# Designing FPLA combinational circuits by conditional rewriting.

**N. Andrianarivelo, W. Bousdira, J. Chabin and Z. Maazouzi**[1]

**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 heuristics 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. The present work is an improvement of our first work [ABCM98a]. Both new goal-oriented deletion criteria inference rules and strategies have been added. 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 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.

[1] LIFO, Dépt. Informatique, Université d'Orléans. 45067 Orléans Cedex 02 (Fr.) e-mail :{andria, bousdira, chabin, maazouzi}@univ-orleans.fr


ECAI 98. *13th European Conference on Artificial Intelligence*
Edited by Henri Prade
Published in 1998 by John Wiley & Sons, Ltd.

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. In order to illustrate the various notions, we deal with the design of the decoder with 7 segments ; so, Section 3 also presents the decoder. In Section 4, we get a general view of heuristic and goal-oriented deletion criteria and strategies. Then we give a brief presentation of CDR in Section 5 and we conclude in Section 6.

## 2 Designing with FPLA devices

The architecture of the method is given in Figure 1 :
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.

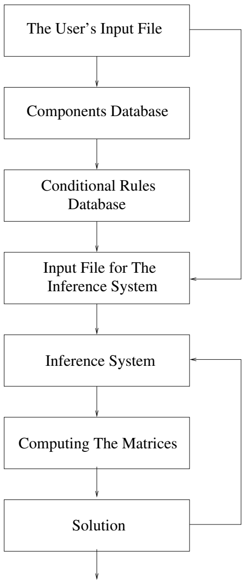



**Figure 1.** The architecture of our expert system

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 $24 \times 50$ 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 : $\mathcal{M}_{or}$ $50 \times 6$ 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.a represents the architecture of a FPLA device.

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 2.b represents the output $o_1 = i_1\bar{\imath}_2 + \bar{\imath}_1 i_2$ 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 $(1,1),\ (2,1)$[2] [3]

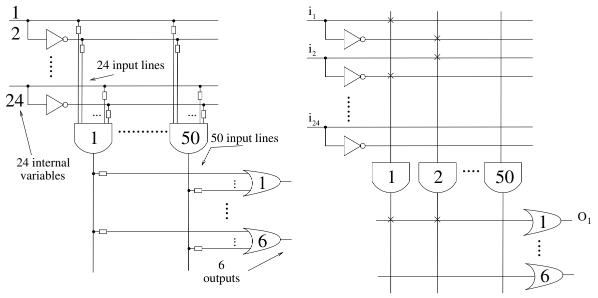


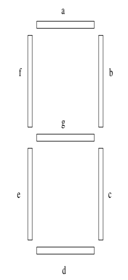

**Figure 2.** fig. 2.a fig. 2.b

Figure 2.b 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 :




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.

**Figure 3.** The decoder with 7 segments

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

[3] 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. In $\mathcal{M}_{or}$, the columns correspond to the outputs and the lines to the products.

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 portion of the circuit solution for the output $e$ represented by the following schema below :

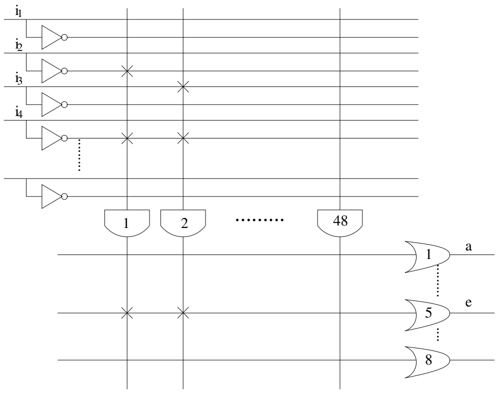



**Figure 4.** One solution for the output $e$ with the FPLA 82S100

## 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. $\vec{x},\ \vec{y},\dots$ 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. $and(\vec{x},\vec{y})$ denotes $and(x_1,y_1),\ and(x_2,y_2),\dots, where, naturally,\ x_i,\ y_i$ 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

The conditional rules are numbered $c_1, c_2, \dots c_7$. Because of a lack of space, we give here only the five first of them. The others can be found in [ABCM98b] ;

Here we explain what appears in the conditional rules : $g$ is a function applied to the list $l$. $g$ transforms $l$ on each inference step according to some principle. This transformation is intended to make the list $l$ a representation of the circuit solution. $l$ is defined in [ABCM98b] and how to compute $g$ is also fully explained in it. $\mathcal{C}_{and}$ 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 $(c_1)$ :

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

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 $i_1 + \bar{\imath}_1.i_2$, a *AND* gate is used only by *clicking* on $i_1$. Thus, we get the schema of Figure 5.
The rules of *OR* are similar to that of *AND*. Here we give only one of them : $c_5$ works with two elementary products and it consists in applying an *OR* gate between two *AND* gates. Like in the *AND* rules,

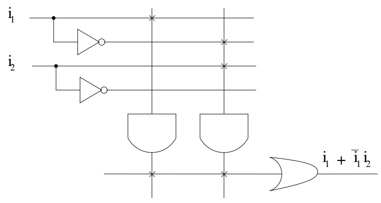



**Figure 5.** Example of the use of $c_4$

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.

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

## 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 $c_1, \ldots, c_7$. 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 all 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).

$$F\$(out(ff,ff,ff,ff,ff,ff,ff,ff,tt,tt,tt,tt,tt,tt,tt,tt), [1], \emptyset, 0, 0) = tt$$
$$F\$(out(ff,ff,ff,ff,tt,tt,tt,tt,ff,ff,ff,ff,tt,tt,tt,tt), [2], \emptyset, 0, 0) = tt$$
$$F\$(out(ff,ff,tt,tt,ff,ff,tt,tt,ff,ff,tt,tt,ff,ff,tt,tt), [3], \emptyset, 0, 0) = tt$$
$$F\$(out(ff,tt,ff,tt,ff,tt,ff,tt,ff,tt,ff,tt,ff,tt,ff,tt), [4], \emptyset, 0, 0) = tt$$

### *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 $o_1,\ o_2, \ldots,\ o_r$ be these outputs, then 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 :

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

$\vec{o}_1, \ldots,\ \vec{o}_r$ 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 :

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

### *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.

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

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

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

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

## 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*. Conditional narrowing 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.

## 3.6 Correctness and completeness

Let us begin by making clear what *complete* means here. That 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]. $\diamond$

When the empty clause is generated, the circuit solution given by the proof process is expressed under the form of two matrices $\mathcal{M}_{and}$ and $\mathcal{M}_{or}$. From these matrices, 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. Initially, $\mathcal{M}_{and}$ and $\mathcal{M}_{or}$ contain only 0. The process '*Computing The Matrices*' consists in examining the circuit solution and the lists $l_1, \ldots, l_i$ to put the value 1 at some positions in $\mathcal{M}_{and}$ and $\mathcal{M}_{or}$. So, everywhere the value is 1. It means that the fuse is intact, and if the value is 0, the fuse is open -not operational-. The way the matrices may be issued from the lists $l_1, \ldots, l_i$ provided by the empty clause is explained in [ABCM98b].

## 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 have developed 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 avoid to generate some useless clauses. On the other hand, from the goal to refute, we have proposed a criterion which can delete some useless clauses or avoid to generate them so we lose neither the correcteness nor the completeness of the design method. All this part is fully detailed in [ABCM98b].

## 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 approach 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.

## 6 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 family of 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 98-1, Laboratoire d'Informatique Fondamentale d'Orléans, 1998.

[ABCM98b] N. Andrianarivelo, W. Bousdira, J. Chabin, and Z. Maazouzi. Designing FPLA combinational circuits by conditional rewriting. Technical Report 98-2, Laboratoire d'Informatique Fondamentale d'Orléans, 1998.

[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.