An experience in proving regular networks of processes by modular model checking*

Nicolas Halbwachs¹, Fabienne Lagnier¹, and Christophe Ratel¹, ² **

¹ IMAG/LGI P.O. Box 53X, F-38041 Grenoble Cedex, France
² Merlin-Gerin, F-38050 Grenoble Cedex, France

Abstract. This paper presents a complete example of the use of the synchronous declarative language LUSTRE for the specification, description and verification of a resource arbiter, which is a regular network of hardware devices. The fact that both programs and properties may be expressed in LUSTRE is used to perform an inductive verification. An invariant property of the network is found, and merged with the description program. Verification is performed by model checking.

1 Introduction

LUSTRE [CPHP87] is a declarative language to describe and program synchronous systems. It was designed to program real-time systems, and to describe circuit behavior. In [HPOG89, RHRg91], we proposed to take advantage of the formal nature of the language to specify and to verify safety properties of programs. The approach consists in specifying the intended properties in the same language, in generating a finite state graph of both the program and the properties, and in verifying the properties by model checking on this state graph. The announced advantages of this approach, with respect to classical model checking [CES86, RRSV87], are the followings:

- There is an obvious ergonomic advantage in using the same language to write programs and their specification.
- The property to be proved is taken into account in the graph generation. In many cases, it reduces the size of the considered graph, which is the main limitation of model checking approaches.
- The proposed method actually converts path properties into state properties – which is possible since we consider only safety properties. As a consequence, the verification may be done by a simple graph traversal. The whole graph does not need to be stored. Such a graph traversal is sometimes called “on the fly model checking” [Hol87, JJ89, FM91].
- Modular proof can be performed: Once a submodule has been shown to satisfy a property, it can be replaced by the property in the whole program.

* This work was partially supported by ESPRIT Basic Research Action “SPEC” and by a grant from the French Department of Research and Technology
** Present address: IMAG/LGI
Moreover, the same approach can be used for inductive proofs about regular networks: if a network of one process satisfies a property $\psi$, and if, under the assumption that a network of $n$ processes satisfies $\psi$, it can be shown that a network of $n+1$ processes also satisfy $\psi$, then $\psi$ is satisfied by a network of any number of processes.

This paper presents a complete example of application of the above techniques: We will describe, specify and verify a distributed resource arbiter, which is a regular network of hardware devices. The paper is organized as follows: Section 2 briefly presents LUSTRE and its trace semantics. The use of the language for expressing safety properties is presented in Sect. 3, together with a set of temporal operators which will be used in the example. Section 4 deals with the available verification tool, called LESAR. The example is informally presented in Sect. 5. The arbiter is then described in LUSTRE and inductively proved (Sects. 6, 7, and 8). The proved properties include mutual exclusion and a priority rule. All the proofs have been mechanically performed using LESAR.

2 The language Lustre

2.1 Overview of the language

We won't give here a detailed presentation of the language LUSTRE, which can be found elsewhere [CPHP87, HCRP91]. We only recall the elements which are necessary to understand the example. The formal semantics of the language will be sketched in the next subsection.

A LUSTRE program specifies a relation between input and output variables. A variable is intended to be a function of time. Time is considered as isomorphic to the set of natural numbers. Relations between variables are defined by equations and assertions: An equation $X=E$, where $E$ is a LUSTRE expression specifies that the variable $X$ is always equal to $E$, and an assertion $\text{assert} (E)$, where $E$ is a boolean LUSTRE expression specifies that the expression $E$ is always true. So $X=E$ is equivalent to $\text{assert} (X=E)$, but since equations are of common usage in a LUSTRE program, they have been given a simplified syntax.

Expressions are made of variable identifiers, constants (considered as constant functions), usual arithmetic, boolean and conditional operators (considered as applying pointwise to functions) and only two specific operators: the "previous" operator and the "followed-by" operator:

- If $E$ is an expression denoting the function $\lambda n. e(n)$ then $\text{pre} (E)$ is an expression denoting the function

$$\lambda n. \begin{cases} \text{nil} & \text{if } n = 0 \\ e(n-1) & \text{if } n > 0 \end{cases}$$

where $\text{nil}$ is an undefined value.

- If $E$ and $F$ are two expressions of the same type, respectively denoting the functions $\lambda n. e(n)$ and $\lambda n. f(n)$, then $E \rightarrow F$ is an expression denoting the function

$$\lambda n. \begin{cases} e(n) & \text{if } n = 0 \\ f(n) & \text{if } n > 0 \end{cases}$$