# Grafcet: A Powerful Tool for Specification of Logic Controllers

René David

*Abstract*—**Basically, a logic controller is a discrete-event system whose purpose is to control the behavior of a process which is itself (seen by the controller as) a discrete-event system, taking into account the state of this process and other information coming from an operator or from other systems.**

**In the early 1970's, the need to describe increasingly complex logic controllers was becoming evident, since the programmable logic controllers (PLC) were becoming more powerful and more extensively used. This paper shows how Grafcet can be used for this purpose. Grafcet is a tool, drawing its inspiration from Petri nets (a general purpose mathematical tool allowing various discrete-event systems to be described), whose aim is the specification of logic controllers. It is the basis of the Sequential Function Chart (SFC), an International Standard in 1987.**

**This paper, introductory in nature, explains the Grafcet model using the classical-state table model and presents a method for interpreting grafcets.**

## I. Introduction

A discrete-event dynamic system (DEDS), or simply discrete-event system, is a system with a discrete state space and a state evolution which is determined by events. For example, the state of a buffer in a manufacturing system is the number of parts in this buffer. This state changes when the event "a part is put into the buffer" or "a part is taken out of the buffer" occurs. A dynamic physical system may be seen either as a continuous variable dynamic system or as a discrete-event system, depending on the aim of the model. For example, if the state of a tank is the liquid level in this tank, the corresponding model is a continuous variable system. This tank may be modeled as a discrete-event system, however, if we consider it may have only three states: empty, between empty and full, and full.

Basically, a logic controller is a discrete event system whose purpose is to control the behavior of a process which is itself (seen as) a discrete-event system, taking into account the state of this process and other information coming from an operator or from other systems. This is illustrated in Fig. 1 where $x$, $y$, $Z$, and $W$ are Boolean variables or vectors.

Logic control is relevant to processes whose measurements and actions are of a logic type. That means that these measurements and actions can only take two values (for example, a contact can be open or closed, a temperature has reached some threshold or not). The measurement is input, and the action is output of a logic controller. The goal of a logic controller is to react to input changes by modifying the output state as a function of control rules. The designer must specify these control rules.

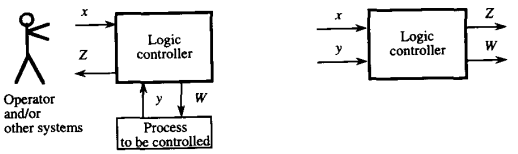



Fig. 1. The logic controller and its environment.

Before the introduction of the first programmable logic controller (PLC) in 1969 [13], logic control was implemented by hard-wired logic. Electromagnetic relays, pneumatic relays, and electronic gates and flip-flops were used. With these technologies, complex problems are difficult to handle, and a logic controller is difficult to modify. The first PLC's worked with specialized processors able to carry out logic operations. Then, general-purpose microprocessors were used in PLC's when microprocessors became cheaper and more powerful. PLC's remain different from general-purpose computers, however, since their hardware is designed for the industrial environment. PLC's are used extensively in many kinds of industries: manufacturing, metallurgy, chemical, paper, food, transportation, handling industries, etc. Today, a PLC can handle several hundreds of inputs–outputs, several PLC's can be linked by a communication network, and some PLC's can carry out additional functions such as calculation and regulation. Many of them are integrated in the production machines themselves, in such a way that they are transparent to the machines' users.

As long as the logic controllers were small systems, the classical models were sufficient to specify the logic control rules. There were two kinds of models. The first kind corresponds to the usual models of asynchronous sequential circuits, state diagrams, or state tables [25], which represent automata, i.e., hardware and software independent specifications. Note that the automata are asynchronous because continuous-time discrete-event systems are considered. The second kind corresponds to the representation as a possible implementation. The most famous is relay ladder logic (RLL) which is a graphical representation based on an analogy to physical relay systems [19]. In the early 1970's, the need to describe increasingly complex logic controllers, the implementation of which would be simplified by recent and future development of microprocessors, was becoming evident. Roughly speaking, it was required to represent large logic controllers with many

Manuscript received August 7, 1994. Recommended by Associate Editor, E. O. King.

The author is with the Laboratoire d'Automatique de Grenoble (Associated with CNRS), Institut National Polytechnique de Grenoble, B.P. 46, 38402 Saint-Martin-d'Hères, France.

IEEE Log Number 9413134.

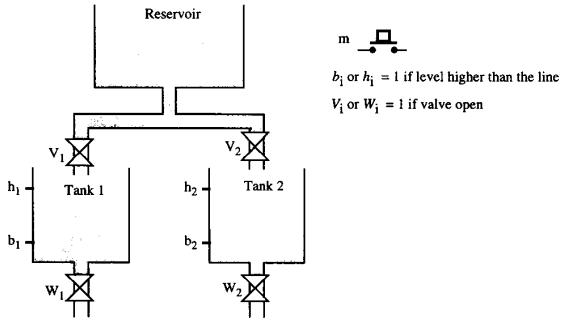



Fig. 2. Example: tank filling.

inputs and many outputs. More precisely the needs were the following:

1) Describe the sequence of states of a discrete-event system which may contain a very large number of states.
2) Take into account the concurrency (for both simplicity and easy understanding), since some subsystems may be partially independent.
3) Generally, given a state of the system, only a few inputs affect the state, and only a few outputs may change. Then, describe only the behavior corresponding to these input changes.
4) Have a clear understanding of the input–output behavior of a logic controller, i.e., what is the control applied to the process to be controlled (output of the PLC) as a function of a change of the process or its environnment (input of the PLC).

The Grafcet[1] model came from this need. It draws inspiration from the Petri nets which were studied in the early sixties by Petri [21]. We assume the reader is familiar with Petri nets [7], [9], [18], [20]. Petri nets are a graphical tool which allow modeling of sequential behavior and concurrency (needs 1 and 2). A place in a Petri net (PN) corresponds to a component of the state. It follows that when a single component changes (marking of a place), the state changes. Then, one can "see" what part of the system changes, and furthermore, a reasonable number of places may provide the representation of a large number of states (the number of markings may increase exponentially with the number of places). In addition, the other needs are satisfied in the Grafcet model.

Let us now present various models with the help of an example. The following notations are used. The complement of the Boolean variable $a$ will be written as $a'$: that means that $a' = 1$ exactly when $a = 0$. The Boolean product (AND function) is represented as a product $(a \cdot b = ab = (a)(b))$. The Boolean sum (OR function) is represented as a sum $(+)$.

*Tank Filling:* The device in question is represented in Fig. 2. Both tanks are used in a similar way. A roman symbol represents a device (sensor or actuator) and an italic symbol the corresponding Boolean variable. Tank 1 is empty when the level is less than $b_1$, i.e., $b_1 = 0$, and is full when the level is greater than $h_1$, i.e., $h_1 = 1$. At the initial state, both tanks are empty. If push button m is pressed, both tanks are filled by opening valves $V_1$ ($V_1 = 1$ means that valve $V_1$ is open) and $V_2$. When a tank is full, e.g., tank 1, filling stops (by closing valve $V_1$) and its contents start to be used (by opening valve $W_1$). When tank 1 is empty, valve $W_1$ is closed. Filling may only start up again when both tanks are empty and if the button m is pressed. Let us now present this specification using four models.

1) State table [ 3(a)]. In this table, a column is associated with every input state and a row with every internal state. An internal state and an input state define a total state. The initial state (noted by a star) is state 1 which is stable for the input state $mh_1h_2b_1b_2 = 00000$ (represented by circling the state number). All the outputs are 0 in this state. If $m$ assumes the value 1, we reach the column where the unstable state 2 (not circled) means that the new internal state 2 will be reached. In this state, the outputs $V_1$ and $V_2$ take the value 1. And so on. A dash indicates a total state which cannot be reached.
   This model is extremely detailed. In Fig. 3(a) we have a complete specification showing all of the total states, even though some cannot be reached from any internal state. For larger systems the state tables become almost empty (i.e., many total states are not reachable) since generally, only a few inputs can change when a state has been reached. This model is very useful for an accurate analysis of a small part of the system. It becomes practically unusable, however, for specifying a large system. For 20 input variables there are more than one million columns!
2) State diagram [Fig. 3(b)]. In this model, a node is associated with every total state. For example, node 2 corresponds to the eight stable total states of the second row in Fig. 3(a). Each arc is labeled by a Boolean variable or function. The corresponding transition is performed as soon as this function assumes the value 1. The initial state is shown by a short incoming arrow. In this model, the information is more concise. We represent internal states (not total states). The input changes which do not cause a state change are not necessarily shown by self-loops. For example, the inputs $m$, $h_1$, and $b_2$ may change in state 3, yet since their changes do not cause a state change, this information is omitted in Fig. 3(b). The concurrency, however, cannot be represented by this model. This is its main weakness. In our example, what happens to tank 1 is independent of what happens to tank 2, as long as they are not empty. This model representing a state for the whole system may become large and difficult to understand when the system is made of several parts, completely or partially independent.
3) RLL [Fig. 3(c)]. In this model, a rung corresponding to a Boolean equation is associated with every internal and output variable. It may be convenient to use the output variables as internal variables. The dependent variable of the equation is represented by a circle. The rung inputs are represented by pairs of vertical parallel bars

[1] We shall write Grafcet (with a capital G) when speaking of the tool in general, and grafcet (with a small g) when referring to a particular logic controller model.

representing relays which close when the corresponding input is active. A diagonal line between the bars means that the complemented value of the variable is used. The AND and the OR functions are constructed by placing variables in series and in parallel, respectively. For example the first rung in Fig. 3(c) corresponds to the equation $q_1 = b'_1 \cdot b'_2$. Then $q_1 = 1$ when both tanks are empty. The second rung corresponds to $V_1 = (q_1 \cdot m + V_1)h'_1$. That means that $V_1$ assumes the value 1 as soon as $q_1 \cdot m = 1$ and keeps this value (because of the memorization by the variable $V_1$ in parallel with $q_1 \cdot m$) as long as $h_1 = 0$ (i.e., as long as tank 1 is not full).

In this model, the concurrency can be exploited (for example one can have $V_1 = V_2 = 1$) even if it is not graphically represented. But the sequential nature of the behavior is not apparent. For example, to know what can happen from the state where $V_1 = V_2 = 1$ and $q_1 = W_1 = W_2 = 0$, a detailed analysis is necessary. This is the main weakness of this representation.

4) Grafcet [Fig. 3(d)]. This model will be explained accurately in Section III. The squares labeled 1 to 6 are steps, i.e., components of states. At initial time, the steps in the set {1, 4} are active. Then transition (1) which follows these steps can be fired as soon as the Boolean variable $m$, associated with (1), has the value 1. After this firing, steps 2 and 5 are active. When step 2 is active, the output $V_1 = 1$ (represented by $V_1$ written in a rectangle associated with step 2). When step 2 is active, transition (2) can be fired if $h_1 = 1$. And so on.

The concurrency is explicitly represented in this model. Steps 1, 2, and 3 correspond to the states of tank 1 (empty, during filling, and during emptying, respectively) and steps 4, 5, and 6 correspond to the states of tank 2. The sequence of states (or active steps) and Boolean conditions leading from one to another are quite apparent (need 1). When step 2 is active, only the input $h_1$ can change the activity of steps 2 and 3, independently of the other steps (need 2) and of the other inputs (need 3). Need 4 is also satisfied and will be explained for more complicated cases in this paper.

The Grafcet model was defined by a French group. Half of the 24 signatories of the final report, dated August 1977 [1], are academics (including the author), and half are from industry.

The Grafcet model has been briefly defined in [1] and then detailed in [2]. A number of concepts and formalisms which enable the original definition to be stated precisely and formalized have been introduced in [6], [7], and [17], to present this model in a manner which is both consistent with the original definition and strict from a theoretical point of view. First a French standard, in 1987 it became an international one [11]. The National Committees of the following countries declared their explicit approval of the publication: Australia, Belgium, Canada, Czechoslovakia, Denmark, Egypt, Finland, France, Germany, Italy, Japan, Netherlands, Norway, Portugal, Spain, Sweden, Switzerland, Turkey, United Kingdom, and Yugoslavia. Basically, a grafcet behaves like a Petri net with just a few differences [5], [17], but above all it proposes a single interpretation of inputs–outputs. This is a specification

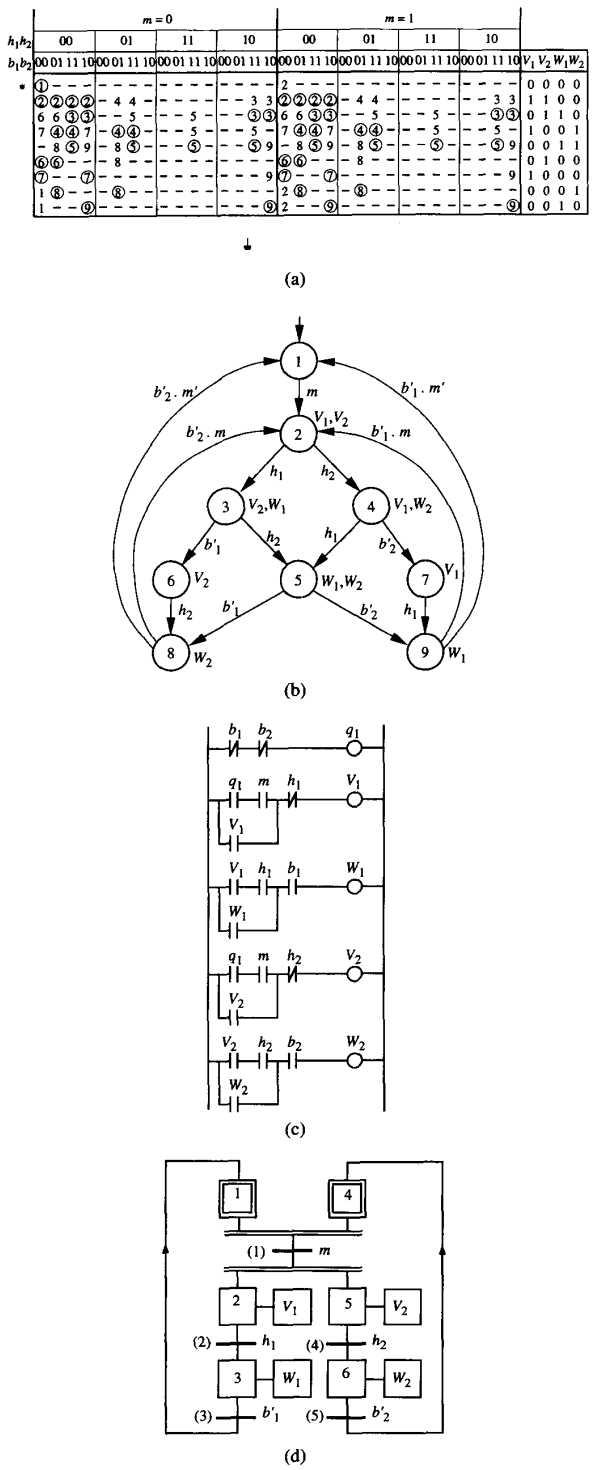



Fig. 3. Specification of tank filling. (a) State table. (b) State diagram. (c) Relay ladder logic. (d) Grafcet.

model which only describes the function to be performed, i.e., a sequential machine in the mathematical sense, free from all hardware and software. The present article is an introductory paper intended for academics and control engineers who want to use formal models for specification of logic controllers. The international standard Sequential Function Chart (SFC) introduces a few modifications and is presented in a way stressing the practical side. This point will be commented on in Section VII.

The paper is organized as follows. Section II presents the concept of condition and the concept of event. The basic notions used in the Grafcet and the interpretation algorithm are presented in Section III and Section IV, respectively. The macrosteps and macroactions, which are abbreviations of the basic Grafcet, are presented in Section V. Section VI compares the Grafcet with other models, and Section VII concludes the paper.

## II. Conditions and Events

### A. General Presentation

The behavior of a logic controller may depend on two kinds of information coming from its environment (process to be controlled or operator or other system): conditions and events. As we shall see later, the state of a logic controller may change if a condition is true (a condition is expressed by a Boolean variable) when an event occurs. These notions have been clearly defined by M. Moalla [17], and they form a basis for Grafcet interpretation [6], [7]. Similar notions are presented in [23].

The first kind of information relates to the state of the environment of the logic controller. For example "the number of parts in the buffer is greater than or equal to 10," or "the level in the tank is high." This kind of information relates to the state of the environment.

*Observation 1:* The state of a discrete-event system (assumed finite state) can always be defined by Boolean values.

This is illustrated with the example in Fig. 4. The level in the tank [Fig. 4(a)] is a continuous variable. If this tank is considered to have three states (low, middle, and high), its state can be modeled as in Fig. 4(b). Obviously, three discrete states can be encoded with two Boolean variables. One of the possible solutions is presented in Fig. 4(c): the Boolean variable $x_1 = 0$ if the level is low, and $x_1 = 1$ otherwise, and the Boolean variable $x_2 = 0$ if the level is low or middle, and $x_2 = 1$ otherwise.

A predicate is a proposition which may be either true or false and which then can be modeled by a Boolean variable. For example, let $x_3$ denote the Boolean variable corresponding to the predicate "the number of parts in the buffer is greater than or equal to 10," i.e., $x_3 = 1$ if this predicate is true. If we assume that the number of parts in the buffer is encoded as a four-digit binary number $y_3\ y_2\ y_1\ y_0$ (maximum number of parts = 15), then $x_3 = y_3(y_1 + y_2)$.

The second kind of information relates to a change in the state of the environment. Such a change is called an event. An event has no duration, whereas the value of a Boolean variable lasts some time. For example, the event "the level in the tank changes from low to middle" occurs at times $t_1$ and $t_2$ illustrated in Fig. 4(d). This event is denoted $\uparrow x_1$ since it corresponds to the rising edge of the Boolean value $x_1$.

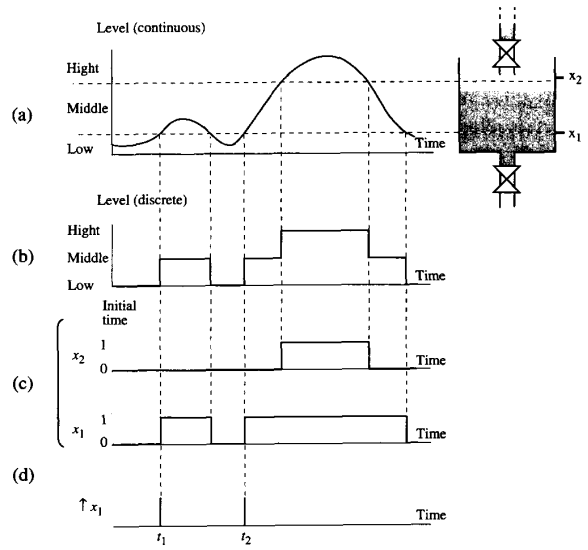



Fig. 4. Illustration of: (a) Continuous state. (b) Discrete state. (c) Boolean encoding of discrete states. (d) Occurrences of an event.

*Observation 2:* An event can always be defined as a rising (or falling) edge of a Boolean value (variable or function).

This is a direct consequence of Observation 1, since an event is a change of discrete variable state. It follows that all the inputs of a logic controller may be Boolean variables, since the events can be defined from their changes.

To have a clear understanding of the relations between conditions and events, let us give some formal definitions and properties which were introduced in [6] and [7].

### B. Algebra of Events

The following definition is illustrated in Fig. 5(a).

*Definition 1:* Let $f(a_1, a_2, \cdots, a_m)$ be a Boolean function whose value is defined from the initial time 0, and such that, for $t_1 < t_2 < t_3 < t_4 \cdots < t_{n-1} < t_n < t_{n+1} \cdots$: 1) $f = 0$ in the time intervals $[0, t_1), [t_2, t_3), \cdots, [t_{n-1}, t_n), \cdots$; 2) $f = 1$ in the time intervals $[t_1, t_2), [t_3, t_4), \cdots, [t_n, t_{n+1}), \cdots$.

If $0 < t_1$, the event $\uparrow f = \uparrow f(a_1, a_2, \cdots, a_m)$ occurs at times $t_1, t_3, \cdots, t_n, \cdots$ and the event $\uparrow f' (= \downarrow f)$ occurs at times $t_2, t_4, \cdots, t_{n-1}, \cdots$.

If $0 = t_1$, i.e., if the initial value of $f$ is 1, the event $\uparrow f = \uparrow f(a_1, a_2, \cdots, a_m)$ occurs at times $t_3, \cdots, t_n, \cdots$ and the event $\uparrow f' (= \downarrow f)$ occurs at times $t_2, t_4, \cdots, t_{n-1}, \cdots$.

From now on, $a, b \cdots$ will denote either a Boolean variable or a Boolean function.

*Notation 1:* The symbol $\uparrow$ takes precedence over the logic symbols $\cdot$ and $+$. That is to say that $\uparrow a \cdot b = (\uparrow a) \cdot b$ and $\uparrow a + b = (\uparrow a) + b$.

Definitions 2-a), b), and c) are illustrated in Fig. 5(b), (c), and (d), respectively.

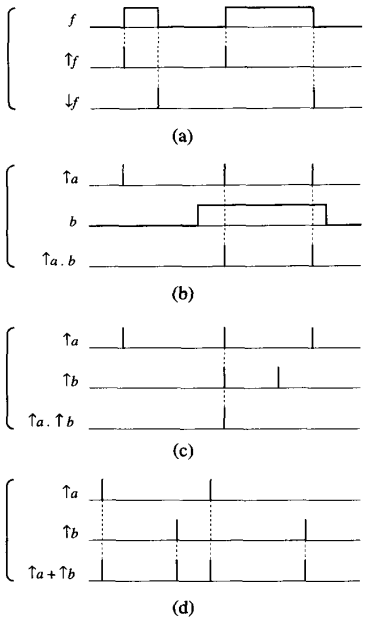



Fig. 5. Illustration of Definitions 1 and 2.

*Definition 2:*

a) $\uparrow a \cdot b$ is an event which occurs at the same time as $\uparrow a$, each time that $b = 1$ at the corresponding time.
b) $\uparrow a \cdot \uparrow b$ is an event which occurs at the times when $\uparrow a$ and $\uparrow b$ occur simultaneously (which is only possible if $a$ and $b$ are not independent, as we shall specify).
c) $\uparrow a + \uparrow b$ is an event which occurs each time that either $\uparrow a$ or $\uparrow b$ occurs.
d) Let $S$ be a system whose behavior depends on the set of events $E$. Let $E(t)$ the event signal associated with $S$ at time $t$: $E(t) \in E \cup \{\varepsilon\}$ where $\varepsilon$ is the absence of an event in $E$ at time $t$.

*Notation 2:*

a) Let $E_1$ and $E_2$ be two events. We write $E_1 = E_2$, if $E_1$ and $E_2$ are equivalent, i.e., if they always occur at the same time.
b) We write $E_1 = 0$, if $E_1$ is an event which never occurs.
c) We denote $e = \sum_{E_i \in E} E_i + \varepsilon$. We call $e$ the "always occurrent" event (it will be used in Section III-B-3).

*Definition 3:* Two events $E_1$ and $E_2$ are independent if there is no event $E_i$ such that $E_1 = X + A \cdot E_i$, $E_2 = Y + B \cdot E_i$, where $X$ and $Y$ are events, $A$ and $B$ are Boolean functions or events such that $X + Y + A \cdot B \cdot E_i > X + Y$.

For example $E_1 = \uparrow f$ and $E_2 = \uparrow g$ may not be independent if both depend on the same Boolean variable $a$, e.g., $f = a \cdot b$ and $g = a$. In this example, the reader may verify (using Property 1-4) below) that $E_i = \uparrow a$, $X = \uparrow b \cdot a'$, $A = b'$, $Y = 0$ and $B = 1$.

*Hypothesis 1:* Two independent events never occur simultaneously. That is to say that $\uparrow a \cdot \uparrow b = 0$, if $a$ and $b$ are two independent variables.

Hypothesis 1 is based on the fact that an event has no duration and that continuous-time models are considered. Then the probability that two independent events occur at the same time is zero.

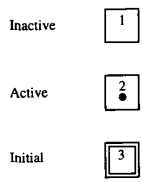



Fig. 6. Representation of a step.

*Remark 1:* It appears that we have the following features:

1) The product of two events is an event [Definition 2-b)].
2) The sum of two events is an event [Definition 2-c)].
3) The product of an event and a condition is an event [Definition 2-a)].

Now, what about the sum of an event and a condition? This will be commented on in Section III-B-3.

*Property 1:*

1) $\uparrow a = \downarrow a'$.
2) $\uparrow a \cdot a = \uparrow a, \uparrow a \cdot a' = 0, \downarrow a \cdot a' = \downarrow a, \downarrow a \cdot a = 0$.
3) $\uparrow a \cdot \uparrow a = \uparrow a, \uparrow a \cdot \uparrow a' = 0, \uparrow a \cdot e = \uparrow a$.
4) If $a$ and $b$ are two independent variables, then

$$\uparrow(a \cdot b) = \uparrow a \cdot b + \uparrow b \cdot a, \qquad \uparrow(a + b) = \uparrow a \cdot b' + \uparrow b \cdot a'.$$

5) If $a$, $b$, and $c$ are three independent variables, then

$$\uparrow(a \cdot b) \cdot \uparrow(a \cdot c) = \uparrow a \cdot b \cdot c.$$

Properties 1-1), 1-2), and 1-3) are obvious from Definitions 1 and 2. The proofs of Properties 1-4) and 1-5), based on Hypothesis 1, can be found in [7]. Property 1-5) shows that the events $\uparrow(a \cdot b)$ and $\uparrow(a \cdot c)$ can occur simultaneously since they are not independent.

## III. Basic Notions

A grafcet is a graph having two types of nodes, i.e., steps and transitions (a grafcet contains at least one step and one transition). Directed arcs either connect a step to a transition or a transition to a step. The original graphical representation was similar to the representation of a Petri net [1]: a step was represented by a circle as a place in a Petri net, and a transition was represented by a bar as in a Petri net. Then a standardized representation was introduced [11]. We shall present only the standardized representation.

### A. Steps and Transitions

*Steps:* A step is represented by a square as shown in Fig. 6. A step may have two states: it may either be active (this is represented by a token in the step) as is the case of step 2 in Fig. 6, or inactive as is the case of step 1 in Fig. 6. The steps which should be active when the system is started and represented by a double square. These are known as initial steps (step 3 in Fig. 6). Actions are associated with the steps, and these are the outputs of the grafcet (this will be specified in Section III-C).

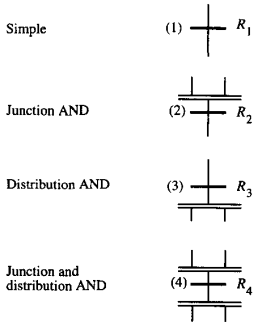



Fig. 7. Representation of a transition.

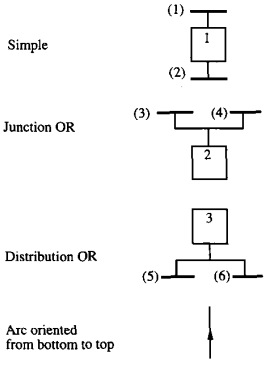



Fig. 8. Representation of directed links.

*Transitions:* A transition is represented as shown in Fig. 7. For a clear understanding of this figure, let us note that a vertical arc without arrow is running from top to bottom. There are a number of cases to be considered. The transition symbol is a bar, but the latter must be preceded or/and followed by a double bar in some cases. It is preceded by a double bar when two or more arcs join this transition [transitions (2) and (4)]; this implies that it involves waiting for multiple input steps to be active before firing the transition. A transition is followed by a double bar when two or more arcs leave this transition [transitions (3) and (4)]; this implies that it involves activating multiple output steps when the transition is fired. A receptivity $R_i$ will be associated with each transition (i). The receptivity is a function of the grafcet input variables, possibly of the internal state (this will be specified in Section III-E).

*Directed Links (or Arcs):* The representation of directed links (or arcs) is illustrated in Fig. 8. A directed link must always run from a step to a transition or from a transition to a step. When two or more directed links join the same step, they are grouped together as is shown in Fig. 8 for arcs $(3) \rightarrow 2$ and $(4) \rightarrow 2$. When two or more directed links leave the same step, they have a common departure point as is shown in Fig. 8 for arcs $3 \rightarrow (5)$ and $3 \rightarrow (6)$. A vertical arc running from bottom to top must be marked by an arrow.

*Remark 2:* In a grafcet, a step may have no input and/or no output transitions (for example, step 7 in Fig. 22 has no input transition). Likewise, a transition may have no input and/or no output steps. A transition without input steps is known as a source transition and a transition without output steps as a sink transition.

On the other hand, a directed link must always have a departure node (transition or step) and an arrival node (step or transition).

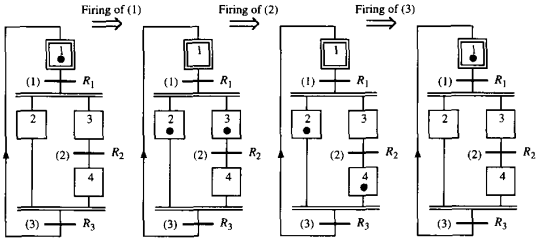



Fig. 9. Evolution of the situation.

### B. Firing of Transitions

An active step contains one and only one token, and an inactive step does not contain any tokens. All the active steps at any given moment, i.e., the marking, define the situation at this moment. A situation corresponds to a system state. The evolution of the situation is achieved by firing of transitions. The inputs of the logic controller are associated with the transitions (they provoke their firings) and the outputs are associated with the steps (they are produced by the active places).

*Definition 4:* A transition is firable if both the following conditions are met:

1) All the steps preceding the transition are active (the transition is said to be enabled.)
2) The receptivity of the transition is true.

Fig. 9 represents the evolution of the situation of a grafcet having four steps and three transitions. (In this grafcet, there is no output. The outputs will be considered in Section III-C.) The situation is represented by the set of active steps, i.e., $\{1\}$ in the initial situation corresponding to the left most grafcet of Fig. 9. In this situation, only transition (1) is enabled. It will be fired as soon as $R_1$ is true. Firing of a transition consists in deactivating all the steps upstream of the transition and activating all the steps downstream. These operations (activation and deactivation) are indissociable and are carried out simultaneously. Firing of (1) leads from the situation $\{1\}$ to the situation $\{2, 3\}$. Then firing of (2) leads to the situation $\{2, 4\}$, and firing of (3) leads back to the initial situation.

We shall now explain when a receptivity is true. We will consider three cases: the receptivity is a condition (i.e., a Boolean function), or an event, or both a condition and an event. The behavior of the grafcet is explained by means of the classical state table model.

*1) The Receptivity is a Condition:* This is illustrated in Fig. 10. The transition under consideration is (3) with a receptivity $R_3 = a$, where $a$ is a Boolean variable. Transition (3) is enabled when step 6 is active. Then the firing of (3) occurs if step 6 is active and as soon as the receptivity $a = 1$. Fig. 10(a) and (b) illustrate a case where $a = 0$ when the step

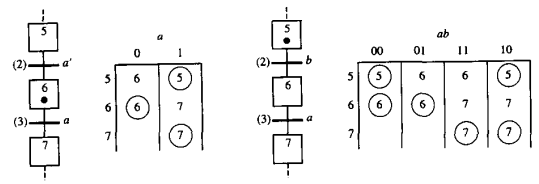


Fig. 10. The receptivity is a condition: $R_3 = a$ (a) and (b) The value of $a$ is unique when step 6 becomes active (c) and (d) The variable $a$ may have any Boolean value when step 6 becomes active.

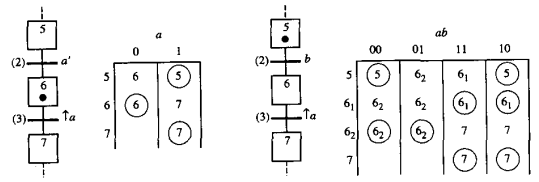


Fig. 11. The receptivity is an event: $R_3 = \uparrow a$ (a) and (b) The value of $a$ is unique when step 6 becomes active (c) and (d) The variable a may have any Boolean value when step 6 becomes active.

becomes active. Step 6 remains active as long as $a = 0$, and the firing occurs as soon as the value of $a =$ changes to 1.

In Fig. 10(c) and (d), the value of $a$ may be either 0 or 1 when step 6 becomes active. Step 5 corresponds to two total states since $a$ may have any Boolean value. These total states are noted (5, 00) and (5, 10) since they are stable in the columns where $ab = 00$ and 10, respectively. When the receptivity $R_2 = b$ takes the value 1, then transition (2) is fired and the situation {6} is reached. If $a = 0$ at this time, then situation {6} is stable. But if $a = 1$, transition (3) is immediately fired and situation {7} is reached; that means that situation {6} is unstable.

This behavior is illustrated in Fig. 10(d). If the total state is (5, 00) when $b$ changes to 1, the unstable state (5, 01) then the stable state (6, 01) are reached. Now, if the total state is (5, 10) when $b$ changes to 1, the unstable state (5, 11), the unstable state (6, 11) and the stable state (7, 11) are successively reached. If the stable state is (6, 01), $b$ may change any number of times; the system state is either (6, 01) or (6, 00). From both cases, if $a$ changes to 1, the state 7 is reached (either (7, 11) or (7, 10)).

*2) The Receptivity is an Event:* This is illustrated in Fig. 11. The transition under consideration is (3) with a receptivity $R_3 = \uparrow a$. The firing of (3) occurs if step 6 is active, and as soon as the event $\uparrow a$ occurs. Fig. 11(a) and (b) illustrate a case where $a = 0$ when the step becomes active. Step 6 remains active until $\uparrow a$ occurs, i.e., until $a$ changes to 1. The behaviors of Fig. 10(a) and 11(a) are quite similar. This is consistent with Definition 1.

In Fig. 11(c) and (d), the value of $a$ may be either 0 or 1 when step 6 becomes active. When the receptivity $R_2 = b$ takes the value 1, then transition (2) is fired, and the situation {6} is reached. If $a = 0$ at this time, then situation {6} corresponds to state $6_2$ in Fig. 11(d). But if $a = 1$, situation {6} corresponds to state $6_1$ in Fig. 11(d). In both cases, the

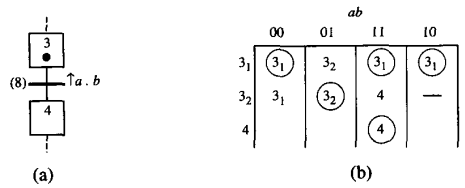


Fig. 12. The receptivity $R_8 = \uparrow a \cdot b$ is the product of an event, $\uparrow a$ and a condition $b$.

situation {6} is stable. If $a = 0$ when situation {6} is reached (state $6_2$ in the state table), firing of transition (3) will occur as soon as $a$ takes the value 1. If $a = 1$ when situation {6} is reached (state $6_1$ in the state table), the event $\uparrow a$ can occur only after a change of $a$ to 0 (leading from $6_1$ to $6_2$).

*Remark 3:* It appears on the example of Fig. 11(c) and (d) that a step (even if it is the only active one) does not always correspond to a single internal state.

*3) The Receptivity is the Product of an Event and a Condition:* This is illustrated in Fig. 12. The receptivity of transition (8) is $R_8 = \uparrow a \cdot b$. When step 3 is active the internal state is either $3_1$ or $3_2$ [Fig. 12(b)]. Assume that step 3 is active. The firing of transition (8) will occur if $b = 1$ when $\uparrow a$ occurs. Then the input state must be $ab = 01$ just before the firing (state $3_2$). When the state $3_2$ is reached, two events can occur: either $\uparrow a$ which leads to situation {4}, or $\downarrow b$ leading back to state $3_1$. Note that from the stable state ($3_2$, 01) the column $ab = 10$ cannot be reached since two independent events cannot occur simultaneously (Hypothesis 1).

*Generalization:* This third case is the most general, whereas the first two are particular cases. Indeed, a receptivity $R_i$ may always be written as $R_i = E_i \cdot C_i$ using:

1) The "always true" condition, written as $C_i = 1$, which is associated with receptivities $R_i$ which only depend on an external event $E_i$. For example, $R_2 = E_2$ means that $C_2 = 1$, thus $R_2 = E_2 \Leftrightarrow R_2 = E_2 \cdot 1$.
2) The "always occurrent" event, written as $e$ (see Notation 2), which is associated with receptivities $R_i$ which only depend on a condition $C_i$. For example, $R_3 = C_3$ means that $E_3 = e$, thus $R_3 = C_3 \Leftrightarrow R_3 = e \cdot C_3$.

*Remark 4:* According to Definition 2-a) and Remark 1-3), the product of an event and a condition is an event. Then, from the above generalization, a receptivity can always be considered as an event. It follows that a receptivity which would be the sum of an event $E_i$ and a condition $R_j$ may be considered as the sum of both events $E_i$ and $e \cdot R_j$, i.e., an event by extension.

*4) Firing Rules:*

*Rule 1)* All firable transitions are immediately fired.
*Rule 2)* Several simultaneously firable transitions are simultaneously fired.
*Rule 3)* When a step must be simultaneously activated and deactivated, it remains active.

These rules are essential for understanding and interpreting Grafcet. The first and third are easy to understand and require no particular commentary. The second is extremely important, first because its application results in an interpretation different

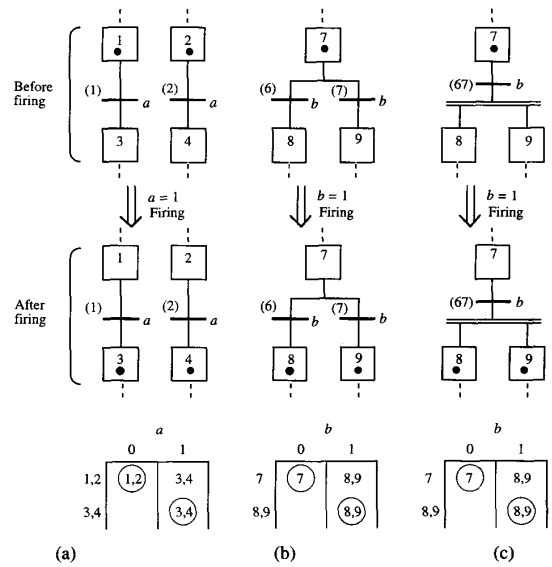



Fig. 13. Illustration of simultaneous firings.

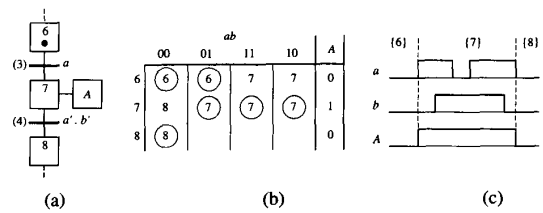



Fig. 14. Illustration of unconditional level action.

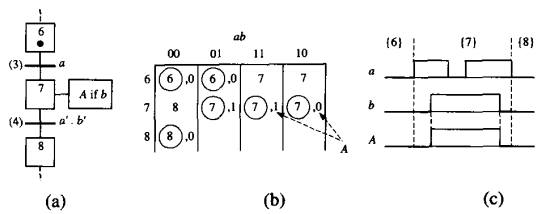



Fig. 15. Illustration of conditional level action.

from that of a Petri net (this point will be commented in Section VI-B), and second because it may be the origin of specification errors.

Fig. 13 presents two examples of simultaneous firing. In Fig. 13(a), transitions (1) and (2), belonging to the same grafcet, are enabled. When the Boolean variable $a$ assumes the value 1, both these transitions are simultaneously firable and are thus simultaneously fired according to Rule 2. This evolution from situation $\{1, 2\}$ to situation $\{3, 4\}$ presents no problems since the enabling conditions for transitions (1) and (2) are independent. In fact, if $R_1 \neq R_2$ and transition (1) were fired before transition (2), transition (2) would remain enabled, and vice versa.

Fig. 13(b) presents a considerably different case. Although transitions (6) and (7) are also simultaneously fired when variable $b$ assumes value 1, we observe that there is no independence between the enablings of these two transitions. In fact, if $R_6 \neq R_7$ and transition (6) were fired before transition (7), step 8 alone would be active and transition (7) would no longer be enabled. This cannot occur since $R_6 = R_7 = b$. We may encounter, however, cases where $R_6 \neq R_7$. The final result may then depend on the order in which the following 3 events take place: 1) step 7 becomes active, 2) receptivity $R_6$ becomes true, and 3) receptivity $R_7$ becomes true.

*Remark 5:* Note that Fig. 13(b) may be replaced by Fig. 13(c) which has an identical evolution (as far as the successive situations are concerned) but is without simultaneous firing. Assume that there are several transitions following some step $i$. It is always possible to modify the grafcet to not have more than one of these transitions which can be fired (this is recommended). Example: assume $R_6 = a$ and $R_7 = b$ in Fig. 13(b). We modify the receptivities as $R_6 = a \cdot b'$ and $R_7 = a' \cdot b$, and we add a transition (8) with an input step 7, with output steps 8 and 9 and with a receptivity $R_8 = a \cdot b$. Since the receptivities are mutually exclusive, at the most one of them may be true at any time.

### C. Actions and Outputs

There are two major categories of actions, namely level and impulse actions.

*1) Level Action:* A level action is modeled by a Boolean variable. It may be unconditional or conditional.

The action $A$ in Fig. 14(a) is unconditional. This means that $A$ assumes the value 1 when step 7 becomes active up to the moment when this step becomes inactive, without any other condition. This is illustrated by the Moore asynchronous state table in Fig. 14(b) (in a Moore machine the value of an output depends only on the internal state), and an example of behavior is presented in Fig. 14(c).

The action $A$ in Fig. 15(a) is conditional. Action $A$ assumes the value 1 when step 7 is active if $b = 1$. This is illustrated by the Mealy asynchronous state table in Fig. 15(b) (in a Mealy machine the value of an output depends on the total state), and an example of behavior is presented in Fig. 15(c).

*2) Impulse Action:* An impulse action is responsible for changing the value of a discrete variable. The latter may be a Boolean variable or any other discrete variable (e.g., the value of a counter). An impulse action associated with a step is carried out as soon as this step changes from the inactive to the active state, regardless of the time during which this step remains active (even infinitely short). An impulse action may be said to be an order (do this...), whereas a level action indicates a state.

In Fig. 16(a), action $B^*$ is an impulse action. The asterisk was recommended at the origin of Grafcet to indicate the impulse character of an action. It does not always, however, serve a purpose. For example, incrementing a counter $C$ can be written as $(C \leftarrow C + 1)^*$ or simply $(C \leftarrow C + 1)$ since there is no ambiguity concerning the impulse character of this action. An example of this behavior is presented in Fig. 16(c): if the situation is $\{6\}$, as soon as $a$ assumes the value 1, the situation $\{7\}$ is reached and action $B^*$ is executed.

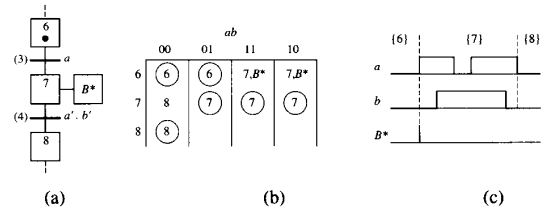



Fig. 16. Illustration of impulse action.

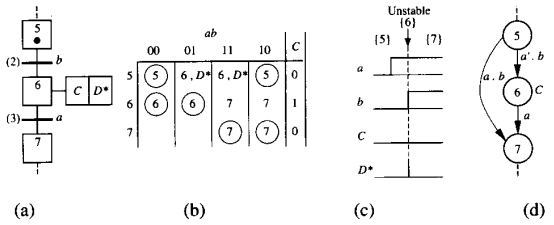



Fig. 17. Actions associated with an unstable step.

We have illustrated this example with the state table in Fig. 16(b), though the representation of an impulse output is not usually used in an asynchronous automaton.

*Remark 6:*

1) A level action has, by definition, a finite duration (or possibly infinitely long). In other words, a level action with an infinitely small duration cannot exist, since this would have no meaning. It follows that a level action is only defined for stable situations.
2) On the other hand, an impulse action is infinitely short. Such an action is executed as soon as the corresponding step becomes active, even if the situation is not stable.

Remark 6 is illustrated in Fig. 17. In Fig. 17(c), receptivity $R_3 = a$ is already true when receptivity $R_2 = b$ becomes true. Transitions (2) and (3) are thus fired successively. Step 6 is active transiently (i.e., for an infinitely short time). Thus the level action $C$ remains at 0, while the impulse action $D^*$ is executed. Note that if the transitions between states 5, 6, and 7 were represented by a state diagram, one would have the representation in Fig. 17(d). It is clear that, from state 5, if $a = 1$ when $b$ changes from 0 to 1, state 7 is directly reached and that the level action $C$ associated state 6 remains at value 0. This is consistent with Remark 6.1. (Representation of an impulse output does not exist in this model.)

*3) Actions and Outputs:* An output corresponds to a signal which acts on the environment of the system described (either on the process controlled, or on a supervision system, or a human operator, for example by means of a visualization). This is shown in diagram form in Fig. 18. The impulse action $C^*$ is not an output but controls an operation on the processing section of the control section which is the logic controller: incrementing a counter, calculation order, etc. Variable $E$ is a logic controller output but is not an action in the Grafcet sense (it may be, for example, a counter value or any Boolean or numerical quantity the evolution of which is controlled by impulse actions such as $C^*$).

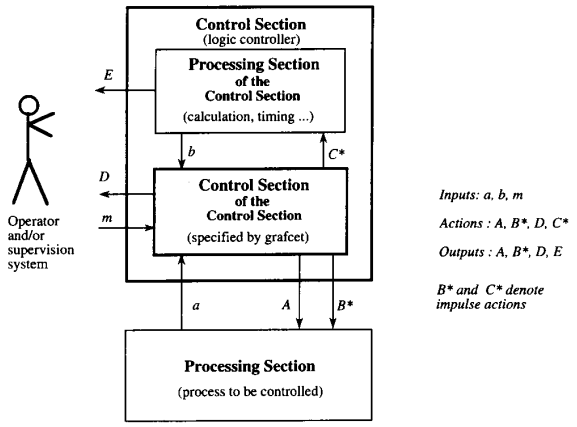



Fig. 18. Logic controller, inputs, actions, and outputs.

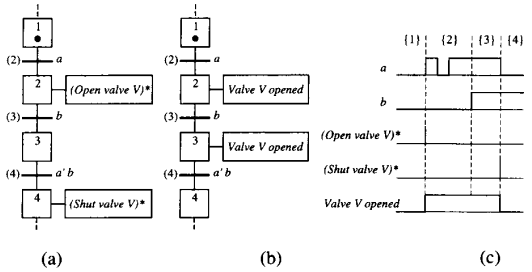



Fig. 19. (a) Impulse action to the processing section. (b) Equivalent behavior described with a level action.

An impulse action such as $B^*$ may be used if there is a device for memorizing the state. This is illustrated in Fig. 19 where $(Open\ valve V)^*$ and $(Shut\ valve V)^*$ are impulse actions. Fig. 19(a) and (b) are equivalent as far as the state of valve V is concerned. It will be observed that valve V must be closed even if step 4 becomes active only transiently.

### D. Concurrency and Synchronization

This is illustrated in Fig. 20 in an example without output. From the situation {1}, the situation {2, 3} is reached by firing transition (1) as soon as $a$ assumes the value 1. From this time, there is a concurrency between what happens in the part {step 2, transition (2), step 4} on the one hand, and in the part {step 3, transition (3), step 5} on the other hand, in Fig. 20(a). Transition (4) corresponds to a synchronization since both steps 4 and 5 must be active before it is fired.

This behavior is illustrated in Fig. 20(b) and (c). It is clear in Fig. 20(b) that four cases must be considered when transition (1) is fired. Case $b = 0$ and $c = 0$ (column 100): the stable situation {2, 3} is immediately reached. Case $b = 0$ and $c = 1$ (column 101): the unstable situation {2, 3} then the stable situation {2, 5} are reached. Case $b = 1$ and $c = 1$ (column 111): the unstable situation {2, 3} then the stable situation {4, 5} are reached. Note that, in this case, transitions (2) and (3) are fired simultaneously (Rule 2, Section III-B-4). Case $b = 1$ and $c = 0$: symmetrical to case $b = 0$ and $c = 1$. In Fig. 20(b), the dash for the total state (2, 3; 011) means that this total state

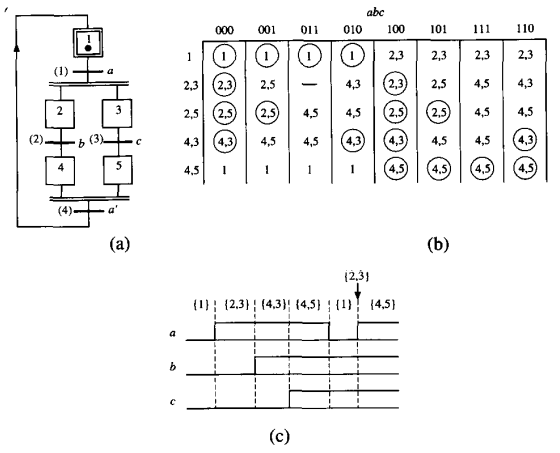



Fig. 20. Concurrency and synchronization.

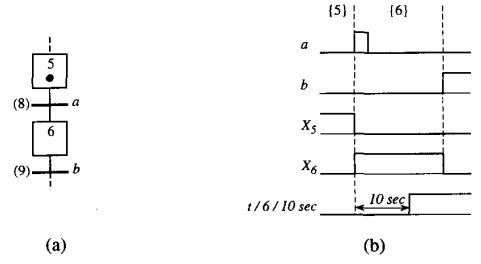



Fig. 21. Internal state and time.

cannot be reached. As a matter of fact, $b$ and $c$ would have to change their value simultaneously to reach this total state from (2, 3; 000), and $a$, $b$, and $c$ would have to change their value simultaneously to reach this total state from (2, 3; 100).

### E. Taking into Account Internal State and Time

We shall now introduce the variable written as $X_i$.

*Definition 5:* The variable $X_i$ is a Boolean variable which is equal to 1 when step $i$ is active.

In Fig. 21(a) we have $X_5 = 1$, and $X_6 = 0$.

In the original Grafcet, a variable $T_i^{\Delta}$ is introduced. The same variable is written as $t/i/\Delta$ in the French standard, and this is the notation that we shall use here. In the international standard, other possibilities have been considered.

*Definition 6:* The variable $t/i/\Delta$ is a Boolean variable which equals 1 if a time at least equal to $\Delta$ has elapsed since the last time that step $i$ moved from the inactive to the active state.

This is illustrated in Fig. 21. Step 5 is active, i.e., $X_5 = 1$. When variable $a$ assumes the value 1, step 6 becomes active, i.e., $X_6$ assumes the value 1. The Boolean variable $t/6/10$ *sec* assumes the value 1 ten seconds later. It will keep this value until the variable $X_6$ next changes from state 0 to state 1. Note that the moment when step 6 becomes inactive again has no influence.

A Boolean variable $X_i$ or $t/i/\Delta$ can be used in a receptivity or as a condition for a level action. An event $\uparrow(t/i/\Delta)$ can be used in a receptivity (but it is recommended to be very careful with an event $\uparrow(X_i)$ [16]).

## IV. Interpretation

A grafcet describes a logic controller (which is a sequential machine): it specifies relations between the inputs and outputs. Using any input timing diagram, the grafcet lets us know the timing diagram of the corresponding outputs. The interpretation must thus be unambiguous. Two persons faced with the same grafcet and the same input timing diagram must come up with the same output timing diagram. This is the aim of the interpretation algorithm given below. All the elements found there have already been presented in simple examples.

The interpretation algorithm is based on two hypotheses, namely Hypotheses 2 and 3 presented below.

*Hypothesis 2:* All the external events are independent (uncorrelated).

Hypothesis 1 and 2 lead to the fact that external events never occur simultaneously (without this assumption the Grafcet model would not be deterministic). This corresponds to a formal model. How to manage this assumption for practical implementations will be commented on in Section VII.

*Hypothesis 3:* A grafcet has the time to reach a stable situation between two distinct external event occurrences (in fact the changeover from one stable situation to another has a zero duration, even if there are several unstable situations between them).

These hypotheses are not new, and they correspond to functioning in the fundamental mode [25] which is classical in the theory of asynchronous sequential systems (roughly speaking, this means that the processing part, i.e., the process to be controlled, represented in Fig. 18, is considered to be slower than the logic controller).

### A. Interpretation Algorithm

*Algorithm: Interpretation of the Grafcet*

*Step 1)* Initialization: activation of the initial steps and execution of the associated impulse actions. Go to Step 5.

*Step 2)* When a new external event occurs, determine the set $\mathbf{T}_1$ of the transitions firable on occurrence of this event. If $\mathbf{T}_1$ is not empty, go to Step 3. Otherwise, modify if necessary the state of the conditional actions associated with the active steps (in fact, certain actions may depend on conditions the values of which may have changed). Wait for a new external event at Step 2.

*Step 3)* Fire all the firable transitions. If the situation remains unchanged after this simultaneous firing, go to Step 6.

*Step 4)* Carry out all the impulse actions associated with the steps having become active at Step 3 (including initialization of the time delays).

*Step 5)* Determine the set $\mathbf{T}_2$ of the transitions firable on occurrence of event e (always occurrent). If $\mathbf{T}_2$ is not empty, go to Step 3.

*Step 6)* A stable situation is reached.

*Step 6.1)* Determine the set $\mathbf{A}_0$ of the level actions which must be deactivated (actions associated with the steps which were active at Step 2 and which are now inactive, and conditional actions associated with the steps having remained active for which the conditions are no longer verified).

*Step 6.2)* Determine the set $\mathbf{A}_1$ of the level actions which must be activated (actions associated with the steps which were inactive at Step 2 and which are now active, possibly under certain conditions, and conditional actions associated with the steps having remained active for which the conditions are verified, whereas they were not so at Step 2).

*Step 6.3)* Set to 0 all the actions which belong to $\mathbf{A}_0$ and do not belong to $\mathbf{A}_1$. Set to 1 all the actions which belong to $\mathbf{A}_1$. Go to Step 2.

### B. Example

*Tank Filling Modified:* The controlled system is the same as in Fig. 2. The control specifications are modified as follows:

1) The filling process is no longer allowed if the button m is pressed (i.e., if $m = 1$) but it is triggered by pressing push button m (i.e., by $\uparrow m$).
2) A counter is added: the value $C$ in the counter represents the number of cycles (emptying and filling of Tank 2).

From this specification, the grafcet in Fig. 22(a) can be obtained. (This grafcet is not the unique solution to the problem of constructing a grafcet representing this specification.) The receptivity $R_1 = 1$ means that it is always true. The operations on the counter are impulse actions.

*Example:* Let us apply the interpretation algorithm from the initial situation {1, 7} for the timing diagram in Fig. 22(b). This diagram corresponds to the string of external events $\uparrow m \downarrow m \uparrow b_1 \uparrow b_2 \uparrow h_1 \cdots$

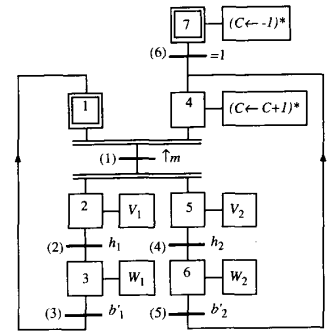
*Step 1)* *Initialization*: we have the initial situation {1, 7}. The impulse action $(C \leftarrow -1)^*$, associated with step 7, is executed (then $C$ = -1).

*Step 2)* When event $\uparrow m$ occurs, we have $\mathbf{T}_1 = \{(1)\}$.

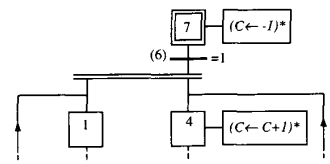
*Step 3)* Firing of the transition (1). Situation {2, 5} is reached.

*Step 4)* No impulse action associated with steps 2 and 5.

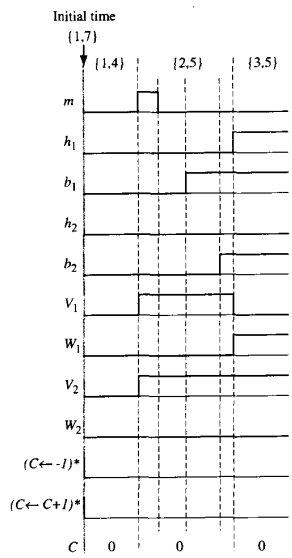
*Step 5)* The enabled transitions are (2) and (4). We have $h_1 = h_2 = 0$. Thus $\mathbf{T}_2$ is empty.




(a)




(b)




(c)

Fig. 22. (a) and (b) Example of specification and interpretation. (c) Same example (partial) with a single initial state.

*Step 5)* Only transition (6) is enabled. It is firable on occurrence of $e$ since the receptivity is always true. Thus $\mathbf{T}_2 = \{(6)\}$.

*Step 3)* Firing of the transition (6). Situation {1, 4} is reached.

*Step 4)* The impulse action $(C \leftarrow C + 1)^*$, associated with step 4, is executed (then $C = 0$).

*Step 5)* $\mathbf{T}_2$ is found to be empty, i.e., $\mathbf{T}_2 = \emptyset$.

*Step 6)* Stable situation {1, 4}. $\mathbf{A}_0$ and $\mathbf{A}_1$ are empty.

*Step 6)* Stable situation {2, 5}. $\mathbf{A}_0$ is empty and $\mathbf{A}_1 = \{V_1, V_2\}$. Then $V_1$ and $V_2$ assume the value 1.

*Step 2)* When event $\downarrow m$ occurs, we have $\mathbf{T}_1 = \emptyset$.
*Step 2)* When event $\uparrow b_1$ occurs, we have $\mathbf{T}_1 = \emptyset$.
*Step 2)* When event $\uparrow b_2$ occurs, we have $\mathbf{T}_1 = \emptyset$.
*Step 2)* When event $\uparrow h_1$ occurs, we have $\mathbf{T}_1 = \{(2)\}$.

*Step 3)* Firing of the transition (2). Situation $\{3, 5\}$ is reached.

*Step 4)* No impulse action associated with step 3.

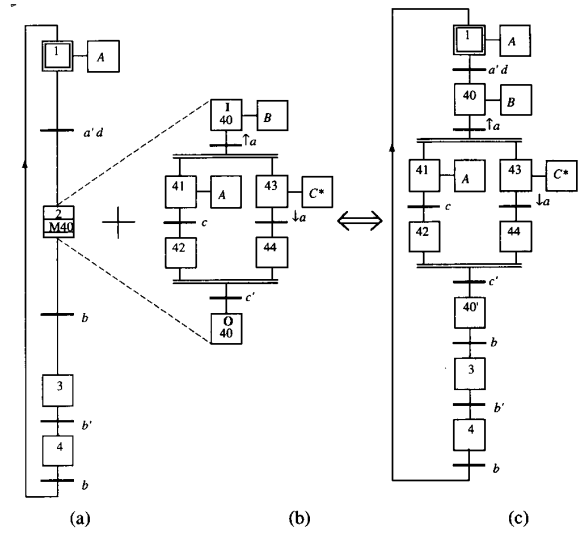



Fig. 23. Macrostep (a) Grafcet with macrostep. (b) Macrostep expansion, (c) Equivalent grafcet without macrostep.

*Step 5)* $\mathbf{T}_2 = \emptyset$.

*Step 6)* Stable situation $\{3, 5\}$. $\mathbf{A}_0 = \{V_1\}$ and $\mathbf{A}_1 = \{W_1\}$. Thus $V_1$ assumes the value 0 and $W_1$ assumes the value 1.

Let us note that, when the stable situation is $\{2, 5\}$, only transitions (2) and (4) are enabled and the grafcet is only receptive to $h_1$ and $h_2$ (then to the events $\uparrow h_1$ and $\uparrow h_2$ since $h_1 = h_2 = 0$). Thus no transition can be fired on occurrence of any different event.

## V. Macrosteps and Macroactions

It may be useful to build a grafcet in an hierarchical manner. Let us comment on the concepts of macrosteps and macroactions [7]. Notice that macrosteps and macroactions correspond to abbreviations of an "ordinary" grafcet, since it is always possible to obtain an equivalent grafcet without any macrostep or macroaction, as illustrated by Figs. 23 and 24.

### A. Macrosteps

The aim of the macrostep concept is to facilitate the description of complex systems. The macrostep makes it possible to lighten the graphical representation of a grafcet by detailing certain parts separately.

The concept of the macrostep is illustrated in Fig. 23. A macrostep is represented by a square divided into three parts by two horizontal lines. A macrostep given as 2/M40 is represented in Fig. 23(a). It represents a grafcet which is detailed elsewhere and which is known as a macrostep expansion. Fig. 23(b) is the expansion corresponding to M40. If this macrostep expansion is used to replace macrostep 2/M40, the grafcet of Fig. 23(c) is obtained. The complete set of Fig. 23(a) and (b) (grafcet with a macrostep, plus expansion

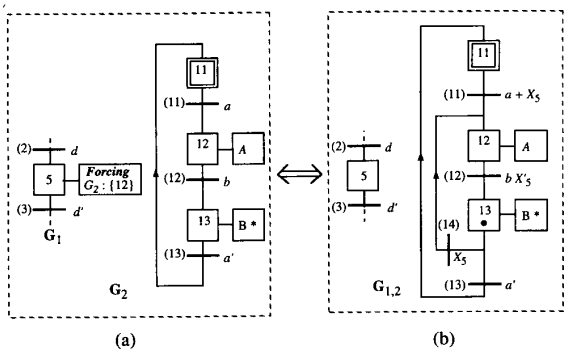



Fig. 24. Illustration of the macroaction forcing.

of the macrostep) is strictly equivalent to Fig. 23(c) in which the macrostep has been replaced by its expansion.

A macrostep and its expansion satisfy the following rules:

1) A macrostep expansion has only one input step (written as **I**) and one output step (written as **O**).
2) All firings of a transition upstream of the macrostep activate the input step of its expansion.
3) The output step of the macrostep expansion participates in the enabling of the downstream transitions, in accordance with the structure of the grafcet containing this macrostep.
4) No directed links either join or leave the macrostep expansion.

Let us now return to Fig. 23 to give further details on certain points. The grafcet of Fig. 23(a) contains a macrostep which is written as 2/M40. Number 2 is the macrostep number, i.e., it indicates its position in the grafcet. Symbol M40 relates to the macrostep expansion. Number 40 is found in the expansion to mark the input step, **I**40, and the output step, **O**40, of expansion M40. When the grafcet part corresponding to expansion M40 has been used to replace macrostep 2 [i.e., in Fig. 23(c)], the symbols **M**, **I**, and **O** have disappeared. We now have an ordinary grafcet in which every step has a number. During replacement, steps **I**40 and **O**40 have become 40 and 40′ (any other numbering would have been possible, provided that no two grafcet steps have the same number).

In general, a grafcet may contain several macrosteps. Eventually two or more macrosteps may have the same expansion, for example 2/M50 and 5/M50 [7].

### B. Macroactions

When describing complex systems, the size of the grafcets may increase so that they become difficult to work out and thus to understand, correct, update, etc. Taking the safety devices into account is an important reason for increasing complexity. We should not treat them like other parameters since they have a different role, which is less frequent (we hope!) and has higher priority. The concept of hierarchy naturally springs to mind. It is easy to imagine that a logic controller (described by a grafcet) has a global influence on

another logic controller (described by another grafcet), this being known as a macroaction.

A macroaction may be a level or an impulse action (just like an ordinary action). It is produced by a grafcet $\mathbf{G}_1$ and has an effect on the behavior of a grafcet $\mathbf{G}_2$. A macroaction is thus homogeneous with an action from the point of view of $\mathbf{G}_1$.

The macroaction forcing is illustrated in Fig. 24. Forcing $\mathbf{G}_2$: {12}, associated with step 5, means that the grafcet $\mathbf{G}_2$ is put into the situation such that step 12 is active (while the other ones are inactive) as long as step 5 is active. It is an level macroaction. The set of the two grafcets $\mathbf{G}_1$ and $\mathbf{G}_2$ of Fig. 24(a) is equivalent to grafcet $\mathbf{G}_{1,2}$ of Fig. 24(b). When step 5 is active, step 12 also becomes active. As a matter of fact, in Fig. 24(b): if step 11 is active, transition (11) is fired because $R_{11} = a + X_5$; if step 13 is active, transition (14) is fired because $R_{14} = X_5$; if step 12 is active it remains active; in all cases step 12 remains active as long as $X_5 = 1$ since $R_{12} = b \cdot X_5'$.

Other macroactions such as force (impulse macroaction), freezing and masking (level macroactions) may be used [7].

## VI. Comparison with Other Models

### A. Comparison with Petri Nets

The main fact we wish to point out here is that the representation of the concurrency in grafcets is inherited from Petri nets. Each model possesses two types of nodes, namely the steps and the transitions for the Grafcet, and the places and transitions for the interpreted $PN$. Interpreted Petri nets can be defined such that the input–output behavior is similar to the input–output behavior of a grafcet [7], [8], [17] (level and impulse actions associated with the places, events, and conditions associated with the transitions).

There are two basic differences between the interpreted $PN$ and the Grafcet.

*First Difference:* The marking of a grafcet step is Boolean, whereas the marking of an interpreted $PN$ place is numerical.

*Second Difference:* In a grafcet, all the simultaneously firable transitions are simultaneously fired (Rule 2). On the other hand, if several transitions are in conflict [their enabling depends on the marking of the same place, as in Fig. 13(b)] and are simultaneously firable in an interpreted $PN$, only one of them can be fired. The behavior of such a $PN$ is not deterministic, since there is no rule for choosing the transition to be fired.

Now, if a $PN$ is safe (i.e., for any reachable marking, the marking of every place is less than or equal to one), the first difference does not exist. If an interpreted $PN$ is such that, for any reachable marking, any pair of transitions in conflict have receptivities which cannot be true at the same time, only one of them can be firable: the behavior is then deterministic, and the second difference does not exist. It follows that a Petri net (with a Grafcet type input–output interpretation), which is safe and deterministic, is equivalent to a grafcet, as illustrated in Fig. 25. This means that if the "net" is interpreted like a $PN$ or a grafcet, the same input–output behavior is deduced from it.

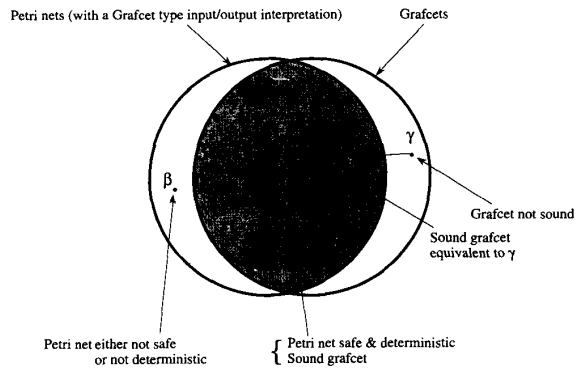



Fig. 25. Comparison between Grafcet and Petri nets.

Roughly speaking, a sound grafcet [5] has the following properties:

1) At no moment can an active step be such that one of its input transitions is firable, and none of its output transitions is simultaneously firable (we say that the grafcet is not reactivable).
2) For any pair of transitions which are simultaneously firable, they have no input step in common.
3) No step can have two simultaneously firable input transitions.

Now, if the marking evolution of a sound grafcet was interpreted as for a $PN$, this $PN$ would be safe from properties 1 and 3 above. If property 2 above is satisfied, then a case similar to Fig. 13(b) cannot exist. It follows that a sound grafcet is equivalent to a $PN$ (with a Grafcet type input–output interpretation), as illustrated in Fig. 25.

Let us note that most of the grafcets which are drawn are naturally sound. If a grafcet is not sound ($\gamma$ in Fig. 25), it can be modified to become sound ($\delta$ in Fig. 25). The transformation presented in Remark 5 is usually sufficient. More details can be found in [7]. Then, in practice, most of the grafcets are also Petri nets, and the few other ones can be modified to have this property. This enables the classical properties of $PN's$ to be applied to them (marking invariants for example).

### B. Comparison with the Classical Models of Sequential Machines

In this section we shall see that the Grafcet is a powerful description tool. The classical models describing sequential systems correspond to asynchronous machines on the one hand and to synchronous machines on the other. We shall see that a grafcet describing the same functioning can immediately be associated with each of these models. On the other hand, however, grafcets exist which cannot be translated immediately by one of these models.

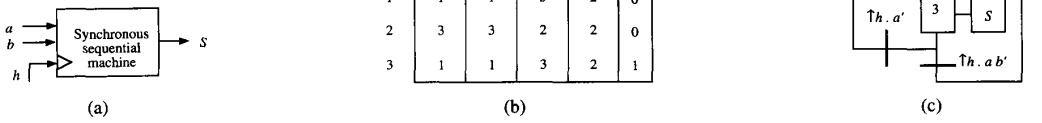



Fig. 26. Grafcet corresponding to a synchronous machine.

Moving from a state table representing an asynchronous machine to a grafcet is immediate. A step of the grafcet is associated with each state of the machine. A transition on the grafcet is associated with each transition of the machine (with a single upstream and a single downstream step). The receptivities correspond to the firing conditions. For the state table in Fig. 11(d), this transformation leads to four states, namely 5, $6_1$, $6_2$, and 7. For a Moore machine, the actions are unconditional, and for a Mealy machine the actions are conditional ones.

Fig. 26(a) represents a synchronous sequential machine with two inputs, $a$ and $b$, and an output, $S$. Input $h$ is a special input which is a clock (usually, this model is used for parts of computers but not for logic controllers). The internal state and/or output state can only change on a rising edge of clock $h$. Since this applies to all the transitions, this condition is implicit and thus is not represented on the Moore state table in Fig. 26(b). For example, if we are in state 1 and $ab = 10$, at the next rising edge of $h$, we shall move to state 2. The notion of a stable state loses all meaning in a synchronous machine. Each time that edge $\uparrow h$ occurs, both the internal state and the output state may well be changed. Note that a synchronous machine can always be transformed into an asynchronous machine (by considering the clock to be an ordinary input), but that the reverse is not true. The transformation from a state table to a grafcet is carried out immediately as for the asynchronous case. The only difference is that each receptivity explicitly mentions that it is on occurrence of event $\uparrow h$ that the transition takes place. For example, with the transition from state 1 to state 2 in the state table of Fig. 26(b) is associated the receptivity $\uparrow h \cdot ab'$ in the grafcet of Fig. 26(c). In the case of Mealy synchronous machine, the outputs of the flow table or diagram correspond to impulse actions on the grafcet.

Whereas a grafcet step can always be made to correspond to a state, the reverse does not apply. This is clear when a grafcet has a number greater than one of active steps (Fig. 20 for example), and there are also examples where the grafcet has a single active step [7].

To summarize, the grafcet includes the descriptive power of both asynchronous and synchronous machines and it can mix Boolean and event inputs, and level and impulse actions (which model Boolean and event outputs).

## VII. Concluding Remarks

The Grafcet model, drawing inspiration from Petri nets, is a very good tool for logic controller specification. It allows modeling of concurrency and synchronization. Above all, the input–output behavior is specified without ambiguity. When some parts of the logic controller can be described separately, one can use macrosteps to simplify the model [7], [16]. This allows stepwise refinements which have already been used for Petri nets [26]. Thanks to macroactions [7], one can model a system through several grafcets having a hierarchical relation. This is particularly useful for taking safety devices into account.

*IEC Standard:* The International Electrotechnical Commission (IEC) Standard on Programmable Controllers [12] includes only some elements of the SFC and refers to [11] which is, then, the basic reference. In [12], the following restriction is made: an SFC must have exactly one initial state. This constraint can easily be taken into account as illustrated in Fig. 22(c).

As was noted in the introduction, the IEC Standard [11] is not presented in a formal way as in this paper; it is presented in a way stressing the practical side. Most of the concepts presented in this paper, however, can be found in it. Here are the main features of this standard.

1) *Receptivities:* They explicitly can be either a condition, or an event, or both (as in this paper). The recommendation in Remark 5 (receptivities mutually exclusive for output transitions of a place) is an obligation in the standard.
2) *Actions:* Basically, an action is a level action. The conditional actions exist. The impulse actions are not formally defined as in this paper; however, the same results can be obtained thanks to the actions which are qualified "stored" [e.g., step 2 in Fig. 19(a)] or "pulse shaped" [e.g., step 4 in Fig. 22(a)].
3) *Macrostep:* The word is not used, but the concept of "detailed representation of a step" is similar.
4) *Macroaction:* Does not exist in the standard.
5) *Time:* The actions can be qualified "delayed" and/or "time limited." The Boolean variable defined in Definition 6 is not used, but the descriptive power is the same.

*Implementation:* When some input of a logic controller changes, this controller must react "immediately" by modifying its output state which controls the process. Hence, any logic controller model is an asynchronous, continuous-time model. This is true for the four models in Fig. 3. Each model is a specification, i.e., a mathematical model where time is not involved. Once a logic controller has been specified, it must be implemented [15], [22].

The implementation is different from the specification since the necessary time for any operation must be taken into account in practice. The important feature is that the input–output behavior of the implementation must be the same as the input–output behavior of the specification (even if the time to reach a new output state is not null). Races between variables is a problem which is well known in hard-wired logic. Similar problems occur in programmed logic, even if their cause is not the same. Usually, a PLC scans the logic controller inputs and computes the next internal and output state before a new scanning of inputs (this is different from the continuous-time model). Let us call a cycle the period between two successive input scans. In an RLL program, all the rungs are scanned and updated in order in a cycle. Obviously, the order is important since the value of an internal variable $P$ depending on another, $Q$, which changes during the cycle, may depend on the order of $P$ and $Q$ in the program. For a Grafcet program, two points must be carefully considered: 1) Be sure that a stable state has been reached before modifying the level outputs and scanning inputs again: this is ensured if the interpretation algorithm (Section IV-A) is implemented. 2) If two independent input variables have changed during the same cycle and if behavior depends on the order of their changes, choose (even arbitrarily) one of the orders. For example, assume a section of railway track, $T$, on which two tracks $T_a$ and $T_b$ converge. When a vehicle approaches $T$ on track $T_i$, the event $\uparrow i$ occurs ($i = a$ or $b$). Since the simultaneous arrival of a vehicle on each of tracks $T_a$ and $T_b$ is considered to be impossible, the designer could have drawn up a grafcet such that two transitions, whose receptivities are $\uparrow a$ and $\uparrow b$, are enabled by the same step. If there is no technological means to distinguish which event took place first when they occur almost simultaneously, the receptivity $\uparrow a$, for example, can be replaced by the redundant receptivity $\uparrow a \cdot b'$. This is coherent with the algebra of events since $\uparrow a \cdot b'$ is an event, and the events $\uparrow a \cdot b'$ and $\uparrow b$ cannot occur simultaneously. The priority of one transition over another one may be specified by the designer [7], then all implementations are similar. If the priority is chosen by the implementer, there are two possible implementations whose behaviors are different only in a case whose probability is close to zero.

A grafcet can be implemented by various literal or graphic language [12]. Some Grafcet-based PLC's have a graphic editor. The structure of the SFC is introduced in the form of a drawing: the user draws the grafcet directly on the screen and adds actions and receptivities in the language he has chosen (literal or RLL) [24]. It is easier to understand behavior and to debug; for example, if a blocking in some situation is observed, the receptivities of the enabled transitions should be examined.

For specification of large real-time systems, a Petri net or a grafcet may become very large, and using synchronous languages may be another way to specify the controllers [4], [14]. Systems of reasonable size, however, may be described by Grafcet, and it appears that this model is extensively used in French industry because it is very easy to understand; hence, debugging and maintenance of programs based on Grafcet are easy. Grafcet has been successfully used in the implementation of supervisors designed on controlled-automata based concepts [3]. Some companies, particularly in the USA, have already developed many PLC programs based on a RLL specification. Since these programs must be maintained, it is difficult for them to switch to a Grafcet specification. This is why some research has been done on automatic translation of RLL programs into Grafcet [10].

## ACKNOWLEDGMENT

The author would like to thank B. H. Krogh for helpful discussions and for suggesting the presentation of Grafcet with the help of the classical state table model.

## REFERENCES

[1] AFCET, "Normalisation de la représentation du cahier des charges d'un automatisme logique," *Final Report of AFCET Commission*, Aug. 1977, published in the *J. Automatique et Informatique Industrielles* (61–62), Nov.–Dec. 1977.
[2] M. Blanchard, "Comprendre, maîtriser et appliquer le Grafcet," Editions Cepadues, Toulouse, 1979.
[3] B. Brandin and F. Charbonnier, "The supervisory control of the automated manufacturing system of the AIP," in *Proc. 4th Int. Conf. Computer Integrated Manufacturing Automation Technol.*, Troy, NY, Oct. 1994, pp. 319–324.
[4] P. Caspi, D. Pilaud, N. Halbwachs, and J. Plaice, "LUSTRE, a declarative language for real-time programming," in *Proc. Conf. Principles Programming Languages*, Munich, 1987.
[5] R. David and H. Deneux, "Deterministic safe interpreted Petri Nets–Relation with the Grafcet model," in *Proc. 4th European Workshop Theory Applications Petri Nets*, Toulouse, Sep. 1983.
[6] R. David and H. Alla, *Du Grafcet aux réseaux de Petri*. Paris: Hermès, 1989.
[7] R. David and H. Alla, *Petri Nets and Grafcet: Tools for Modelling Discrete-Event Systems*. London: Prentice-Hall, 1992.
[8] R. David, "Petri nets and Grafcet for specification of logic controllers," *IFAC Congress*, Sydney, vol. 3, pp. 335–340, July 1993.
[9] R. David and H. Alla, "Petri nets for modeling of dynamic systems—A survey," *Automatica*, vol. 30, no. 2, pp. 175–202, Feb. 1994.
[10] A. Falcione and B. H. Krogh, "Design recovery for relay ladder logic," in Proc. *IEEE Conf. Contr. Appl.*, Dayton, OH, Sep. 1992.
[11] IEC, International Electrotechnic Commission, *Preparation of Function Charts for Control Systems*, publication 848, 1988.
[12] IEC, International Electrotechnic Commission, *Programmable Controllers, Part 3: Programming Languages*, publication 1131-1, 1992.
[13] H. Lecocq, "Programmable logic controllers," in *Handbook of Manufacturing Automation and Integration*, Stark, Ed. Boston: Auerbach Publishers, Ch. VII.11, pp. 805–821, 1989.
[14] P. Le Guernic, A. Benveniste, P. Bournai, and T. Gauthier, "SIGNAL: A data flow oriented language for signal processing," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34 no. 2, pp. 362–374, Apr. 1986.
[15] J. Martinez and M. Silva, "A package for computer design of concurrent logic control systems," in *Symp. IFAC Software Computer Contr., SOCOCO*, Madrid, Sep. 1982, pp. 221–226.
[16] M. Moalla and R. David, "Extension du Grafcet pour la représentation de systèmes temps réel complexes," *J. RAIRO, Automat. Contr. Series*, vol. 15, no. 2, pp. 159–192, 1981.
[17] M. Moalla, "Réseaux de Petri interprétés et Grafcet," *Technique Sci. Informatiques*, vol. 14, no. 1, 1985.
[18] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, 1989.
[19] J. D. Otter, *Programmable Logic Controllers: Operation, Intrefacing, and Programming*. New York: Prentice-Hall, 1988.

[20] J. L. Peterson, *Petri Net Theory and the Modelling of Systems*. Englewood Cliffs, NJ: Prentice Hall, 1981.
[21] C. A. Petri, *Kommunikation mit Automaten, Schriften des Rheinisch, Westfalischen Institutes fur Intrumentelle Mathematik and Der Universitat Bonn*, 1962, translation by C. F. Greene, *Applied Data Research Inc., Suppl.1 to Tech. Report RADC-TR-65-337*, N.Y., 1965.
[22] M. Silva and S. Velilla, "Programmable logic controllers and Petri nets," *Symp., IFAC Software Computer Contr.*, Madrid, Sep. 1982, pp. 29–34.
[23] R. S. Sreenivas and B. H. Krogh, "On condition/event systems with discrete state realizations," *Discrete Event Dynamic Syst.*, vol. 1, no. 2, pp. 209–236, 1991.
[24] Telemecanique, *Terminal TSX Y607, Programmation PL7-3, Language Grafcet*, Télémécanique, 50/64 av. Arago, F92022 Nanterre cédex, 1986.
[25] S. H. Unger, *Asynchronous Sequential Switching Circuits*. New York: Wiley, 1969.
[26] R. Valette, "Analysis of Petri nets by stepwise refinements," *J. Computer Syst. Sci.*, no. 18, pp. 35–46, 1979.

**René David** was born in Saint-Nazaire, France, in 1939. He received the Engineering Degree from the Ecole Nationale Supérieure d'Arts et Métiers, the Automatic Control Engineering degree, and the Doctorat d'Etat ès-Sciences degree, both from the University of Grenoble, in 1962, 1965, and 1969, respectively.

He is now Directeur de Recherche at the Centre National de la Recherche Scientifique, working at the Laboratoire d'Automatique de Grenoble (in Institut National Polytechnique de Grenoble). He was Deputy Director of this laboratory from 1987 to 1990. His research interests include design of logic controllers, testing of digital circuits, and modeling and performance evaluation of manufacturing systems. He is the coauthor of *Du Grafcet aux réseaux de Petri* (Hermès, 1989) and *Petri Nets & Grafcet: Tools for Modelling Discrete-Event Systems* (Prentice-Hall, 1992).