Hierarchical approach to GRAFCET using forcing order

Jean-Jacques Lesage, Jean-Marc Roussel

Hierarchical approach to GRAFCET using forcing order

Jean-Jacques Lesage, Jean-Marc Roussel

Laboratoire Universitaire de Recherche en Production Automatisée (L.U.R.P.A.)
ENS de Cachan, 61 avenue du Président Wilson
94235 Cachan Cedex - France
Tel. 33 1 47 40 22 15, Fax. 33 1 47 40 22 20
emaillesage@lurpa8.ens-cachan.fr

ABSTRACT. In 1987, the function chart "GRAFCET" was endowed with two new concepts intended to supplement the IEC 848 Standard : the macro-step and the forcing order. In this paper, our field of interest is the forcing order and more particularly the hierarchy between partial grafcets which is engendered.
Initially, we describe the basic principles of the forcing order and of the hierarchical structure of a global grafcet. A method, which uses graph theory, is then developed. The proposed algorithm is intended to prove the coherence of forcing orders. If a hierarchical incoherence is detected, forcing orders involved are identified.

KEYWORDS : function chart "GRAFCET", forcing order, forcing hierarchy, graph theory, identification of circuits.

1 Introduction

Industrial experience shows that the use of GRAFCET improves the description of the functional specifications of automated systems control. Since 1984, AFCET and ADEPA have worked together in order to facilitate the modelling of complex control systems. In a synthesis document [AFC.87], the French working group "AFCET Logical Systems" proposes two concepts - often called "extensions" of GRAFCET because they are not included in the IEC 848 Standard [IEC.88] - the macro-step and the forcing order.

Complex systems may be represented on several levels by "macro-representation" depicting the functions to be performed, without the analyst

worrying about the superfluous details on this level of description. The macro-step permits such a description which is compatible with the spirit of GRAFCET.

The forcing order allows us to impose the situation of a grafcet from the situation of another grafcet. It is the easier to model the systems that comprise a decisional hierarchy.

After five years of industrial and university use, we can affirm that these two extensions have profoundly consolidated the modelling power of GRAFCET and offer interesting possibilities for the structuring and the hierarchisation of the models.

However, if the use of the macro-step does not present any difficulty, the modelling of strongly hierarchical systems with several decision relations translated by forcing orders introduces an important problem: *how to verify the coherence of the forcing hierarchy between partial grafcets ?*

This verification must be automated and implanted into GRAFCET editors to permit the verification of important models. Without computer assistance, this operation is humanly impossible.

In this paper, we describe a method using the graph theory. This method brings the proof of hierarchical coherence or identifies forcing orders which are incoherent. An algorithm is given for implantation.

Firstly, the principles, the rules and the hierarchical implications of the forcing order will be given.

2 Forcing order : principles and rules

2.1 Definitions [UTE.92]

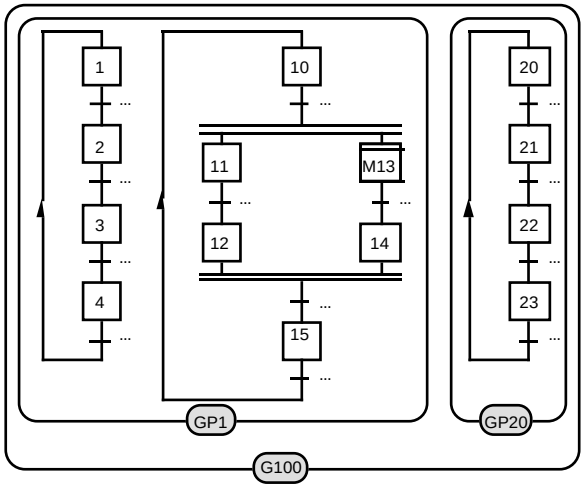
The following definitions are excerpts from UTE Project Ref. C03-191-E. This document is intended to supplement the publication UTE C03-190. It is to be submitted to IEC as a new proposal for study in view of publishing a practical and theoretical supplement to the IEC 848 publication.

2.1.1Partitions of a grafcet

A *"connected grafcet"* is a grafcet such that there always exists explicit directed links between any two elements (step, macro-step or transition).

A *"partial grafcet"* is a set of one or several connected grafcets.

The *"global grafcet"* of a system consists of all partial grafcets defining the behaviour of the controlling unit. Each connected grafcet of a global grafcet belongs to one and only one partial grafcet.



the grafcet GP1 consists of steps 1, 2, 3, 4, 10, 11, 12, M13, 14 and 15.

the grafcet GP20 consists of steps 20, 21, 22, and 23.

Figure 1.*Global grafcet G100 consisting of partial grafcets GP1 and GP2*

2.1.2Situations of a grafcet

The situation of a grafcet defines the activity of all its steps for a considered instant. It is characterized by the exhaustive list of its active steps at this instant and/or by the following notations :

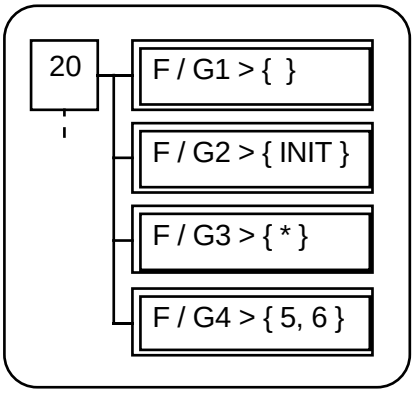
The *"Present situation"* of a grafcet may be noted {*}. It corresponds to the whole of its active steps at the considered instant.

The *"Empty situation"* of a grafcet noted { }, is the situation in which none of its steps is active.

A *"Given situation"* of a grafcet noted {i, j, ...}, is the situation in which the steps {i, j, ...} are solely active.

The *"Initial situation"* of a grafcet noted {INIT}, is the situation in which the initial steps are solely active.

The proposed symbol associated with forcing orders is shown below.



Forcing orders associated with STEP20

→ forces grafcet G1 to its empty situation.

→ forces grafcet G2 to its initial situation.

→ forces grafcet G3 to its present situation.

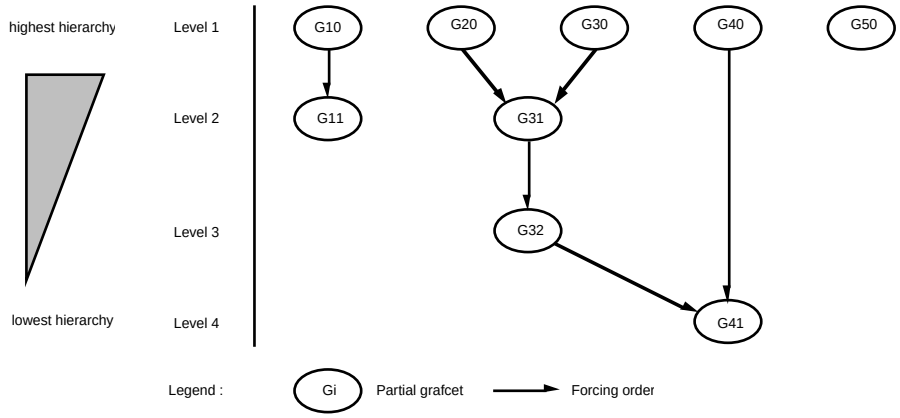
→ forces grafcet G4 to its given situation.

Figure 2.*Symbol associated with forcing orders*

2.1.3Forcing the situation of a grafcet

A forcing order makes it possible to impose the situation of a *partial grafcet* from the situation of another *partial grafcet.*

Writing a forcing order entails a different hierarchical level between the grafcet transmitting the order and the forced grafcet.



Note : As the partial grafcets {G10, G11} on the one hand, and the partial grafcet {G50} on the other, are not linked by a forcing order, no hierarchical forcing level could result.

Figure 3. *Representation of hierarchical levels due to forcing order*

Forcing order is an "internal order" having priority over evolution rules [G7W.92] : a forced grafcet can not evolve as long as the forcing order is present.

2.2 Hierarchical implications of forcing order

The writing of a forcing order entails a hierarchical relation between the two grafcets. The grafcet transmitting this order must be higher than the forced grafcet to represent the hierarchy of decision translated by the forcing order. One immediate consequence is that if the partial grafcet Gi "forces" the partial grafcet Gj, then Gj can not "force" Gi or any other grafcet superior to Gi.

The hierarchical structure of a global grafcet is then a consequence of the writing of forcing orders and it only is meaningful in relation to the forcing order. This structure is often called a partial order.

3 How to prove the coherence of a forcing hierarchy

3.1 Terms of the problem

Consider a representation of the hierarchical structure of a global grafcet (Figure3). This structure is coherent - from the point of view of the forcing hierarchy - if and only if no grafcet "forces" any higher grafcet.

Proving the coherence of a hierarchical structure consists in showing that there does not exist a "forcing loop" (e.g. in Figure 3, if G41 "forces" G31, the forcing hierarchy will be incoherent because a "forcing loop" will exist {G31, G32, G41, G31}).

Detecting possible incoherence consists in identifying these loops.

Thus expressed, this problem can easily be translated into a graph theory problem. To do this, every global grafcet can be associated to a graph where every vertex is an image of a partial grafcet and every oriented arc is an image of a forcing order. Arcs are oriented according to this rule:

- the initial extremity is the vertex image of the grafcet transmitting the forcing command,

- the terminal extremity is the vertex image of the forced grafcet.

In Figure 4, there is the translation of two propositions used for the resolution of our problem, in the consecrated language of graph theory [TUT.84] [CHR.75].

"GP1 is hierarchically superior to GP2"	becomes ⇒	*"There is a path from the vertex GP1 to the vertex GP2"*
"GP1 forces a partial grafcet which is hierarchically superior"	becomes ⇒	*"There is a circuit going through the vertex GP1"*

Figure 4. *Translation of two essential propositions of our problem into graph theory vocabulary*

A *path* in a directed graph is a sequence of arcs where the final vertex of one is the initial vertex of the next.

A *circuit* is a path in which the initial vertex of the first arc coincides with the final vertex of the last arc.

Our problem can now be expressed in these terms (from the point of view of the forcing hierarchy):

- if the graph is without any circuit, then all forcing orders are coherent between themselves,

- If there are circuits, then the arcs of these circuits represent the incoherent forcing orders involved.

The resolution of our problem is therefore essentially the search for and the identification of circuits in a graph.

3.2 Description of our method

In the field of graph theory, a lot of research has been related to the development of computation algorithms. The main algorithms connected with the search for circuits in a directed graph are essentially based on the existence of *strongly connected components*[1] in this graph. In particular, R.Tarjan proposed a very efficient algorithm for identification of these strongly connected components [TAR.72].

1 Let G be a directed graph. Suppose that for each pair of vertices v, w in G, there exist paths p1 (from v to w) and p2 (from w to v). Then G is said to be strongly connected. A strongly connected component of G is a strongly connected subgraph of G which is not enclosed within another strongly connected subgraph.

Moreover, the circuits we are searching for are generally non *Hamiltonian*[2] circuits. Consequently, we can not use classical results like the method of Latin composition (Kaufman & Malgrange) [KAU.63].

Classical methods seem then not to be optimal because of our objective (to prove the non existence of circuits in a general case) and the very specific topology of our graphs (most of the arcs do not belong to any one circuit).

The main idea of our method is to delete each arc of the graph when it is established that it can not be included in a circuit.

This approach procures us a double advantage : first, a very fast convergence to the final result, and second, an iterative method applied on more and more simple graphs in which the identification of circuits is made after their existence has been proved.

The general principle of our method is given in the following algorithm. The procedures written using italic characters are developed in the next sections of this paper.

```
BEGIN
      Construct the graph which represents forcing orders
      Reduce the graph
      IF the graph has no arc THEN
            the hierarchical structure is coherent (No detected circuit)
            Calculate the hierarchical levels
      ELSE
            there are circuits
            DOUNTIL the graph has no arc
                  Choose a vertex with a maximum degree
                  Search for circuits which pass by this vertex
                  Delete incident arcs to this vertex
                  Reduce the graph
            ENDO
      ENDIF
END
```

The principal sequence of this algorithm consists in detecting if there are circuits in the graph. To do that, we reduce the graph by suppressing the arcs which can not be included in a circuit (cf § 3.2.1). This reduction is iterative.

If the final result of this reduction is a graph without arcs, then the initial graph does not have a circuit. The hierarchical structure of the global grafcet can be calculated (cf § 3.2.3). In the opposite case, the circuits which are in the graph must be identified (cf § 3.2.2).

[2] A circuit is called *Hamiltonian* if it passes through every vertex in the graph.

3.2.1The reduction of the graph

An arc Ui can not belong to a circuit if any other arc Uj has for its terminal extremity the initial extremity of Ui **OR** if any other arc Uj has for its initial extremity the terminal extremity of Ui.

To delete those arcs, the procedure of reduction explores successively all non-null degree vertices (these vertices have at least one incident arc). For every one of these vertices V, the *in-degree* (number of arcs of which V is the terminal extremity) and the *out-degree* (number of arcs of which V is the initial extremity) are calculated. If the in-degree or the out-degree is null, all the vertices incident to V are deleted.

The procedure of reduction is reiterated as long as arcs can be deleted.

```
BEGIN
      DOUNTIL no arc can be deleted
            FOR each non-null degree vertex V
                  IF the in-degree OR the out-degree of V is null THEN
                        Delete incident arcs to V
                  ENDIF
            ENDFOR
      ENDO
END
```

3.2.2Identification of circuits

Now, we are going to identify elementary circuits going through a considered vertex. The choice of the maximal degree vertex allows us to delete the maximum amount of arcs during the next phase.

The identification of circuits begins with the identification of maximal elementary paths starting from the considered vertex.

Elementary paths are obtained step by step by searching to extend each old elementary path.

For that, it is necessary to verify that the initial extremity of the arc is the same as the terminal extremity of the path (In order to prove that it is really a path) and to verify that the terminal extremity of the arc is not the terminal extremity of one of the arcs of the current path (in order to prove that it is an elementary path). Initial paths are constructed with arcs which are incident to exterior to the considerated vertex.

After the construction of all the elementary paths, the identification of circuits is easy because all that is needed is to save each elementary path in which the initial extremity is the same as the terminal extremity.

After the identification of circuits going through a considered vertex V, it is possible to delete incident arcs to V because they can not be included in other

circuits. Indeed, only circuits going through G can contain the incident arc to this vertex.

3.2.3Calculation of the hierarchical levels

When all forcing orders are coherent between themselves, it is possible to define automatically the hierarchical structure for the global grafcet. This definition takes place when the hierarchical levels are calculated. For that, it is necessary to start from the graph representing the forcing orders.

The level of a partial grafcet is immediately superior to the level of its highest-level predecessor[3]. If a partial grafcet has no predecessor, its level is 1. The hierarchical levels are numbered from 1 to n by decreasing hierarchy [AFC.87] (Figure 3).

```
BEGIN
      FOR each partial grafcet G
            level of G = + ∞
      ENDFOR
      DOUNTIL a level is + ∞
            FOR each partial grafcet G of which the level is + ∞
                  level of G = 1
                  FOR each predecessor P of G
                        level of G = SUP {level of G, (level of P) +1}
                  ENDFOR
            ENDFOR
      ENDO
END
```

4 Example

To illustrate this method, we will apply it on a voluntarily simple example. Consider the global grafcet (Figure 5). Nine partial grafcets are described and there is a structure of decision with twelve forcing orders.

[3] P is a predecessor of V if there exists an arc which has P as its initial extremity and V as its terminal extremity.

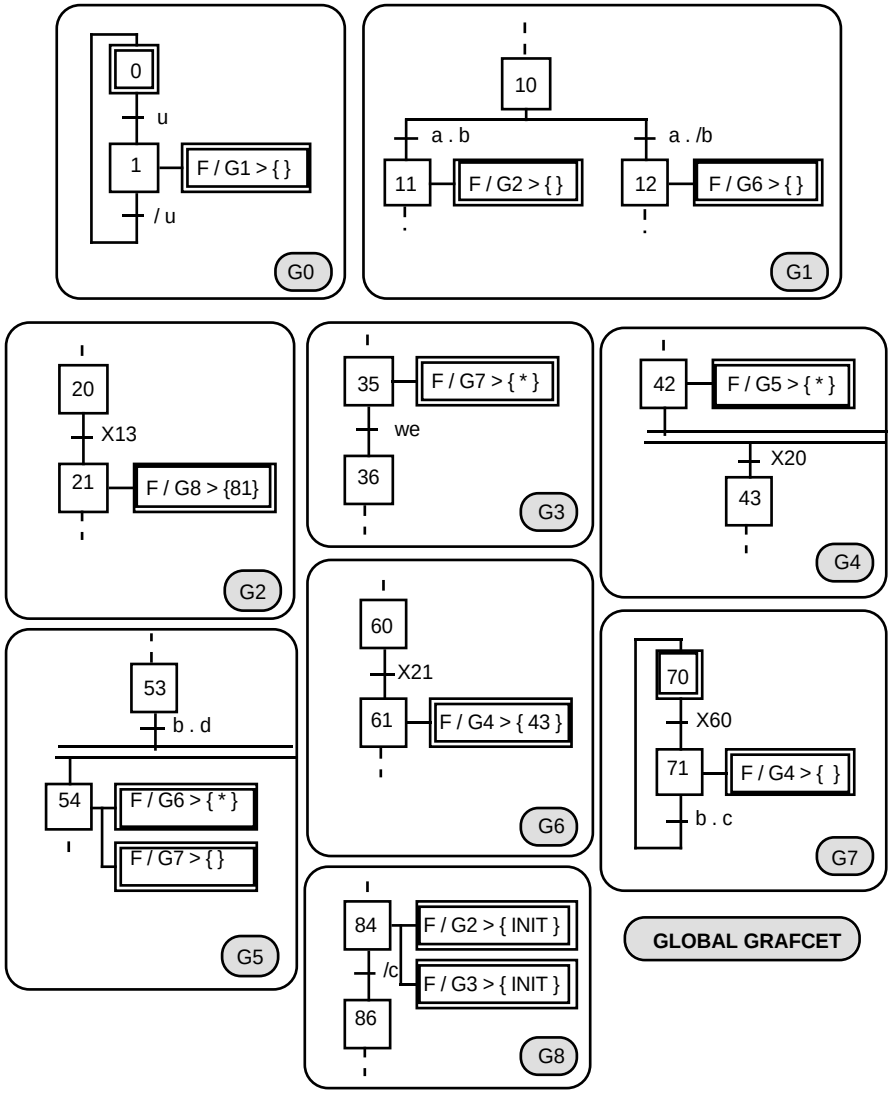
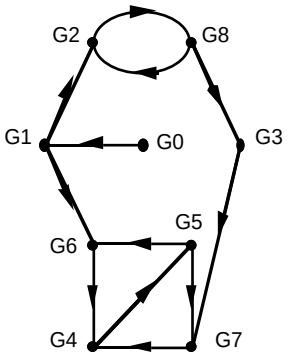


Figure 5. *An example of forcing order utilization.*

Now, we will run through the proposed algorithm and present the results of the principal stages.

First stage

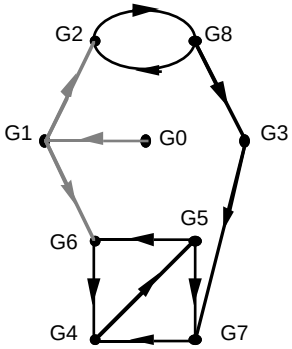
Construction of the graph



The graph comprises nine vertices (one for each partial grafcet) and twelve arcs (one for each forcing order).

Second stage

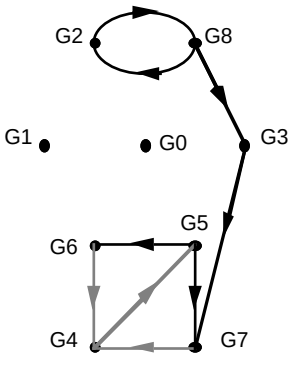
Reduction of the graph



The arc (G0,G1) and then the arcs (G1,G2) and (G1,G6) are deleted. The result of this reduction is not a graph without arcs. Consequently we are certain that circuits exist.

Third stage

Search for circuits going through the highest degree vertex and deletion of arcs which are incident to it.



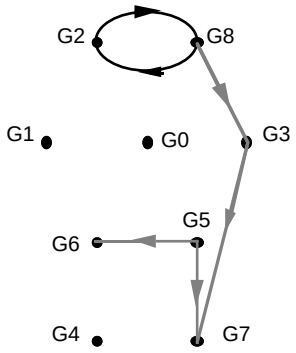
The chosen vertex is G4.

Two circuits are identified.
{(G4,G5), (G5,G6), (G6,G4)}
{(G4,G5), (G5,G7), (G7,G4)}

The arcs (G4,G5), (G6,G4), (G7,G4) are deleted.

Fourth stage

Reduction of the graph

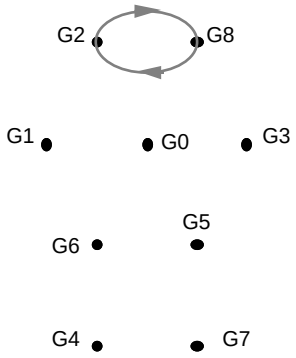


The arcs (G5,G7), (G5,G6) and (G3,G7), then the arc (G8,G3) are deleted.

The result of this reduction is not a graph without arcs. Consequently we are certain that other circuits exist.

Fifth stage

Search for circuits going through the highest degree vertex and deletion of arcs which are incident to it.



The chosen vertex is G2.

One circuit is identified. {(G2,G8), (G8,G2)}

The arcs (G2,G8), (G8,G2) are deleted. The result of this reduction is a graph without arcs. Consequently no more circuits exist.

Finally, three circuits have been identified:

{(G4, G5), (G5, G6), (G6, G4)}
{(G4, G5), (G5, G7), (G7, G4)}
{(G2, G8), (G8, G2)}

The global grafcet has then seven forcing orders incoherent between themselves.

5 Conclusions

As regards the implementation, of this algorithm, two distinct strategies are considered:a systematic invocation to prove the coherence after the writing of all forcing orders or an occasional invocation when the analyst decides it.

The first technique brings "in-line" assistance to the analyst and consist in verifying that the new forcing order is compatible with the others. The second provides "out-line" assistance and proves that a global grafcet is coherent.

Through the deep algorithm described in this article is destined for "out-line" utilization, we tested these two strategies with complex cases (several dozen partial grafcets and several dozen forcing orders).

These tests have been successfully carried out with experimental software developed in C language.

So far, we have dealt with the search for a hierarchical coherence of forcing orders. On-going research is trying to prove temporal coherence as well. Indeed, a partial grafcet may be simultaneously bound by several forcing orders. In this case, it is necessary to establish that the involved forcing orders do not contradict each other.

References

[AFC.87] French AFCET working group, «Formalisation d'extensions du GRAFCET : macro étape et forçage», AFCET Ed., 1987

[CHR.75] N. Christofides, «*Graph theory : an algorithmic approach*», Academic press Ed., 400 p., 1975

[G7W.92] ADEPA-AFCET, «*Le GRAFCET*», Cépadues Ed. (France), 144 p., 1992 (This book is intendet to be translated in English)

[IEC.88] IEC standard, «*Preparation of function charts for control systems*», First edition, 1988

[KAU.63] A. Kaufman, Y. Malgrange, «*Recherche des chemins et des circuits Hamiltoniens d'un graphe*» : Revue Française de Recherche Opérationnelle, N°26 - pp. 61, 73, 1963

[TAR.72] R. Tarjan, «*Depth-first search and linear graph algorithms*», SIAM J. Comput., Vol. 1, N°2 - pp. 146,160, 1972

[TUT.84] W. T. Tutte, «*Graph theory*», Addison-Wesley Publishing Company, 333 p., 1984

[UTE.92] UTE C03-191 E, French UTE project, «*Function chart GRAFCET : extension of basic principles*», UTE Ed., 1992