15

Supervisory Control of Manufacturing Systems

The focus of this chapter is the autonomous supervisory control of part flow within networked flexible manufacturing systems (FMSs). In manufacturing industries that employ FMSs, automation has significantly evolved since the introduction of computers onto factory floors. Today, in extensively networked environments, computers play the role of planners as well as that of high-level controllers. The preferred network architecture is a hierarchical one: in the context of production control, a hierarchical network of computers (distributed on the factory floor) have complete centralized control over the sets of devices within their domain, while receiving operational instructions from a computer placed above them in the hierarchical tree.

In a typical large manufacturing enterprise, there may be a number of FMSs, each comprising, in turn, a number of flexible manufacturing workcells (FMCs) (Fig. 1). These FMCs will be connected via (intercell) material handling systems such as automated guided vehicles (AGVs) and conveyors (Chap. 12).

FMCs have been, commonly configured for the fabrication and/or assembly of families of parts with similar processing requirements. A traditional FMC comprises a set of programmable manufacturing devices with their own controllers that are networked to the FMC's host computer for the downloading of production instructions (programs) as well as to a supervisory controller for the autonomous control of parts flow (Fig. 2).

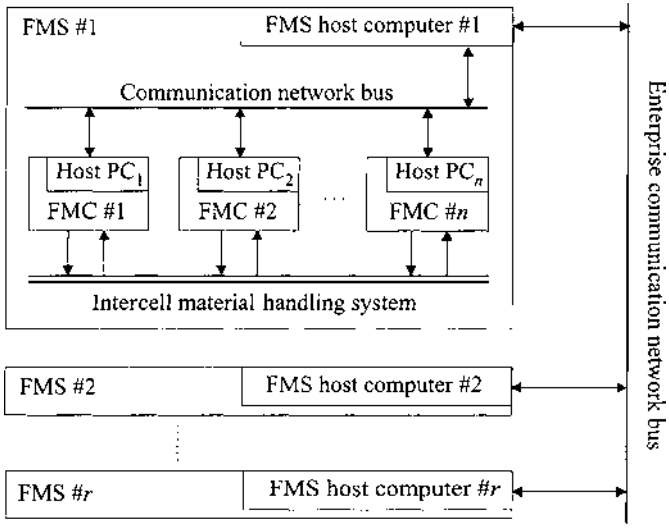


FIGURE 1 A networked manufacturing environment.

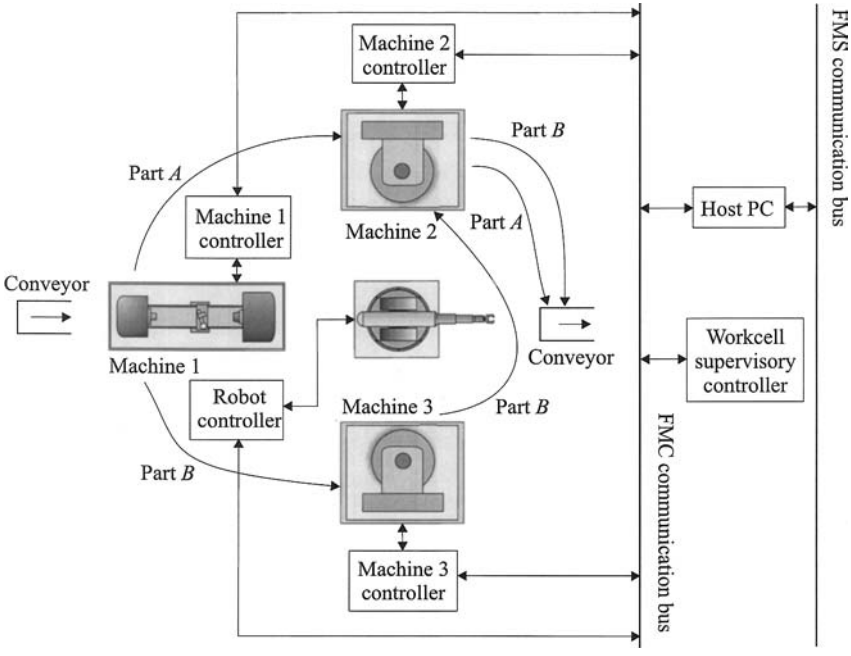


FIGURE 2 A flexible manufacturing workcell.

Although human operators have been traditionally used in the past century in the traffic control of part movements on the factory floor, personal computers (PCs) and programmable logic controllers (PLCs) have been replacing them since the early 1980s at a rapid pace. Such autonomous traffic controllers can be programmed with high-level instructions to make (correct) decisions in fractions of a second and communicate these decisions to the individual FMC devices with no delays. In turn, these devices can carry out their expected tasks as preprogrammed in their respective controllers, which will have been downloaded a priori or on-line from the host PC of the FMC. An FMC "supervisor" initiates/terminates device operations, though it does not interfere with the accomplishment of these tasks.

In contrast to time-driven (continuous variable) control of the individual devices in an FMC, the supervisory control of the FMC is event driven. The future actions of the FMC are solely dependent on the past events, as opposed to being clock driven. Thus manufacturing systems can be considered as discrete event systems (DESs) from a supervisory control perspective. DESs (also known as discrete event dynamic systems, DEDSs) evolve according to the (unpredictable) occurrence of events that are instantaneous, asynchronous, and nondeterministic.

The state of a DES changes in a deterministic manner based on the physical event that has just been observed, but the system overall is nondeterministic, since in any one state there may be several possible routes of actions ("enabled" events) that can take place. Nondeterminism implies that we may not know a priori which event (among the several possible) will take place, though once observed, this event can lead to only one future state of the DES (i.e., deterministic transition). For example, when a machine is working (state = Working), it may either complete its operation (event = Task completion) or break down (event = Failure), we do not know in advance which one will happen. However, we do know that the former will take the machine to its "Idle" state and the latter will take the machine to its "Down" state.

There exist three interested parties to this practical and very important manufacturing problem: users, industrial controller developers, and vendors and academic researchers. The users (customers) have been always interested in controllers that will improve productivity and impose minimal restrictions. Effective (supervisory) controllers are necessary for them to implement existing flexible manufacturing strategies. Industrial controller vendors have almost exclusively relied on the marketing of PLCs in the past two decades in response to the control needs of FMSs. Their efforts have largely concentrated on hardware improvements and better user interfaces, though continuously lagging behind developments by the PC

industry by several years. The programming of PLCs must still be carried out in ad hoc manner (versus mathematical formalism), and thus it is prone to human error.

The academic community has spent the past two decades developing very effective formal control theories that are suitable for the supervisory control of manufacturing systems. Control strategies determined by invoking any one of these theories can be software coded and downloaded onto a PC or PLC for real-time (DES) control of limited-size FMCs. Naturally, although the successful control of such manufacturing systems have been shown in academic laboratory settings, appropriate software tools must be developed by current industrial controller developers/vendors prior to their adoption by the users (i.e., the manufacturing industry).

In this chapter, we will address two of the most successful DES control theories developed by the academic community: Ramadge–Wonham automata theory and Petri-nets theory. As proposed in Fig. 3, it is expected

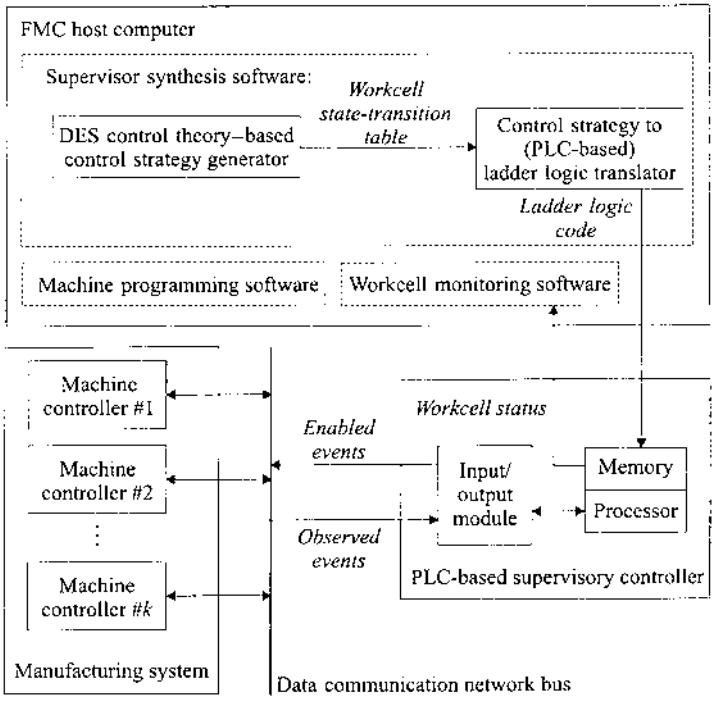


FIGURE 3 Software architecture for FMC control.

that in the future industrial users will employ such formal DES control theories in the supervisory control of their FMCs. The description of PLCs, used for the autonomous DES-based supervisory control of parts flow in FMCs, concludes this chapter. In Fig. 3 the term ladder logic refers to the programming language used by most current PLC vendors.

15.1 AUTOMATA THEORY FOR DISCRETE EVENT SYSTEM MODELING

Automata theory generally refers to the study of the dynamic behavior of information systems that can be described by a finite number of states and with discrete inputs and outputs. Although our focus in this chapter is on manufacturing systems, the field of automata theory was originally developed in response to the needs of computer science. It is of interest to note, however, that the first published work in the field of finite-state systems (''machines'') by A. M. Turing in 1936 preceded all (digital) computers.

Significant advancements in the field of automata were reported in the 1950s and the early 1960s in the works of N. Chomsky, G. H. Mealy, and E. F. Moore. The application of automata theory to the supervisory control of manufacturing systems, though, was made possible only after the pioneering works of P. J. G. Ramadge and W. M. Wonham in the late 1980s (today known as the R–W theory). Thus, in this section, following a brief background review on the theories of languages and automata, we will present an overall description of the R–W theory.

15.1.1 Formal Languages and Finite Automata

Automata theory deals with systems whose dynamics is dependent on the occurrence of events that cause the system to change its state. Abstract algebra is an essential tool in the modeling and analysis of such DESs, in contrast to the use of differential calculus in time-varying systems.

Sets: A set is a collection of elements with a common property:

$S = \{s \mid s \text{ has property } P\}$ or $s \in S$

Most common operations on sets include

Union (sum): $A \cup B = \{a \mid a \in A \text{ or } b \mid b \in B\}$.

Intersection: $A \cap B = \{a \mid a \in A \text{ and } b \mid b \in B\}$.

Cartesian product: $A \times B = \{(a, b) \mid a \in A, b \in B\}$.

For example, let $A = \{\alpha, \beta\}$ and $B = \{\gamma, \delta\}$, then

$$A \cup B = \{\alpha, \beta, \gamma, \delta\} \qquad A \cap B = \phi$$
$$A \times B = \{(\alpha, \gamma), (\alpha, \delta), (\beta, \gamma), (\beta, \delta)\}.$$

(The elements of $A \times B$ are termed as "ordered pairs").

Mapping: f: $A \rightarrow B$; the function, f, maps the elements of A into B.

For example

$$f(\alpha) = \delta \qquad \text{and} \qquad f(\beta) = \gamma \qquad \alpha \in A \text{ and } \delta \in B.$$

Combinational logic: Logic elements can be used to perform logical operations on multiple inputs in order to yield a desired output. In binary-valued logic, the two most commonly used operations are AND and OR:

Input		Output, y				
x_1	x_2	AND	OR	NAND (not AND)	NOR (not OR)	Exclusive OR
0	0	0	0	1	1	0
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	1	0	0	0

The "not" operation, also known as the complementation, negates the value of the output (0 to 1, or 1 to 0). Although the above table only shows two input variables for clarity of discussion, there may be multiple input variables (≥ 2), on which the logical operations would be applied in the same manner.

Languages: In a DES, the set of all possible events can be considered as the alphabet, E, from which sequences of events, strings or words, can be generated. An (artificial) language is a collection set of strings (events). For example, for $E = \{\alpha, \beta, \gamma, \delta\}$, a language could be $L = \{\alpha\beta, \alpha\gamma\delta\}$.

Finite automata: A finite automaton comprises a finite set of states and a set of transitions (events) that occur according to the alphabet of the DES. Finite automata are also known in the literature as finite-state machines describing the dynamics of sequential machines (i.e., DESs). Automata are also considered as generators of languages according to well-defined rules. Formally, a finite-state automaton (FA) is defined by a quintuple,

$$\text{FA} = (S, E, f, s_0, F)$$

where S is a finite (nonempty) set of states, E is a finite (input) alphabet (events), f is a state-transition (mapping) function, f: $S \times E \rightarrow S$, s_0 is the initial state, $s_0 \in S$, and F is the set of final states, $F \subseteq S$.

For example, let us consider the finite automaton, M, shown in Fig. 4, where $S = \{s_0, s_1, s_2\}$, $E = \{0, 1\}$, $F = \{s_0\}$ and

$$f(s_0, 1) = s_2 \qquad f(s_1, 1) = s_0 \qquad f(s_2, 1) = s_1$$
$$f(s_0, 0) = s_1 \qquad f(s_1, 0) = s_2 \qquad f(s_2, 0) = s_0$$

In Fig. 4, the initial state is marked by an arrow labeled "start" and the final state is marked by two concentric circles. An input sequence (string) of $w = 000$ into M would yield the state s_0, $w = 00100$ would also yield s_0, etc.

A string w is said to be "accepted" by a FA, if $f(s_0, w) = p$, where $p \in F$. The language accepted by the FA, L(FA), is the set of all (accepted) strings satisfying this condition.

There exist two common finite-state machines with user-specified outputs at all of their states: Moore and Mealy machines. In Moore machines, the output at a specific state is defined regardless of how that state has been reached, while in Mealy machines, the output is dependent on the state as well as how it has been reached (i.e., the specific input transition to this state). Typical Moore and Mealy machines are given in Fig. 5a and Fig. 5b, respectively.

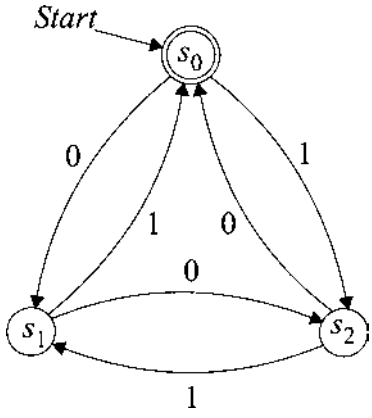


FIGURE 4 A finite-state automaton.

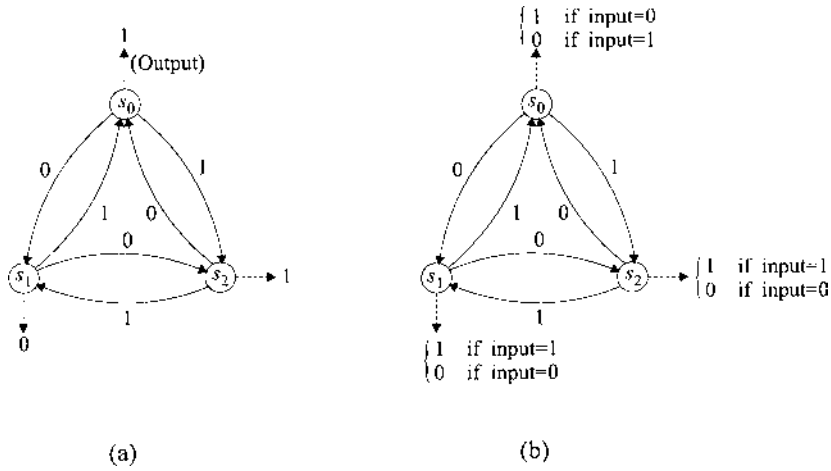


FIGURE 5 (a) A typical Moore machine; (b) a typical Mealy machine.

Formally, both Mealy and Moore machines are defined by a sextuple,

$$M = (S, E, O, f, g, s_0)$$

where S is a finite set of states, E is a finite input alphabet, O is a finite output alphabet, f is a state-transition function, g is output (mapping) function and s_0 is the initial state. In Mealy machines g is a function of the input as well, $g(s,e)$, $e \in E$. For example, in Fig. 5b, $g(s_0,1) = 0$, $g(s_0,0) = 1$, $g(s_1,1) = 1$, etc. Thus an input sequence of $w = 0011$ would yield an output of 1 in the Moore machine, while it would yield an output of 0 in the Mealy machine.

15.1.2 Ramadge–Wonham Supervisory Control Theory

Supervisory control of a DES, in the context of finite-state automata theory, can loosely be defined as the enablement (or disablement) of events at the latest reached state of the system. That is, a supervisor (a finite-state automaton) changes its state according to the latest event observed within the DES and informs the (controlled) DES what future events are enabled (or disabled). (Fig. 6). Naturally, only a subset of all events (defined in the alphabet, E) are controllable and only they can be enabled/disabled. For example, the start of an operation is a controllable event, whereas a breakdown event is uncontrollable by the supervisor.

The Ramadge–Wonham (R–W) controlled automata theory allows users to synthesize supervisors that are correct by construction. That is, all the system states within the supervisor are reachable through a

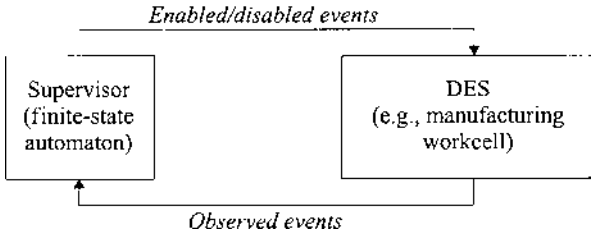


FIGURE 6 Supervisory control of a DES.

sequence of events (strings) included in the (''supremal-controllable'') language of the automaton—a deadlock-free controller. Prior to the description of the controller synthesis process, the fundamentals of R–W (DES modeling) theory will be briefly described here. For consistency with the existing literature, the nomenclature introduced by Ramadge and Wonham will be utilized.

The R–W finite-state automaton, G, is defined by a quintuple,

$$G = (Q, \Sigma, \delta, q_0, Q_m)$$

where Q is the finite set of states, Σ is the finite alphabet of events, δ: $Q \times \Sigma \rightarrow Q$ is the (one-to-one mapping) function defining the transition between states according to observed events, q_0 is the initial state, $q_0 \in Q$, and $Q_m \subseteq Q$ is a subset of marker (completed task) states. A transition event is formally defined as a triple (q, σ, q'), where $\delta(\sigma,q) = q'$, for $\sigma \in \Sigma$ and $q, q' \in Q$.

The alphabet of events, Σ, is further partitioned into two disjoint subsets of controllable, Σ_c, and uncontrollable, Σ_u, subsets, where $\Sigma_{\hat{c}} \cup \Sigma_{\hat{u}}$. In an automaton, controllable events can be enabled (shown by a ''tick'' across the transition line in a directed graph), while uncontrollable events can be observed but not enabled or disabled. Fig. 7 illustrates a model of a machine with three states (idle, I, working, W, down, D) and four events (start to operate, α; finish, β; breakdown, λ; get repaired, μ), of which the breakdown and finish events are not controllable.

An automaton, G, is said to be nonblocking (deadlock free) if the language $L(M)$ includes the marked language accepted by M. The marked language, L_m, includes all strings that commence and terminate at the automaton's marker states (e.g., state I in Fig. 7). If the language, L, includes a string that leads to a nonmarker state with no controllable or uncontrollable event exiting it, then the DES is deadlocked at this state. Such (deadlock) states are labeled as not reachable and/or coreachable in R–W theory.

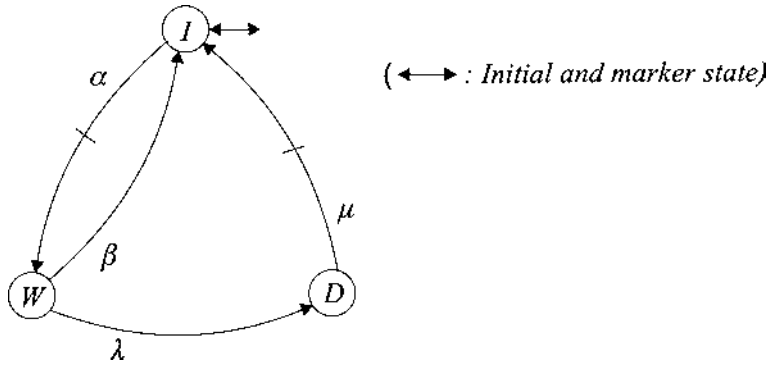


FIGURE 7 A (finite-state) automaton model for a machine.

The synthesis of a (controllable) supervisor is a two-step procedure: first, all the automata representing the individual machines of a DES are combined into one overall (uncontrolled) system automaton through a "shuffle" operation, while in parallel all automata representing the control specifications of this system are combined through a "meet" operation into one overall specifications automaton; second, the intersection of the languages of these two (system and specification) automata is obtained through a meet operation to determine the supremal- controllable language of the supervisor. This procedure is illustrated below through a simple manufacturing workcell example—two machines with a buffer of capacity one in between:

Shuffle operation: The shuffle operation (also known as the synchronous product) of two languages, $L_1||L_2$, yields a language comprising all possible interleavings of the strings of L_1 with those of L_2. The shuffled automaton of two machines, shown in Fig. 7, is given in Fig. 8. All shown system states (*II*, *WI*, *DI*, etc.) refer to the individual states of the two machines. For example, *IW* implies that the first machine, M_1, is idle, while M_2 is working. The indices of the events correspond to the machine numbers, $i = 1, 2$.

Meet operation: The meet operation applied on two languages yields their intersection, namely, a language comprising all the strings accepted by both their automata, $L = L(G_1) \cap L(G_2)$. As an example, the meet operation is applied herein on the (uncontrolled) system automaton shown in Fig. 8 and the control specification automaton shown in Fig. 9. This workcell specification does not allow M_1 to start operating unless the buffer, B, is already empty (preventing overflow) and does not allow M_2 to start operating unless the buffer contains a part that can be drawn by M_1

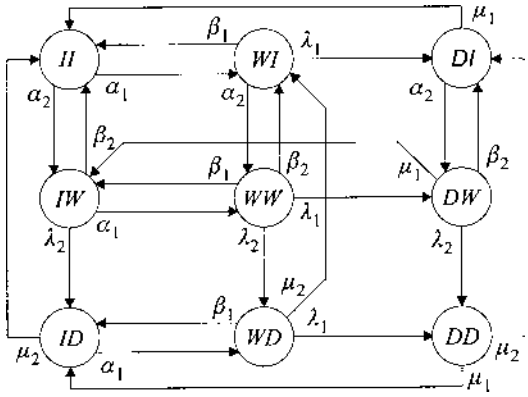
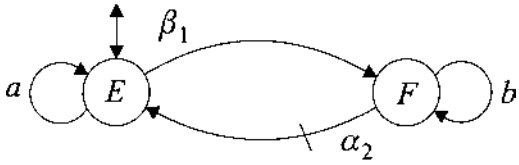


FIGURE 8 A shuffled automaton.

(preventing underflow). The resulting controllable supervisor for the specific M_1—B—M_2 DES is given in Fig. 10.

As shown in Fig. 10, the finite-state automaton (supervisor) of the overall manufacturing workcell, *SUP*, has 12 states and 25 transitions. The supervisor is nonblocking (deadlock-free) by construction. It enables controllable events and changes states by the observation of both controllable and uncontrollable events. A system state (label) in Fig. 10 is the



Self-loop-a: $\alpha_1, \lambda_1, \mu_1, \beta_2, \lambda_2, \mu_2$

Self-loop-b: $\lambda_1, \mu_1, \beta_2, \lambda_2, \mu_2$

E: Empty; F: Full

FIGURE 9 A control specification automaton, *B*.

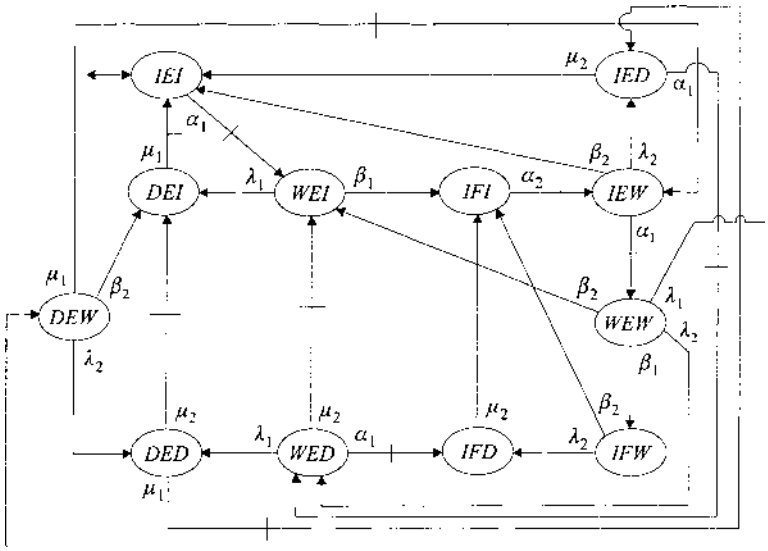


FIGURE 10 Supervisor, SUP, automaton.

concatenation of the individual states of the devices. For example, *IEI* refers to the machines, M_1 and M_2, being idle and the buffer, B, being empty.

15.2 PETRI NETS

Petri nets (PNs) provide engineers with a mathematical formalism for the modeling and analysis of DESs, such as manufacturing systems. They provide a simpler alternative to automata theory for the graphical representation of parts flow in a manufacturing system in terms of (system) states and transitions (events). (This graphical representation can be expressed by a set of linear algebraic equations.) However, the academic community has yet to illustrate clearly whether the formalism of PNs is superior to that of automata theory. In this section, we will only discuss the fundamentals of PNs and refrain from declaring a winner.

PNs were originally developed in the late 1950s and early 1960s by C. A. Petri. Petri's Ph.D. dissertation on the use of automata for the modeling and analysis of communications (events) within computer systems was published in 1962 in the Federal Republic of Germany. The use of PNs in manufacturing system modeling, however, started only in the early 1980s,

coinciding with the start of the widespread use of computers in manufacturing planning and control activities.

Since the 1980s, significant advancements have been reported by the academic community in the use of PNs for queuing simulations (performance analysis), scheduling, and supervisory control of manufacturing systems using ordinary (event-based) PNs, timed PNs (stochastic or deterministic), and "colored" PNs, where colors (differentiators) are used for the modeling of a number of different parts within a PN. However, except for an isolated success in developing a PN-based programming language (GRAFCET) for sequential logic controllers, the implementation of PNs in industrial manufacturing environments has been sparse.

Our focus in this book will be on the modeling of manufacturing systems using deterministic (versus stochastic), nontime (versus timed), ordinary (versus colored) PNs. Furthermore, the emphasis will be on the potential use of PNs for the supervisory control of manufacturing systems (versus their performance evaluation).

15.2.1 Discrete Event System Modeling with Petri Nets

PNs allow engineers to model asynchronous (event-driven) manufacturing systems, with concurrent operations and shared resources, by formalizing precedence relations. A PN is a directed bipartite graph comprising nodes, places, and transitions joined by directed arcs. Places (states) are represented by circles and transitions (events) by bars/rectangles.

The dynamics of a PN is achieved by tokens that are moved from one place to another by a transition connecting them. A transition can be weighted to transfer multiple tokens at one instance. (For example, a transition can cause two tokens to leave a place, but arrive at the next place as only one token.) The marking of a PN is an n-component vectorial representation of the number of tokens stored in each of its places. An example PN with its initial marking, $m_0 = (3,1,1)$, is shown in Fig. 11. For ordinary PNs all the weights are equal to 1.

Formally, a marked PN can be represented by a quintuple,

$$PN = \{P, T, I, O, m_0\}$$

where $P = (p_1, p_2, \ldots, p_n)$ is a finite set of places, $T = (t_1, t_2, \ldots, t_p)$ is a finite set of transitions, I is an input function representing all directed arcs from P to T, $P \times T$, O is an output function representing all directed arcs from T to P, $T \times P$, and m_0 is the initial marking. Both I and O can be expressed as (incidence) matrices, whose elements are 0 or 1 for ordinary PNs representing the absence or presence of a joining arc, respectively.

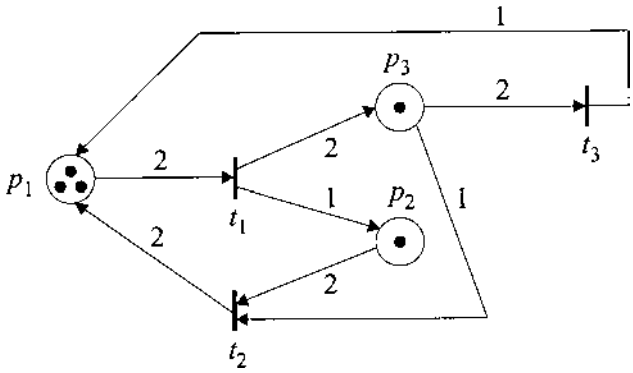


FIGURE 11 A marked Petri net.

A transition t is enabled if all places connected to it by input arcs contain tokens in numbers equal to or greater than the weights attached to the arcs. An event within the modeled system causes the corresponding transition to "fire." A fired transition causes transfer of tokens between places according to the specific weights. For example, in Fig. 11, the firing of transition t_1 yields the following marking: $m_1 = (1,2,3)$. A sequence of transitions, for example, $\sigma = \langle t_1, t_2, t_3, t_1 \rangle$, takes the same PN from its initial marking $m_0 = (3,1,1)$ to $m_4 = (2,1,2)$.

A transition without an input place is called a source and is always enabled. Similarly, a transition without an output place is called a sink that can be fired for the pure removal of tokens from the PN when enabled (Fig. 12). A self-loop is a circular representation of one place and one transition connected by an input as well as an output arc (Fig. 12). For example, a self-loop used in the modeling of a production machine would not allow the start of a new operation until the current operation is concluded.

Properties of PN Models

The properties of PNs can be classified as behavioral and structural. The former depend on the structure and the initial marking of the PN, while the latter depend only on the structure of the PN. Here we review several PN properties pertinent to manufacturing systems.

Reachability: A PN marking, m_k, (i.e., a specific system state) is said to be reachable if there exists a sequence of transitions, σ, that leads from m_0 to m_k. The (behavioral) reachability property of a PN can be analyzed by generating the corresponding reachability tree/graph, starting from the initial marking, m_0. In order to limit the size of the tree, markings (states),

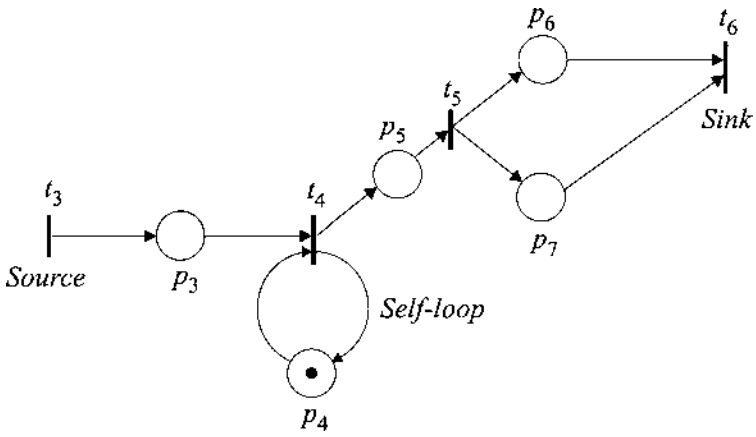


FIGURE 12 An example of an ordinary PN with a self-loop (all weights are 1 and thus not shown).

reached by (random) firing of transitions (events), that have already been noted as an earlier-encountered node on the tree branch from m_0, are labeled as old. No further transitions are fired from old markings. This elimination of duplicate markings result in a more compact coverability tree, which is equivalent to the reachability tree. In generating a reachability/coverability tree, one must note that a PN's marking can be changed by the simultaneous firing of multiple enabled transitions, as opposed to sequential firing.

Boundedness: Given the reachability set of all possible markings, a place, p_i, is l-bounded if it receives a maximum number of l tokens. The number l may or may not need to be a function of the initial marking. In manufacturing applications, boundedness can define the necessary capacity of a buffer or show its overflow. If the place examined is a machine, the term safeness is used to indicate a boundedness of $l = 1$, (i.e., only one operation at a time is allowed on that machine).

Liveness: A transition, t, is live if at any marking defined by the reachability tree there exits a sequence of subsequent transitions, σ, whose firing will lead to a marking that will reenable it. The PN is live as a whole if all of its transitions are live, i.e., the system is free of deadlock. A transition, t, is dead at a specific marking (also called dead marking) if there exits no subsequent sequence of transitions, σ, that will reenable it. A PN may have multiple dead markings, i.e., deadlock states. In the most common deadlock situations, called circular waiting, two or more processes, arranged in a circular closed-loop chain, each wait for resource availability next in the

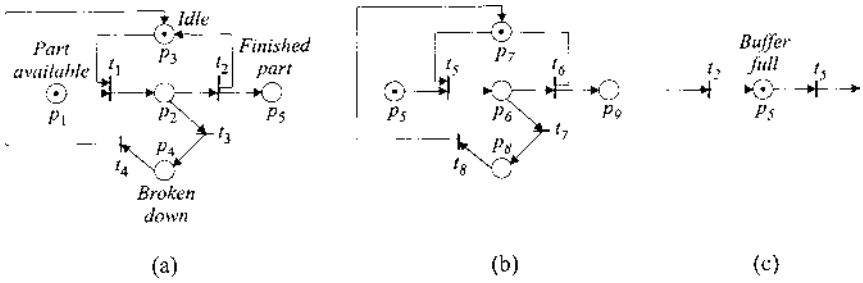


FIGURE 13 (a) PN model for M_1; (b) PN model for M_2; (c) PN model for B.

chain. A possible solution to such a practical problem is the utilization of buffers with sufficient storage capacity.

15.2.2 Synthesis of Petri Nets

The modeling of multiresource DESs, such as manufacturing workcells, can be carried out either by modeling the system as a whole or by modeling the individual resources first and then connecting them using a synthesis method. There are two primary PN synthesis methods: bottom-up and top-down.

A typical bottom-up approach would connect (live and bounded) multiple individual PNs into a larger system PN by merging common places into a new place. An alternative bottom-up approach would connect simple elementary paths shared by the individual PNs: for example, merging common paths terminated on both ends by a transition or by a place. A PN for a manufacturing line that comprises two machines (prone to failure), M_1 and M_2, and a buffer of size 1, B, that are combined in an M_1—B—M_2 configuration, whose PNs are given in Fig. 13, can be synthesized using a bottom-up approach as shown in Fig. 14.

In Figs. 13a and 13b, the PN model of the machine allows it to work if the machine was previously idle and a part is available (e.g., placed on its worktable). Once working, the machine can either finish its operation or break down. The machine returns to its idle state and the finished part is made available for the next resource/buffer/etc. after the machine is finished working. The reachability tree for such a machine model is given in Fig. 15. [As one can note, an external transition, t_e, making a part available to the machines (i.e., supplying a token to p_1 or p_5, respectively) is not included in the tree. Such a transition could happen only once the finished part is removed form the machine.]

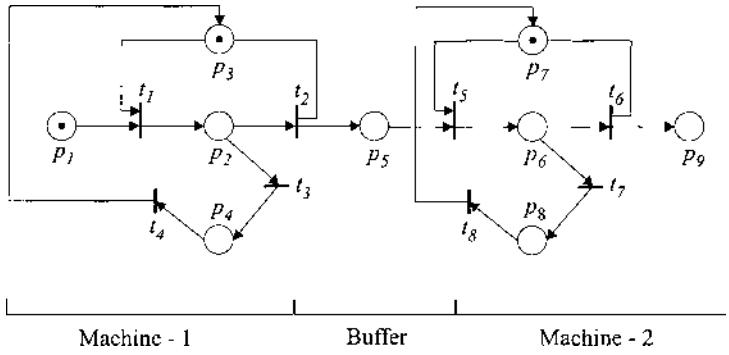


FIGURE 14 A PN for a M_1—B—M_2 manufacturing line.

In top-down synthesis techniques, the overall PN is developed in a gradual manner by (stepwise) refinement of places or transitions on an existing in-process PN model. That is, a more detailed submodel is inserted into the latest PN at hand as a block to replace a transition or a place. Unlike the bottom-up synthesis approaches, which provide users with flexible tools for the modular construction of large PNs, the top-down methods are more suitable for the minor refinement of already existing PNs, for example, the replacement of a resource or an operation.

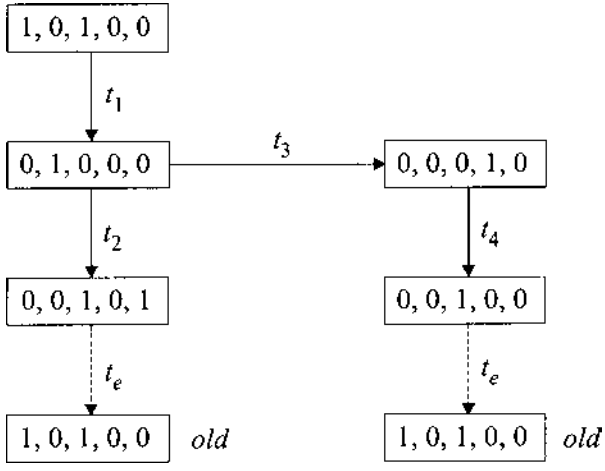


FIGURE 15 Reachability tree for M_1.

In regard to behavioral properties, although most synthesis methods (bottom-up or top-down) are expected to preserve liveness and boundedness, in practice it would be advisable to reexamine the properties of the resultant synthesized PN.

15.2.3 Supervisory Control Using Petri Nets

PNs have evolved since the early 1980s to cope with a variety of manufacturing constraints, such as modeling the flow of several different part types through the manufacturing system, sharing of resources, and so on. Although the primary utilization area of PNs has been in the simulation of manufacturing systems for performance analysis, there have also been supervisory control cases: for example, (1) the direct implementation of PNs through the use of a companion execution software and (2) the translation of PNs into sequential programming codes that can be implemented on a PLC or on an industrial PC with external I/O capability. The former method has also been known as playing the token game, i.e., keeping track of the locations of tokens in a PN as new transitions (events) occur in the corresponding physical system.

As one can infer, the synthesis of overall system PNs by combining the sub-PNs of their individual resources would yield very large nets, where places refer to the individual states of the resources as opposed to the overall state of the system, as would be the case with Ramadge–Wonham's supervisory control theory (Sec. 15.1.2). Thus, in any token game, one must keep track and examine all pertinent places for token movements. Transitions should be enabled based on firing rules, and such information must be effectively transmitted to individual device controllers. In PC-based control, the receipt of input signals from the manufacturing system, in regard to the actual occurrence of events, and the sending of output signals, in regard to the enablement of events, can only be achieved via multichannel I/O interface cards.

Among the efforts for generating a sequential programming code based on PNs, the work of a group of French academics and industrial participants in the mid 1970s stands out as unique. This programming standard, officially established in 1980, is today known as GRAFCET (graphe de commande étape transition). GRAFCET is a graphical programming tool directly derived from ordinary PNs for implementation on PLCs. The basic elements of GRAFCET are steps (places with capacity 1), transitions, and receptivities (logical conditions that need to be satisfied before a transition can fire). Directed arcs connect transitions and steps. The dynamics of the GRAFCET (net) is achieved by enabling transitions, whose

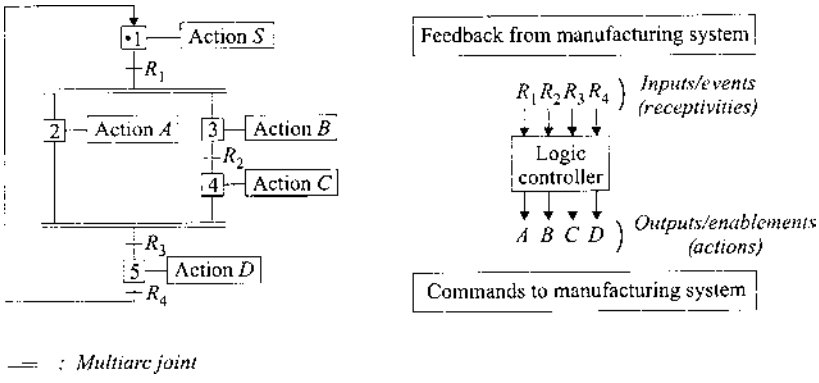


FIGURE 16 A GRAFCET example.

input steps satisfy the firing rules (i.e., have tokens in them), when the associated receptivities are true. A receptivity, R, may be an (external) event or a logic condition, or a combination of both. An example GRAFCET (net) is shown in Fig. 16.

As will be further discussed in subsection 15.3.2, PNs in general and GRAFCET in particular have led to the development of several industrial standards for PLC programming. The primary reason for this close relationship has been the similarities between PLC ladder logic coding and PNs in programming sequential systems via the use of logical expressions (AND, OR, NOR, etc.) that can be easily expressed in graphical form.

15.3 PROGRAMMABLE LOGIC CONTROLLERS

A PLC is a sequential controller that ensures (allows) the occurrence of events in a programmed sequence, through its output unit, based on feedback it receives from the system it is controlling, through its input unit. A control program stored in the memory of the PLC is continuously scanned (run in an endless loop) while examining all the inputs and "energizing" appropriate output ports (Fig. 17).

The first commercial PLC was developed and installed in 1969 at General Motors Hydra-Matic division by Modicon (Gould Electronics). The primary objective from GM's perspective was rapid retooling needed by product model changes. Electromechanical relays used on the factory floor to control the flow of parts prevented such rapid retooling owing to

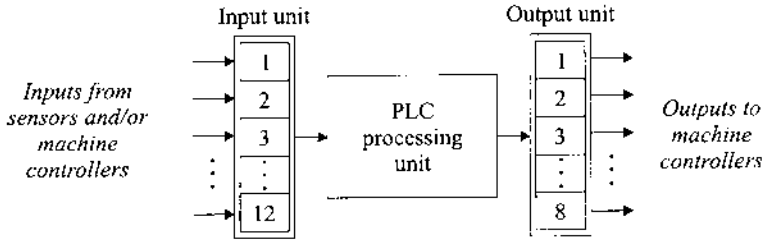


FIGURE 17 A typical PLC structure.

extensive (and expensive) rewiring. Thus GM engineers set the following specifications for the design of a sequential event controller:

Ease of on-the-floor reprogrammability
Modularity and ease of interface to factory devices
Ruggedness and cost effectiveness

All the above specifications were met by the original PLC designs, and their use became very widespread with the introduction of Intel's processors. Today the market for commercial PLCs is several billions of (US) dollars.

PLCs have long been accepted as industrial computers with efficient input/output (I/O) interface capability, and they competed successfully as an alternative to the use of PCs as process controllers. Recent advances in PLC technologies blur the differences even more in favor of PLCs. Today, PLCs, like PCs, can be networked (via Ethernet) to other remote PLCs and PCs, execute multiple programs (using coprocessors), communicate in digital and analog format, with a very large number of machines, and so on. Their modular structure also allows them efficient expandability to handle thousands of inputs and outputs, while in mini-PLC configuration (up to 24-32 I/O ports) they can be purchased for $100 to $150 (USA). PLCs' current primary weakness is their expected programmability using a low-level, device-specific language (i.e., lack of programmability by a high-level language) and difficulty of creating large programs that are verifiable, for example, for deadlocks.

15.3.1 PLC Hardware Structure

A PLC is an industrial computing device that continuously and sequentially checks its input ports to determine the most recent events that have occurred within the system it is controlling, and it activates (or deactivates) its output ports to allow (or disallow) other events to happen within the system (Fig. 17). The core unit of the PLC, as with any other computing

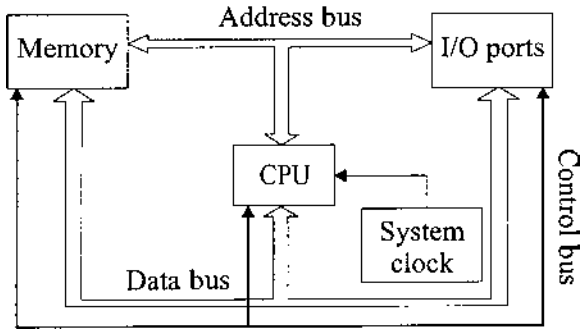


FIGURE 18 PLC communications.

device, is the central processing unit (CPU). The CPU controls all the operations within the PLC based on instructions specified by the user in the form of a program stored in its memory. The CPU communicates with the data- and program-memory modules and I/O units through the various (address, data, control) buses, according to its clock frequency (Fig. 18). Although CPU units used in PCs today can have a clock speed of above 2 GHz, most PLCs still use CPU units that are a few generations old (e.g., Intel's 80486) with clock speeds that are less than 200 MHz.

I/O System

The I/O system of a PLC is its interface (physical connection) to the entity it is controlling. The majority of input signals are received from a variety of sensors/switches and individual device controllers in the form of 5 V (TTL* level) or 24V DC and occasionally 100/120 V or 230/240 V AC. It is expected that all incoming (low- or high-voltage) signals are isolated through optical couplers in order to protect the PLC against voltage spikes. The output signals of the PLC also vary from 5 V (TTL level) DC to 230/240 V AC and are applied in the reverse order of the input signals. Every I/O point (port) is assigned a unique address that is utilized for its monitoring via the user supplied program. For example, Allen-Bradley's Series 5 PLCs denote addresses as I or O: two-digit rack number_one-digit module group number/two-digit port number: I:034/03, I:042/01, O:034/08, O:042/12, and so on.

As mentioned above, PLCs can be configured as single boxes that house all the logic and I/O units in one casing (with minimal variety on I/O

* Transistor–transistor logic (TTL).

signals) or as modular structures that allow users to choose from a variety of I/O modules (analog and discrete signal). The latter configuration is, naturally, flexible and expandable.

Memory Units

As do PCs, PLCs employ a variety of memory devices for the permanent or temporary storage of data and operating instructions. However, as with CPUs, these devices are generally several generations older than their counterparts in PCs in terms of capacity and speed. Most manufacturer-provided information, including the operating system of the PLC, is stored in a read-only memory (ROM) module. All user-provided programs and collected input data are stored in a random-access memory (RAM) module—such CMOS-based memory modules are battery supported and easily erasable and rewritable. Users can also choose to store certain programs and data on erasable programmable read-only memory (EPROM) modules for better protection. EPROMs can be completely erased through external intervention (e.g., using an ultraviolet light input through a window on the memory device) and reprogrammed for future read-only access cases. (Electrically erasable EPROMs are often called EEPROMSs.)

Almost all commercial PLCs provide users with a PC-based interface capability for their programming. Thus PLC programs can be developed on a host PC, stored on its hard drive, and downloaded to the PLC's RAM module when needed.

External Data Communications

Modern PLCs allow users to network their controllers for data communications between multiple PLCs as well as between PLCs and computers or other controllers on a factory floor. PLCs can be placed on local area networks (LANs) utilizing proprietary software/hardware (e.g., Allen-Bradley's Data Highway, Mitsubishi's Melsec-NET, and General Electric's Net Factory LAN) or Ethernet (nonproprietary network protocol and interface developed by Xerox, DEC, and Intel) (Fig. 19).

PLCs can also communicate between themselves and with other devices using serial communication interfaces. RS232 (also known as EIA 232) is the most commonly used serial interface standard, it uses a 25-pin connector—$\pm$12 V signals indicate 0/1. Data transfer rates of up to 25 kilobaud (but typically only 9600 baud) can be achieved over short distances (less than 50 feet, 15 meters). The majority of PLCs also provide users with RS422 serial interface capability. RS422 can yield a transmission rate of up to 10 megabaud over a distance of up to 4,000 feet (1,200 meters). Other standards include the RS485 and the 20 mA current loops.

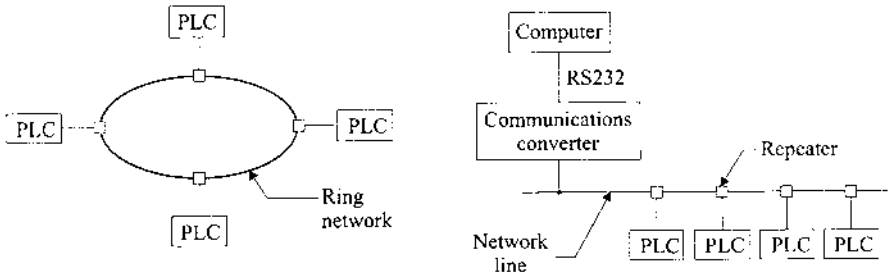


FIGURE 19 Networked PLCs.

15.3.2 PLC Programming

Ladder logic (LL) programming of PLCs is the current industrial standard, though instruction sets vary from one PLC make to another. LL programming is based on the logical representation of output decisions based on binary (0/1) values of the inputs. LL programs can be represented in a ladder diagram form or simply as sets of instructions. A rung typically is a combination of inputs that affect one output, though some commercial systems allow a rung to have multiple outputs.

Common logic gates (e.g., AND, OR, NOR) are utilized in the LL programming of PLCs. Fig. 20 illustrates typical logic symbols used in LL diagrams, while Table 1 lists some instructions used by commercial PLC manufactures.

A series of typical rungs are shown in Fig. 21. The first two correspond to examining multiple inputs and energizing the corresponding outputs, while the third illustrates a multiple output case.

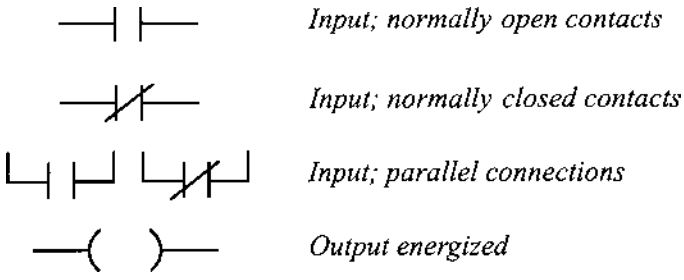


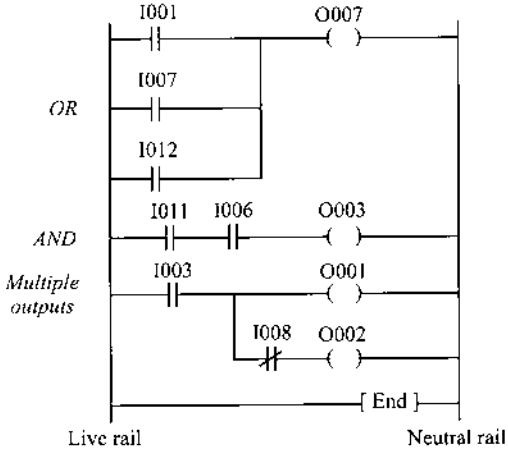
FIGURE 20 Ladder logic symbols.

TABLE 1 Ladder Logic Instructions

Action	Mitsubishi	Omron	Texas Instruments
Start a new rung	LD	LD	STR
Logical AND	AND	AND	AND
Logical OR	OR	OR	OR
Logical NOT	NOT	NOT	I
Output	OUT	OUT	OUT

As discussed above, a LL program loaded to the PLC's RAM module runs in an endless loop. During each scan cycle, the processor sequentially examines (reads) all the inputs and accordingly energizes or deenergizes the corresponding outputs. For example, in the third rung of Fig. 21, the output port with the address O001 is energized only if the PLC detects an "ON" (1) signal at input port address I003, while O002 is energized only if additionally the input port address I008 does not have an "ON" (1) value.

Timer and counter instructions, as they affect the outputs of a PLC, can also be programmed using LL. Timer instructions can be used to delay the activation of the output port, to deenergize it after a certain period of time and so on. Counters can perform similar tasks as timers by counting (up or down) the instances of signals generated to energize the output



Address	*Instruction*	*Data*
0	*LD*	I001
1	*OR*	I007
2	*OR*	I012
3	*OUT*	O007
4	*LD*	I011
5	*AND*	I006
7	*OUT*	O003
8	*LD*	I003
9	*OUT*	O001
10	*AND NOT*	I008
11	*OUT*	O002
12	*END*	

FIGURE 21 Example ladder logic rungs.

port. Simple arithmetic functions (add, subtract, multiply, divide, square root, negate) can also be programmed using LL for data comparison instructions (e.g., equal, greater than or equal). Most PLCs also allow the creation and use of subroutines/procedures (external subprograms) within main programs.

The majority of PLC manufacturers today provide customers with interface software that runs on PCs for the writing of LL programs and their easy downloading to the PLCs. Although they differ in their graphical user interface capability, most software modules are very similar in aiding the programmer for syntax error detection and so on. One must realize, however, that having such an off-line programming tool does not guarantee the correct operation of the manufacturing system, for human errors in programming are normally detected only once the system is running and rarely ahead of time (i.e., off-line).

During the 1990s, two themes of research and development have been pursued by the industrial and academic communities. The former group have made some efforts in allowing users to program PLCs via high-level languages, while the academic community primarily concentrated on the automatic translation of supervisory controllers, developed via PN or automata tools, into LL programs.

The sequential function chart (SFC), made available to users by a number of major PLC manufacturers, is the most commonly used high-level language alternative to LL. This graphical sequencing language is defined within the IEC* 1131–3 standard and is derived from IEC 848 GRAFCET—a technique based on PN modeling of DESs.

The first revision of the IEC 1131-3 standard was published in 1993 for PLC programming that specifies the syntax, semantics, and display for several languages: LL, SFC, function block diagram (FBD), structured text (ST), and instruction list (IL). A PLC program can be built with any of these languages. Typically, such a program would consist of a network of functions and function blocks that are capable of exchanging data.

In particular, the SFC consists of steps linked with action blocks and transitions. Each step represents a particular state of the system that is being controlled. A transition causes the system to change states (steps). Steps are linked to action blocks that perform certain control actions. Steps and transitions can be arranged in series or in parallel (Fig. 22). Parallelism and other features of SFC allow the scanning of only the active states (steps) instead of sequential scanning of the entire logic, as is the case with LL programming.

*International Electrotechnical Commission.

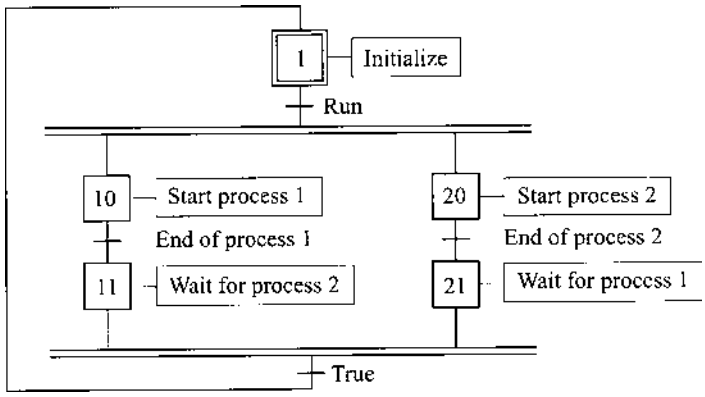


FIGURE 22 Sequential function chart programming.

REVIEW QUESTIONS

1. Define flexible manufacturing workcell (FMC) and discuss its autonomous supervisory control.
2. What is discrete event system (DES) control and how does it differ from time-driven control? Discuss *determinism* versus *nondeterminism* in the state transition of DES systems.
3. What is automata theory?
4. Define (artificial) language in the context of automata theory.
5. What is a finite-state automaton?
6. Define Moore (finite-state) machines versus Mealy machines.
7. What is the primary feature of Ramadge–Wonham (R–W) automata theory that makes it suitable to the supervisory control of manufacturing systems?
8. How do you obtain a R–W supervisor for an FMC?
9. What is a Petri net (PN) and how can it be utilized for the DES modeling of FMCs?
10. How is a transition enabled in PNs? What is the consequence of a fired transition?
11. Describe the following properties of PNs in terms of the control of FMCs: reachability, boundedness, and liveness. Furthermore, define a deadlock state.
12. How do you synthesize a PN supervisor for an FMC and how would you implement it?
13. What is a programmable logic controller (PLC) and how would it control an FMC?

14. Discuss the primary elements of a PLC and compare its architecture to that of a personal computer (PC). Furthermore, discuss PLC communications over an Ethernet-based network.
15. Discuss the programming of PLCs using ladder logic and other higher level programming languages.
16. How could one use automata or Petri net theory in the automatic generation of PLC programs versus their compilation manually by an experienced operator?

DISCUSSION QUESTIONS

1. Computers and other information management technologies have been commonly accepted as facilitators for the integration of various manufacturing activities. Define/discuss integrated manufacturing in the modern manufacturing enterprise and address the role of computers in this respect. Furthermore, discuss the use of intranets and extranets as they pertain to the linking of suppliers, manufacturers, and customers.
2. Manufacturing flexibility can be achieved on three levels: operational flexibility, tactical flexibility, and strategic flexibility. Discuss operational flexibility. Is automation a necessary or a desirable tool in achieving this level of flexibility?
3. The factory of the future will be a totally networked enterprise. Information management in this enterprise will be a complex issue. In regards to planning, monitoring, and control, discuss the level of detail of information that the controllers (humans or computers) would have to deal with in such environments. For example, some argue that in a hierarchical information management environment, activities are more of the planning type at the higher levels of the hierarchy and of the control type at the lower levels. It has also been argued that the level of details significantly decreases as you ascend the enterprise ladder.
4. Job shops that produce one-of-a-kind products have been considered the most difficult environments to automate, where a product can be manufactured within a few minutes or may require several days of fabrication. Discuss the role of computers in facilitating the transformation of such manual, skilled-labor-dependent environments to intensive automation-based environments.
5. Several fabrication/assembly machines can be physically or virtually brought together to yield a manufacturing workcell for the production of a family of parts. Discuss the advantages of adopting a cellular manufacturing strategy in contrast to having a departmentalized

strategy, i.e., having a turning department, a milling department, a grinding department, and so on. Among others, an important issue to consider is the supervisory control of such workcells.

6. In the factory of the future, it is envisioned that production and assembly workcells will be frequently reconfigured based on the latest manufacturing objectives without the actual physical relocation of their machines/resources. Discuss the supervisory control options that should be available to the users of such workcells, whose boundaries may exist only in the (computer's) virtual space.
7. The primary advantage of PLCs has been their robust modular design that allows large numbers of inputs and outputs to be directly connected to the controller. However, this advantage has also been discouraging users from implementing PLCs widely in flexible manufacturing workcells, who frequently opt for networkable industrial PCs. Compare PCs versus PLCs for the control of manufacturing systems that are configured for mass production versus those that are configured (and occasionally reconfigured) for small batch production.
8. In the factory of the future, it is expected that manufacturing workcells will operate autonomously under the control of an overall computer-based supervisor; all devices, including fixtures and jigs, will be equipped with their own (device) controllers that can effectively communicate with the workcell supervisor. Discuss difficulties in the integration and utilization of a large number of sensors and other communication components in remotely controlled (via two-way communications) FMCs.
9. Computer-based controllers (PC or PLC based) have been advocated as better supervisors than human controllers based on a number of factors including reaction time. Besides manufacturing environments, such controllers have been widely used in the health care, nuclear, aviation, security, and military industries. Discuss the advantages/disadvantages of using computer-based controllers in today's manufacturing industries. You may also extrapolate and discuss the role of such controllers in the factories of the future.
10. Discrete event system (DES)-based autonomous supervisory control of flexible manufacturing workcells require the planning of a new (software-based) controller, the reprogramming of the control hardware using this controller, and, frequently, rewiring of the communication lines between the control hardware and the input/output devices in the workcell every time the workcell is reconfigured to manufacture different products. All these three problematic issues and others discourage manufacturers from implementing computer-

based autonomous DES controllers. The manufacturers mostly continue to rely on limited feedback received from the workcell that is monitored by a human supervisor. Discuss remedies to these problems that would allow widespread use of autonomous computer-based supervisors.

11. Manufacturing systems, supported by computers, can be classified as manual versus automated and flexible (reconfigurable, reprogrammable, etc.) versus rigid. Discuss these classifications and elaborate on the intersections of their domains (e.g., manual and flexible, etc.). Note that each classification may have sublevels and subclassifications (i.e., different ''levels of gray'').

12. In the factory of the future, no unexpected machine breakdowns will be experienced! Such an environment, however, can only be achieved if a preventive maintenance program is implemented, in which all machines and tools are modeled (mathematically and/or using heuristics). These models would allow manufacturers to schedule maintenance operations as needed. Discuss the feasibility of implementing factory-wide preventive maintenance programs in the absence of our ability to model completely all existing physical phenomena and furthermore in the lack of a large variety of sensors that can monitor the states of these machines and provide timely feedback to such models.

13. Human factors (HF) studies encompass a range of issues spanning from ergonomics to human–machine (including human–software) interfaces. Discuss the role of HF in the autonomous factory of the future, where the impact of human operators is significantly diminished and emphasis is switched from operating machines to supervision, planning, and maintenance.

BIBLIOGRAPHY

AFCET (Association Française pour la Cybernétique Economique et Technique). (1977). Normalisation de la représentation du cahier des charges d'un automatisme logique. Journal Automatique et Informatique Industrielles. pp. 61–62, November–December.

Baker, A. D., Johnson, T. L., Kerpelman, D. I., Sutherland, H. A. (1987). GRAFCET and SFC as factory automation standards: advantages and limitations. Minneapolis, MN: Proceedings of the American Control Conference, pp. 1725–1730.

Batten, George (1994). *Programmable Controllers: Hardware, Software, and Applications*. New York: McGraw-Hill.

Boel, R., Stremersch G., eds. (2000). *Discrete Event Systems: Analysis and Control*. Boston, MA: Kluwer.
Booth, Taylor L. (1968). *Sequential Machines and Automata Theory*. New York: John Wiley.
Cao, X.-R., Ho, Y.-C. (June 1990). Models of discrete event dynamic systems. *IEEE Control Systems Magazine* 1(4):69–76.
Carrow, Robert S. (1998). *Soft Logic: A Guide to Using a PC as a Programmable Logic Controller*. New York: McGraw-Hill.
Cassandras, Christos G. (1993). *Discrete Event Systems: Modeling and Performance Analysis*. Homewood, IL: Aksen Associates.
Chomsky, N. (1956). Three models for the description of language. *IRE Transactions of Information Theory* 2(3):113–124.
Costanzo, Marco. (1997). *Programmable Logic Controllers: The Industrial Computer*. New York: Arnold.
Crispin, Alan J. (1997). *Programmable Logic Controllers and Their Engineering Applications*. New York: McGraw-Hill.
David, René, Alla, Hassane. (1992). *Petri Nets and GRAFCET: Tools for Modeling Discrete Event Systems*. New York: Prentice Hall.
Desrochers, A. Alan, Al-Jaar, Robert Y. (1994). *Applications of Petri Nets in Manufacturing Systems: Modeling, Control, and Performance Analysis*. Piscataway, NJ: IEEE Press.
Dicesare, F., et al. (1993). *Practice of Petri Nets in Manufacturing*. New York: Chapman and Hall.
Filer, Robert, Leinomen, George. (1992). *Programmable Controllers and Designing Sequential Logic*. New York: Saunders.
Ho, Yu-Chi, ed. (1992). *Discrete Event Dynamic Systems: Analyzing Complexity and Performance in the Modern World*. New York: Institute of Electrical and Electronics Engineers.
Hopcroft, John E. (1979). *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley.
Hughes, Thomas A. (2001). *Programmable Controllers. Instrument Society of America*. NC: Research Triangle Park.
Johnson, David G. (1987). *Programmable Controllers for Factory Automation*. New York: Marcel Dekker.
Kain, Richard Y. (1972). *Automata Theory: Machines and Languages*. New York: McGraw-Hill.
Kohavi, Zvi (1970). *Switching and Finite Automata Theory*. New York: McGraw-Hill.
Lauzon, Stéphane C. (1995). An Implementation Methodology for the Supervisory Control of Manufacturing Workcells. M.A.Sc. thesis, Department of Mechanical Engineering, University of Toronto, Toronto, Canada.
Lauzon, S. C., Mills, J. K., Benhabib, B. (1997). An implementation methodology for the supervisory controller for flexible manufacturing workcells. *SME J. of Manufacturing Systems* 16(2):91–101.
Lewis, R. W. (1998). *Programming Industrial Control Systems Using IEC 1131-3*. Stevenage, UK: IEE Books.

Mealy, G. H. (1955). A method for synthesizing sequential circuits. *Bell System Technical Journal* 34(5):1045–1079.
Meduna, Alexander (2000). *Automata and Languages: Theory and Applications*. New York: Springer Verlag.
Moody, John O., Antsaklis, Panos J. (1998). *Supervisory Control of Discrete Event Systems Using Petri Nets*. Boston: Kluwer.
Moore, E. F. (1956). Gedanken-experiments on sequential machines. *Automata Studies, Annals of Mathematical Studies* (No. 34, pp. 129–153). Reading, MA: Princeton University Press.
Moore, Edward F., ed. (1964). *Sequential Machines: Selected Papers*. Reading, MA: Addison-Wesley.
Murata, T. (Apr. 1989). Petri nets: properties, analysis and applications. *Proceedings of the IEEE* 77(4):541–580.
Parr, E. A. (1999). *Programmable Controllers: An Engineer's Guide*. Boston: Newnes.
Petri, Carl Adam (1962). Kommunication mit Automaten (Communication with Automata). Schriften des IIM Nr. 3, Bonn: Inst. fur Instrumentelle Mathematik, Ph.D. diss.
Petruzella, Frank D. (1998). *Programmable Logic Controllers*. New York: Glenco.
Proth, Jean-Marie, Xie, Xiaolan (1996). *Petri Nets: A Tool for Design and Management of Manufacturing Systems*. New York: John Wiley.
Ramadge, P. J. G., Wonham, W. M. (Jan. 1989). Control of discrete event systems. *Proceedings of the IEEE* 77(1):81–98.
Ramirez-Serrano, Alejandro (2000). Extended Moore Automata for the Supervisory Control of Virtual Manufacturing Workcells. Ph.D. diss., Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Canada.
Ramirez, A., Zhu, S. C., Benhabib, B. (Aug. 2000). Moore automata for flexible routing and flow control in manufacturing workcells. *J. of Autonomous Robots* 9(1):59–69.
Ramirez, A., Benhabib, B. (Oct. 2000). Supervisory control of multi-workcell manufacturing systems with shared resources. *IEEE Transactions on Systems, Man and Cybernetics* 30(5):668–683.
Ramirez, A., Sriskandarajah, C., Benhabib, B. (Dec. 2000). Discrete-event-system modeling and control synthesis using extended Moore automata. *IEEE Transactions on Robotics and Automation* 16(6):807–823.
Rathmill, K. (1988). *Control and Programming in Advanced Manufacturing*. Bedford, UK: IFS.
Silva, M., Velilla, S. (1982). Programmable logic controllers and Petri nets: a comparative study. *Proceedings of the 3rd IFAC/IFIP Symposium on Software for Computer Control, Madrid, Spain, 29–34*.
Sobh, T. M., Benhabib, B. (June 1997). Discrete-event and hybrid systems in robotics and automation: an overview. *IEEE Robotics and Automation Magazine* 4(2):16–19.
Stenerson, Jon (1999). *Fundamentals of Programmable Logic Controllers, Sensors, and Communications*. Upper Saddle River, NJ: Prentice Hall.
Tzafestas, Spyros G. (1997). *Computer-Assisted Management and Control of Manufacturing Systems*. New York: Springer Verlag.

Turing, A. M. (1936–1937). On computable numbers with an application to the entscheidungs problems. *Proceedings of the London Mathematical Society* 43: 230–265 (correction ibid, Vol. 42, pp 544–546).
Viswanadham, N., Narahari, Y. (1992). *Performance Modeling of Automated Manufacturing Systems*. Englewood Cliffs, NJ: Prentice Hall.
Warnock, Ian G. (1988). *Programmable Controllers: Operation and Application*. Englewood Cliffs, NJ: Prentice-Hall.
Williams, Robert A. (1993). A DES-Based Hybrid Supervisory Control System for Manufacturing Systems. M.A.Sc. thesis, Department of Mechanical Engineering, University of Toronto, Toronto, Canada.
Williams, R. A., Benhabib, B., Smith, K. C. (1996). A DES-based supervisory control system for manufacturing systems. *SME J. of Manufacturing Systems* 15(2): 71–83.
Wonham, W. Murray (1997). *Notes on Control of Discrete-Event Systems*. University of Toronto: Department of Electrical and Computer Engineering.
Zhou, MengChu, DiCesare, Frank (1993). *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*. Boston: Kluwer.
Zhou, MengChu, ed. (1995). *Petri Nets in Flexible and Agile Automation*. Boston: Kluwer.
Zhou, MengChu, Venkatesh, Kurapati (1999). *Modeling, Simulation, and Control of Flexible Manufacturing Systems. A Petri Net Approach*. River Edge, NJ: World Scientific.