Elaboration of invariant safety properties from fault-tree analysis

Sébastien Henry, Jean-Marc Faure

Elaboration of invariant safety properties from fault-tree analysis

S. HENRY*, ** and J.M. FAURE**
* LAG - ENSIEG - Rue de la Houille Blanche, F38402 Saint Martin d'Heres
sebastien.henry@lag.ensieg.inpg.fr
** LURPA – ENS de Cachan - 61, Avenue du President Wilson, F94230 Cachan
jean-marc.faure@lurpa.ens-cachan.fr

Abstract – Formal verification of PLC programs using model-checking requires to elaborate previously temporal logic formulae that state in a formal way the properties that must be checked. Unfortunately temporal logic is a formalism totally unknown by automation engineers. This explains why PLC programs developers willing to verify the behavior of their programs are unable to use the existing model-checking tools. Temporal logic formulae elaboration is a too difficult task. In order to overcome this problem and to bridge the gap between PLC programs development and model-checking, this paper proposes a methodology enabling to obtain invariant safety properties from fault-tree analysis. Fault-tree analysis is a quite popular analysis method often used in industry when designing critical systems. Hence using the results of this analysis to elaborate formal properties may contribute to increase the use of formal verification techniques.

I. INTRODUCTION

Programmable Logic Controllers (PLC) are widely employed nowadays to implement industrial control systems. The first PLC systems have been used in the 1980s to control simple processes, mainly in production systems where they have replaced hard-wired switching networks based on relays. The new capabilities of PLCs, the comfort of PLC programming languages and of programming environments as well as economic constraints have led to an increasing use of PLC-based systems in all the industrial fields and even for safety-related applications. Examples can be found in transport systems, in critical industries like chemical processes, oil industry, power production and distribution. The growing demand of the society for risk prevention combined with this expanding use of PLC systems explain why PLC programs verification [FAU, 01], [FRE, 00] is to-day a major industrial concern.

PLC program verification does not aim merely at checking the intrinsic properties of the program, e.g. no infinite loop, no locking point, …, regardless the application, but mainly at checking that the program behaves as required. This paper focuses only on this last kind of property: the compliance of a given PLC program with the properties required for the application. Moreover these properties are often ranked in two categories: safety properties (what shall not be done) and liveliness properties (what must be done). In this paper only safety properties will be dealt with.

The LURPA has achieved since ten years several works on PLC programs verification using model-checking. A significant result of these works is a formal semantics of the SFC and ladder diagram languages [LAM, 00], [DES, 02]. This formal definition enables to represent any program written in these languages as a state automaton and consequently to translate such programs into model-checkers compliant languages. The underlying theory of model checking [MAC, 93] [SCH, 99] is indeed state automata theory. Hence, using a model-checker for PLC program verification implies to have a state model of the program as well as formal properties, in some temporal logic, that expresses in a formal way the application requirements and that must be satisfied by the program (Fig. 1).

Obtaining these formal properties is unfortunately a major problem. The application requirements are indeed expressed in industry in a quite informal way, i.e. some sentences in natural language or drawings., but never with sound mathematical formulae. Moreover, temporal logic [PNU, 77] [EME, 90] is not a classical logic and is difficult to grasp for automation engineers. It really matters to highlight that the use of temporal logic for expressing properties acts as a brake for the industrial use of model-checking of PLC programs.

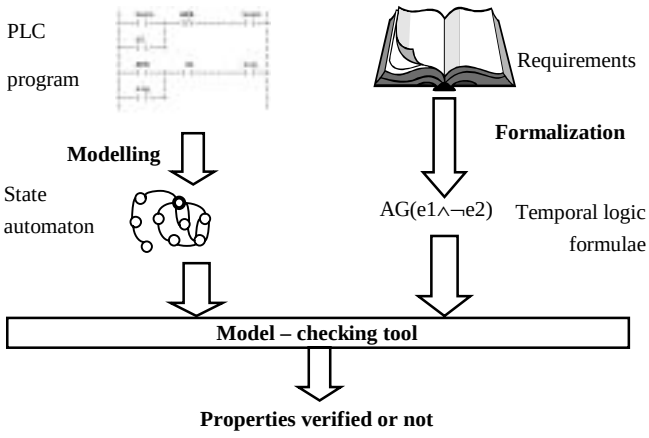


Fig. 1. Formal verification using model-checking

The works presented in this paper are aimed at easing the elaboration of formal properties of PLC programs. Other methods, developed with the same objective, are presented in [FIL, 99], [JUL, 99], [KLE, 01]. Though they may help usefully designers looking for formal properties, they require some experience and are not directly connected to any existing industrial method for dependability or safety analysis. On the opposite the method presented hereafter has been developed in order to link formal properties elaboration to a method commonly employed in industry: fault-tree analysis. This constraint has been put to facilitate the acceptance of industrial users.

In the scope of this paper, only invariant safety properties will be considered. An invariant (or static) property is true for each state of the state automaton modeling the program. It does not depend on time or on events sequence and is written in CTL temporal logic [CLA, 81] [EME, 82]: AG (present formula) or AG ¬(present formula), where the present formula is a combinatory formula. A textual description of that kind of property is: "For each path, in each state, the formula holds (or does not hold)".

This paper is structured as follows. The section 2 gives an overview on the developed method. A small example used in the rest of the paper to illustrate this method is presented in the section 3. The section 4 shows the design of a fault-tree taking into account PLC program faults. Formal statements are then derived from this fault-tree in section 5. The scope of these statements as well as program invariants elaboration are discussed in the last section.

II. METHOD OVERVIEW

Lots of techniques have been developed in order to assess or to enhance systems dependability when specifying and designing [VIL, 88]. Fault-tree analysis [IEC, 90] has been chosen for the following reasons:

- This analysis technique is widely used for critical systems design.
- It aims to find the different causes of a given fault.
- It enables to determine how failures combinations may lead to faults.
- Several research works have been developed to facilitate fault–trees design from functional analysis [DUT, 95] or to generate automatically fault-trees [GAL, 99] [GAU, 01].

The objective of the work presented in this paper is to take benefit of the results of a fault-tree analysis to elaborate invariant safety properties. Figure 2 depicts the proposed method. The general idea is to perform from the requirements a fault-tree analysis taking into account PLC programs faults and then to deduce temporal logic formulae from the obtained fault-trees.

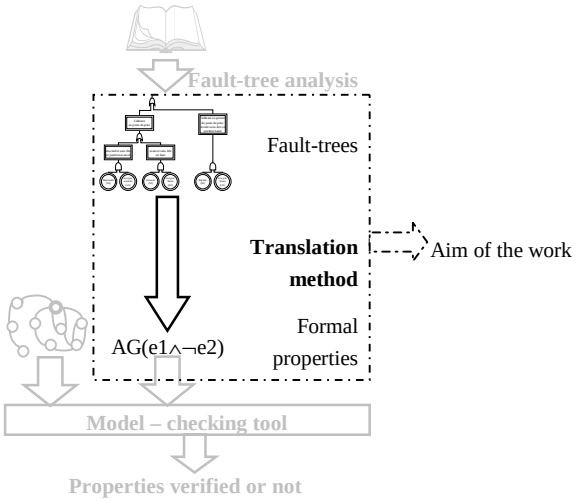


Fig. 2. Using fault-tree analysis to elaborate formal properties

III. ILLUSTRATIVE EXAMPLE

This method for properties elaboration will be presented thanks to a small example used in the framework of a benchmarking project by the French working group COSED [COS]. This example, sketched in figure 3, is a pick-and-place manipulator, part of an assembly line located at the Mechanical Engineering Department of the ENS Cachan. The aim of this manipulator is to pick up gearwheels with suction cups and to transfer the gearwheels to gear housings using two pneumatic cylinders. Only automatic operations will be considered.

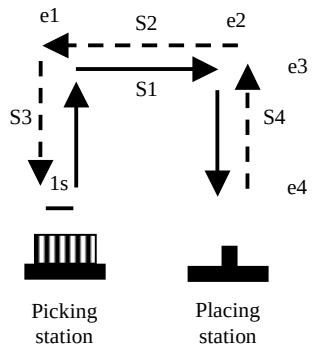
The controlled system may be decomposed into three parts:

- Horizontal movement:
  - 1 double-acting cylinder
  - 1 dual-solenoid valve
  - 2 magnetic sensors (rightmost and leftmost positions)
- Vertical movement:
  - 1 double-acting cylinder
  - 1 dual-solenoid valve
  - 2 magnetic sensors (upper and lower positions)
- Drawing up:
  - suction cups with a venturi system
  - 1 single-solenoid valve
  - no sensor

The control system is implemented by a control program running on a PLC. The inputs and outputs of this program are given in figure 3.

In order to avoid collisions with mechanical systems located between the two stations, all the horizontal movements must be performed with the drawing up system in the upper position. Hence any downward

movement between the two stations shall be forbidden and considered as a fault during safety analysis.



INPUTS

Leftmost Position	e1
Rightmost Position	e2
Upper Position	e3
Lower Position	e4

OUTPUTS

Move to the Left	S1
Move to the Right	S2
Move Up	S3
Move Down	S4

Fig. 3. Studied example

IV. System fault-tree elaboration

The aim of fault-tree analysis [IEC, 90] is to determine all the possible causes of a considered fault, called "undesirable event". To reach this objective, fault-tree analysis looks for how combinations of elementary faults (faults that do not come from other faults) can lead to this fault. The result of a fault-tree analysis can be represented in the form of a tree including faults and logical gates. The root of this tree is the "undesirable event", its leaves are the elementary faults. The figure 4 presents an extract of a fault-tree designed for the "Collision with the environment at the picking station" fault. Similar analysis can be carried out for other faults, like: "Collision with the environment at the placing station" or "Drawing up system fault".

In a classical fault tree [NIE, 02], any fault is related to a physical component failure or to an operator fault or error. When dealing with automatic systems, faults can come from:

- The controlled system. In this case only physical components failures are considered.
- The control system. Faults can be then either physical failures of the components of this system (sensors, input/output cards, processor, …), or faults or errors of the control software.

It is the reason why the fault-tree depicted in the figure 4 includes physical components failures and software faults. The fault "Move down with Leftmost Position false" is related to the control system and is a fault of the control software, whilst the fault "Move down with Leftmost Position information true", is also related to the control system but is a failure of an hardware component.

Hence, the obtained fault-tree includes three kinds of faults: control software faults, physical components failures and faults that are combination of these two kinds of faults (Move down without being at the leftmost position for instance). In order to elaborate properties dealing only with control software faults, control program faults shall be decoupled from other faults (coming from physical components failures). This can be achieved by introducing the following constraint, that the fault-tree shall verify:

Any control program fault must be combined with physical components faults only using OR logical gates.

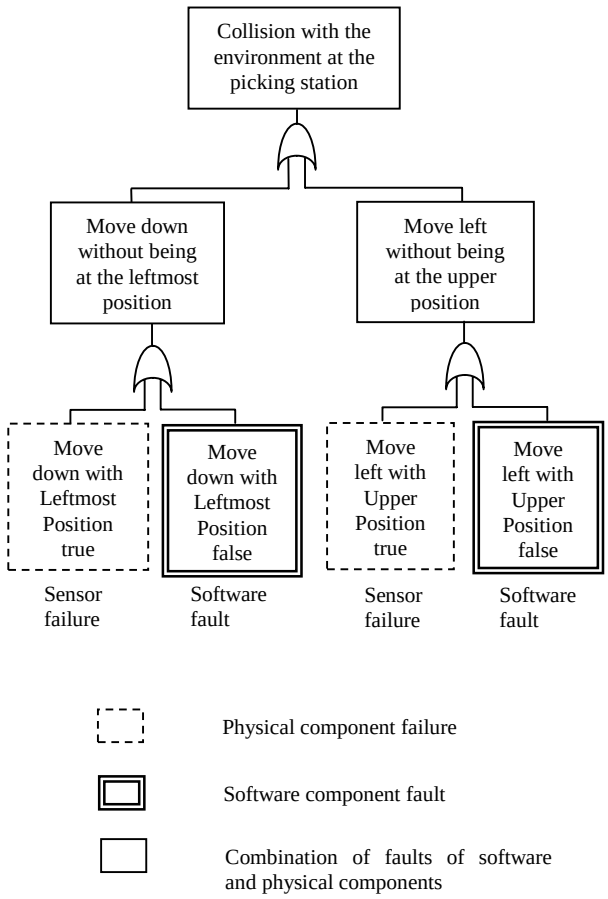


Fig. 4. Fault-tree for the "Collision with the environment at the picking station" fault

V. From fault-tree to formal statements

A System properties

Assuming that the global system, that includes the controlled system and the control system, can be described in the form of a state automaton, the following property in temporal logic can be derived from the previous fault-tree:

AG ¬ (Collision with the environment at the picking station)

This property states that the considered fault must never happen or, with other words, that for all paths, for each state, this property shall always hold. Using the decomposition provided by the fault-tree it is then possible to obtain another equivalent expression:

AG ¬ (Move down without being at the leftmost position

∨

Move to the left without being at the upper position)

That becomes when applying the De Morgan theorem:

AG (¬Move down without being at the leftmost position

∧

¬ Move to the left without being at the upper position)

This property may be interpreted as follows: "For all paths, in each state, it is impossible to move down without being at the leftmost position and to move to the left without being at the upper position", and be rewritten as a system of two properties: "For all paths, in each state, it is impossible to move down without being at the leftmost position" **and** "For all paths, in each state, it is impossible to move to the left without being at the upper position":

$$\begin{cases} \text{AG (}\neg\text{Move down without being at the leftmost position)} \\ \text{AG (}\neg\text{Move to the left without being at the upper position)} \end{cases}$$

When a fault is indeed the output of an OR gate, this fault will never occur if none of the faults inputs of this gate occurs.

B Control software properties

In order to obtain program properties, only software faults will be kept in the following. In the case of the studied fault-tree, the two hereafter mentioned faults will be retained:

- Move down with Leftmost Position false
- Move left with Upper Position false.

The root of this fault-tree (the "undesirable event") is indeed the output of a OR logical gate. Each input of this gate is itself output of a OR gate. This structure enables to state that the undesirable event will never occur if none of the control program faults occurs, that leads to the following properties system:

$$\begin{cases} \text{AG } \neg\text{(Move down with Leftmost Position false)} \\ \text{AG } \neg\text{(Move left with Upper Position false)} \end{cases}$$

This system means by reasoning in the state automaton describing the program, that, for all paths, in each state, the two properties must hold to avoid the fault for which the fault-tree analysis has been made.

C Taking into account PLC inputs-outputs

It is now possible to replace textual expressions by the input and output variables of the control program, that leads to:

$$\begin{cases} \text{AG } \neg(\ S3 \wedge \neg e1\ ) & (1) \\ \text{AG } \neg(\ S2 \wedge \neg e3\ ) & (2) \end{cases}$$

VI. Program invariants elaboration

The properties system obtained in the previous section has been deduced from a fault-tree related to collision at the picking station. Nevertheless checking invariant properties of a program implies that the properties to check shall hold for all the states of the automaton representing the program, i.e. in our case when the drawing up system is at the placing station or moves from a station to the other.

It matters now to ensure that the above obtained properties are really invariants and, if not, to modify them so as to obtain real invariants.

A. Model-based verification: introducing conditions on inputs

[FRE, 00] describes two approaches for PLC programs verification:

- The first one does not use any plant model. All inputs combinations are possible.
- The other one uses a plant model that limits inputs combinations to physically plausible ones. Hence constraints on inputs are introduced, stating for instance that two sensors used to indicate the rightmost and leftmost positions of a cylinder must never give simultaneously a true signal.

The main advantage of this second approach is to reduce the number of reachable states and therefore the checking time. It is the reason why this approach is often used and will be taken in what follows. In the case of the studied example, it is possible to introduce the two following conditions that constraint PLC input variables:

$$e1 \wedge e2=0 \qquad (c1)$$

the horizontal cylinder can not be at the same time at the rightmost position and at the leftmost position

$$e3 \wedge e4=0 \qquad (c2)$$

the vertical cylinder can not be at the same time at the upper position and at the lower position.

B. Scope of the previous properties

The properties system (1), (2) has been deduced considering the "Collision with the environment at the picking station" fault. But a collision with the environment can also occur at the placing station. From the fault-tree of the "Collision with the environment at the placing station" a system of two invariant properties can be derived in a similar way:

$$\begin{cases} AG \neg(S3 \wedge \neg e2) & (3) \\ AG \neg(S1 \wedge \neg e3) & (4) \end{cases}$$

The first attempt to obtain a program invariant is to consider a generic fault "Collision with the environment" and to merge the four previously obtained properties (1), (2), (3) an (4):

$$AG \neg(\text{Collision with the environment})$$

$$AG \neg[(S3\wedge\neg e1)\vee(S3\wedge\neg e2)\vee(S1\wedge\neg e3)\vee(S2\wedge\neg e3)]$$

$$AG \neg(S3\wedge\neg e1)\wedge\neg(S3\wedge\neg e2)\wedge\neg(S1\wedge\neg e3)\wedge\neg(S2\wedge\neg e3)$$

$$AG (\neg S3\vee e1)\wedge(\neg S3\vee e2)\wedge\neg(S1\wedge\neg e3)\wedge\neg(S2\wedge\neg e3)$$

$$AG [\neg S3\vee(e1\wedge e2)]\wedge\neg(S1\wedge\neg e3)\wedge\neg(S2\wedge\neg e3)$$

It is easy to see that this attempt was wrong. The condition c1 [e1 $\wedge$ e2] is always false. The property is true only if $\neg$S3, i.e. if the drawing up system never goes down.

The properties systems previously obtained are true only in the conditions for which the fault-tree analysis has been performed. The system (1), (2) is true only for a collision at the picking station and the system (3), (4) is true only for a collision at the placing station. It is not allowed to merge them so as to obtain a program invariant, true whatever the state of the controlled system would be.

C. Program invariants

In order to obtain program invariants (properties true for all states of the program), previous statements are to be rewritten by introducing a condition indicating from which fault-tree analysis they have been deduced, i.e. for which situation of the plant (controlled system) there are true. This implies to add to each present formula coming from a given fault-tree an other present formula stating the situation of the plant considered when designing the fault-tree. Hence program invariants will be written in the following way:

$$AG \neg(\text{present formula \#1}\wedge \text{present formula \#2})$$

The first formula is merely that deduced from the fault-tree. The second one indicates the plant situation.

In the case of the studied example, a collision at the picking station does not imply that the drawing up system is strictly at this station, but that it arrives, leaves or is waiting for at this station. Hence this situation must be translated by "Not at the placing station" with a control point of view and will be characterized with PLC inputs by using the $\neg$e2 condition. The properties system (1) and (2) becomes consequently:

$$\begin{cases} AG \neg(S3 \wedge \neg e1 \wedge \neg e2) & (5) \\ AG \neg(S2 \wedge \neg e3 \wedge \neg e2) & (6) \end{cases}$$

In a similar way, "Collision at the placing station" will be translated by $\neg$e1 (not to be at the picking station). The properties system (3) and (4) becomes consequently:

$$\begin{cases} AG \neg(S3 \wedge \neg e2 \wedge \neg e1) & (7) \\ AG \neg(S1 \wedge \neg e3 \wedge \neg e1) & (8) \end{cases}$$

The property (5) is identical to the property (7) and means that the control program shall never set the "Move down" output if neither the input variable e1 (drawing up system at the leftmost position), nor the variable e2 (drawing up system at the rightmost position) are true. Properties (6) and (8) mean respectively that no movement to the left (to the right) can occur if he drawing up system is not at the upper position.

These four properties are the searched program invariants.

VII. Conclusion

This paper has presented the first results of a work whose objective is to facilitate the elaboration of invariant safety properties using a preliminary fault-tree analysis. The presented method has been tested on other examples, e.g. the control of a water distribution system including several pumps. The main difficulty of this method lies in the scope of the statements directly obtained from the fault-tree. Nevertheless it has been shown in section VI that it is possible to obtain from these statements program invariants by introducing a formula clearly stating the scope of the fault-tree analysis from which the statement has been obtained.

The perspectives of this work are numerous. First of all the robustness of this method has to be evaluated thanks to several examples. The next step will aim at easing dynamic properties elaboration. This implies to design fault-trees that do not include only logical gates, but also gates dealing with time and events sequences. Some works taking into account that kind of specific elements are presented in [HEN, 02]. The last step, that must be carried out in strong cooperation with safety engineers and PLC developers, may be the development of a software tool supporting the method and linked to existing fault-tree design commercial software.

REFERENCES

[CLA, 81] E.M. Clarke, E.A Emerson., "Design and synthesis of synchronization skeletons using branching time temporal logic", In Proc. Logics of Programs Workshop, Yorktown Heights, New-York, May 1981, volume 131 of Lecture Notes in Computer Science, page 52-71. Springer, 1981.

[COS] Site Internet du groupe de travail COSED du Club EEA,http://lurpa.ens-cachan.fr/cosed/

[DES, 02] O. De Smet, O. Rossi, "Verification of a controller for a flexible manufacturing line written in Ladder Diagram via model-checking", In Porc. Of the American Control Conference, Anchorage, AK May 8-10, 2002.

[DUT, 95] Y. Dutuit, F. Chatelet, J. Dos Santos, T. Bouhoufani, "Les diagrammes-blocs fonctionnels: une aide à la construction manuelle des arbres de defaillances - Systemes sans boucle de regulation", Revue Europeenne Diagnostic et Sûreté de Fonctionnement, vol. 5, n°2, p. 181-200, 1995.

[EME, 90] E.A. Emerson "Temporal and modal logic" In J. Van Leeuwen, editor, Handbook of Theoretical Computer Science, vol. B, chapter 16, pages 995-1072. Elsevier Science Publishers, 1990.

[EME, 82] E.A. Emerson, J.Y. Halpern, "Decision procedures and expressiveness in the temporal logic of branching time", In Proc. 14th ACM Symp. Theory of Computing, STOC'82, San Francisco, CA, May 1982, pages 169-180, 1982.

[FAU, 01] J.M. Faure, J.J. Lesage, "Methods for safe control systems design and implementations", 10th IFAC Symposium on Information Control Problems in Manufacturing, INCOM 2001, Vienna (Austria), CDRom paper, 6pages, September 2001.

[FIL, 99] T. Filkorn, M. Holzein, P. Warkentin, M. Weiss, "Formal verification of PLC-programs", Proceedings of the 14th IFAC, 1999.

[FRE, 00] G. Frey,L. Litz, "Formal methods in PLC programming", Proceedings of IEEE int. conf. on Systems, Man and Cybernetics, Nashville, USA, pp. 2431-2436.

[GAL, 99] M. Gallois, M. Pilliere, "Benefits expected from automatic studies with KB3 at EDF", Proceding of PSA'99, International Topical Meeting on Probalistic Safety Assessment, vol. 2, p. 1061-1067, Washington D.C., 1999.

[GAU, 01] J. Gauthier, "Outil de conception et analyse systeme", Aralia workshop, manuel d'utilisation, Dassault Aviation, 2001.

[HEN, 02] S. Henry, "Elaboration de proprietes de surete a partir de la methode de l'arbre des causes", Memoire de recherche de DEA de l'Ecole Normale Superieure de Cachan, septembre 2002.

[IEC, 90] CEI/IEC - 1025, "Analyse par arbre de panne (AAP), Geneve, Bureau Central de la CEI, 1990.

[JUL, 99] J. Julliand, F. Bellegarde, B. Parreaux, "De l'expression des besoins à l'expression formelle de proprietes dynamiques", Technique et Sciences Informatiques, V. 18, n°7, p747-776, juillet 1999.

[KLE, 01] Klein, S., "A case study in design and formal verification of control algorithms using Interpreted Petri Nets and SFC", Memoire de recherche de DEA de l'Ecole Normale Superieure de Cachan, juin 2001.

[LAM, 00] S. Lamperiere - Couffin, J.J. Lesage, "Formal Verification of the Sequential Part of PLC Programs", 5th Workshop on Discrete Event Systems (WODES 2000), , Ghent, Belgium, August 21 - 23, 2000.

[MAC, 93] Mac Millan, K.L., "Symbolic model checking", Kluwer Academic, 1993.

[NIE, 02] E. Niel, E. Craye, "Maitrise des risques et surete de fonctionnement des systemes de production", Traite IC2 productique, ed. Hermes - Lavoisier, 2002.

[PNU, 77] A. Pnueli, "The temporal logic of programs", In proc. 18th IEEE Symp. Foundations of computers Science (FOCS'77), Providence, RI, USA, Oct. – Nov. 1977, pages 46-57.

[SCH, 99] P. Schnoebelen, "Verification de logiciels – Techniques et outils du model-checking", Ed. Vuibert Informatique, 1999.

[VIL, 88] A. Villemeur, " Surete de fonctionnement des systemes industriels", Ed. Eyrolles, 1988.