

## The GRAFCET Specification Language

---

### 64.1 Introduction

### 64.2 The GRAFCET Standards

### 64.3 The GRAFCET Context

### 64.4 Basic Elements

Introduction • Structure • Interpretation • Evolution Rules  
• Tank Filling Example (I)

### 64.5 Advanced Elements

Sequence Structures • Variable Type Extensions • Internal  
Variables • Events • Time Representation • Action Types  
• Particular Structures • Graphical Composition • Tank Filling  
Example (II)

### 64.6 Hierarchical Grafcets

Forcing • Enclosure • MacroSteps • Tank Filling Example (III)

Acknowledgments

Paulo Portugal and Adriano  
Carvalho

*Faculdade de Engenharia da  
Universidade do Porto*

---

## 64.1 Introduction

Due to the complexity of automated systems and the increasing application requirements, the use of appropriate modeling tools becomes essential to achieve a correct system design. These tools must be capable of simultaneously specifying the system behavior in a concise and unambiguous manner, and of validating this specification. Support of a method to easily map the formal model into the final implementation, on possibly varying hardware, is also required.

For several years, the absence of a common modeling tool led system designers to use different methodologies, each supported by a specific tool. This approach resulted in a large number of problems, such as higher development costs, incompatible specifications, absence of formal validation techniques, and difficulties in translating specifications into implementations.

GRAFCET was initially proposed by several academic and industrial research groups, and later standardized by International Electrotechnical Commission (IEC), in the hope that it would be accepted as the common modeling tool. Its aim is to allow the designers to describe and specify a control system's sequential behavior.

This chapter presents a description of the GRAFCET language, from an automated systems' perspective. All aspects of the language are described, although the number of application examples was reduced

---

to a minimum due to space constraints. Interested readers are encouraged to refer to [3][5][6] for more detailed examples.

## 64.2 The GRAFCET Standards

---

In 1975, a working group called Logical Systems from AFCET (*Association Française pour la Cybernétique Economique et Technique*) decided to create a committee in charge of defining the requirements an automated system's modeling tool should fulfill. Having evaluated the state-of-the-art modeling tools of that time — flowcharts, petri nets, and state graphs — they concluded that a new more appropriate modeling language was required. In 1977, GRAFCET (*Graphe Fonctionnel de Commande Etapes-Transitions*), which is translatable to Functional Graph of a Step-Transition Command, was proposed to fill the gap [4].

In 1982, a French initiative created the first GRAFCET standard: NF C 03-190 — *Diagramme fonctionnel "GRAFCET" pour la description des systèmes logiques de commande*. Later, the widespread use of GRAFCET led IEC to promote a standardization process, which resulted in 1988 in the first international GRAFCET standard: IEC 848 Ed. 1 — *Preparation of function charts for control systems*. This document was later revised, and a second edition was published in 2001: IEC 60848 (2001) Ed. 2 — *Specification Language GRAFCET for Sequential Function Charts* [1].

Since the IEC 60848 Ed. 2 does not define how to implement GRAFCET in control systems, other initiatives were adopted to implement it as a programming language for programmable logic controllers (PLCs). These efforts resulted in the definition of the language SFC — *Sequential Function Chart*, which is part of the IEC 61131-3 (2001) *Programmable Controllers — Programming Languages* standard [2].

## 64.3 The GRAFCET Context

---

A system can be defined as a set of organized interacting components that work together to achieve some defined objective, which is to confer an added value — physical modification, particular arrangement, position change, data processing — to a group of raw materials — products, energy, data, people — in a specific environment. When the operations necessary to achieve the system objective are executed automatically under the control of computational resources, the system is defined as an *automated system*.

From a conceptual viewpoint, the structure of an automated system is usually considered as being composed of two cooperating parts: *operative* and *control* (Figure 64.1). The *operative part*<sup>1</sup> represents the physical process that creates the added value. It is composed of a set of elements such as sensors, actuators, machines, robots, transportation systems, etc. The *control part* is composed of computational resources that coordinate the sequence of operations performed by the operative part. Additionally, it interacts with other external systems in its environment, such as human-machine interfaces and communication networks. All interactions may be modeled by the sending of commands and the reading of data.

The implementation of an automated system requires a formal, accurate, and complete description of its behavior. Without this understanding, the system designer could introduce faults during the development and implementation phases, which could lead to serious problems during system operation. It is important to notice that automated systems are commonly found operating in very sensitive areas — nuclear power plants, railways, industrial facilities — where a design fault could have a catastrophic impact on people and the environment.

The description of any system component's behavior is performed by first defining a virtual interface that isolates it from the external elements with which it interacts, followed by characterizing the interactions that occur across that interface. Therefore, interactions with the control part, which control engineers will typically be focusing on, can be functionally represented by isolating it from other system elements. Data and commands exchanged with both the operative part and the environment are, respectively, represented by *input* and *output variables* (Figure 64.2).

---

<sup>1</sup>Also commonly referred to as the controlled part.

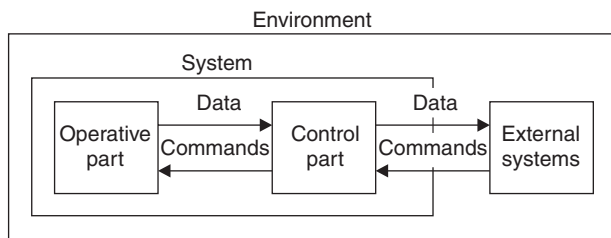


FIGURE 64.1 Automated system structure.

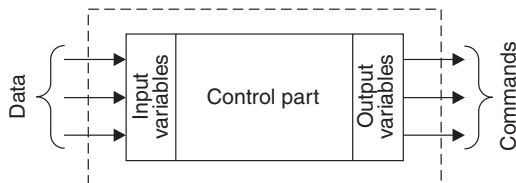


FIGURE 64.2 Control part interactions.

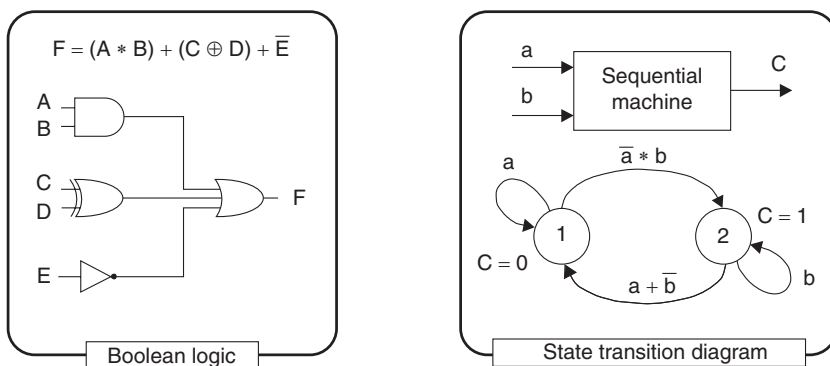


FIGURE 64.3 System behavior modeling.

The control part behavior may be described by the way in which the output variables depend on the input variables, which is necessarily different among distinct system types. Nevertheless, it is possible to find behavioral patterns common to every system and to use appropriate modeling tools to describe it. Among these patterns, two of them are distinguished by being present in any automated system and by representing the majority of interactions related to the control part — *combinational* and *sequential behaviors*. Both of them assume that system inputs and outputs are boolean variables, but other variable types can also be considered within specific contexts.

The *combinational behavior* is characterized by system outputs being a logical combination of their inputs at any given instant. Several modeling tools are available to describe this behavior — boolean logic, Karnaugh maps — which can be easily implemented (Figure 64.3).

The *sequential behavior* introduces the notion of state to describe the system evolution. At any given instant, the system outputs are logical functions of the inputs and the internal system state. This behavior can be modeled as a *sequential machine* by means of a *state transition diagram*.

The diagram is a simple directed graph consisting of circles and arcs, where circles represent the system states (Figure 64.3). At any given instant, only one state is active, representing the current system state, referred to as *situation*<sup>2</sup>. For each system state, a group of actions operating on the output variables

<sup>2</sup>Adopting the IEC 60848 Ed. 2 terminology.

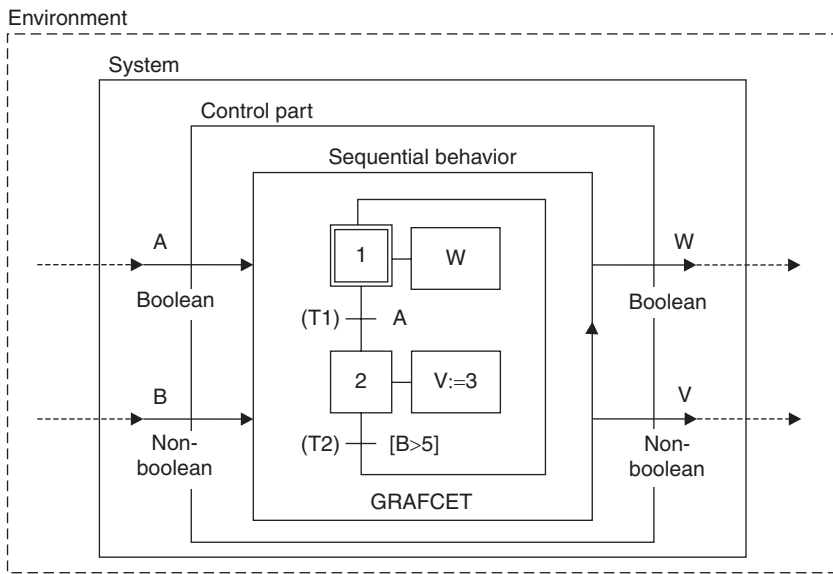


FIGURE 64.4 GRAFCET and system behavior.

may be defined, which are only executed while the state is active. The evolution between system states is modeled by directed arcs connecting the states. A boolean expression composed of input variables is associated to each arc, representing an evolution condition. When this condition is true, a transition from one state to another occurs. The definition of output variables and evolution conditions imply that the combinational behavior is embedded in the sequential behavior.

Although a state transition diagram could describe any type of sequential behavior, they are difficult to use when it becomes necessary to model behaviors commonly found in automated systems, which usually involve a certain degree of complexity, such as synchronization, parallelism, hierarchical relationships, and different detail levels.

In order to overcome these limitations, a new formalism is required, with the same descriptive power as sequential machines, but also capable of representing complex behaviors in a simple, concise, and intuitive way. It should also be well-adapted to represent the sequential evolutions of a system, which will potentially ease the mapping to the final hardware/software implementation. With all these properties, it becomes the appropriate tool for both automated system designers and end users.

These requirements resulted in the specification of the GRAFCET standard — IEC 60848 Ed. 2 [1].

GRAFCET's aim can be synthesized as to describe and specify, in a consistent and unambiguous manner, the sequential behavior of the system control part (Figure 64.4). The description is performed by using a graphical language composed of charts, which includes graphical elements to represent the system states and possible evolutions, associated with an alphanumerical representation of the system input and output variables. Therefore, it can be said that GRAFCET is a *behavior specification language* for sequential systems.

Although any kind of system that exhibits a sequential behavior can be modeled by using GRAFCET, until now its main application domains have been in the specification of control applications running on PLCs.

## 64.4 Basic Elements

### Introduction

In a sequential system at any given instant the output variables are established from the system state and the input variables<sup>3</sup>. Since GRAFCET models the behavior of these systems, it becomes necessary that it includes

<sup>3</sup>Both variables are assumed to be of the boolean type, unless otherwise stated.

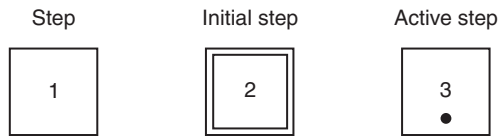


FIGURE 64.5 Step representation.

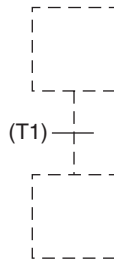


FIGURE 64.6 Transition representation.

all system data in its representation. This is accomplished by producing a chart (or group of charts), referred to as *Grafsets*<sup>4</sup>, using a set of graphical elements that represent different, but complementary, viewpoints of the system behavior. This representation distinguishes two perspectives: *structure* and *representation*.

## Structure

The structure models the system evolution between situations. It is represented as a graph composed of *steps* and *transitions*, disposed in alternating mode, which are interconnected by *directed links*.

### Steps

A *step* represents part of the system state. It may have two states: *active* or *inactive*. The set of active steps at any given instant represents the system, and *Grafcet*, situation at that instant.

A step is graphically represented by a square box, which is identified by means of a distinct label (Figure 64.5). The initial situation, *initial step*, represents the state of the system at the initial time ( $t = 0$ ). The corresponding steps are represented by a double box. During the *Grafcet* evolution, the active states, at any given instant, are represented with a small dot inside the box.

### Transitions

A *transition* represents a possible system *evolution* from one step to another, which is a change in the *Grafcet* situation. The evolution is performed by *clearing*<sup>5</sup> the transition. It is graphically represented by a thin horizontal bar (Figure 64.6), and it is identified by means of a distinct label, between brackets, placed to the left. This is known as its *designation*.

### Directed Links

A *directed link* is used to connect steps to transitions and vice versa. Therefore, two steps cannot be directly connected, unless a transition exists between them. This property is known as the *alternation step-transition* rule.

A directed link is graphically represented by a line, with the direction of evolution path, assumed from top to bottom (Figure 64.7). An arrow can be used to avoid misunderstandings.

<sup>4</sup>We have adopted “GRAF CET” to refer to the language in general, and “*Grafcet*” to refer to a particular chart model.

<sup>5</sup>Also commonly referred to as *firing*.

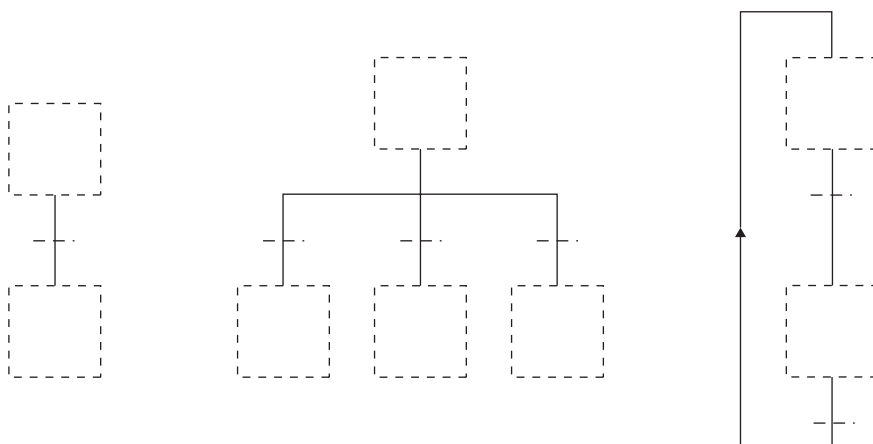


FIGURE 64.7 Directed link representation.

## Interpretation

The interpretation establishes the relationships between the input and output variables and the *Grafcet* structure. It is composed of alphanumerical elements — *transition-conditions* and *actions* — representing the functional aspect of GRAFCET.

### Transition-Conditions

A *transition-condition*<sup>6</sup> is a logical expression composed of input variables and/or *internal variables* (Section 64.5) representing an *evolution condition*. This condition is associated with a transition and establishes the necessary conditions to its clearing. Input variables are provided by both the operative part and the environment, while internal variables represent data internal to the system control part.

Transition-conditions are graphically represented by a logical expression placed at the right of the transition (Figure 64.8). A condition that is always true is represented as ‘1’.

### Actions

An *action* represents an operation performed on an output variable, corresponding to a command that is sent either to the operative part or to the environment. The action is executed only while the associated step is active. Several actions can be associated to the same step.

Actions are graphically represented by a rectangle connected to the associated step (Figure 64.9). A literal or symbolic label inside it designates the output variable that takes the true value.

## Evolution Rules

Since GRAFCET specifies the behavior of a dynamic system, it is necessary to define a set of rules that establish, without any ambiguity, the system evolution from one situation to another. The following five rules apply:<sup>7</sup>

- *Rule 1 — Initial Situation.* This situation is defined by the system designer and represents the state of the system at initial time ( $t = 0$ ).
- *Rule 2 — Clearing of a Transition.* A transition is cleared if and only if both the following conditions occur:
  - All immediately preceding steps, linked to the transition, are active. The transition is said to be *enabled*.
  - The condition associated with the transition is true.

<sup>6</sup>Also commonly referred to as *receptivity*.

<sup>7</sup>According to IEC 60848 Ed. 2.

- *Rule 3 — Evolution of Active Steps.* The clearing of a transition results in the simultaneous deactivation of all immediately preceding steps and the activation of all immediately succeeding steps.
- *Rule 4 — Simultaneous Evolutions.* All transitions that can be cleared simultaneously are simultaneously cleared.
- *Rule 5 — Simultaneous Activation and Deactivation of a Step.* If during rule 4 phase, an active step is simultaneously deactivated and activated, it remains active.

The systematic application of the previous rules to a *Grafcet* describes all possible evolution situations, which helps the system designer to verify if the model's behavior diverges from the initial specification. Application examples are presented in Figure 64.10.

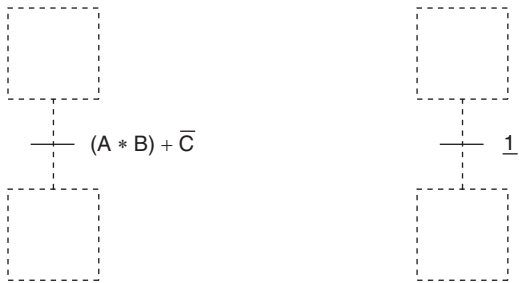


FIGURE 64.8 Transition-condition representation.

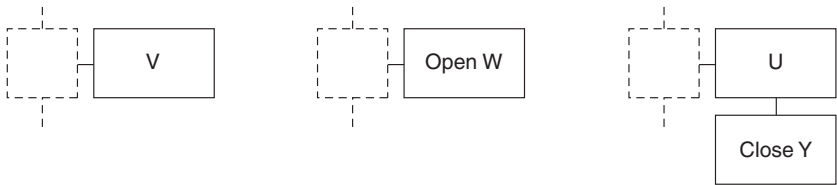


FIGURE 64.9 Action representation.

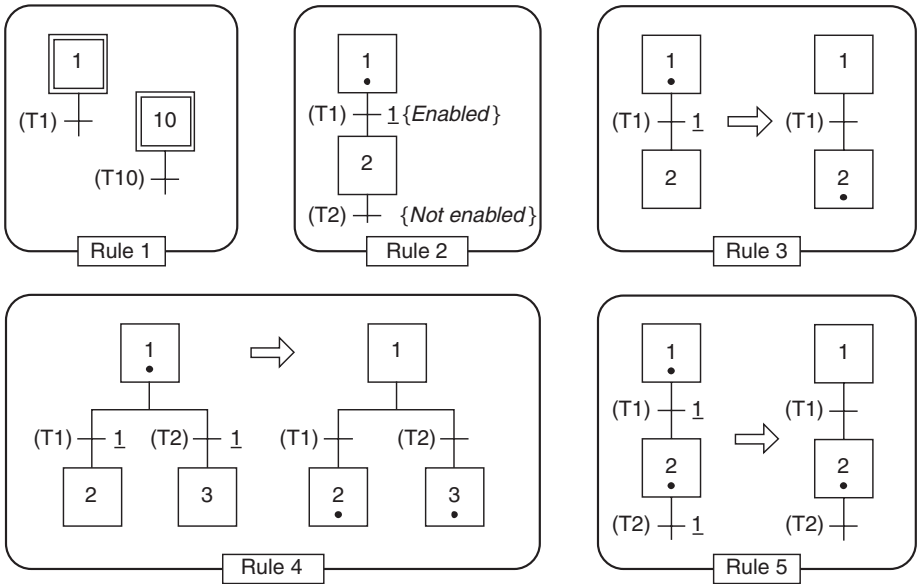


FIGURE 64.10 Evolution rules.

Unstable Situations

The application of the evolution rules, combined with the existence of particular situations, may lead the *Grafcet* to a scenario where an *unstable situation* will occur.

As an example, consider the *Grafcet* presented in Figure 64.11, where at a given instant the situation is defined by: **step 1** is active,  $A=0$  and  $B=1$ . Now assume that  $A$  changes to 1, initiating the *Grafcet* evolution (Figure 64.11). From the instant where  $A$  changes to 1, the *Grafcet* situation instantaneously changes from **step 1** active to **step 3** active. The simultaneous activation and deactivation of **step 2** is referred to as an *unstable situation*. Actions associated to steps in unstable situations are not executed, unless they are *stored actions* (Section 64.5).

Evolution Algorithm

The evolution rules establish a simple algorithm that defines the GRAFCET evolution:

- 1. Initialization (assumed stable). All initial steps are activated and the actions associated are executed.
- 2. Define the set of cleared transitions.
- 3. Clear all cleared transitions, until a stable situation occurs. During this phase, execute eventual stored actions associated with transitions clearings, step activations, and deactivations, including unstable situations.
- 4. Execute the actions associated with the active steps.
- 5. Goto to step 2.

Tank Filling Example (I)

The following example was chosen to demonstrate the basic GRAFCET capabilities, when used to specify the behavior of an automated system.

The system represented in Figure 64.12 is composed of a tank that must be filled with a liquid stored in a reservoir. Two valves (V and W), respectively, control the input and output of liquid in the tank. Two level sensors (H and L) monitor the tank level. A button (S) is used to start the system operation.

The tank is empty when the level is less than L (i.e.,  $L=0$ ) and is full when the level is greater than H (i.e.,  $H=1$ ). Initially, the tank is empty. When button S is pressed (i.e.,  $S=1$ ), the tank is filled by opening V and closing W. The filling operation stops when the tank is full, by closing V, and its contents are drained by opening W. When the tank is empty, W is closed. Filling may only restart when S is pressed again. The valves are open when the respective outputs are true, otherwise are closed.

A *Grafcet* that meets these specifications is also presented in Figure 64.12. It has three input variables: L, H, and S, and two output variables: V and W. Initially, the system waits for button S to be pressed. Only **step 1** is active — the initial step — without any associated action. Therefore, both valves will be closed. When the button is pressed, S becomes true, T1 is cleared, and **step 2** is activated. The associated action — open V — is executed. When H becomes true, transition T2 is cleared and **step 3** is activated. In this situation, W is opened and V is closed. The system is maintained in this situation until L becomes true. When this happens, T3 is cleared and **step 1** is activated, returning to the initial situation.

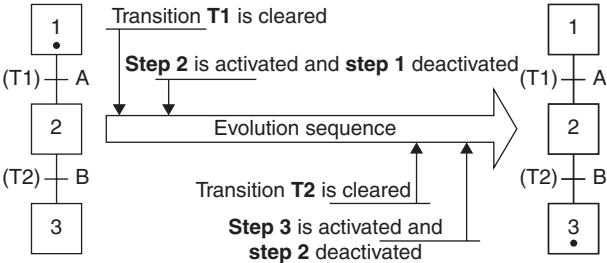


FIGURE 64.11 Unstable situation.



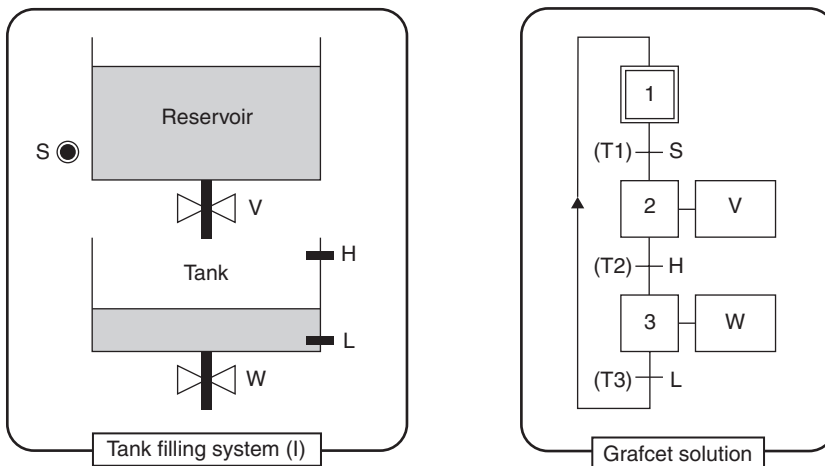


FIGURE 64.12 Tank filling example (I).

## 64.5 Advanced Elements

The GRAFCET elements already described are not a significant improvement to state transition diagrams. This may be easily understood considering that the GRAFCET basic elements are in fact based on these diagrams; therefore, the modeling capabilities must be similar. The introduction of a new set of advanced elements becomes necessary to enhance the GRAFCET modeling power.

### Sequence Structures

A sequence consists of a succession of steps, where each step has only one preceding and succeeding transition, and represents the simplest sequential behavior that can be found. However, the behavior of an automated system is frequently more complex, requiring the use of advanced modeling structures. GRAFCET provides this by defining extensions to the graphical notation that are able to represent those behaviors:

- *Selection.* This structure is intended to represent a choice between alternative sequences. Example (Figure 64.13): transitions T1, T2, and T3 are simultaneously enabled. One or more can be cleared simultaneously. Mutually exclusive transition clearing can only be achieved if their transition-conditions behave accordingly.
- *Parallelism.* This structure is intended to describe the behavior of parallel sequences. It is graphically represented as a double bar after a transition. Example (Figure 64.13): when transition T4 is cleared, step 5 is deactivated and steps 6, 7, and 8 are simultaneously activated. After their simultaneous activation, the evolution of the active steps in each parallel sequences becomes independent.
- *Synchronization.* This structure can be interpreted as the inverse of parallelism, since its intention is to join parallel sequences. It is graphically represented as a double bar before a transition. Example (Figure 64.13): transition T5 is only enabled when all previous steps (9, 10, and 11) are simultaneously active. When transition T5 is cleared, steps 9, 10, and 11 are simultaneously deactivated and step 12 is activated.

### Variable Type Extensions

The interactions between the control part and the remaining system elements have been formalized by means of boolean input and output variables. Nevertheless, in a real system not all the variables are of the

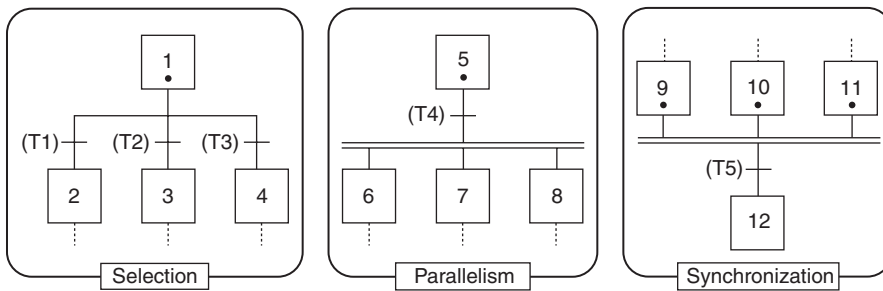


FIGURE 64.13 Sequence representation.

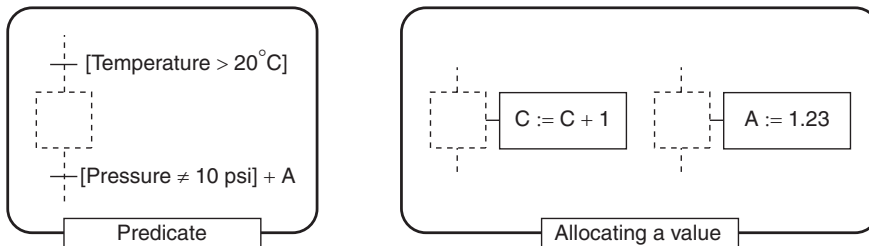


FIGURE 64.14 Predicate and allocation representation.

boolean type (e.g., temperature, pressure, counters, etc.). The system specification must include references to these non-boolean variables; otherwise, it would become inconsistent.

To overcome this limitation, GRAFCET provides an extension to the types of variables supported, without, however, establishing a formal definition of their type (e.g., integer, float, string) as classic programming languages do. Support for the extended types was achieved by naturally extending the use of boolean variables.

Since input variables are associated with transition-conditions, which are logical expressions, supporting these non-boolean input variables only requires that they be included as boolean *predicates*, graphically represented as a logical condition between square brackets (Figure 64.14).

Output variables are associated with actions. When an action is executed, a value is assigned to the variable. In the case of a boolean variable, the inclusion of its label is sufficient to establish its true value. Non-boolean variables are processed in a straightforward way, by *allocating* them a value (Figure 64.14).

## Internal Variables

Another extension related with variables reflects the possibility of accessing data that are internal to the *Grafcet* structure. This is performed by distinguishing between *external* and *internal* variables:

- *External variables* are those that are external to the control part — the previously defined input and output variables.
- *Internal variables* are associated with the GRAFCET structure. For each step, a boolean variable is defined with the following syntax: *Xstep\_label*. When the corresponding step is active the variable takes a true value; otherwise, it is false (Figure 64.15). These *step variables* have the same application scope as boolean input variables and they are used to access the step state. They become useful when synchronizing several *Grafquets* (Section 64.6).

The logical value of a boolean expression that comprises one or several boolean variables — input variables, predicate, step variables, partial *Grafcet* variables or macrostep variables (Section 64.6) — can be represented by a *logical variable*.

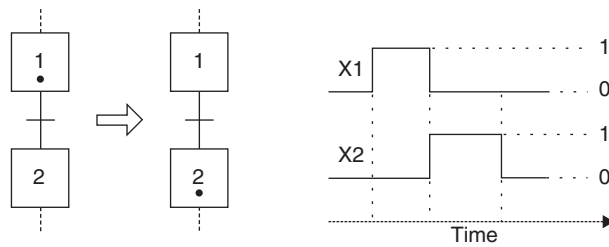


FIGURE 64.15 Step variable behavior.

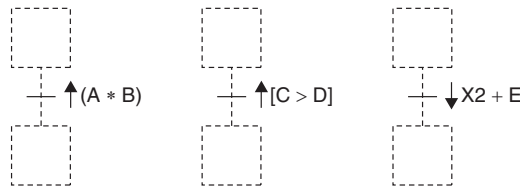


FIGURE 64.16 Input event representation.

## Events

At first, it may seem that the input variable definition is sufficient to represent all the conditions that enable the *Grafcet* evolution. Nevertheless, there are scenarios where this representation is insufficient — for example, how would it be possible to express the rising edge of a variable? To overcome this, GRAFCET introduces the notion of *event*. An event is characterized by the falling or rising edge of a variable and takes a boolean value.

It is possible to distinguish between two types of events:

- *Input events*, associated with the falling or rising edge of a logical variable. When the respective edge occurs, this event takes a true value; otherwise, it is false. Since this event is related to logical variables, it has the same application scope. It is graphically represented as a  $\uparrow$  (rising edge) or  $\downarrow$  (falling edge) symbol, followed by the associated logical variable (Figure 64.16).
- *Internal events*, associated with a step activation, deactivation, and transition clearing. These events are closely related with *stored actions*. They are interpreted as follows: when the internal event occurs, the associated action is executed. A detailed explanation is presented in the Stored Actions Section.

## Time Representation

Time plays a central role in the operation of automated systems, intervening in many aspects of their evolution. In order to track time, GRAFCET defines a *time-dependent condition*. This is a boolean condition related with an input or internal event by the notation: T1/variable/T2. The semantics is as follows: the condition is true after a time T1 of the occurrence of the rising edge of the logical variable, and becomes false after a time T2 from the occurrence of the falling edge (Figure 64.17). T2 may be omitted. The logical value of this condition can be represented by a *timed variable*.

Time-dependent conditions have the same application scope as standard boolean conditions — transition-conditions and timed *continuous actions*.

## Action Types

Actions represent the connection between the GRAFCET evolution and the system outputs. Although the action types presented until now are capable of describing the behavior of the most common output

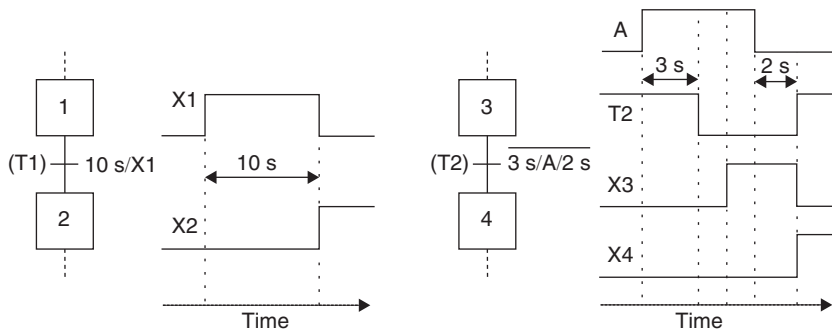


FIGURE 64.17 Time-dependent condition behavior.

types — on, off, assigning a value — they cannot represent more complex behaviors, such as timed or event actions. GRAFCET approaches this issue by extending the basic action definition into *continuous* and *stored actions*.

### Continuous Actions

Continuous actions are continuously executed while an associated *assignment condition* is true. This boolean assignment condition has the same characteristics of a logical or timed variable. When the assignment condition is true, and the step associated to the action is active, a true value is assigned to the boolean output variable referenced in the action. Otherwise, the output variable value is considered false. If the assignment condition is omitted, it is considered as always true, which corresponds to the basic action definition.

A continuous action is graphically represented as a basic action with the assignment condition placed at the right of a vertical line, which is connected to the action. Some application examples are presented in [Figure 64.18](#).

### Stored Actions

Stored actions are only executed when an event occurs. A value is allocated to the output variable indicated in the action whenever the event occurs. The output variable remains unchanged until a new allocation modifies its value. Typically, this kind of action is used to process non-boolean data such as incrementing/decrementing counters, computing expressions, starting of internal tasks, etc.

A stored action is graphically represented as basic action with symbols referring to the event type. Some application examples are presented in [Figure 64.19](#).

### Output Values

From the previous discussions, the output variables behavior results as follows:

*Continuous actions.* For a given situation, the value of the outputs variables is assigned:

- to the true value, for each output related to an action associated with an active step for which the assignment condition is true.
- to the false value, for the remaining outputs.

*Stored actions.* The output variable is allocated to a value when the associated event occurs, and remains unchanged until a new allocation is performed.

When an output variable is referred in more than one action, a conflict could occur if there are actions being simultaneously executed and if a contradictory assignment or allocation is performed. This problem can only be circumvented if the system designer takes the necessary measures to avoid it; otherwise, an inconsistent behavior results.

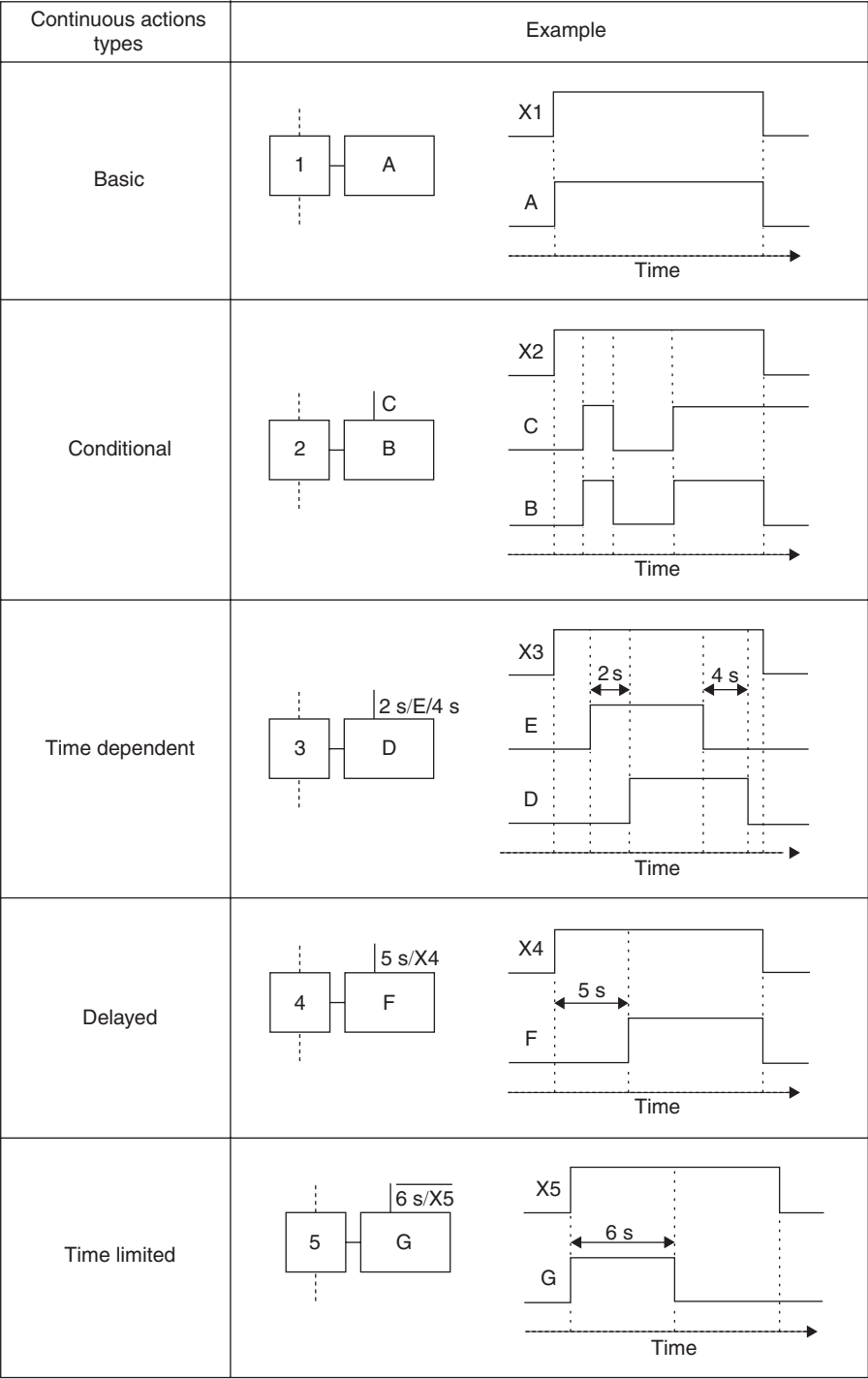


FIGURE 64.18 Continuous actions.

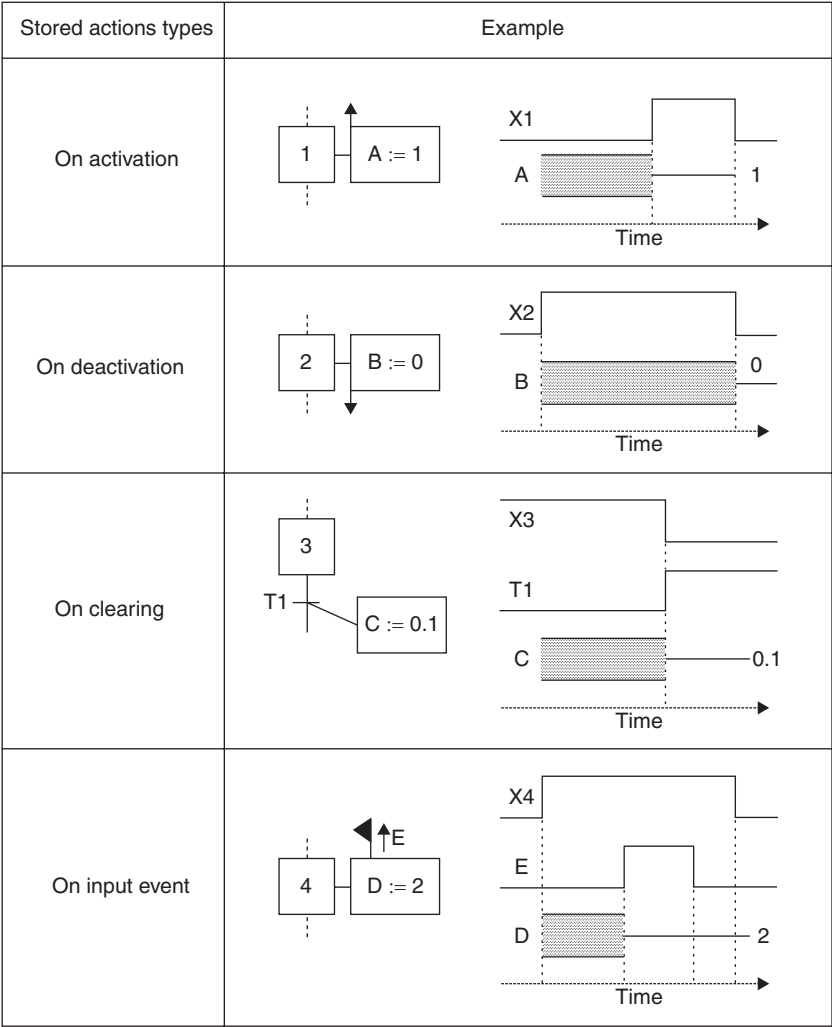


FIGURE 64.19 Stored actions.

### Particular Structures

Some GRAFCET structures represent an exception to the *alternation step-transition* rule (Figure 64.20). Their use is associated with the implementation of hierarchical *Grafjets* (Section 64.6), and can be classified as follows:

- *Source step*, is characterized by the absence of any preceding transition.
- *Pit step*, is characterized by the absence of any succeeding transition.
- *Source transition* does not have any preceding step. It is assumed that it is always enabled, being cleared when its condition-transition is true.
- *Pit transition* does not have any succeeding step.

### Graphical Composition

Like any other language, GRAFCET includes some elements that help the reader to have a clear description of the charts either textually, using *comments*, or graphically – using *page references* (Figure 64.21).

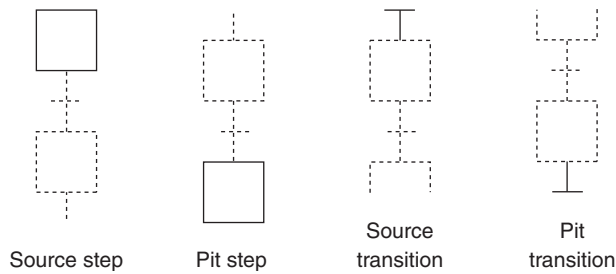


FIGURE 64.20 Particular structures.

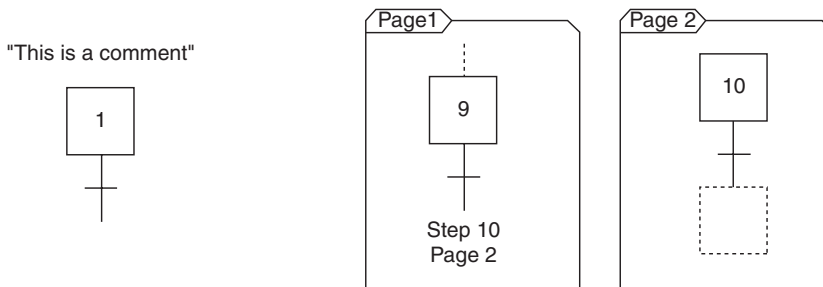


FIGURE 64.21 Graphical composition.

## Tank Filling Example (II)

Let us now consider the following extensions to the specifications of the tank system presented previously. The system is now composed of two tanks used in a similar way (Figure 64.22), with the following changes:

1. The filling process is triggered by the rising edge of S.
2. Both tanks are filled simultaneously.
3. Filling may only start up again when both tanks are empty.
4. V1 is closed only 2 seconds after H1 becomes active.
5. W2 is opened only 10 sec after H2 becomes active.

A *Grafset* that meets these specifications<sup>8</sup> is presented in Figure 64.22. Initially, with **step 1** active, the system waits for button S to be pressed. When this occurs, the event  $\uparrow S$  becomes true, and **steps 2** and **6** are simultaneously activated. This initiates two parallel sequences where both tanks are filled.

On the left sequence, **step 2** is activated and a stored action is used to open V1. When H1 becomes true, V1 is closed 2 seconds after that. This delay is represented by **step 3** and the time-dependent condition-transition T3. When T3 is cleared, V1 is closed by a stored action. Immediately, **step 4** is activated and the associated action, open W1, is executed. The tank 1 is maintained in this situation until L1 becomes true. When this happens, **step 5** is activated and W1 is closed. This is a waiting step required to allow synchronization between tanks.

In the sequence on the right, **step 6** is activated and the associated action, open V2, is executed. When H2 becomes true, **step 7** is activated and V2 is closed. This step has a delayed action, open W2, which is only executed 10 seconds after its activation. The tank 2 is maintained in this situation until L2 becomes true. When this happens, **step 8** is activated and W2 is closed. Just like with the sequence on the left, a waiting step is required to synchronize behaviors.

<sup>8</sup> Our aim is to present several alternative solutions, and not necessarily the simplest one.

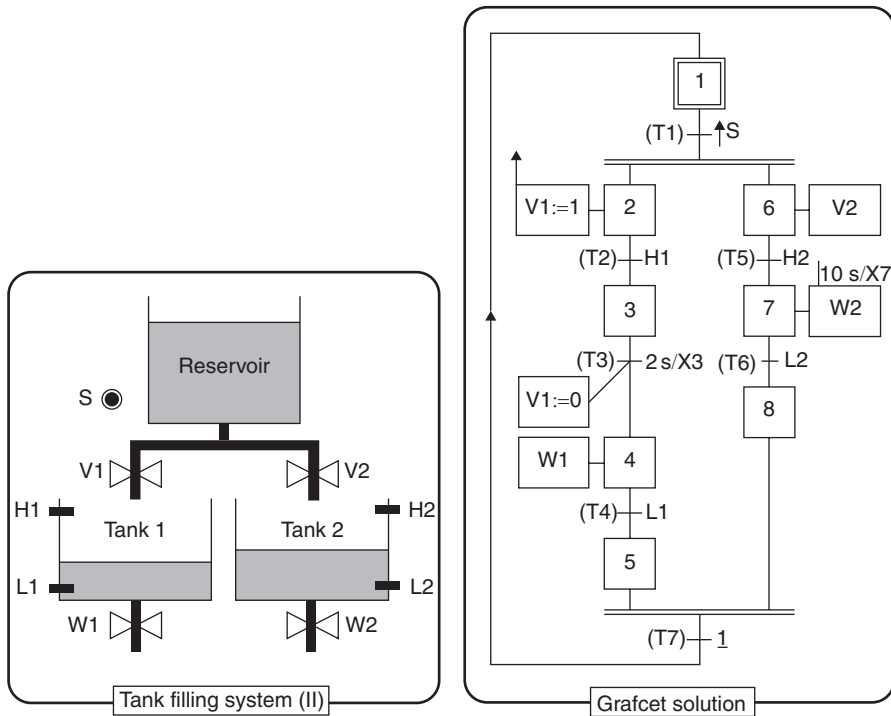


FIGURE 64.22 Tank filling example (II).

When both tanks are empty — steps 5 and 8 are both activated — the system immediately returns to the initial situation. This is achieved by using a synchronization structure associated to a transition, T7, with a condition that is always true.

## 64.6 Hierarchical Grafcets

Some automated systems present very complex behavior. In these cases, an attempt to model the behavior using a single abstraction level results in a model almost impossible to understand. To overcome these scenarios, system designers usually use a methodology based on a top-down approach, which decomposes a single complex system in several less complex subsystems. This results in several advantages, such as: (i) better specifications, since the behavior of any subsystem is easier to understand; (ii) definition of different detail levels, which helps to identify the relationships between subsystems; and (iii) reuse of specifications — if a subsystem is replicated, a single specification suffices.

In order to support this methodology, GRAFCET defines *hierarchical Grafcets*. Using this approach, a complex *Grafcet* is decomposed into smaller ones, *partial Grafcets*, each representing part of the system behavior. Since a hierarchical relationship exists between *Grafcets*, it is naturally asymmetrical, with high-level *Grafcets* sending commands to the lower ones, and accessing their internal situation. These mechanisms allow the *Grafcets* to synchronize their evolutions (Figure 64.23).

A partial *Grafcet* is formed by one or several *connected Grafcets*. A connected *Grafcet* is one where there is always a continuity of links between any pair of elements (steps or transitions). It is identified by means of a name, with the following syntax: *Gname*. Figure 64.24 presents two partial *Grafcets*, G1 with a single connected *Grafcet* and G2 with two connected *Grafcets*.

The global state of a partial *Grafcet* is represented by a boolean variable, *XGname*, which is true if at least one of their steps is active. Their internal situation is represented by *Gname{steps}*, where *steps* is a list of the currently active steps. Some application examples are presented in Figure 64.24.



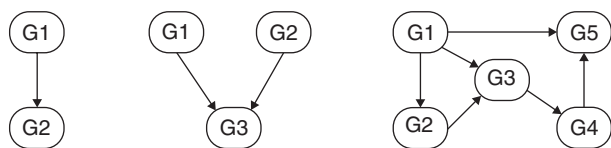


FIGURE 64.23 Hierarchical *Grafkets*.

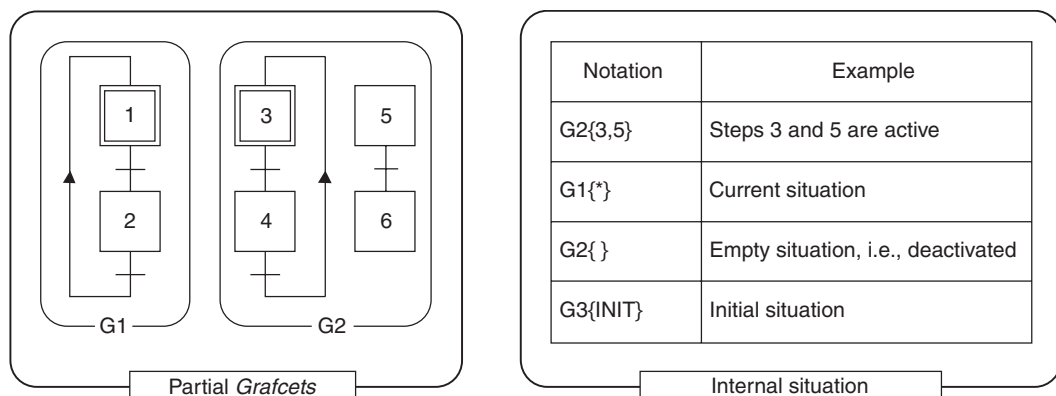


FIGURE 64.24 Partial *Grafket* representation.

As stated previously, structured *Grafkets* need to synchronize their evolution. The GRAFCET notation was therefore extended in order to allow a partial *Grafket* to modify the current situation of another partial *Grafket*. The following subsections present the three distinct mechanisms that were defined to support this feature.

## Forcing

With this approach, a partial *Grafket* forces another to a specific situation. This is accomplished by associating a *forcing order* to a step. While the step is active the order is continually executed. It has precedence over the evolution rules and during their execution the forced *Grafket* cannot evolve. This is referred as a *frozen Grafket*.

A forcing order is graphically represented as a basic action, but with a double rectangle to distinguish it. Some application examples are presented in [Figure 64.25](#).

## Enclosure

An enclosure is an alternative to forcing. It is more powerful, although a bit more complex. The concept is based on the definition of an *enclosure* — a set of steps that are enclosed in an *enclosing step*. The set of enclosed steps can be considered a *partial Grafket*.

The enclosing step has the same properties of a basic step and a similar graphical representation, but with an octagon inside it ([Figure 64.26](#)). The enclosure is graphically represented as partial *Grafket* inside a rectangle, which has two labels referring to the enclosing step and the enclosure name.

When the enclosing step is activated, the steps in the enclosure identified by an *activation link* are also activated. This activation initiates the evolution of the partial *Grafket*. When the enclosing step is deactivated, all steps in the enclosure are also deactivated. The activation link is graphically represented by an asterisk “\*” at the left of the enclosed steps ([Figure 64.26](#)). This behavior is analogous to the release of a task in a programming language.

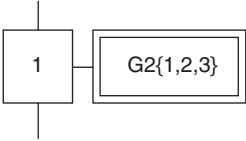
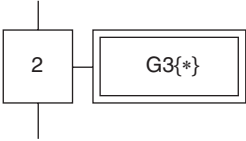
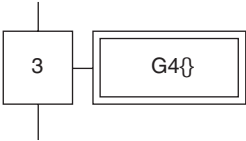
Forcing orders types	Description
	While step 1 is active, Grafcet G2 is forced to a situation where steps 1, 2, and 3 are active.
	While step 2 is active, Grafcet G3 is forced to the same situation it was in when step 2 became active.
	While step 3 is active, Grafcet G4 is deactivated, i.e., it is forced to the empty situation.

FIGURE 64.25 Forcing orders.

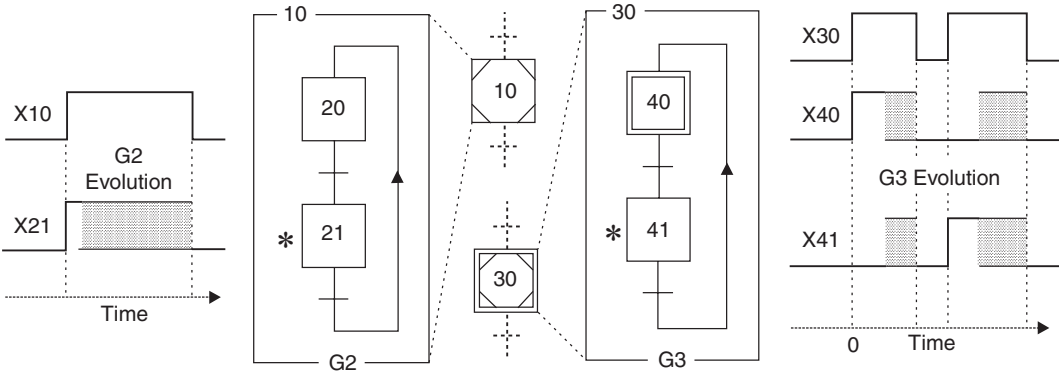


FIGURE 64.26 Enclosure representation.

An enclosure participates in the initial situation if, at least, one of the enclosed steps in each of its enclosures is also an initial step. In this scenario ( $t = 0$ ), only the initial steps are activated (and not the activation links), the enclosure evolution being defined by the standard *Grafcet* rules. However, if the initial enclosing step is deactivated, the enclosure is also deactivated. Posterior reactivations of the enclosing step follow the defined enclosure activation rules (Figure 64.26). Its graphical representation results from merging both the initial and enclosure step representations.

An enclosing step can contain several enclosures simultaneously, which enables the parallel execution of several partial *Grafcets*. An enclosure can itself contain other enclosures, enabling the modeling of hierarchical abstraction layers.

### MacroSteps

A partial *Grafcet* may be encapsulated by a special step, referred as a *macrostep*, which when activated, is expanded and substituted by the partial *Grafcet* (Figure 64.27). This encapsulation allows a gradual

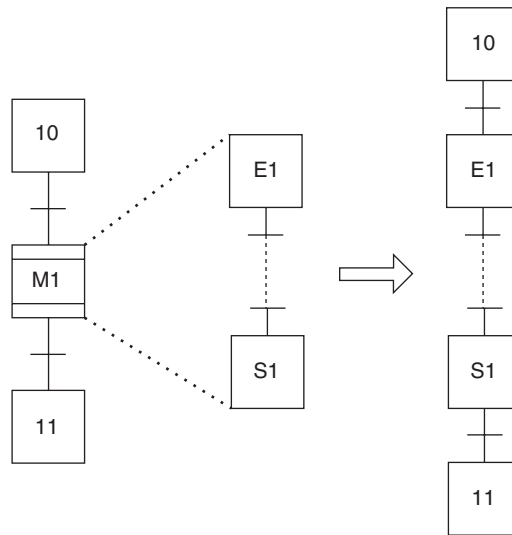


FIGURE 64.27 Macro-step representation.

description from general to particular, with a behavior analogous to the execution of a subroutine in a classical programming language.

To perform this expansion, the partial *Grafcet* must have a particular structure, being initiated by an *entry step*, and terminated by an *exit step*. The entry step becomes active when one of the preceding transitions of the macrostep is cleared. The succeeding transitions of the macrostep are enabled when the exit step becomes active. The expansion of a macrostep can have initial steps, and can recursively include other macrosteps.

A macrostep is graphically represented as a basic step with two horizontal bars inside, and is identified by means of a label with the following syntax: *Mlabel*. Their global state is represented by a boolean variable, *XMlabel*, which is true if at least one of the encapsulated steps is active.

### Tank Filling Example (III)

Consider the following extensions to the system specified in Tank filling example (II):

- (i) A suspend–resume button (SR) that when activated suspends the filling operation. If SR is once again activated, the operation is resumed.
- (ii) An emergency button (EM) that when activated forces the system to the initial situation. This button has priority over all others.

The *Grafcet* that meets these specifications is presented<sup>9</sup> in Figure 64.28. The system behavior is structured through the use of three *Grafcets* in a hierarchical relationship — G1, G2, and G3.

The *Grafcet* G1 is at the highest level, and related to the operation of the emergency button (EM). On the second level, *Grafcet* G2 describes the operation of the suspend–resume button (SR). Finally, *Grafcet* G3 describes the tank filling operation. Initially steps 1, 21, and 31 are active. This initiates the independent evolution of *Grafcets* G1, G2, and G3.

When the emergency button (EM) is pressed, step 2 is activated, which disables the enclosure G2 and executes a forcing order on *Grafcet* G3, forcing it to its initial situation. During this period, any activations of the start button (S) or the suspend–resume button (SR) are ignored, since G3 is frozen and G2

<sup>9</sup> Our aim is to present several alternative solutions, and not necessarily the simplest one.

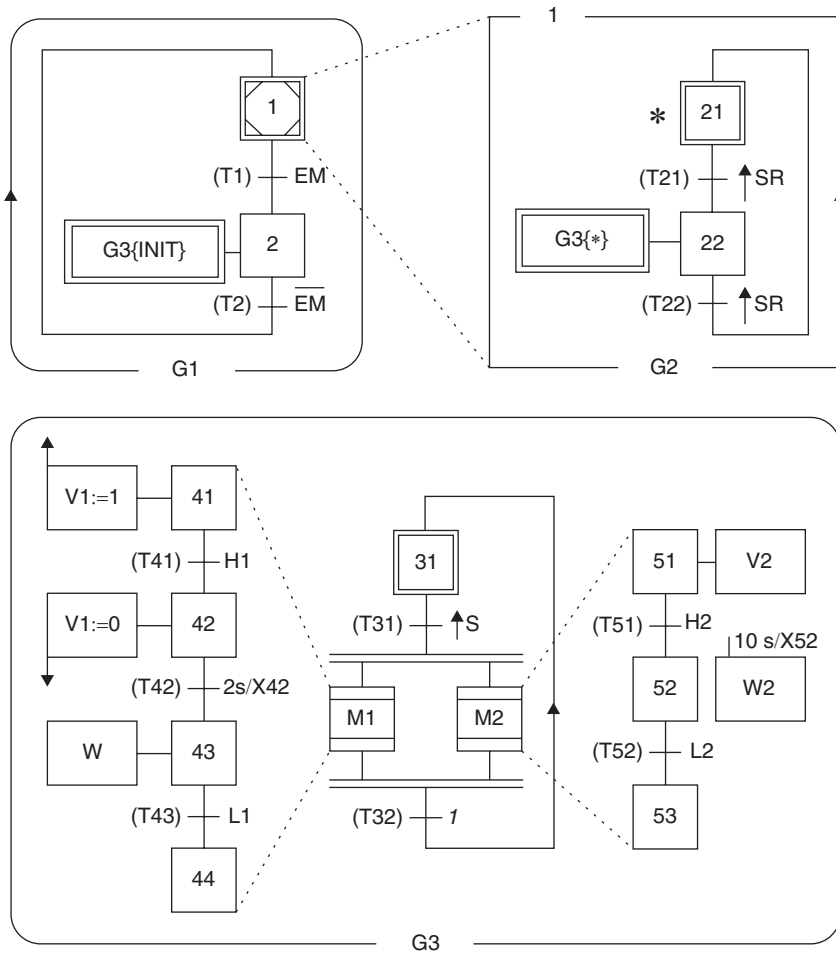


FIGURE 64.28 Example (III): *Grafset* solution.

is deactivated. When the emergency button (EM) is released, the system returns to the initial situation. Since **step 1** is also an enclosing step, its activation initiates the evolution of enclosure **G2** by activating **step 21**.

If during normal system operation the suspend–resume button (SR) is pressed, **step 22** is activated. Since its associated action is a freezing order, *Grafset* **G3** cannot evolve and maintains the current situation. If during this order it is necessary to disable all output variables, then the use of *conditional actions* must be employed. Therefore, the internal variable XZZ should be used as an *assignment condition* of the actions in steps 41, 42, 51 and 52. For clearance reasons this is omitted. When the suspend–resume button (SR) is pressed once again, *Grafset* **G3** is allowed to resume its normal evolution.

*Grafset* **G3** has a structure similar to the example presented in the Tank filling example (II) section, with the exception that the parallel sequences were replaced by macrosteps.

## Acknowledgments

The authors wish to thank Mário de Sousa for his contribution, including discussions and revision of the document. He also has contributed a chapter to this book — Programming with the IEC 61131-3 Languages and the MatPLC.

## References

- [1] IEC 60848 Ed. 2, Specification language GRAFCET for sequential function charts, International Electrotechnical Commission, 2001.
- [2] IEC 61131-3, Ed. 2, Programmable Controllers — Programming Languages, Final Draft, International Electrotechnical Commission, 2001.
- [3] David, R. and Alla, H., *Petri Nets and Grafcet — Tools for Modelling Discrete Event Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1992.
- [4] Home of Grafcet, <http://www.lurpa.ens-cachan.fr/grafcet.html>
- [5] Reeb, B., *Développement des grafjets*, Éditions Ellipse, Paris, 1999 (in French).
- [6] Grepa, *Le Grafcet, de nouveaux concepts*, Cepadues Éditions, Toulouse, 1985 (in French).