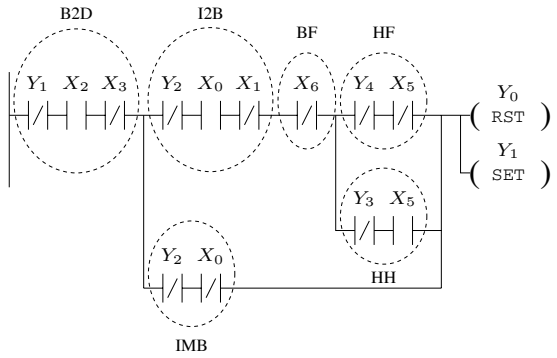


Figure 7. Part of the Ladder code

on the state-based supervisory control paradigm. It places the emphasis on a technique that avoids implementing a transition structure into the PLC. This approach does not preclude modular supervision. The basis for code generation is an SFBC f synthesized using standard SCT procedures, therefore inheriting its state explosion problem. However, once the SFBC has been obtained, a naive procedure would require scanning the SFBC table once per controllable event to generate code for each rung of the PLC program. It is a computation of complexity $O(\mid \Sigma_c \mid \times \mid f \mid)$. Therefore, it seems more scalable than the previous approaches because the code size grows linearly with respect to the number of controllable events. This schema works under the assumption that an SFBC can be derived for the process, that is, feedback only depends on the current state of the process. A larger class of systems using a dynamic state feedback control [17], which is an extension of SFBC, allows to take into consideration situations where memory elements are needed. In such systems, information about data structures (e.g., a FIFO queue) or workpiece attributes are stored in a memory, thus a wider range of constraints could be considered, but with some extensions to the new code generation schema. Some more work is also required to include fault management, diagnostic, and recovery.

7. Acknowledgment

The research described in this paper was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] S. Balemi, G. J. Hoffman, P. Gyugyi, H. Wong-Toi, and G. Franklin. Supervisory control of a rapid thermal multiprocessor. *IEEE Transactions on Automatic Control*, 38(7):1040–1059, 1993.

[2] M. Barbeau, G. Custeau, and R. St-Denis. Requirements engineering and synthesis of a control system. *Automatic Control Production Systems*, 28(1):37–52, 1994.

[3] M. Barbeau, F. Kabanza, and R. St-Denis. A method for the synthesis of controllers to handle safety, liveness, and real-time constraints. *IEEE Transactions on Automatic Control*, 43(11):1543–1559, 1998.

[4] B. Brandin. The real-time supervisory control of an experimental manufacturing cell. *IEEE Transactions on Robotics and Automation*, 12(1):1–14, 1996.

[5] E. Brinksma, A. Mader, and A. Fehnker. Verification and optimization of a PLC control schedule. *International Journal on Software Tools for Technology Transfer*, 4(1):21–33, 2002.

[6] V. Chandra, Z. Huang, and R. Kumar. Automated control synthesis for an assembly line using discrete event system control theory. *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews*, 33(2):284–289, 2003.

[7] F. Charbonnier, H. Alla, and R. David. The supervised control of discrete-event dynamic systems. *IEEE Transactions on Control Systems Technology*, 7(2):175–187, 1999.

[8] M. H. de Queiroz and J. E. R. Cury. Modular supervisory control of large scale discrete event systems. In R. Boel and G. Stremersch, editors, *Discrete event systems – analysis and control*, pages 103–110, Boston, 2000. Kluwer Academic Publishers.

[9] M. H. de Queiroz and J. E. R. Cury. Sysnthesis and implementation of a local modular supervisory control for manufacturing cell. In *Proceedings of 6th International Workshop on Discrete Event Systems*, pages 377–382, Zarogaza, Spain, 2002.

[10] G. Frey and L. Litz. Formal methods in PLC programming. In *Proceedings of the IEEE Conference on Systems, Man and Cybernetics*, pages 2431–2436, Nashville, 2000.

[11] R. Kumar and V. K. Garg. *Modeling and Control of Logical Discrete Event Systems*. Kluwer, Boston, MA, 1995.

[12] S. C. Lauzon, J. K. Mills, and B. Benhabib. An implementation methodology for the supervisory control of flexible manufacturing workcells. *Journal of Manufacturing Systems*, 16(2):91–101, 1997.

[13] J. Liu and H. Darabi. Ladder logic implementation of Ramadge-Wonham supervisory controller. In *Proceedings of the 6th International Workshop on Discrete Event Systems*, pages 383–389, Zaragoza, Spain, 2002.

[14] P. J. G. Ramadge and W. M. Wonham. Modular feedback logic for discrete event systems. *SIAM Journal on Control and Optimization*, 25(5):1202–1218, 1987.

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

[16] D. Tilbury and P. Khargonekar. Challenges and opportunities in logic control for manufacturing systems. Engineering research center for reconfigurable machining systems, University of Michigan, 2000.

[17] W. M. Wonham. Notes on control of discrete-event systems. System Control Group ECE 1636F/1637S, University of Toronto, revised 2003.

[18] W. M. Wonham and P. J. G. Ramadge. Modular supervisory control of discrete event systems. *Mathematics of Control, Signals, and Systems*, 1(1):13–30, 1988.

[19] J. Zaytoon and V. Carré-Ménétrier. Synthesis of control implementation for discrete manufacturing systems. *International Journal of Production Research*, 39(2):329–345,, 2001.