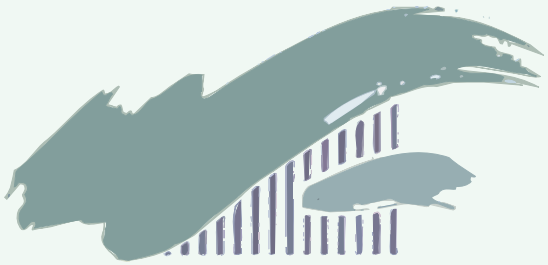


UNIVERSITE D'ORLEANS
Faculté des Sciences
LIFO

Laboratoire d'Informatique Fondamentale d'Orléans
4, rue Léonard de Vinci, BP 6759
F-45067 Orléans Cedex 2
FRANCE

Rapport de Recherche

www : http://www.univ-orleans.fr/SCIENCES/LIFO/

Designing FPLA combinational circuits by conditional rewriting.

N. Andrianarivelo, W. Bousdira, J. Chabin, Z. Maazouzi.
Université d'Orléans, LIFO

Rapport N° **98-01**

Abstract

We present a rewrite based method to design FPLA combinational circuits. It is based on the maximal unit strategy method with built-in algebras. Basic operations on natural integers and boolean algebra are done by the use of efficient usual methods. The formalism of constraint is used; efficient, correct heuristic and goal-oriented deletion criteria and strategies are used without losing correctness and completeness. A beta version named CDR (for Circuit Design by Rewriting), of our method is under experiment. A brief description of CDR and some results are presented at the end of the paper.

1 Introduction

Palasm, Abel, ... are two of the many software applications to design FPLA circuits. One of their main problems is they constantly need test on vector tests to check the correctness of each computed circuit-solution. As a result, they are time consuming.

In this paper, a new method to avoid the use of vectors to check the correctness is presented. It is based on conditional and constrained rewriting, and more precisely, on the well-known maximal unit strategy. As all the inference system at the heart of the method is correct, all the process results are correct. Both new goal-oriented deletion criteria inference rules and strategies are included. By means of the inequations representing the circuit to design, we can suppress some rules. Furthermore, we can forbid *conditional-narrowing* between some types of rules. This is more than a deletion criteria. All this helps the process to reduce the search space tremendously. A short version of this report is published in the proceedings of ECAI'98 [ABCM98b].

A beta version of our work named CDR, for Circuit Design by Rewriting, has been experimented on some examples. Some results are presented at the end of this paper. CDR is user-friendly. Users need not master rewriting technique to design a circuit by CDR. They only have to input both the truth table of the circuit to design and the reference of the FPLA device to use for. Like in Abel, the results of CDR are matrices which will be the input data of dedicated machines to build the chips.

All the circuit solutions using the given FPLA device may be generated by CDR. In fact, CDR computes the solutions one after the other. If a solution is not convenient for some reasons, the users can ask CDR to compute the next solution.

If no solution exists, within a finite period of time, CDR stops computing by pointing out to the users the failure of the design process. In this case, the users only have to replace the FPLA input device and run the process again.

With each result, some significant information is also given by CDR to the users to assist them. One of them is the occupation rate of the FPLA device input using the computed solution. Such rate will yield information on the physical behaviour of the circuit-solution without actually building it.

In this paper, we suppose readers are familiar with basic notions in conditional and constrained rewriting. For further details see [DJ90]. We use the same notation as in the latter. The formalism of constraint used here is also the usual one. For more details see [KKR90], and [Smo89]. All the notations used here are the same as in our previous work [ABT96]. Let us recall that a constrained conditional rule is noted $\mathcal{C} \wedge L \Rightarrow s = t$, in which $\mathcal{C}$ is a list of constraints to be solved by using algorithms in the predefined algebras, or by unification, and L is a list of equations to be solved by rewriting techniques. When the L part does not exist then we call it an equation.

We also suppose readers are familiar with all the usual notations and principles in logic circuit design. For more details, see [Wak91].

The layout of this paper is as follows: Section 2 presents the architecture of our circuit design method. Each part of this architecture is fully detailed in Section 3. The decoder with 7 segments will be presented in this section. Then heuristic and goal-oriented deletion criteria and strategies are developed in Section 4. In Section 5, we give a brief presentation of CDR. Section 6 is devoted

to some excerpts of the design of the decoder of 7 segments by CDR. Then we conclude in Section 7.

2 Designing with FPLA devices

The architecture of the method is given in Figure 1 :

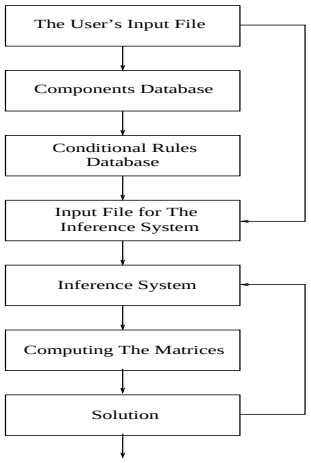


Figure 1: The architecture of our expert system

Let us precise the contents of each entity in the above schema.

1. The user's input file contains the description of the circuit to design in the form of a truth table.
2. The electronic components database is a list of all the references of FPLA devices with their basic characteristics (the number of inputs, and the number of each of the three types of gates).
3. The conditional rules are those schematizing the use of the gates in a FPLA device.
4. The input file for the inference system is the *real* input file for the inference system and it is deduced automatically from the three previous entities. It is made up by constrained and conditional rules and a negative clause to represent the circuit to design.
5. The process '*Computing the Matrices*' provides a solution to the circuit to design in the form of two matrices. The reason is that such matrices make up a satisfactory representation of circuits designed from FPLA devices and in practice, they may be more easily used by electronicians to design the circuits. We will see later what matrices are.
6. The solution is the result of the design process in the form of two matrices.

The truth table given by the user may be *reduced* by usual methods of *minimization* like those of Karnaugh or McCluskey. The general process is : from all the information contained in the input file, we infer in our inference system until we get a solution. If the given solution is not convenient for the user, we infer again in our system until we get another solution.

Now let us comment the above architecture.

3 Description of the method

3.1 The components database

It is the list of all the available FPLA device references. For example, the FPLA 74330 device which includes 12 inputs and 6 outputs looks like an electronic torch of 20 pins [Tan88]. Each of the

12 inputs undergoes a reversing inner operation, extending to 24 the number of internal variables. The heart of the device consists in 50 *AND* gates, each of them receives the 24 internal variables. All of this make up a 24×50 bits matrix we call $\mathcal{M}_{and}$. Each entry of the 50 gates is endowed with a fuse ; when the device is built, the 1200 components are untouched. The programming of the matrix leads to break up some fuses in order to get the product terms of the function to realize. The device presents 6 external outputs (6 *OR* gates). Each of these *OR* gates receives the outputs of the 50 *AND* gates. We have once again a matrix : $\mathcal{M}_{or}$ 50×6 bits, i.e. 300 fuses, and it is necessary to break up some of them to get the sum terms of the logical function. Figure 2 represents the architecture of a FPLA device.

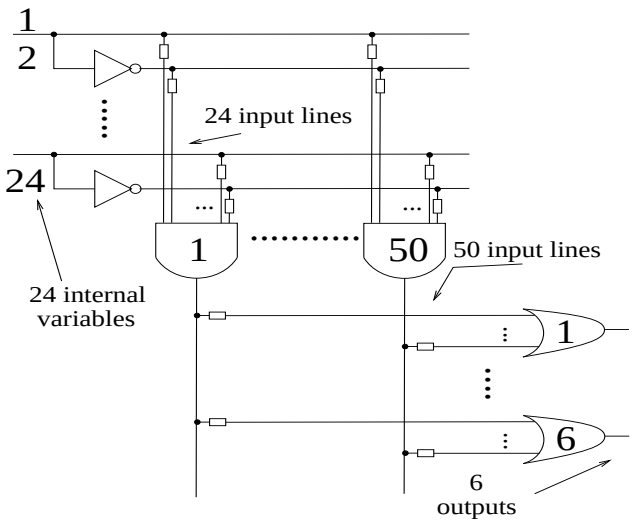


figure 2

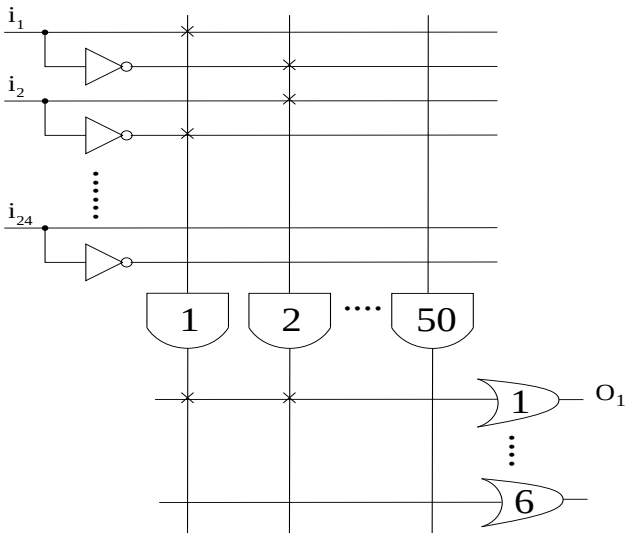


figure 3

For the sake of readibility, we represent more simply a FPLA device -we do not represent explicitely the fuses and all the input wires-. However, in order to point out that a fuse is intact, we schematize this thanks to a cross. For example, figure 3 represents the output $o_1 = i_1\bar{i}_2 + \bar{i}_1 i_2$ and in this case, $\mathcal{M}_{and}$ contains only 0 except at positions $(1,1)$, $(4,1)$, $(2,2)$, $(3,2)$ and $\mathcal{M}_{or}$ contains only 0 except at positions $(1,1)$, $(2,1)$[1] [2]

Figure 3 shows clearly that a FPLA device is organized into a hierarchy matching naturally the particular form of the logical functions it carries out. Indeed, in order to program a logical function under the form of sum of products, first, the products are programmed by taking a number of inputs -or their complements- and then, the sum of such products is programmed. This hierarchy in examplified in the following conditional rules schematizing the proof procedure. There are 4 hierarchy levels. Level 0 corresponds to the inputs, level 1 to the inputs on which a negation inference has been executed. From levels 0 or 1, it is possible to reach level 2 by performing a product. From one or several products, it is possible to reach level 3 carrying out the sum of such products.

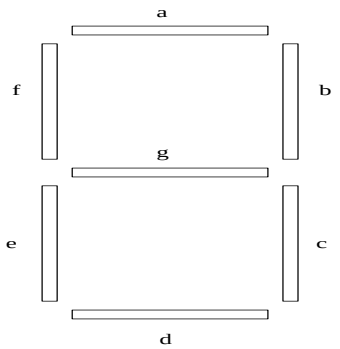
Let us see how to design the decoder with 7 segments with such kind of device. To do so and to avoid confusions let us recall what a decoder with 7 segments is according to [Wak91].

3.2 The decoder with 7-segments

This device aims at displaying 0,1, ...,15 in hexadecimal forms. Its inputs are 4-binary numbers, and its outputs are 7 segments (lights) a,b,c,d,e,f,g represented as follows :

[1] (i,j) stands for the line i and the column j

[2] The lines of $\mathcal{M}_{and}$ correspond successively to $i_1, \bar{i}_1, i_2, \bar{i}_2, \ldots$ Each column of $\mathcal{M}_{and}$ represents one *AND* gate. In $\mathcal{M}_{or}$, the columns correspond to the outputs and the lines to the products.



Digital display table with 7-segments

	Inputs				Outputs						
	D	C	B	A	a	b	c	d	e	f	g
0	ff	ff	ff	ff	tt	tt	tt	tt	tt	tt	ff
1	ff	ff	ff	tt	ff	tt	tt	ff	ff	ff	ff
2	ff	ff	tt	ff	tt	tt	ff	tt	tt	ff	tt
3	ff	ff	tt	tt	tt	tt	tt	tt	ff	ff	tt
4	ff	tt	ff	ff	ff	tt	tt	ff	ff	tt	tt
5	ff	tt	ff	tt	tt	ff	tt	tt	ff	tt	tt
6	ff	tt	tt	ff	ff	ff	tt	tt	tt	tt	tt
7	ff	tt	tt	tt	tt	tt	tt	ff	ff	ff	ff
8	tt	ff	ff	ff	tt	tt	tt	tt	tt	tt	tt
9	tt	ff	ff	tt	tt	tt	tt	ff	ff	tt	tt
A	tt	ff	tt	ff	ff	ff	ff	tt	tt	ff	tt
b	tt	ff	tt	tt	ff	ff	tt	tt	ff	ff	tt
C	tt	tt	ff	ff	ff	tt	ff	ff	ff	tt	tt
d	tt	tt	ff	tt	tt	ff	ff	tt	ff	tt	tt
E	tt	tt	tt	ff	ff	ff	ff	tt	tt	tt	tt
F	tt	tt	tt	tt	ff	ff	ff	ff	ff	ff	ff

Table 1 : the thruth table of the decoder with 7 segments.

For instance, to display 5, which is 0101 in 4-binary form, the outputs a, f, g, c and d must be at the state *tt* (for *true*) and all the other outputs must be at the state *ff* (for *false*). The initial truth table of the decoder is represented by Table 1. The latter is from [Wak91]. Nothing is displayed for the decimal value 15. Let us choose the FPLA 82S100 to design the circuit ; then the number of *AND* gates is 48. For the sake of readibility, we only give the proof for the output e represented by the following schema below :

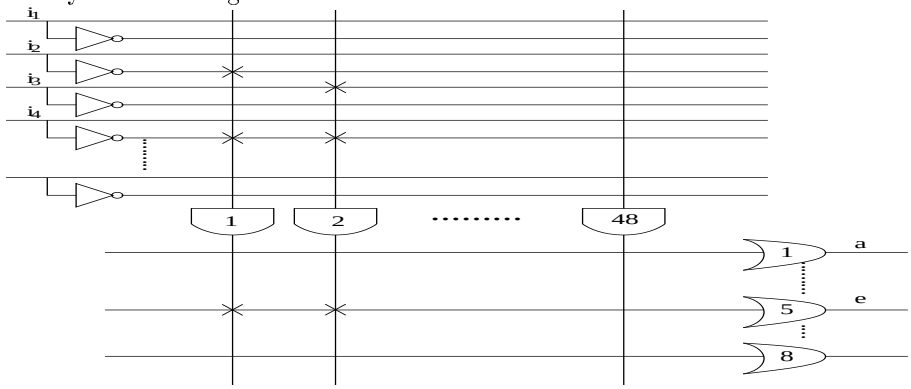


figure 4 : One solution for the output e with the FPLA 82S100.

3.3 The conditional rules

There are 7 rules specifying the behaviour of the logical gates in the FPLA device. Let us precise some notations that will be used all through this paper. $\vec{x}$, $\vec{y}, \ldots$ denote finite ordered sequences of *'tt'* (for the boolean value *true*), and *'ff'* (for the boolean value *false*). *not*, *and* and *or* are the usual logic connectors. $and(\vec{x}, \vec{y})$ denote $and(x_1, y_1)$, $and(x_2, y_2), \ldots, where, naturally$, x_i, y_i are boolean variables. *and* and *or* are considered binary functions. $F\$(out(\vec{x}), l, s, m, N)$ is a term representing the state of the device at a given time ; (i) $\vec{x}$ corresponds to a *vector* input or output, (ii) l is a *list* of list of integers whose elements are the identifiers for all the *rules* (inputs and outputs) which have taken part in the deduction of $\vec{x}$, (iii) s is a set *representing* all the *AND* gates used to build the circuit from the inputs to $\vec{x}$, (iv) m is the number of *AND* gates used to build the circuit, (v) N is the hierarchy level and as stated above : $0 \leq N \leq 3$.

In the conditional rules, we do not have to specify either the number of *NOT* gates or the number of *OR* gates used in the device to build the circuit. The reason is that depending on the specification of a user's problem, we can see directly if a reference of a FPLA device is suitable or not. If the thruth table has n inputs, then the FPLA device must provide at least n inputs, and if there are p outputs in the table, the device must include at least as many outputs. For example, to design the circuit of the decoder with 7 segments, the FPLA 74330 device is not suitable since it only has 6 outputs while we have 7 functions to carry out from a to g. On the other hand, the

FPLA 82S100 device which has 16 inputs, 48 *AND* gates and 8 *OR* gates will be *a priori* suitable -if the number of *AND* gates is sufficient- to design the circuit for the decoder.

The conditional rules are unique, they do not depend on a specific FPLA device. What depends on a particular reference of FPLA device is: its *capacity* (the number of internal variables, *AND* and *OR* gates).

3.3.1 The rules of the logical gates

Here, we give the 7 conditional rules; g is a function applied to the list l. g transforms l on each inference step according to the principle explained below (section 3.5.1). $\mathcal{C}_{and}$ expresses the number of *AND* gates of the device. The condition '$l_1 \notin l_2$' means that l_1 is not a sub-list of l_2 (it is the usual notion of sub-list for the type list). The third argument of $F\$$ is a set of integers. The latter are given by the process to rules in order to identify them. When $Current_id$ is used, it simply refers to the integer attributed to the clause by the process to identify it. Each element of this set is used to identify a *complete* product of the output. So, finally the cardinal of this set is exactly the number of *AND* gates used to built such output.

The rule for NOT is:

(c_1) $\qquad F\$(out(\vec{x}), l, \emptyset, 0, 0) = tt \Rightarrow F\$(out(not(\vec{x})), g(l), \emptyset, 0, 1) = tt$

not works necessarily at the level 0. $\vec{x}$ represents any input i and executing *not* consists in using the input $\bar{\imath}$. Then we reach the hierarchy level 1.

The rules for *AND* are:

(c_2) $\quad N_1 \leq 1 \ \wedge N_2 \leq 1 \ \wedge l_1 \notin l_2 \ \wedge F\$(out(\vec{x}), l_1, \emptyset, 0, N_1) = tt \ \wedge F\$(out(\vec{y}), l_2, \emptyset, 0, N_2) = tt$
$\Rightarrow F\$(out(and(\vec{x}, \vec{y})), g(l_1, l_2), \{Current_id\}, 1, 2) = tt$

This first form of *AND* works with two inputs i, j or their negations $\bar{\imath}, \bar{\jmath}$ (or a combination of both) represented by $\vec{x}$ and $\vec{y}$. We use an *AND* gate, thus m is set to 1, and we reach the hierarchy level 2.

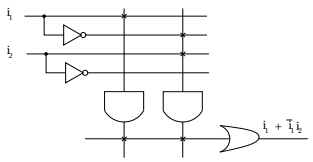
The condition $l_1 \notin l_2$ forbids the use of the idempotency law of and. Indeed, $and(x, x) = x$ can be inferred indefinitely and the process will never terminate!

(c_3) $\quad N_1 \leq 1 \ \wedge l_1 \notin l_2 \ \wedge F\$(out(\vec{x}), l_1, \emptyset, 0, N_1) = tt \ \wedge F\$(out(\vec{y}), l_2, s, 1, 2) = tt$
$\Rightarrow F\$(out(and(\vec{x}, \vec{y})), g(l_1, l_2), \{Current_id\}, 1, 2) = tt$

This second form of *AND* works with one of its components that is already a product (represented by $\vec{y}$) and the other one is necessarily an input i (or its negation $\bar{\imath}$), represented by $\vec{x}$. This is the same gate as the one used to complete the product by adding i (or $\bar{\imath}$). Consequently, m does not increase.

(c_4) $\quad N_1 \leq 1 \ \wedge F\$(out(\vec{x}), l, \emptyset, 0, N_1) = tt \Rightarrow F\$(out(\vec{x}), g(l), \{Current_id\}, 1, 2) = tt$

This last form of *AND* carries out a product from an input i (or from its negation $\bar{\imath}$). One *AND* gate is used and we find the same input at the exit of this *AND*. This inference rule is correct because of the idempotency law of and since $and(x, x)$ simplifies to x.
Example: if the circuit to design corresponds to the function $i_1 + \bar{\imath}_1.i_2$, an *AND* gate is used only by *clicking* on i_1. Thus, we get:



The rules of OR are :

$$(c_5) \quad 2 \leq \mathcal{C}_{and} \ \wedge l_1 \notin l_2 \ \wedge F\$(out(\vec{x}), l_1, s_1, 1, 2) = tt \ \wedge F\$(out(\vec{y}), l_2, s_2, 1, 2) = tt$$
$$\Rightarrow F\$(out(or(\vec{x}, \vec{y})), g(l_1, l_2), s_1 \ \cup \ s_2, 2, 3) = tt$$

c_5 works with two elementary products and it consists in applying an OR gate between two AND gates. Like in the AND rules, the condition $l_1 \notin l_2$ allows to control the use of the idempotency law of or, and we reach the hierarchy level 3.

$$(c_6) \quad 1 + m_2 \leq \mathcal{C}_{and} \ \wedge l_1 \notin l_2 \ \wedge F\$(out(\vec{x}), l_1, s_1, 1, 2) = tt \ \wedge F\$(out(\vec{y}), l_2, s_2, m_2, 3) = tt$$
$$\Rightarrow F\$(out(or(\vec{x}, \vec{y})), g(l_1, l_2), s_1 \ \cup \ s_2, m_2 + 1, 3) = tt$$

One of its inputs is at level 3 (represented by $\vec{y}$), then it is already a sum of products. We add one more product to this sum by applying c_6 between a sum and an elementary product.

$$(c_7) \quad 1 \leq \mathcal{C}_{and} \ \wedge F\$(out(\vec{x}), l, s, 1, 2) = tt \Rightarrow F\$(out(\vec{x}), g(l), s, 1, 3) = tt$$

This form of OR is only needed to get an elementary product in an output. For example, if a logical function is expressed by $i_1\bar{i}_2$, to get this expression as an output, we need to have it at level 3 since the outputs are necessarily at level 3. This is correct since $or(x, x) = x$.

3.4 The input file for the inference system

The input file for our inference system contains

1. the set of all conditional rules $c_1, \ldots, c_7$,
2. the set of inputs expressed by clauses, which represent the thruth table,
3. an inequation representing the outputs of the circuit to design.

The conditional rules are those given in the previous section. Now, let us make clear the inputs and the inequation :

3.4.1 The inputs

The number k of the inputs to specify depends on the problem the user wishes to solve. These k inputs are numbered from 1 to k, and the equations introducing the inputs are in the following form for any input of number j :

$$(j) \qquad F\$(out(\vec{i}_j), [j], \emptyset, 0, 0) = tt$$

For the example of the decoder with 7 segments, since we have 4 inputs A, B, C, D, we introduce 4 equations, numbered from (1) to (4) : (the 4 input vectors that correspond to A, B, C, D in Table 1).

$$(1) \qquad F\$(out(ff, ff, ff, ff, ff, ff, ff, ff, tt, tt, tt, tt, tt, tt, tt, tt), [1], \emptyset, 0, 0) = tt$$

$$(2) \qquad F\$(out(ff, ff, ff, ff, tt, tt, tt, tt, ff, ff, ff, ff, tt, tt, tt, tt), [2], \emptyset, 0, 0) = tt$$

$$(3) \qquad F\$(out(ff, ff, tt, tt, ff, ff, tt, tt, ff, ff, tt, tt, ff, ff, tt, tt), [3], \emptyset, 0, 0) = tt$$

$$(4) \qquad F\$(out(ff, tt, ff, tt, ff, tt, ff, tt, ff, tt, ff, tt, ff, tt, ff, tt), [4], \emptyset, 0, 0) = tt$$

3.4.2 The outputs of the circuit to design

The number of the outputs to specify also depends on the problem the user wishes to solve. Let o_1, o_2, ..., o_r be these outputs, the theorem to prove is an inequation that we get from the output columns of the truth table. To achieve this, a new variadic symbol '$A\$$' is introduced so that all the outputs are the subterms of the $A\$(\ldots)$ term. The '$out(\ldots)$' first subterm of it is the first output column of the truth table and so on. The inequation is as follows:

$$\mathcal{C}ard(s_1 \cup \ldots \cup s_r) \leq \mathcal{C}_{and} \wedge A\$(F\$(out(\vec{o}_1), l_1, s_1, m_1, 3), \ \ldots, \ F\$(out(\vec{o}_r), l_r, s_r, m_r, 3)) = tt \Longrightarrow$$

$\vec{o}_1, \ldots, \vec{o}_r$ are vectors with 2^k components if k is the number of the inputs of the problem, (for a problem with 3 inputs, the outputs are 8 value vectors). Note that the outputs are necessarily in the hierarchy level 3. The constraint $\mathcal{C}ard(s_1 \cup \ \ldots \ \cup s_r) \leq \mathcal{C}_{and}$ is to deal globally with the total number of the used AND gates.

For the example of the decoder with 7 segments, the inequation is the following:

$$\begin{array}{l}
\mathcal{C}ard(s_1 \cup \ \ldots \ \cup s_7) \leq \mathcal{C}_{and} \wedge \\
A\$(F\$(out(tt,ff,tt,tt,ff,tt,ff,tt,tt,tt,ff,ff,ff,tt,ff,ff), l_1, s_1, m_1, 3), \\
\quad F\$(out(tt,tt,tt,tt,tt,ff,ff,tt,tt,tt,ff,ff,tt,ff,ff,ff), l_2, s_2, m_2, 3), \\
\quad F\$(out(tt,tt,ff,tt,tt,tt,tt,tt,tt,tt,ff,tt,ff,ff,ff,ff), l_3, s_3, m_3, 3), \\
\quad F\$(out(tt,ff,tt,tt,ff,tt,tt,ff,tt,ff,tt,tt,ff,tt,tt,ff), l_4, s_4, m_4, 3), \\
\quad F\$(out(tt,ff,tt,ff,ff,ff,tt,ff,tt,ff,tt,ff,ff,ff,tt,ff), l_5, s_5, m_5, 3), \\
\quad F\$(out(tt,ff,ff,ff,tt,tt,tt,ff,tt,tt,ff,ff,tt,tt,tt,ff), l_6, s_6, m_6, 3), \\
\quad F\$(out(ff,ff,tt,tt,tt,tt,tt,ff,tt,tt,tt,tt,tt,tt,tt,ff), l_7, s_7, m_7, 3)) = tt \Longrightarrow
\end{array}$$

3.4.3 The demodulators

The demodulators are rewrite rules used to simplify the subterms the form of '$not(\ldots)$' or '$and(\ldots)$' or '$or(\ldots)$' which are subterms of subterms the form of '$F\$(\ldots)$'. In the present work, 'and' and 'or' are declared binary functions.

$$not(ff) \rightarrow tt\,; \quad not(tt) \rightarrow ff\,; \quad not(not(x)) \rightarrow x$$

$$and(ff,x) \rightarrow ff\,; \quad and(x,ff) \rightarrow ff$$

$$and(tt,x) \rightarrow x\,; \quad and(x,tt) \rightarrow x$$

$$or(ff,x) \rightarrow x\,; \quad or(x,ff) \rightarrow x\,; \quad or(tt,x) \rightarrow tt\,; \quad or(x,tt) \rightarrow tt$$

3.5 The inference system

All the inference rules of the maximal-unit-strategy described in [ABT96] are included in our system. The main inference rules we use are *deduction of new rules by Conditional-narrowing*, and *simplifying conditional rules and inequations*. Let us recall the first one. Let us start by precising that the terms the form of $F\$(\ldots)$ or $A\$(\ldots)$ are called terms of the high hierarchy.

Definition 1 (Conditional-narrowing) *If* (1) $\mathcal{C} \wedge s[u] = t \wedge L \Rightarrow A = B$ *is a conditional rule,* u *a non-variable subterm of* s *in the high hierarchy, and* (2) $\mathcal{C}' \Rightarrow \ v = w$ *an equation, then* $\mathcal{C} \wedge \mathcal{C}' \wedge \mathcal{C}'' \wedge u = v \wedge s[w] = t \wedge L \Rightarrow A = B$ *is a conditional-narrowing consequence from (1) and (2).* $\mathcal{C}''$ *is* $s = t \not\leftarrow a = b \ \forall a = b \in L$ *and* $\mathcal{C} \wedge \mathcal{C}' \wedge \mathcal{C}'' \wedge u = v$ *is satisfiable.*

Let us point out that this inference rule is to be performed between a conditional rule or an inequation (a pure negative clause) and an equation. As usual in refutation based methods, we infer in our system until the empty clause is generated. The history of the proof of this empty clause yields the design of the circuit. Consequently, through all of this paper, the terms *proof* and *circuit-design* have the same meaning. Now, we may define exactely how we compute the list l which is a representation of the circuit solution.

3.5.1 The list l and how to compute g ?

l is initialized to the empty list [ ]. As stated previously, each clause which is deduced during the proof is granted a number. According the different cases that may occur, we compute g after each conditional narrowing step as follows:

- conditional narrowing between (c_1) and an input identified by i :
  then $g(l) = [c_1, [i]]$.
- conditional narrowing between $c_i \in \{c_2, c_3, c_5, c_6\}$ and an equation : then $g(l_1, l_2) = [c_i, \nu l_1, l_2]$ if the narrowing is performed in the first component $F\$(out(or(\vec{x})), \ldots)$ of the gate else $g(l_1, l_2) = [c_i, l_1, \nu l_2]$ if it is performed in the second component $F\$(out(or(\vec{y})), \ldots)$. ν is the substitution of the evaluation.
- conditional narrowing between an equation and c_j, such that $c_j = c_4$ or $c_j = c_7$:
  then $g(l) = [c_j, \nu l]$. ν is the substitution of the evaluation.
- conditional narrowing between two clauses i and j such that $i \notin \{c_1, \ldots, c_7\}$ and $j \notin \{c_1, \ldots, c_7\}$,
  then $g(l_1, l_2) = l_2[\nu l_1]$ (resp. $g(l_1, l_2) = l_1[\nu l_2]$) if '$i$ narrows in j' (resp. 'j narrows in i'). ν is the substitution of the evaluation.

3.6 Correctness and completeness

Let us begin by making clear what *complete* means here. This is : if solutions exist we get all of them within a finite period of time, and if no solutions exist, within a finite period of time, we stop by pointing out to the users that there are no solutions. By assuming the usual fairness hypothesis, we get the following :

Theorem 1 *The design method decribed above is correct and complete.*

Proof : The correctness of this method is trivial. As regards completeness, it results straightly from that of the maximal-unit-strategy in [AA91] or [Der90] or [ABT96]. $\diamond$

Now, let us explain how the matrices may be issued from the lists provided by the empty clause.

3.7 Building the matrices

From matrices $\mathcal{M}_{and}$ and $\mathcal{M}_{or}$, it is easy for the user to break up some fuses in order to program the corresponding logical functions since they are, so to speak, a picture of the FPLA.

In $\mathcal{M}_{and}$, each line k corresponds to an input i (if k is odd) or to $\bar{\imath}$ (if k is even). The lines of $\mathcal{M}_{and}$ correspond successively to $i_1, \bar{\imath}_1, i_2, \bar{\imath}_2, \ldots$ Each column of $\mathcal{M}_{and}$ represents one *AND* gate. $\mathcal{M}_{and}[k, j] = 1$ means that the entry that corresponds to the line k is a component of the j^{th} *AND* gate.

In $\mathcal{M}_{or}$, the columns correspond to the outputs and the lines to the products. If $\mathcal{M}_{or}[k, j] = 1$ then the product k is a component of the sum j. $\mathcal{M}_{and}$ and $\mathcal{M}_{or}$ derive from the lists provided by the empty clause at the end of the proof.

The way to build the matrices $\mathcal{M}_{and}$ and $\mathcal{M}_{or}$ is the following :

3.7.1 The algorithm

$\mathcal{M}_{and}$ and $\mathcal{M}_{or}$ contain only 0. We build them by analyzing all the lists corresponding to the outputs and which are provided by the empty clause.

Let l_i be the list obtained for the output o_i ; we examine l_i from left to right, and let φ be the function that works on l_i, and such that φ modifies $\mathcal{M}_{and}$ and $\mathcal{M}_{or}$. Then, φ is specified as

follows :

$\varphi([k])$ translates to $\mathcal{M}_{and}[2*(k-1)+1, j] = 1$
$\varphi([a,[k]])$ translates to : if $a = c_1$ then $\mathcal{M}_{and}[2*k, j] = 1$ if $a = c_4$ then $\varphi(k)$ furthermore, $\mathcal{M}_{or}[j,i] = 1$ (i is the index of l_i) if $a = c_7$, then $\varphi(k)$
$\varphi([a,b,c])$ translates to : if $a \in \{c_2, c_3\}$ (a is a binary form of AND-inference) then $\varphi(b)$ and $\varphi(c)$ for the same value of j. Furthermore, for this value of j, $\mathcal{M}_{or}[j,i] = 1$ (i is the index of l_i) otherwise, a is a binary form of OR-inference, then $\varphi(b)$ and $\varphi(c)$.

j is an index of one of the columns in $\mathcal{M}_{and}$ that we increment by adding to it the value 1 if and only if the update of $\mathcal{M}_{and}$ provides a matrix that is different from the one described at the previous step of the calculus. In other words, if we put the value 1 somewhere in $\mathcal{M}_{and}$, and we do not change the existing matrix, the index of column provided by the function φ is the index of such column. On the other hand, if $\mathcal{M}_{and}$ is actually modified, then the corresponding modifications are updated in the first column in $\mathcal{M}_{and}$ that still is equal to 0 -the index of such column is incremented to 1- and then j is equal to this value.

j is initialized to 1.

Here are improvements to make our method more efficient.

4 Heuristic and Goal-oriented deletion criteria

The search space computed by the inference system seen above is still too large. For this reason, we will develop some heuristic and goal-oriented deletion criteria. Two cases will be focused on. The first one is based on some properties extracted from the system of rules. From this, we deduce a method to delete some useless clauses. In this case, we talk about *heuristic*. On the other hand, from the goal to refute, we will propose a criterion which can delete some useless clauses. Note with the deletion criteria presented here, we lose neither the correcteness or the completeness of the design method. Let us deal with the first case.

4.1 Heuristic deletion criteria

To extract the properties of the system of rules, one way is to observe its evolution during the process.

4.1.1 Gate properties

It is natural to take advantage of the commutativity of the AND and OR gates. For instance, it is useless to deduce two clauses which correpond respectively to the products AB and BA. These two clauses are syntactically different, but they are semantically the same.

Recall that the list l which is the second argument of a term the form of $F\$(out(...), l, s, m, N)$, is viewed as a binary tree. When it is of the form $[c_i, l_1, l_2]$, the root is c_i, the left (resp. the right) subtree is l_1 (resp. l_2). If l is of the form $[c_i, l_1]$, its root is c_i, and l_1 its left subtree. In this case its right subtree is the empty tree. When l is of the form $[n]$, the binary tree is simply the leaf n. After this, let us now define the notion of measure.

Definition 2 (measure) *1. Let be $\mathcal{L}$ the set of lists, $\mathcal{S}$ the set of lists of lists of integers, and "leaves" a function which gives the list of the leaves of a binary tree. Let us define $meas, meas : \mathcal{L} \to \mathcal{S}$ such that:*

$$meas([n]) = [[n]],$$

$$meas([c_i, l]) = [leaves(l)] \ \ for \ \ c_i \in \{c_1, c_4\},$$

$$meas([c_i, l_1, l_2]) = [leaves([c_i, l_1, l_2])] \ \ where \ \ c_i \in \{c_2, c_3\},$$

$meas([c_i, l_1, l_2]) = concat(meas(l_1), meas(l_2))$ *for* $c_i \in \{c_5, c_6\}$ *, where concat is the naturally concatenation on lists.*

and $meas([c_7, l]) = meas(l)$*.*

2. *Let us extend this notion on ground equations of the form* $u = tt$ *where the top operator of* u *is* $F\$$*,* $meas(u = tt) = meas(u_{|2})$*, the measure of the subterm of* u *at position* 2*.*

Definition 3 (lexicographic ordering) *1. Let be* $l = [l_1, \ldots, l_k]$ *a list of integer lists.* l *is lexicographically correct if each sublist* l_i *is ordered, and* $\forall i \in \{1, \ldots, k-1\}$*,* $l_i \ll l_{i+1}$*, where* $\ll$ *is the natural lexicographic order.*

2. *An equation* $u = tt$ *where the top operator of* u *is* $F\$$ *is* useful*, iff* $meas(u = tt)$ *is lexicographically correct.*

Lemma 1 *Deleting all not* useful *equations then the proof is preserved.*

For instance, for the following two equations :

(i) $F\$(out(\vec{A}\vec{B}), [c_3, [1], [2]], \{i\}, 1, 2) = tt$

(j) $F\$(out(\vec{B}\vec{A}), [c_3, [2], [1]], \{j\}, 1, 2) = tt$

where A and B are inputs, the measures are : $meas(i) = [[1, 2]]$ and $meas(j) = [[2, 1]]$. We only keep the first one because it is lexicographically correct.

Definition 4 (measure) *1. Let be* G *the set of all ground terms with top operator* $F\$$ *and* $\mathcal{L}$ *the set of lists of integers. Let us define* $meas, meas : G \to \mathcal{L}$ *such that* $\forall u \in G, meas(u) = leaves(u_{|2})$*, where* $u_{|2}$ *is the subterm of* u *at position 2, and "leaves" a function which gives the list of the leaves of a binary tree.*

2. *Let us extend this notion on ground equations of the form* $u = tt$ *where the top operator of* u *is* $F\$$*,* $meas(u = tt) = meas(u)$*.*

Definition 5 (lexicographic ordering) *1. Let be* l *a list of integers.* l *is lexicographically correct if each element of* l *is less than its successor.*

2. *An equation* $u = tt$ *where the top operator of* u *is* $F\$$ *is useful, iff* $meas(u = tt)$ *is lexicographically correct.*

Lemma 2 *All not useful equations can be deleted without losing the proof.*

For instance, for the following two equations :

(i) $F\$(out(\vec{A}\vec{B}), [c_3, [1], [2]], \{i\}, 1, 2) = tt$

(j) $F\$(out(\vec{B}\vec{A}), [c_3, [2], [1]], \{j\}, 1, 2) = tt$

where A and B are inputs, the measures are : $meas(i) = [1, 2]$ and $meas(j) = [2, 1]$. We keep only the first because it is the lexicographically correct one.

4.1.2 The intruders

The aim of the rules c_4 and c_7 is to increase the hierarchy level of an equation without changing its *boolean values*. Let us see the meaning of this *intrusion* through the example below. It is possible to deduce an equation which correponds to the product AB in two ways.

- The inputs A and B are :

  (1) $F\$(out(\vec{A}), [1], \emptyset, 0, 0) = tt$

  (2) $F\$(out(\vec{B}), [2], \emptyset, 0, 0) = tt$

- Conditional narrowing between (1) and (c_2) gives:

  (3) $N_2 \leq 1 \wedge [1] \notin l_2 \wedge F\$(out(\vec{y}), l_2, \emptyset, 0, N_2) = tt \Rightarrow F\$(out(and(\vec{A}, \vec{y})), [c_2, [1], l_2], \{3\}, 1, 2) = tt$

- Conditional narrowing between (2) and (3) gives:

  (4) $F\$(out(\vec{A}\vec{B}), [c_2, [1], [2]], \{4\}, 1, 2) = tt$

The other way to deduce the same product AB is as follows:

- Conditional narrowing between (2) and (c_4) gives:

  (5) $F\$(out(\vec{B}), [c_4, [2]], \{5\}, 1, 2) = tt$

- Conditional narrowing between (1) and (c_3) gives:

  (6) $[1] \notin l_2 \ \wedge \ F\$(out(\vec{y}), l_2, s, 1, 2) = tt \Rightarrow F\$(out(and(\vec{A}, \vec{y})), [c_3, [1], l_2], \{6\}, 1, 2) = tt$

- Conditional narrowing between (5) and (6) gives:

  (7) $F\$(out(\vec{A}\vec{B}), [c_3, [1], [c_4, [2]]], \{7\}, 1, 2) = tt$

Note that the case of OR gate is similar to the rule c_7.
To avoid such redundancy, we append two boolean arguments to the $F\$$ function. The rules $c_1, \ldots, c_7$ become:

(c_1) $\quad F\$(out(\vec{x}), l, \emptyset, 0, 0, ff, ff) = tt \Rightarrow F\$(out(not(\vec{x})), g(l), \emptyset, 0, 1, ff, ff) = tt$

(c_2) $\quad N_1 \leq 1 \wedge N_2 \leq 1 \wedge l_1 \notin l_2 \wedge F\$(out(\vec{x}), l_1, \emptyset, 0, N_1, ff, ff) = tt \wedge F\$(out(\vec{y}), l_2, \emptyset, 0, N_2, ff, ff) = tt$

$\Rightarrow F\$(out(and(\vec{x}, \vec{y})), g(l_1, l_2), \{CURRENT_ID\}, 1, 2, ff, ff) = tt$

(c_3) $\quad N_1 \leq 1 \wedge l_1 \notin l_2 \wedge F\$(out(\vec{x}), l_1, \emptyset, 0, N_1, ff, ff) = tt \wedge F\$(out(\vec{y}), l_2, s, 1, 2, ff, ff) = tt$

$\Rightarrow F\$(out(and(\vec{x}, \vec{y})), g(l_1, l_2), \{CURRENT_ID\}, 1, 2, ff, ff) = tt$

(c_4) $\quad N_1 \leq 1 \wedge F\$(out(\vec{x}), l, \emptyset, 0, N_1, ff, ff) = tt \Rightarrow F\$(out(\vec{x}), g(l), \{CURRENT_ID\}, 1, 2, tt, ff) = tt$

(c_5) $\quad 2 \leq \mathcal{C}_{and} \wedge l_1 \notin l_2 \wedge F\$(out(\vec{x}), l_1, s_1, 1, 2, IAND1, ff) = tt \wedge F\$(out(\vec{y}), l_2, s_2, 1, 2, IAND2, ff) = tt$

$\Rightarrow F\$(out(or(\vec{x}, \vec{y})), g(l_1, l_2), s_1 \cup s_2, 2, 3, or(IAND1, IAND2), ff) = tt$

(c_6) $\quad 1 + m_2 \leq \mathcal{C}_{and} \wedge l_1 \notin l_2 \wedge F\$(out(\vec{x}), l_1, s_1, 1, 2, IAND1, ff) = tt \wedge F\$(out(\vec{y}), l_2, s_2, m_2, 3, IAND2, ff) =$

$tt \Rightarrow F\$(out(or(\vec{x}, \vec{y})), g(l_1, l_2), s_1 \cup s_2, m_2 + 1, 3, or(IAND1, IAND2), ff) = tt$

(c_7) $\quad 1 \leq \mathcal{C}_{and} \wedge F\$(out(\vec{x}), l, s, 1, 2, IAND, ff) = tt \Rightarrow F\$(out(\vec{x}), g(l), s, 1, 3, IAND, tt) = tt$

(Inequation) $\quad \mathcal{C}ard(s_1 \cup \ \ldots \ \cup s_r) \leq \mathcal{C}_{and} \wedge$

$A\$(F\$(out(\vec{o_1}), l_1, s_1, m_1, 3, IAND_1, IOR_1), \ldots, F\$(out(\vec{o_r}), l_r, s_r, m_r, 3, IAND_r, IOR_r)) = tt \Rightarrow$

The fifth (resp. sixth) new argument in F$(...) is to carry out the redundancy problem on the AND (resp. OR) gate. At the beginning of the process they are both set to ff. After applying conditional-narrowing between an input and the rule c_4 (resp. c_7), it is set to tt. As a result, the conditional-narrowing can not be performed between the new deduced rule and all the AND rules (resp. OR rules).

Remark For the sake of readability, those new arguments are not reproduced in the remaining part of the paper.

Let us now deal with the goal-oriented deletion criteria.

4.2 Goal-oriented deletion criteria

The idea of this strategy is that, according to the goal to refute, how to forecast the rules that will never take part in the proof, and avoid their deduction. First of all, we present the strategy for circuits with one output ; next, we extend this strategy for more generalized ones. Let us in the following part, define some useful notions.

Definition 6 (Boolean Projection) *1. Let be $\mathcal{T}$ the set of all ground terms with top operator "out". $\forall i \in \{1, \ldots, arity(out)\}$, let us define π_i (for ith projection), $\pi_i : \mathcal{T} \rightarrow \{ff, tt\}$ such that $\forall u \in \mathcal{T}$, $\pi_i(u) = u_{|i}$, the subterm of u at position i.*

2. Let us extend this notion on ground terms whose top operator is "F$". $\pi_i(F\$(out(\vec{x}), l, e, m, N)) = \pi_i(out(\vec{x}))$.

3. Let us extend the boolean projection on ground equations of the form $u = tt$ where the top operator of u is "F$". We say: $\forall i \in \{1 \ldots arity(out)\}$, $\pi_i(u = tt) = \pi_i(u)$.

Definition 7 (Generalization to inequations) *Let us extend this notion on $\mathcal{I}$, ground terms whose top operator is "A$". $\forall i \in \{1 \ldots arity(out)\}$, $\forall u \in \mathcal{I}$, $\tilde{\pi}_i^j(u) = \pi_i(u_{|j})$, $\forall j \in \{1 \ldots arity(u)\}$, the boolean projection of the subterm of u at position j.*

Example : If a circuit has two inputs A and B, the product AB is represented by the equation (4) $F\$(out(ff, ff, ff, tt), [c_2, [1], [2]], \{4\}, 1, 2) = tt$ then $\pi_1(4) = ff$.

4.2.1 The case of circuits with one output

Lemma 3 *Let be [0] ($\mathcal{C}ard(s) \leq \mathcal{C}_{and} \wedge A\$(F\$(out(\vec{o}), l, s, m, 3)) \Rightarrow$) a cicruit to design with one output. Let be (eq) $u = tt$ an equation where the top operator of u is $F\$$, such that $\tilde{\pi}_i^1([0]) = \pi_i(eq) = tt$, for some i in $\{1 \ldots arity(out)\}$. Failing to perform conditional narrowing between (eq) and* OR *rules ($c_5, c_6,$ or c_7) then the proof is preserved.*

Proof of the lemma

To each equation $F\$(out(\vec{x}), l, s, m, N) = tt$ corresponds an algebraic expression. For instance, if the hierarchy level of the equation (the value of N) is equal to 2, the equation represents a product of the form $I_1 I_2 \cdots I_k$, where I_j are inputs or their negations. When the value of N is equal to 3, the equation corresponds to a sum of products of the form $P_1 + P_2 + \cdots + P_l$, where P_j are products as above.
Let us extend the boolean projection to algebraic representation to make the proof more readeable.

Definition 8 *Let be X an algebraic representation of a boolean expression. Let be $u = tt$ the equation where the top operator of u is $F\$$, and which corresponds to the expression X. We define Π_i, $\Pi_i(X) = \pi_i(u = tt)$.*

Property 1 *Let be X an algebraic expression in the normal form, i.e. $X = P_1 + \cdots + P_l$, where P_j are products. If $\exists$ i such that $\Pi_i(X) = ff$, then $\forall P \in \{P_1, \ldots, P_l\}$, $\Pi_i(P) = ff$.*

The meaning of this property in conditional rewriting is : for the output [0], where $\tilde{\pi}_i^1([0]) = ff$, conditional-narrowing between any equation eq such that $\pi_i(eq) = tt$ and an OR rules will not give rules which can take part in the design of [0]. That is why the above lemma is correct. $\diamond$
Example : Let be a circuit to design represented by the inequation :

$$[0]\ \mathcal{C}ard(s) \leq \mathcal{C}_{and} \wedge$$

$$A\$(F\$(out(ff,ff,tt,tt,tt,tt,tt,ff,tt,tt,tt,tt,tt,tt,tt,ff),l,s,2,3)) = tt \Rightarrow$$

Conditional narrowing between $(eq)\ u = tt$ such that $\pi_i(eq) = tt$, where $i \in \{1,2,8,16\}$, and OR rules should not be performed.

Note: It is worth noting that, we do not delete each equation $u = tt$ such that $\pi_i(u = tt) = tt$, since this equation would deduce a new one $u' = tt$ such that $\pi_i(u' = tt) = ff$. That is because $and(tt, ff) = ff$. However, in the following case we can delete this kind of equations.

Definition 9 (size of products) *Let be u a term with top operator $F\$$.*

1. *Let us define size, $size(u) = occ(u_{|2}, \{c_2, c_3\}) + 1$, where occ gives the number of occurrences of c_2 and c_3 in the list $s_{|2}$.*
2. *Let us extend the notion of size to equations $u = tt$, where u is a term whose top operator is $F\$$. We say: $size(u = tt) = size(u)$.*

If the hierarchy level of $u = tt$ is 2, then $size(u = tt)$ corresponds to the size of the product represented by the equation. This is illustrated by the example below.
Example: If at one step of the deduction process we get the product $p = B\bar{C}D$ which corresponds to the equation

$$(26)\ F\$(out(\vec{p}), [c_3, [2], [c_2, [c_1, 3], [4]]], \{26\}, 1, 2) = tt,$$

then $size(26) = 2 + 1 = 3$.

Lemma 4 *Let be $[0]\ (\mathcal{C}ard(s) \leq \mathcal{C}_{and} \wedge A\$(F\$(out(\vec{o}), l, s, m, 3)) \Rightarrow)$ a circuit to design with one output, and n the number of its inputs. Let be $(eq)\ u = tt$ an equation where the top operator of u is $F\$$, such that $size(eq) = n$ and $\tilde{\pi}_i^1([0]) = \pi_i(eq) = tt$, for some i in $\{1 \ldots arity(out)\}$. Deleting such equation (eq) then the proof is preserved.*

Proof: When all the inputs or their negation are present in a product, then appending an input or its negation do not yield a new interesting product. That is why the deletion is correct. $\diamond$
Example: For inputs A, B, and C, all the possibilities to expand the product $A\bar{B}C$ are $A\bar{B}CA \equiv A\bar{B}C$, $A\bar{B}C\bar{A} \equiv 0$, $A\bar{B}C\bar{B} \equiv 0$, ..., the latter do not generate a new different interesting product.

4.2.2 The case of circuits with several outputs

Lemma 5 *Let be*

$$[0]\ \ \mathcal{C}ard(s_1 \cup \ \ldots \ \cup s_r) \leq \mathcal{C}_{and} \wedge$$

$$A\$(F\$(out(\vec{o}_1), l_1, s_1, m_1, 3),\ \ldots,\ F\$(out(\vec{o}_r), l_r, s_r, m_r, 3)) \Rightarrow$$

a circuit to design, E the set of all i such that for each $j \in \{1, \ldots, r\}$, $\tilde{\pi}_i^j([0]) = ff$, n the number of inputs. Let be $u = tt$ an equation where the top operator of u is $F\$$, such that $\pi_i(u = tt) = tt$ for some i in E.

- *Failing to perform conditional narrowing between $u = tt$ and* OR *rules,*
- *and deleting all the the equations $u = tt$ such that $size(u = tt) = n$,*

then the proof is preserved.

Proof : It is deduced naturally by generalizing lemma 2 and lemma 3. $\diamond$
Example : Let be A, B, C, and D the inputs of a circuit to design which has two outputs represented by the inequation

$$[0] \quad \mathcal{C}ard(s_1 \cup s_2) \leq \mathcal{C}_{and} \wedge$$

$$A\$(F\$(out(tt, ff, ff, ff, tt, tt, tt, ff, tt, tt, ff, ff, tt, tt, tt, ff), l_1, s_1, m_1, 3),$$

$$F\$(out(ff, ff, tt, tt, tt, tt, tt, ff, tt, tt, tt, tt, tt, tt, tt, ff), l_2, s_2, m_2, 3)) = tt \Rightarrow$$

In this case $E = \{2, 8, 16\}$. Conditional narrowing between equation (eq) $u = tt$ such that $\pi_i(eq) = tt$, where $i \in E$, and OR rules should not be performed. Furthermore, if $size(eq) = 4$ (the number of inputs), then eq should be deleted.

4.3 Presentation of some strategies

4.3.1 Singular refutation

This consists on priviledging the deduction of outputs containing the maximal number of ff. To do so, let us note for each output o_l, E_l the set of $i \in \{1, \ldots, arity(out)\}$, such that $\pi_i(o_l) = ff$. The set of $\{E_l\}$ is sorted according to a discreasing order of their number of elements. Consequently, let us assume that we have: $\mathcal{C}ard(E_1) > \mathcal{C}ard(E_2) > \cdots > \mathcal{C}ard(E_n)$, where n is the number of outputs. We start by designing the output o_1, then o_2, ...
In the example above, we try to deduce the first subterm of the term $A\$$, where $E_1 = \{2, 3, 4, 8, 11, 12, 16\}$, and after, the second with $E_2 = \{1, 2, 8, 16\}$. In this manner, we generate a smaller number of clauses at each step of the deduction.

4.3.2 Saturation of products

The architecture of the FPLA imposes us to deduce equations in a normal form (sum of products). Our conditional rules are written in such a way that the circuit is designed in this normal form. This strategy consists in deducing all the products before the sums. We start deducing all the clauses of hierarchy level 2, and next, we generate all the ones of hierarchy level 3. This method is interesting when the number of inputs of the circuit is small enough and the sizes of the products are too large.

5 A brief presentation of CDR

CDR is a software to design combinational circuits by means of SSI elements or FPLA devices. The approach on the SSI case is the one developed in [ABCM98a]. To design an FPLA circuit the above appraoch is used. The inference system of CDR is written in C and its interface in TCL/TK. It runs under SunOs or SunSolaris operating system and its size is about 500 Kb. CDR is fully automatic. To design a FPLA circuit, the users' inputs are only the truth table of the circuit to design and the reference of the FPLA device to be used. Thus the users need not master any notion of rewriting. The results of CDR are two matrices which will be the input data of dedicated machines to build the actual chips. 0 and 1 are the only components of such matrices. 0 is for open fuse and 1 is for intact one.

As an example, for the decoder with 7 segments, CDR computes a first solution for the output e in about 73 seconds, and a complete solution for the decoder in about 10 minutes.

For every circuit-solution, CDR provides users with information to help them to make a decision. One significant peace of information refers to the occupation rate of the device in the case of the proposed solution. If the solution is not suitable for some reasons, CDR will compute another solution. When no solution exist CDR will stop by pointing out the non-existence of a solution by means of the input device. In this case, the users have to replace the FPLA device and rerun CDR. In the future, CDR will be installed on the Web. Now, let us see some excerpts of the design of the decoder with 7 segments by CDR.

6 Example

We start with the full contents of the input file for the inference system (i.e. all the conditional rules of sections 3.3 and 3.4 and the reference 82S100 of the device). We append the following demodulators in this file.

$$(c_8)\ not(ff) \to tt\ ;\quad (c_9)\ not(tt) \to ff\ ;\quad (c_{10})\ not(not(x)) \to x$$

$$(c_{11})\ and(ff,x) \to ff \quad (c_{12})\ and(x,ff) \to ff$$

$$(c_{13})\ and(tt,x) \to x \quad (c_{14})\ and(x,tt) \to x$$

$$(c_{15})\ or(ff,x) \to x\ ;\ (c_{16})\ or(x,ff) \to x\ ;\ (c_{17})\ or(tt,x) \to tt\ ;\quad (c_{18})\ or(x,tt) \to tt$$

The inequation is : (0) $\mathcal{C}ard(s_1 \cup \ \ldots\ \cup s_7) \leq 48 \wedge$

$$\begin{aligned}
&A\$(F\$(out((tt,ff,tt,tt,ff,tt,ff,tt,tt,tt,ff,ff,ff,tt,ff,ff)),l_1,s_1,m_1,3),\\
&\quad F\$(out((tt,tt,tt,tt,tt,ff,ff,tt,tt,tt,ff,ff,tt,ff,ff,ff)),l_2,s_2,m_2,3),\\
&\quad F\$(out((tt,tt,ff,tt,tt,tt,tt,tt,tt,tt,ff,tt,ff,ff,ff,ff)),l_3,s_3,m_3,3),\\
&\quad F\$(out((tt,ff,tt,tt,ff,tt,tt,ff,tt,ff,tt,tt,ff,tt,tt,ff)),l_4,s_4,m_4,3),\\
&\quad F\$(out((tt,ff,tt,ff,ff,ff,tt,ff,tt,ff,tt,ff,ff,ff,tt,ff)),l_5,s_5,m_5,3),\\
&\quad F\$(out((tt,ff,ff,ff,tt,tt,tt,ff,tt,tt,ff,ff,tt,tt,tt,ff)),l_6,s_6,m_6,3),\\
&\quad F\$(out((ff,ff,tt,tt,tt,tt,tt,ff,tt,tt,tt,tt,tt,tt,tt,ff)),l_7,s_7,m_7,3)) = tt \Rightarrow
\end{aligned}$$

6.1 Some significant steps of the proof process

For the sake of space, we will only consider the output e, computed by CDR in about 73 seconds. All the other outputs are processed similarly. The main steps of the design process of e are the following.

By conditional-narrowing between 4 and c_1 simplifying with c_8 and c_9, we get :

(27) $F\$(out(tt,ff,tt,ff,tt,ff,tt,ff,tt,ff,tt,ff,tt,ff,tt,ff),$
$[c_1,[4]],\{27\},0,1,ff,ff) = tt$

By conditional-narrowing between 2 into c_1 simplifying with c_8 and c_9, we get :

(29) $F\$(out(tt,tt,tt,tt,ff,ff,ff,ff,tt,tt,tt,tt,ff,ff,ff,ff),$
$[c_1,[2]],\{29\},0,1,ff,ff) = tt$

By conditional-narrowing between 3 into c_2 simplifying with c_{11} and c_{12}, we get :

(32) $N_2 \leq 1 \wedge [3] \notin l_2 \wedge F\$(out(y_1,\ldots,y_{16}),l_2,\{27\},0,N_2,ff,ff) = tt$
$\Rightarrow F\$(out(and(ff,y_1),\ldots,and(tt,y_{16})),[c_2,[3],l_2],\{32\},1,2,ff,ff) = tt$

By conditional-narrowing between 29 into c_2, we get :

(51) $N_2 \leq 1 \wedge [c_1,[2]] \notin l_2 \wedge F\$(out(y_1,\ldots,y_{16}),l_2,\{29\},0,N_2,ff,ff) = tt$
$\Rightarrow F\$(out(and(tt,y_1),\ldots,and(ff,y_{16})),[c_2,[c_1,[2]],l_2],s_2,1,2,ff,ff) = tt$

By conditional-narrowing between 27 and 32 simplifying with c_{11} and c_{12}, we get :

(55) $F\$(out(ff,ff,tt,ff,ff,ff,tt,ff,ff,ff,tt,ff,ff,ff,tt,ff),$
$[c_2,[3],[c_1,[4]]],\{55\},1,2,ff,ff) = tt$

By conditional-narrowing between 27 into 51 simplifying with c_{11} and c_{12}, we get :

(91) $F\$(out(tt,ff,tt,ff,ff,ff,ff,ff,tt,ff,tt,ff,ff,ff,ff,ff),$
$[c_2,[c_1,[2]],[c_1,[4]]],\{91\},1,2,ff,ff) = tt$

By conditional-narrowing between 91 into c_5, we get :

(213) $[c_2,[c_1,[2]],[c_1,[4]]] \notin l_2 \wedge F\$(out(y_1,\ldots,y_{16}),l_2,s_2,1,2,IAND,ff) = tt$
$\Rightarrow F\$(out(or(tt,y_1),\ldots,or(ff,y_{16})),[c_5,[c_2,[c_1,[2]],[c_1,[4]]],l_2],$
$\{91\} \cup s_2,2,3,IAND,ff) = tt$

By conditional-narrowing between 55 into 213 simplifying with c_{14} and c_{15}, we get :
(816) $F\$(out(tt,ff,tt,ff,ff,ff,tt,ff,tt,ff,tt,ff,ff,ff,tt,ff),$
$[c_5,[c_2,[c_1,[2]],[c_1,[4]]],[c_2,[3],[c_1,[4]]]],\{91,55\},2,3,ff,ff) = tt$

Simplifying [0] by (816), we get the fifth subterm with F\$ of the inequation, which is to capture the output e of the circuit.

Notes

- At some step of the design process we get the following rule.
  By conditional-narrowing between 4 and c_4, we obtain :
  (39) $F\$(out(ff,tt,ff,tt,ff,tt,ff,tt,ff,tt,ff,tt,ff,tt,ff,tt),$
  $[c_4,[4]],\{39\},1,2,tt,ff) = tt$
  This equation (39) is the representation of the initial input A at the hierarchy level 2. It also represents the product A (i.e. A=AND(A,A)). This equation is deduced via c_4, it is an intruder, since the fifth argument is set to tt. That way, it is not possible to perform a conditional narrowing between it and the rule c_3 in which the fifth argument is ff.
  On the other hand, since $\forall j \in \{1,\ldots,7\}, \tilde{\pi}^j_{16}([0]) = ff$, where [0] is the inequation representing the circuit to design, any conditional narrowing between (39) and a *OR* rule (c_5, c_6, *or* c_7) should not be performed.

- At some stage of the deduction we also get the following equation :
  (303) $F\$(out(ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,tt),$
  $[c_3,[1],[c_3,[2],[c_2,[3],[4]]]],\{303\},1,2,ff,ff) = tt$

  With (303) we have $\pi_{16}(303) = tt$, and $size(303) = 3+1 = 4$ *(the number of inputs).* On the other hand, $\forall j \in \{1,\ldots,7\}, \tilde{\pi}^j_{16}([0]) = tt$, then this equation is deleted.

7 Conclusion

To the best of our knowledge, the method we have described above is the first rewrite based method to design FPLA combinational circuits. It is based on the maximal unit-strategy extended to the case where built-in algebras are to be considered. The inference system is basically the same as the one presented in [ABT96]. Our method is efficient because original heuristic and goal-oriented deletion criteria and subtle strategies are included. By means of the inequations representing the circuit to design, we can decide that some deduced clauses are to be suppressed because they are not necessary in the design process. Furthermore, by using the inequations we need not apply conditional-narrowing between some kinds of clauses. Through those two mechanisms, our search space is reduced tremendously. This approach is implemented in CDR which, though it cannot currently support graphical input, is user-friendly and users need not master rewriting to design circuits.

In the future, we will work on both the improvement of the method and its extension to the case of sequential FPLA circuits.

References

[AA91] N. Anantharaman and N. Andrianarivelo. A semi-decision procedure in Horn Theories. Technical report, Laboratoire d'Informatique Fondamentale d'Orléans–Laboratoire de Recherche en Informatique d'Orsay, 1991.

[ABCM98a] N. Andrianarivelo, W. Bousdira, J. Chabin, and Z. Maazouzi. A conditional rewrite-based method for designing combinational logic circuits. Technical Report 98-4, Laboratoire d'Informatique Fondamentale d'Orléans, 1998.

[ABCM98b] N. Andrianarivelo, W. Bousdira, J. Chabin, and Z. Maazouzi. Designing FPLA Combinational Circuits by Conditional Rewriting. In H. Prade, editor, *Wiley publisher*, pages 373–377, Brighton, UK, 1998. 13th European Conference on Artificial Intelligence.

[ABT96] N. Andrianarivelo, W. Bousdira, and J-M. Talbot. On theorem-proving in Horn theories with built-in algebras. In J. Calmet, J.A. Campbell, and J. Pfalzgraf, editors, *Lecture Notes in Computer Science*, volume 1138, pages 320–338, Steyr, Austria, 1996. Third International Conference on Artificial Intelligence and Symbolic Computation.

[Der90] N. Dershowitz. A maximal-literal strategy for Horn Clauses. In S. Kaplan and M. Okada, editors, *2nd International Workshop on Conditional Term Rewriting Systems*, volume 516 of *Lecture Notes in Computer Science*, pages 143–154. Springer-Verlag, June 1990.

[DJ90] N. Dershowitz and J-P. Jouannaud. Rewriting Systems. In J. Van Leuven, editor, *Handbook of Theoretical Computer Science*. Elsevier Science Publishers North-Holland, 1990.

[KKR90] C. Kirchner, H. Kirchner, and M. Rusinowitch. Deduction with Symbolic Constraints. *Revue Française d'Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on Automatic Deduction.

[Smo89] G. Smolka. *Logic Programming over Polymorphically Order-Sorted Types.* PHD Thesis, Universitat Kaiserslautern, FB Informatik, West Germany, 1989.

[Tan88] A. Tanenbaum. *Architecture de l'ordinateur. Du circuit logique au logiciel de base.* InterEditions, 1988.

[Wak91] John F. Wakerly. Digital design principles and practices. Prentice Hall International Editions, 1991.

Publications récentes du LIFO
LIFO's recent publications

97-7 E. MELIN, B. RAFFIN, X. REBEUF, B. VIROT
A Unifying Approach of Data and Task Parallelism Based on Structural Clock Communication Paradigm (extended version)

97-8 S. ANANTHARAMAN, Editor
Proceedings of UNIF'97, International Workshop on Unification

97-9 F. BENTAYEB, D. LAURENT
Inversion de l'algrèbre relationnelle et mises à jour

97-10 I. PARFENOFF
An efficient parallel algorithm for maximum matching in P_4-tidy graphs

97-11 F. ROUSSEL, I. RUSU
An O(m2+mn) algorithm to recognize Meyniel graphs

97-12 M. BAHMA, G. HAINS
Prevision des performances de la θ-jointure et de la multi-jointure pour des machines SN

97-13 F. LOULERGUE, G. HAINS
Design of an event-oriented functionnal parallel language : the finite case

97-14 S. LIMET, F. SAUBION
A proof method for logic programming

97-15 E. MELIN, B. RAFFIN, X. REBEUF, B. VIROT
An Automatic Distribution of Sequential Code Fitting Irregular Data Accesses

Pour récupérer les rapports de recherche du LIFO / To get LIFO's research reports :

- Par WWW : http://www.univ-orleans.fr/SCIENCES/LIFO/Recherche/Publis/RR/index.html
- Par FTP : ftp://ftp-lifo.univ-orleans.fr/pub/RR
- Par courrier papier adressé à / By surface mail adressed to :
  Arnaud Lallouet, LIFO, Université d'Orléans
  4, rue Léonard de Vinci, BP 6759
  F-45067 Orléans cedex 2, FRANCE
  en mentionnant votre adresse.
- Par courrier électronique à / By e-mail to : Arnaud.Lallouet@lifo.univ-orleans.fr