On the symbolic analysis of combinational loops in circuits and synchronous programs*

Nicolas Halbwachs, Florence Maraninchi
VERIMAG†
Miniparc-Zirst, 38330 Montbonnot, France
e-mail: {Nicolas.Halbwachs,Florence.Maraninchi}@imag.fr

Abstract: Combinational loops are very likely to appear in circuits compiled from high level languages, especially imperative synchronous languages like Esterel, Statecharts or Argos. These loops result from the fact that, both in circuits and in synchronous languages, the response of the system to incoming events can be viewed as a fixpoint of some function. Such a loop can be accepted if a unique fixpoint exists in any state which is reachable from the initial state of the system. Moreover, some "care-set", which expresses an assumption about inputs, should be taken into account. In this paper, we propose a complete Bdd-based procedure to check the consistency of combinational loops and, when possible, to remove these loops without changing the semantics of the circuit. The method is presented in the framework of Boolean-encoded finite state machines.

1 Introduction

Combinational loops are generally forbidden in design methodologies, since they can involve undesirable circuit behaviors (oscillations, non-determinism), and make formal reasoning difficult, since usual laws of Boolean calculus don't apply. However, such loops are very likely to appear in circuits compiled from a high level language. For instance, *synchronous languages* [11] have been shown to be very convenient for circuit description, and silicon compilers have been proposed that take such languages as input [1, 18, 10]. Synchronous languages are based on the idea that a system is made of a set of concurrent, deterministic, reactive processes, that communicate by *instantaneous broadcast* of signals. A well-known problem with such a powerful communication mechanism is the existence of instantaneous looping communications. Such loops can be meaningful, but can also involve either deadlocks or non-determinism, and produce combinational loops when translated into circuits. Some people [14, 15] argue that combinational loops should be allowed in some cases, as they can allow circuit optimization. Here, we do not adopt this point of view, but we propose a method for removing combinational loops, when possible, without changing the abstract semantics of the circuit. Let us first introduce the problem with some examples.

Let us consider the circuit of Fig. 1 (extracted from [15]), which can be described by the system of Boolean equations:

$$\begin{aligned} z &= x.y \\ y &= x+y \end{aligned}$$

It is easy to see that the output z is equal to the input x, even though the local variable y is non determined when x=0. Now, consider the circuit shown in Fig. 2, or the equivalent system of Boolean equations:

$$\begin{aligned} a &= \text{if } y \text{ then } b \text{ else } x \\ b &= \text{if } z \text{ then } a \text{ else } x \end{aligned}$$

*This work has been partly supported by EEC under ESPRIT-BRA action "REACT".

†Verimag is a joint laboratory of CNRS, Institut National Polytechnique de Grenoble, Universite J. Fourier and Verilog SA associated with IMAG.

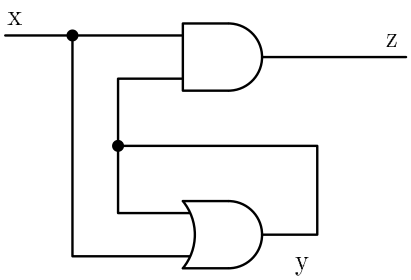

Figure 1: Combinational loop, example 1

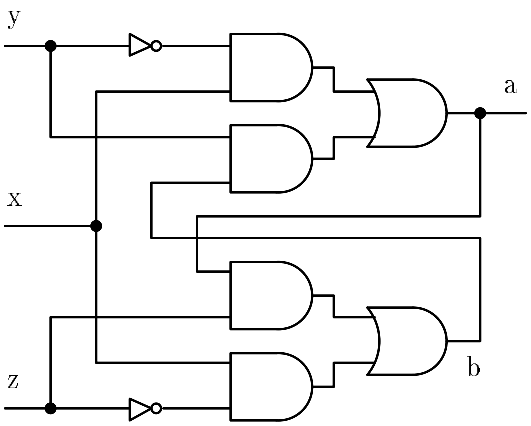

Figure 2: Combinational loop, example 2

This circuit has a perfectly defined behavior, under the condition that y and z are never both true. This condition may be satisfied, for instance because y and z are state variables, and no state where $y = z = 1$ can be reached from the initial state. An other case happens when y and z are input variables and $y = z = 1$ belongs to a "don't care" set of the circuit.

Of course, such stupid circuits would not be designed by hand, but they are often obtained by automatic translation from high level descriptions. As a matter of fact, such meaningful combinational loops are a major problem in compiling synchronous languages, even toward software. In the current compilers, they are either syntactically forbidden (as in LUSTRE [12]), or partially checked using heuristics (as in ESTEREL [4]), or completely checked by an expensive exhaustive enumeration of reachable states (as in ARGOS [16]).

The paper is organized as follows: In Section 2 we formalize, on a very general model of *finite state machines* (FSM, for short), what kind of circuits we want to accept, namely those which are *reactive* and *deterministic*. Then, we make our model more precise by defining *Boolean automata*, a special kind of FSM, where the outputs and the transition function are defined by means of a fixpoint equation on local variables. Two notions of consistency are defined: A Boolean automaton is *strongly consistent* if, in any reachable state, for any input configuration of its "care set", each local variable can take one and only one value. It is *weakly consistent* if, in any reachable state, any input configuration of its "care set" uniquely determines the output values and the next state. For instance, the circuit of Fig. 1 is weakly consistent, but not strongly consistent. In Section 3, we give a Bdd-based procedure to check the strong consistency of a Boolean automaton. The procedure returns an acyclic definition of the local variables, together with a necessary and sufficient condition on state and input variables for this definition to be consistent with the initial one. This condition can be checked against the "care set" of the automaton, or proved invariant using standard verification techniques. In Section 4, the procedure is adapted to check weak consistency. These procedures have been implemented in an automatic tool, called BAC (for "Boolean automaton checker"), which is briefly described in Section 5. Section 6 deals with related works. The two propositions which are used are proved in the appendix.

2 Definitions and notations

We start with a very general notion of finite state machines:

Definition 1 *A* **finite state machine** *(FSM, for short) is a tuple* $M = (S, I, O, init, \sigma, \omega)$

where,

- S *is a finite set of* **states**
- I *is a finite set of* **inputs**
- O *is a finite set of* **outputs**
- $init \subseteq S$ *is the set of* **initial states**
- $\sigma \subseteq S \times I \times S$ *is the transition relation*
- $\omega \subseteq S \times I \times O$ *is the output relation*

A **run** *of the FSM is a finite or infinite sequence* $(s_0, i_0, o_0), (s_1, i_1, o_1) \ldots, (s_n, i_n, o_n) \ldots$ *of elements of* $S \times I \times O$*, with* $s_0 \in init$*, and where* $\forall n \geq 0$*,* $(i_n, s_n, o_n) \in \omega$ *and* $(s_n, i_n, s_{n+1}) \in \sigma$*.*

A **restricted FSM** *is a pair* $(M, \mathcal{C})$*, where* $M = (S, I, O, init, \sigma, \omega)$ *is a FSM and* $\mathcal{C} \subseteq S \times I$ *is a* **care set**. *A run of the restricted FSM* $(M, \mathcal{C})$ *is a run of* M*, all the elements of which belong to* $\mathcal{C} \times O$*.*

Notice that the model allows non-determinism since outputs and transitions are defined by *relations*. Care sets are commonly used to express some knowledge about the inputs of the machine. Here, we allow a very general kind of care sets, since they can involve state variables.

A pair $(s, i) \in S \times I$ is called a **configuration**. A configuration (s, i) is **reachable** in the FSM M, if there exists an output o such that (s, i, o) belongs to a run of M (the same definition applies to restricted FSMs).

For each configuration (s, i), we note $\sigma(s, i)$ (respectively $\omega(s, i)$) the set of states s' (resp. the set of outputs o) such that $(s, i, s') \in \sigma$ (resp. $(s, i, o) \in \omega$). Let $|X|$ denote the cardinality of a set X.

Definition 2 *A FSM* $M = (S, I, O, init, \sigma, \omega)$ *is said to be* **reactive** *if it cannot be blocked (or it always accept inputs), i.e., if for all reachable configuration* (s, i)*,* $|\sigma(s, i)| \geq 1$ *and* $|\omega(s, i)| \geq 1$ *(The same definition applies to restricted FSMs).*

A FSM is said to be **deterministic** *if each reachable configuration uniquely determines the current output and the next state, i.e., if for all reachable configuration* (s, i)*,* $|\sigma(s, i)| \leq 1$ *and* $|\omega(s, i)| \leq 1$ *(The same definition applies to restricted FSMs).*

These two properties have been considered separately for synchronous programs: For instance, some semantics proposed for STATECHARTS [13] forbids non reactivity, but allows non determinism. Of course, for circuits, we need both reactivity and determinism. We will say that a FSM is **consistent** if it is both reactive and deterministic (i.e., if each reachable configuration exactly determine one output and one next state). We will consider the problem of consistency checking on a class of Boolean-encoded FSMs, which we call "Boolean automata".

Definition 3 *A* **Boolean automaton***, with* n *state variables,* m *input variables,* p *local variables, and* q *output variables is a tuple* $A = (init, \Sigma, \Omega, \Lambda, \mathcal{C})$*, where*

- $init \subseteq \mathbb{B}^n$ *is the set of initial states (where* $\mathbb{B} = \{0, 1\}$*)*
- Σ *is a total function from* $\mathbb{B}^{m+n+p}$ *to* $\mathbb{B}^n$ *(transition function), given as a vector* $(\Sigma_k)_{k=1\ldots n}$ *of functions from* $\mathbb{B}^{m+n+p}$ *to* $\mathbb{B}$*. If* $s \in \mathbb{B}^n$*,* $i \in \mathbb{B}^m$ $\ell \in \mathbb{B}^p$*,* $\Sigma(s, i, \ell)$ *denotes the vector* $[\Sigma_k(s, i, \ell)]_{k=1\ldots n}$*.*
- *In the same way,* Ω *is a total function from* $\mathbb{B}^{m+n+p}$ *to* $\mathbb{B}^q$ *(output function), given as a vector* $(\Omega_k)_{k=1\ldots q}$ *of functions from* $\mathbb{B}^{m+n+p}$ *to* $\mathbb{B}$*.*
- Λ *is a total function from* $\mathbb{B}^{m+n+p}$ *to* $\mathbb{B}^p$ *(local variable definition), given as a vector* $(\Lambda)_{k=1\ldots p}$ *of functions from* $\mathbb{B}^{m+n+p}$ *to* $\mathbb{B}$*.*
- $\mathcal{C} \subseteq \mathbb{B}^{m+n}$ *(care set)*

Such a Boolean automaton $A = (init, \Sigma, \Omega, \Lambda, \mathcal{C})$ can be viewed as a restricted FSM $M(A) = (S, I, O, \sigma, \omega)$, with $S = \mathbb{B}^n$, $I = \mathbb{B}^m$, $O = \mathbb{B}^q$, and

$$\begin{aligned}
(s, i, s') \in \sigma \;\Leftrightarrow\; & \exists \ell \in \mathbb{B}^p \text{ such that} \\
& \ell = \Lambda(s, i, \ell) \text{ and } s' = \Sigma(s, i, \ell) \\
(s, i, o) \in \omega \;\Leftrightarrow\; & \exists \ell \in \mathbb{B}^p \text{ such that} \\
& \ell = \Lambda(s, i, \ell) \text{ and } o = \Omega(s, i, \ell)
\end{aligned}$$

So, a Boolean automaton consists of three systems of Boolean equations. The system

$$s'_k = \Sigma_k(s, i, \ell)\,,\ k = 1 \ldots n$$

defines the next values of state variables, the system

$$o_k = \Omega_k(s,i,\ell)\,,\ k = 1\ldots q$$

defines the current outputs, and the system

$$\ell_k = \Lambda_k(s,i,\ell)\,,\ k = 1\ldots p$$

defines the current values of local variables. Notice that the next state and the output are defined as *total functions* of the current state, the current input, and the current values of local variables. Only local variables are defined by a possibly cyclic system of (fixpoint) equations. So, all the problems of deadlock (absence of fixpoint) and non-determinism (multiple fixpoints) are localized in the definition of local variables.

Definition 4 *Let* loc *be the function from* $\mathbb{B}^{m+n}$ *to the powerset of* $\mathbb{B}^p$ *associating with each configuration* (s,i) *the set of solutions of the system of equations* $\ell = \Lambda(s,i,\ell)$. *A Boolean automaton is* **strongly consistent** *if and only if, for each reachable configuration* (s,i), $|loc(s,i)| = 1$. *It is* **weakly consistent** *if and only if, for each reachable configuration* (s,i),

$$\begin{aligned} &|\{\Omega(s,i,\ell)\mid \ell\in loc(s,i)\}| &=& \ 1\\ \textit{and}\ \ &|\{\Sigma(s,i,\ell)\mid \ell\in loc(s,i)\}| &=& \ 1\end{aligned}$$

In other words, in a strongly consistent automaton, the value of all the local variables is uniquely determined in each reachable configuration; in a weakly consistent automaton, the value of some local variables may not be completely determined, as long as it neither influences the value of the output nor of the next state. For instance, the circuit given in Fig. 1 is only weakly consistent if it can reach a configuration where $x = 0$. Of course, strong consistency implies weak consistency.

3 Strong consistency

In this section, we will give a method to check strong consistency. More precisely, we propose a procedure, taking as input a system of recursive equations $\ell = \Lambda(s,i,\ell)$, and returning two results:

- A condition $E(s,i)$ which is true if and only if $|loc(s,i)| = 1$
- A new, non recursive, system $\ell = \Lambda'(s,i)$ such that

$$E(s,i) \implies loc(s,i) = \{\Lambda'(s,i)\}$$

For instance, consider the system

$$\begin{aligned}\ell_1 &= \ell_1\\ \ell_2 &= \overline{\ell_2}.\ell_1.s_1\end{aligned}$$

where ℓ_1, ℓ_2 are local variables and s_1 is a state variable. Clearly, $s_1 = 1$ is a necessary and sufficient condition for this system to admit a unique solution, which is $\ell_1 = \ell_2 = 0$.

Let F be the BDD expressing the definitions of the local variables. In the above example, F is the BDD of the formula

$$(\ell_1 \equiv \ell_1)\,.\,(\ell_2 \equiv (\,\overline{\ell_2}.\ell_1.s_1))$$

For each local variable, say ℓ_k, we want to know under which condition

1. there exists a solution for ℓ_k in F
2. this solution is unique, i.e., there exists a Boolean function g, independent of ℓ_k, such that $F \Rightarrow (\ell_k = g)$.

Let $F = \ell_k.F_{\ell_k} + \neg\ell_k.F_{\overline{\ell_k}}$ be the Shannon decomposition of F according to ℓ_k. Point (1) is straightforward: $F_{\ell_k} + F_{\overline{\ell_k}}$ is a necessary and sufficient condition on other variables for the existence of a solution for ℓ_k. For point (2), we use the following classical property [5]:

Proposition 1 *Let* $F = \ell_k.F_{\ell_k} + \overline{\ell_k}.F_{\overline{\ell_k}}$ *be the Shannon decomposition of* F *according to* ℓ_k. *Then* $F_{\overline{\ell_k}}.F_{\ell_k} = 0$ *is a necessary and sufficient condition for the existence of a function* g, *independent of* ℓ_k, *such that* F *implies* $\ell_k = g$. *Moreover, under the condition* $F_{\overline{\ell_k}}.F_{\ell_k} = 0$, *for any* g *satisfying* $F_{\ell_k} \Rightarrow g \Rightarrow \overline{F_{\overline{\ell_k}}}$, *we have* $F = (F_{\ell_k} + F_{\overline{\ell_k}}).(\ell_k = g)$.

In practice, a good choice for g is $F_{\ell_k} \uparrow (F_{\ell_k} + F_{\overline{\ell_k}})$, where "$\uparrow$" denotes the "restrict" operator introduced by [9], since it produces a small BDD for g. A dual good choice could be $\overline{F_{\overline{\ell_k}} \uparrow (F_{\ell_k} + F_{\overline{\ell_k}})}$. Now, let $\{\ell_1, \ell_2, \ldots, \ell_p\}$ be the set of local variables[1]. From the BDD $F^{(0)} = F$, we successively compute, for all $k = 1 \ldots p$,

$$\begin{aligned}
F^{(k)} &= F_{\ell_k}^{(k-1)} + F_{\overline{\ell_k}}^{(k-1)} \\
g^{(k)} &= F_{\ell_k}^{(k-1)} \uparrow F^{(k)} \\
E^{(k)} &= \neg(F_{\ell_k}^{(k-1)} . F_{\overline{\ell_k}}^{(k-1)})
\end{aligned}$$

then we backward substitute $g^{(j)}$ to ℓ_j , $j > k$, in each $g^{(k)}, E^{(k)}$. The result is the system of equations $\ell_k = g^{(k)}, k = 1 \ldots p$, and the necessary and sufficient condition $E = \bigwedge_{k=1}^{p} E^{(k)}$. If E is not identically true, one can first check if it is implied by the care set $\mathcal{C}$. Otherwise, a standard verification technique (e.g., [8, 6]) can be applied to check its invariance.

Example: Let us come back to the example of Fig. 1:

$$\begin{aligned}
y &= x + y \\
z &= x.y
\end{aligned}$$

and assume that y and z are local variables, whereas x is either an input or a state variable. We have

$$\begin{aligned}
F^{(0)} &= \overline{y} . \overline{z} . \overline{x} + y.(\overline{z} . \overline{x} + z.x) \\
F^{(1)} &= \overline{z} . \overline{x} + z.x \\
E^{(1)} &= z + x \\
g^{(1)} &= (\overline{z} . \overline{x} + z.x) \uparrow (\overline{z} . \overline{x} + z.x) \\
&= 1 \\
F^{(2)} &= 1 \\
E^{(2)} &= 1 \\
g^{(2)} &= x \uparrow 1 \\
&= x
\end{aligned}$$

Backward substitution of $z = g^{(2)} = x$ in $g^{(1)}$ and $E^{(1)}$ provides $y = g^{(1)} = 1$ and $E^{(1)} = x$. So the final result is $z = x$, $y = 1$, under the condition $E = E^{(1)}.E^{(2)} = x$. *[End of example]*

[1] In fact it is enough to select one local variable in each loop. Moreover, it is much more efficient to consider separately each strongly connected component of the graph of the dependency relation.

4 Weak consistency

For weak consistency, we have to find under which condition the output and the next state of the automaton are uniquely determined by the current state and the current input, without regards to possible non-determinism in local variables. Let F_Ω be the BDD expressing the full system of equations of the outputs:

$$\bigwedge_{k=1}^{q} o_k = \Omega(s, i, l) \wedge \bigwedge_{k=1}^{p} \ell_k = \Lambda_k(s, i, l)$$

and F_Σ be the BDD expressing the full system of equations of the next state:

$$\bigwedge_{k=1}^{n} s'_k = \Sigma_k(s, i, l) \wedge \bigwedge_{k=1}^{p} \ell_k = \Lambda_k(s, i, l)$$

We want to find under which conditions E_Ω and E_Σ there exist two (vectorial) functions Ω' and Σ', independent of ℓ such that

$$\begin{aligned}
E_\Omega &\implies \big(F_\Omega \Rightarrow (o = \Omega'(s, i))\big) \\
E_\Sigma &\implies \big(F_\Sigma \Rightarrow (s' = \Sigma'(s, i))\big)
\end{aligned}$$

The following proposition shows that we just have to apply the previous procedure, respectively to $\exists \ell(F_\Omega)$ and $\exists \ell(F_\Sigma)$ (where $\exists \ell_1(F)$ is $F_{\ell_1} + F_{\overline{\ell_1}}$ and $\exists(\ell_1, \ldots, \ell_p)(F)$ is $\exists \ell_1(\exists(\ell_2, \ldots \ell_p)(F))$).

Proposition 2 *A formula F implies an equation $x = g$, where g neither depends on x nor on y, if and only if there exists g, independent of x, such that $\exists y(F)$ implies $x = g$.*

Example: If, in the circuit of Fig. 1, we consider z as the only output, we have $F_\Omega = \overline{y} . \overline{z} . \overline{x} + y.(\overline{z} . \overline{x} + z.x)$ and $\exists y(F_\Omega) = \overline{z} . \overline{x} + z.x$. Applying the first procedure to this function provides $z = x$ under the condition $E = 1$. *[End of example]*

Notice that weak consistency is not sufficient to ensure circuit correctness: Since the possible loops on local variables are not removed by the procedure, they can still produce bad circuit behaviors, like oscillations. However, since the result of the above procedure is to express outputs and next state independently of local variables, local variables can be simply dropped out.

5 Implementation

The proposed procedures have been implemented into a prototype verification tool called Bac (for "Boolean automaton checker"). This tool takes as input a Boolean automaton, described in a format very similar to Blif [3]. Other functionalities of Bac include computing the set of reachable configurations (taking fully into account our general notion of care sets), and verification of the invariance of Boolean formulas either by forward or by backward traversal. Bac makes use of the Bdd package described in [17], and of standard techniques [7, 2, 9, 17] for image and back-image computation. Moreover, translators from Argos and Lustre to Boolean automata are available, which have made possible experiments on large examples (hundreds variables). These experiments show that the loop analysis procedure is quite efficient. The expensive part is clearly the verification of invariance of the consistency condition E. However, notice that valuable results can often be obtained without performing this verification, when the consistency condition is either identically true, or is implied by the care set.

6 Related works

The use of combinational loops in circuit optimization has been advocated a long time ago [14]. In circuit design, consistent combinational loops are often introduced for resource sharing, when some resource is used for different purposes at different cycles in an exclusive way. The problem of avoiding such loops has been studied in [19]. [15] checks whether combinational loops can be kept as they are, without introducing bad circuit behaviors. In that paper, the problem of oscillations introduced by propagation delays is considered. For instance, the circuit shown in Fig. 3 can have an oscillating behavior on y because of the inverter propagation delay. Notice that this circuit is weakly consistent in our sense, and the loop will be removed by our procedure. Of course, there are cases where removing loops increases the size of the circuit.

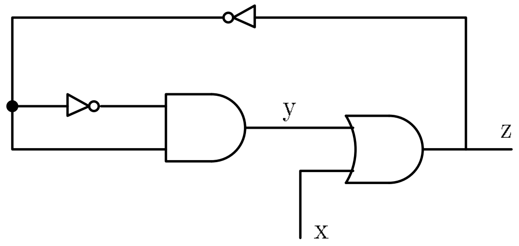

Figure 3: An oscillating circuit

References

[1] G. Berry. A hardware implementation of pure Esterel. In *ACM Workshop on Formal Methods in VLSI Design, Miami*, January 1991.

[2] C. Berthet, O. Coudert, and J. C. Madre. New ideas on symbolic manipulations of finite state machines. In *International Conference on Computer Design (ICCD), Cambridge*, September 1990.

[3] Blif-MV: An interchange format for design verification and synthesis. Technical report, Berkeley Logic Synthesis Group, 1990.

[4] F. Boussinot and R. de Simone. The Esterel language. *Proceedings of the IEEE*, 79(9):1293–1304, September 1991.

[5] F. M. Brown. *Boolean Reasoning: The logic of Boolean Equations.* Kluwer Academic Publishers, 1990.

[6] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Fifth IEEE Symposium on Logic in Computer Science, Philadelphia*, 1990.

[7] O. Coudert, C. Berthet, and J. C. Madre. Verification of sequential machines using boolean

functional vectors. In L. J. M. Claesen, editor, *Formal VLSI Correctness Verification*. North-Holland, November 1989.

[8] O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines based on symbolic execution. In *International Workshop on Automatic Verification Methods for Finite State Systems, Grenoble*. LNCS 407, Springer Verlag, 1989.

[9] O. Coudert and J. C. Madre. A unified framework for the formal verification of sequential circuits. In *International Conference on Computer Aided Design (ICCAD), Santa Clara*, 1990.

[10] D. Drusinsky and D. Harel. Using Statecharts for hardware description and synthesis. *IEEE Trans. on Computer-Aided Design*, 8(7):798–807, July 1989.

[11] N. Halbwachs. *Synchronous programming of reactive systems*. Kluwer Academic Pub., 1993.

[12] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[13] C. Huizing. Semantics of reactive systems: comparison and full abstraction. Thesis Eindhoven University, 1991.

[14] W. H. Kautz. The necessity of closed loops in minimal combinatorial circuits. *IEEE Trans. on Computers*, pages 162–164, 1970.

[15] S. Malik. Analysis of cyclic combinational circuits. In *ICCAD'93, Santa Clara*, 1993.

[16] F. Maraninchi. Operational and compositional semantics of synchronous automaton compositions. In *CONCUR'92, Stony Brook*. LNCS 630, Springer Verlag, August 1992.

[17] C. Ratel. Définition et réalisation d'un outil de vérification formelle de programmes Lustre: Le système Lesar. Thesis, Université Joseph Fourrier, Grenoble, June 1992.

[18] F. Rocheteau and N. Halbwachs. POLLUX, a LUSTRE-based hardware design environment. In P. Quinton and Y. Robert, editors, *Conference on Algorithms and Parallel VLSI Architectures II, Chateau de Bonas*, June 1991.

[19] L. Stok. False loops through resource sharing. In *ICCAD'92, Santa Clara*, 1992.

A Appendix

Proof of Proposition 1

Assume $F_{\ell_k} \Rightarrow g \Rightarrow \overline{F_{\overline{\ell_k}}}$. When F is true, then either ℓ_k and F_{ℓ_k} are true — so g is true and $\ell_k = g$ —, or ℓ_k is false and $F_{\overline{\ell_k}}$ is true — so g is false and $\ell_k = g$. Conversely, if $F_{\overline{\ell_k}}.F_{\ell_k} \neq 0$, if the other variables satisfy $F_{\overline{\ell_k}}.F_{\ell_k}$, F is true whatever be the value of ℓ_k. So the value of ℓ_k is undetermined, and is certainly not a function of the other variables. Finally, under the hypotheses that $F_{\overline{\ell_k}}.F_{\ell_k} = 0$ and $F_{\ell_k} \Rightarrow g \Rightarrow \overline{F_{\overline{\ell_k}}}$, we have

$$\begin{aligned}
&(F_{\ell_k} + F_{\overline{\ell_k}}).(\ell_k = g) \\
&\quad = \quad \overline{\ell_k}\,.\,\overline{g}\,.\,\overline{F_{\ell_k}}\,.F_{\overline{\ell_k}} + \ell_k.g.\,\overline{F_{\overline{\ell_k}}}\,.F_{\ell_k} \\
&\quad \Rightarrow \quad \overline{\ell_k}\,.F_{\overline{\ell_k}} + \ell_k.F_{\ell_k} \\
&\quad = \quad F
\end{aligned}$$

Proof of Proposition 2

Let $F = x.y.F_{xy} + x.\overline{y}\,.F_{x.\overline{y}} + \overline{x}\,.y.F_{\overline{x}.y} + \overline{x}\,.\,\overline{y}\,.F_{\overline{x}.\overline{y}}$ be the Shannon decomposition of F according to x and y. Let $F_0 = (F_{\overline{x}.y} + F_{\overline{x}.\overline{y}})$ and $F_1 = (F_{xy} + F_{x.\overline{y}})$. We have $\exists y(F) = x.F_1 + \overline{x}\,.F_0$. So, from Prop. 1,

$$\begin{aligned}
&(F_1.F_0 = 0) \iff \\
&\quad \forall g \text{ such that } F_1 \Rightarrow g \Rightarrow \overline{F_0} \\
&\qquad \exists y(F) \Rightarrow (x = g)
\end{aligned}$$

Now, again from Prop. 1, there exists g such that $F \Rightarrow (x = g)$ if and only if

$$(y.F_{xy} + \overline{y}\,.F_{x.\overline{y}}).(y.F_{\overline{x}.y}\,.\,\overline{x}\,.\,\overline{y}\,.F_{\overline{x}.\overline{y}}) = 0$$

and in that case, $(y.F_{xy} + \overline{y}\,.F_{x.\overline{y}}) \Rightarrow g \Rightarrow \neg(y.F_{\overline{x}.y} + \overline{y}\,.F_{\overline{x}.\overline{y}})$. So, for g to be independent of y, there must not exist y_1, y_2 such that

$$(y_1.F_{xy} + \overline{y_1}\,.F_{x.\overline{y}}).(y_2.F_{\overline{x}.y} + \overline{y_2}\,.F_{\overline{x}.\overline{y}})$$

is satisfiable (Otherwise, g should be true for y_1 and false for y_2). This condition reduces to $(F_{xy} + F_{x.\overline{y}}).(F_{\overline{x}.y} + F_{\overline{x}.\overline{y}})$, i.e., $F_1.F_0 = 0$.