Symbolic Design of Combinational and Sequential Logic Circuits Implemented by Two-Level Logic Macros

GIOVANNI DE MICHELI, MEMBER, IEEE

Abstract—**This paper presents a method for the optimal synthesis of combinational and sequential circuits implemented by two-level logic macros, such as programmable logic arrays. Optimization consists of finding representations of switching functions corresponding to minimal-area implementations. The design of optimization is based on two steps: symbolic minimization and constrained encoding. Symbolic minimization yields an encoding-independent *sum of products* representation of a switching function which is minimal in the number of product terms. The minimal symbolic representation is then encoded into a compatible Boolean representation. The algorithms for symbolic minimization and the related encoding problems are described. The computer implementation and the experimental results are then presented.**

I. Introduction

THE AUTOMATED synthesis of regular modules for very large scale integrated (VLSI) circuit design decreases design time and ensures functional correctness. In order to compete with manual designs, computer-aided synthesis techniques must include optimization procedures which attempt to minimize both silicon area and circuit switching times.

We present here a method for the optimal synthesis of digital modules implementing combinational and/or sequential switching functions. Modules are implemented by two-level logic macros, such as programmable logic arrays (PLA's) and synchronous registers. Optimization consists of finding representations of switching functions at the logic level that minimize the silicon area taken by their physical implementation (referred to as *cost* in short), without considering the interconnection area. We assume that the digital modules to be implemented can be described by tables. In general, tabular descriptions can be obtained in a straightforward way from structural-level systems descriptions in Hardware Description Languages (HDL), as in the case of the Yorktown Silicon Compiler [3].

In the standard approach to synthesis [10], Boolean representations of switching functions are obtained from the structural description by representing each mnemonic entry in a table (or each variable in a HDL program) by Boolean variables. The optimization of logic functions (and, in particular, two-level logic minimization) is performed on the Boolean representation. The result of logic optimization is heavily dependent on the representation of the variables. As an example, the complexity (in particular, the minimal cardinality of a two-level implementation) of the combinational component of a finite-state machine depends on the assignment of Boolean variables to the internal states [9].

The **symbolic design** methodology presented here avoids the dependence on the variable representation in the optimization process and consists of two steps: 1) determine an optimal representation of a switching function independently on the encoding of its inputs and outputs; 2) encode the inputs and outputs so that they are compatible with the optimal representation. This technique can be applied to solve the following problems of logic design:

- P1) find an encoding of the inputs (or some inputs) of a combinational circuit that minimizes its cost;
- P2) find an encoding of the outputs (or some outputs) of a combinational circuit that minimizes its cost;
- P3) find an encoding of both the inputs and the outputs (or some inputs and some outputs) of a combinational circuit that minimizes its cost;
- P4) find an encoding of both the inputs and the outputs (or some inputs and some outputs) of a combinational circuit that minimizes its cost and such that the encoding of the inputs is the same as the encoding of the outputs (or the encoding of some inputs is the same as the encoding of some outputs).

Finding an optimal state assignment of a sequential circuit is equivalent to solving problem P4, when the sequential circuit is implemented by feeding back (possibly through registers) some outputs of a combinational circuit to its inputs. Similarly, finding the encodings of the signals connecting two (or more) combinational circuits, that minimize the total cost, can be reduced to problem P4. The author presented in [4] and [5] an approximation to the solution of the state assignment problem, in which the

Manuscript received January 29, 1986; revised May 19, 1986. This research is a part of the project "Research on topological design tools for structured logic arrays," sponsored by the National Science Foundation under Contract ECS-8121446.

The author is with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

IEEE Log Number 8609826.

cost was minimized with regard only to the encoding of the inputs. In particular, the technique presented in [4] and [5] solved only problem P1. Problem P2 was attacked by Nichols [13], but the algorithm he presented could deal only with small-scale circuits. We refer the reader to [5] for an extended set of references and a critical survey of most of the previous techniques for state assignment.

Though the symbolic design methodology is fairly general, we restrict our attention here to two-level *sum of products* implementations. Since the area of the physical implementation has a complex functional dependence on the function representation (even by using PLA implementations [5], we consider a simplified optimization technique that leads to quasi-minimal areas. In particular, we attempt to find first *a sum of products* representation that is minimal in the number of products, and then a representation of the input/output that is minimal in the number of Boolean variables.

The difficulty in solving problems P2–P4 is related to finding a minimal two-level representation of a switching function independently of the encoding of both inputs and outputs. We introduce here a technique called **symbolic minimization**. Symbolic minimization consists of determining a minimal encoding-independent two-level *sum-of-products* representation of a switching function. It is minimal in number of product terms and independent of the encoding of all (or part of) the inputs and outputs [6]. The minimal symbolic representation is an intermediate step towards the determination of a corresponding Boolean representation. For this reason, three encoding problems are introduced to transform the minimal symbolic cover into an equivalent Boolean representation.

Section II contains a general overview of the symbolic design methodology and is an informal introduction to the problem. Then, in Section III, we present in detail the properties of the symbolic representation and an algorithm for symbolic minimization. The three new encoding problems are introduced in Section IV, as well as an algorithm for constrained encoding. The computer implementation of the symbolic minimization and the encoding algorithms is also presented along with experimental results.

II. Overview

In this section, we present an informal overview of symbolic design. The methodology is introduced by elaborating on an example. We consider first a combinational circuit (Fig. 1) and we use symbolic design to find a representation of its inputs and outputs and a corresponding two-level implementation that minimize its cost. This is equivalent to solving problem P3 of Section I. Note that problem P1 or P2 can be derived from problem P3 by considering only the circuit inputs or ouputs.

Example 1: The following truth table specifies a combinational circuit; in particular, an instruction decoder. There are three fields: the first is related to the addressing mode, the second to the operation code, and the third one to the corresponding control signal. The circuit has two

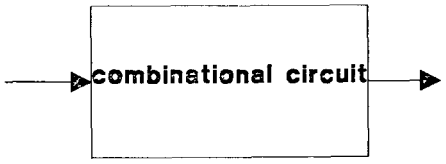


Fig. 1.

inputs and one output. Each row specifies a symbolic output for any given combination of symbolic inputs

INDEX	AND	CNTA
INDEX	OR	CNTA
INDEX	JMP	CNTA
INDEX	ADD	CNTA
DIR	AND	CNTB
DIR	OR	CNTB
DIR	JMP	CNTC
DIR	ADD	CNTC
IND	AND	CNTB
IND	OR	CNTD
IND	JMP	CNTD
IND	ADD	CNTC

In the standard approach to synthesis, each word (mnemonic string) in the table would be encoded by a string of binary symbols (i.e., 1's and 0's). Then, the encoded table would be minimized by a logic minimizer. In symbolic logic design, the table is minimized first (i.e., a table consisting of a minimal number of rows is computed) and then the table is encoded.

A first approach to symbolic minimization can be achieved by grouping the set of inputs that correspond to each output symbol. This process of reducing the size of the table is called here **disjoint minimization** because the table is considered as a set of independent subtables corresponding to each output symbol. Remarkably, disjoint minimization can be achieved by using techniques of multiple-valued logic minimization [2], [5], as shown in Section III.

Example 2: From the table of Example 1, we can see that the addressing mode INDEX and any operation codes AND OR ADD JMP correspond to the control CNTA. Similarly either one of the following conditions

addressing mode DIR and operation codes AND or OR
addressing mode IND and operation code AND

correspond to control CNTB. The entire table can be expressed as a set of conditions

INDEX	AND OR ADD JMP	CNTA
DIR	AND OR	CNTB
IND	AND	CNTB
IND	OR JMP	CNTD
DIR IND	ADD	CNTC
DIR	JMP	CNTC

Note that this table is more compact than the previous one, because it requires only six rows instead of twelve.

The problem now is to find a Boolean representation of the symbols, corresponding to a Boolean cover representation of the function, with as many rows as the compacted table. While the encoding problem will be presented in detail in Section IV, we show here by an example the consequence of the choice of a particular encoding.

Example 3: Consider this particular Boolean encoding of the words

INDEX = 00	AND = 00	CNTA = 11
DIR = 01	OR = 01	CNTB = 01
IND = 11	ADD = 10	CNTC = 10
	JMP = 11	CNTD = 00

Then the function can be represented by a Boolean cover as

00	**	11
01	0*	01
11	00	01
11	*1	00
*1	10	10
01	11	10

where a *don't care* condition on a binary input variable is represented by *. Note that the fourth Boolean implicant can be deleted because its output part is 00. Moreover, note that by deleting this implicant this cover is minimum, i.e., there exist no Boolean covers corresponding to this encoding with fewer than five implicants. (This can be proven experimentally by running an exact minimization algorithm [14], like that implemented by program ESPRESSO-II with the "exact" flag [18].)

We question now to what extent symbolic design guarantees the minimality[1] of the Boolean cover, obtained by replacing the words by their corresponding binary encoding.

Example 4: Consider this other Boolean encoding of the words, in which we changed only the encoding of the output symbols

INDEX = 00	AND = 11	CNTA = 00
DIR = 01	OR = 01	CNTB = 01
IND = 11	ADD = 10	CNTC = 10
	JMP = 11	CNTD = 11

Then the function can be represented by a Boolean cover as

00	**	00
01	0*	01
11	00	01
11	*1	11
*1	10	10
01	11	10

[1]The optimality of a Boolean cover is measured by its cardinality, i.e., by the number of its implicants. A Boolean cover of a function is **minimum** if there exists no cover of that function having a smaller cardinality. A Boolean cover of a function is **minimal** if its cardinality is minimum with regard to some local criterion. Usually, a Boolean cover of a function is said to be minimal if no proper subset is a cover of the same function [2].

Note that the first Boolean implicant can be deleted because its output part is 00. However, note that this cover is not minimum: there exist now a minimum Boolean cover corresponding to this encoding with three implicants and that can be computed from the above one by a standard minimization technique, namely

*1	0*	01
*1	1*	10
11	*1	11

Note that the first cover has two pairs of Boolean implicants with 01 and 10 in the third field and that are merged into two single implicants in the minimum cover. This is possible because the third implicant of the minimum cover has 11 in the third field, and 11 covers 01 and 10.

The reason for this additional reduction is in the covering relations among the encoded output symbols. Note that when we optimized the symbolic table by disjoint minimization, our goal was only to group the input symbols corresponding to each output symbol independently. The relations among the output symbols were neglected. For this reason, it is important to exploit the relations among the output symbols at the symbolic level. **Symbolic minimization**, formally defined in Section III, is a technique that determines an optimal ordering of the output symbols. This ordering is related to the covering relations among the binary encodings of the output symbols, and is responsible for the additional reduction of the table size, as shown by Example 4.

Example 5: Consider the following table:

INDEX	AND OR ADD JMP	CNTA
DIR IND	AND OR	CNTB
DIR IND	ADD JMP	CNTC
IND	JMP OR	CNTD

Here, an ordering relation is assumed that allows control CNTD to override control CNTB and CNTC when both are specified. The table, together with this ordering relation, is an equivalent representation of the function specified by Example 1. It is an example of the result of a symbolic minimization. Note that this table can be transformed into the Boolean cover of Example 4 by replacing each symbol by its encoding. Moreover, the encoding of CNTD covers bit-wise the encoding of CNTB and CNTC and allows CNTD to override CNTB and CNTC. Note that the first implicant can be deleted by assuming that CNTA is the default output and that the encoding of CNTA is 00, which is covered bit-wise by the encoding of all other outputs.

Once a minimal table has been found, the encoding of the words into Boolean variables is driven by the grouping of the input symbols (group of symbols appear on the same row of a minimal table) and the ordering of the output symbols generated by symbolic minimization. Note that disjoint minimization deals with each output symbol independently, and therefore does not provide information for an encoding of the output symbols that optimize the table size. Therefore, disjoint minimization can be used only to solve problem P1 of symbolic design.

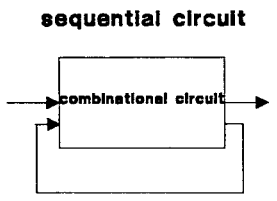


Fig. 2.

We describe in detail in Section IV how to compute an encoding of the input and output symbols that is compatible with a minimal table. Such an encoding allows us to transform the minimal symbolic cover into a Boolean representation with as many product terms as the minimal symbolic cover. Even though this mapping is not sufficient to imply the minimality of the Boolean cover, the encoded cover can be considered a good solution to the problem. It is important to remark that the length of the encoding (i.e., the number of binary variables) needed to encode each symbol may have to be larger than the minimum length required to distinguish all the symbols in each field (i.e., the ceiling of the logarithm in base 2 of the number of elements in each field). Therefore, it is interesting to compute minimal length encodings compatible with a minimal symbolic representation and to tradeoff possibly the minimality of the number of rows in a table for the number of bits required to encode each field.

Let us now consider the design problem P4 of Section I. Any sequential circuit can be implemented by feeding back the (some of the) outputs of a combinational circuit to its inputs, possibly through a register (Fig. 2). A general model of a sequential circuit is the *finite-state machine* model; the machine is synchronous if the feedback path contains synchronous registers. Finite-state machines are generally represented by **state tables**. State tables consist of four fields related to the primary inputs/outputs and to the present/next state representation. While, in general, all these fields may be represented by symbols, it is customary to represent the primary inputs and outputs in terms of binary variables and the internal states in terms of mnemonic symbols. A classical problem is to find an optimal encoding of the state symbols that correspond to an optimal implementation. This problem is referred to as optimal state assignment [9], [10].

Optimal state assignment can be solved by symbolic design by minimizing the state table using symbolic minimization and by computing a state encoding compatible with the minimal table [5]. The feedback path makes this problem different from designing combinational units. In particular, the state symbols appear in both an input and an output column of the state table and must be encoded consistently: the set of state symbols must be encoded while satisfying the group and the ordering constraints simultaneously. The limitations and the encoding procedure is described in Section IV.

Eventually symbolic design can be applied to interconnected logic circuits. Consider two units, to be implemented by two-level logic macros, that communicate

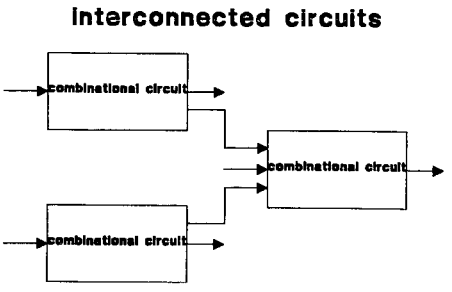


Fig. 3.

through a bus (Fig. 3). If the representation of the information is transmitted across the bus is irrelevant to the design, symbolic optimization can be used as follows. The transmitting unit can be represented by a table with a symbolic output field and the receiving unit by another table with a symbolic input field. The tables corresponding to both units are optimized by symbolic minimization and the set of symbols, representing the communication signals, can be encoded as in the previous cases. Needless to say, this method can be extended to the interconnection among any number of modules, implementing combinational or sequential functions.

III. Symbolic Minimization

A. Definitions

Symbolic functions are switching functions whose variables can take a finite set of values. Each value is represented by a **word** (or mnemonic), i.e., by a string of characters. A symbolic variable s has a set of admissible values S. The symbol ϕ is reserved to denote that variable s does not take any value of S. For functions of n input variables and m output variables, let S_i^I, $i = 1, 2, \cdots, n$, and S_i^O, $i = 1, 2, \cdots, m$, be the set of admissible values for the corresponding variables s_i^I and s_i^O. Then the domain of the symbolic function is the Cartesian product $S^I \equiv S_1^I \times S_2^I \times \cdots \times S_n^I$ and the range is the Cartesian product $S^O \equiv S_1^O \times S_2^O \times \cdots \times S_m^O$. A generic element of the domain is denoted by s^I and one of the range by s^O.

A **completely specified symbolic function** of n input variables and m output variables is a function $f: S^I \rightarrow S^O$ that maps each element of the domain to an element of the range. An **incompletely specified symbolic function** is a function having the property that, for some inputs, some output variables can take any value in the corresponding range. The collection of these points of the domain is called the *don't care* set of that particular output variable.

Example 6: The truth table of Example 1 is a representation of a completely specified symbolic function with $n = 2$ inputs and $m = 1$ outputs. Here,

$S_1^I = \{\text{DIR, IND, INDEX}\}$; $S_2^I = \{\text{AND, OR, ADD, JMP}\}$;

$S^O = \{\text{CNTA, CNTB, CNTC, CNTD}\}$.

Boolean or **binary-valued** functions are symbolic functions whose variables can take the values $S = \{0, 1\}$.

The domain is $\{0, 1\}^n$. The range of a completely specified function is $\{0, 1\}^m$. For incompletely specified functions, let the symbol * represent the *don't care* condition. Then the range is $\{0, 1, *\}^m$. Similarly, the variables of **multiple-valued** functions can take the values $S = \{0, 1, \cdots, p-1\}$, where p is the radix of the representation [17], [12]. Algebras have been developed for both the Boolean and the multiple-valued [15] representations. The representation of the result of Boolean operations are based on a **linear order** of S. Let r: $S \rightarrow N$ be an enumeration consistent with the linear order [16], where N is the set of natural numbers. Then

i) Product (AND): $s \wedge s' \equiv r^{-1} \min (r(s), r(s'))$

ii) Sum (OR): $s \vee s' \equiv r^{-1} \max (r(s), r(s'))$

iii) Complement (NOT): $\bar{s} \equiv r^{-1}(p - 1 - r(s))$.

Note that the order does not affect the semantics of the representation of a switching function; however, the order may strongly affect the size of the representation. For example, canonical representations, such as *sum of products* or *product of sums* depend on the linear order of S; in particular, the minimal representations of a Boolean function and of its complement as *sum of products* have different sizes, and algorithms have been developed to exploit this fact [19].

Representations of symbolic functions depend on the definitions of the operations among words. Unfortunately, no order relation is meaningful *a priori* among the elements of a symbolic description. For this reason, operations on symbolic representations are related to order relations among words, and appropriate order relations are introduced to obtain convenient representations of symbolic functions. In the following presentation, single-output functions are considered first, i.e., $m = 1$, to simplify the notations. The extension to multiple-output functions is then shown.

Symbolic functions are represented here in a particular canonical notation: *sum of products* or more exactly *sum of products of symbolic literal functions*. Let S be the set of admissible values for a variable s. A **symbolic literal** is a nonempty subset $\sigma \subseteq S$. For any variable $s \in S$, the **symbolic literal function** is defined as follows:

$$l(s, \sigma) \equiv \begin{cases} \textbf{TRUE} & \text{if } s \in \sigma \\ \textbf{FALSE} & \text{else.} \end{cases}$$

Example 7: Consider the set $S_1^I = \{\text{DIR, IND, INDEX}\}$ corresponding to the set of admissible values of the first input variable in Example 6. An example of a symbolic literal is DIR IND. The corresponding literal function is TRUE for either s = DIR or s = IND.

By using a *sum of product of symbolic literal functions* representation, only the order in S^O affects the representation because the literal function maps words into the pair of values (TRUE, FALSE) independently of the order in S^I. Note that if a linear order relation is applied on the range, symbolic function representations in this canonical form are equivalent to multiple valued logic function representations [17] in the same form. In particular, a multiple-valued representation can be obtained from a symbolic representation by interchanging each symbol s with $r(s)$, where $r(\cdot)$ is the appropriate enumeration.

Since an order in S^O is not necessarily given, the definitions of the representation of a symbolic function are compatible with a set, possibly empty, of **partial order relations** among the elements of the range. Let $R = \{(s, s'); s, s' \in S^O\}$ be a partial order on S^O. We say that s **covers** s' if either $s = s'$ or $s = \phi$ or $(s, s') \in TR$, where TR is the transitive closure of R [1]. The **symbolic sum** of two words s and s' is well defined only if a covering relation exists among the elements involved. In particular

$$s \vee s' = \begin{cases} s & \text{if } s \text{ covers } s' \\ s' & \text{if } s' \text{ covers } s \end{cases}$$

else the symbolic sum is ambiguous or ill-defined.

A **symbolic product-term** (or symbolic product) of literals is the $n + 1$-tuple $(\sigma_1, \cdots, \sigma_n, \tau)$, where $\sigma_i \subseteq S_i^I$, $i = 1, 2, \cdots, n$; $\tau \in S^O$. The word τ is called the **output-part** of the literal. A **symbolic product** $p(s^I, \tau)$ **of literal functions** $l_i(s_i^I, \sigma_i)$, $i = 1, 2, \cdots, n$, is a function

$$p(s^I, \tau) = \begin{cases} \tau & \text{if } l_i(s_i^I, \sigma_i) = \textbf{TRUE}, \\ & \quad \forall i = 1, 2, \cdots, n \\ \phi & \text{else.} \end{cases}$$

Example 8: A symbolic product of the function in Example 1 is

DIR AND CNTB.

The symbolic product function takes the value CNTB when the two inputs take the values DIR and AND, respectively.

Two products p_1, p_2 **intersect** ($p_1 \cap p_2 \neq \phi$) if $\exists s^I \in S^I$ such that $p_1(s^I, \tau_1) \neq \phi$ and $p_2(s^I, \tau_2) \neq \phi$. Two sets of products P_1 and P_2 intersect ($P_1 \cap P_2 \neq \phi$) if $\exists p_1 \in P_1$ and $\exists p_2 \in P_2$ such that p_1 and p_2 intersect. Two products are **output-disjoint** if either they do not intersect or they have the same output-part, i.e., $p_1(s^I, \tau_1)$ intersects $p_2(s^I, \tau_2)$ implies $\tau_1 = \tau_2$. A set of products is output-disjoint if the products are pair-wise output-disjoint.

Example 9: Consider the two symbolic products

DIR IND	ADD	CNTC
DIR	ADD JMP	CNTC

The two symbolic product intersect because, for the symbolic input s^I = DIR ADD, the corresponding symbolic product functions specify a symbolic value. Since both symbolic product functions take value CNTC, the product terms are output-disjoint.

A symbolic function can be represented in a *sum of product* form; if $\forall s^I \in S^I$ for which the function is specified, the operation of symbolic sum among products is well defined. In particular, such a representation always exists in the following two cases: i) for any linear order on S^O, ii) if the representation is a sum of pair-wise out-

put-disjoint products. In the former case, symbolic sum is always well defined because a covering relation is defined between each pair of symbols in S^O. In the latter, only the symbolic sum of identical values is required.

Sum of product representations are conveniently represented in tabular forms, as a stack of product terms. A **symbolic implicant** is a symbolic product $p(s^I, \tau)$ such that $\forall s^I \in S^I$ for which the symbolic function is specified, $f(s^I)$ covers $p(s^I, \tau)$. A **symbolic cover** of a symbolic function is a set of implicants $P = \{p_1, p_2, \cdots, p_{|P|}\}$ whose sum is $f(s^I)$, $\forall s^I \in S^I$ for which the symbolic function is specified. Since symbolic sum depends on the order R on S^O, we denote a symbolic cover by the pair $C(P, R)$. The **cardinality** of a symbolic cover is $|P|$ and depends on R. A **minimum** symbolic cover of a symbolic function is a cover of minimum cardinality. A **minimal** (local minimum) symbolic cover of a symbolic function is a cover such that no proper subset is a cover of the same function.

Example 10: The following table is a symbolic cover of the function specified in Example 6:

INDEX	AND OR ADD JMP	CNTA
DIR	AND OR	CNTB
IND	AND	CNTB
IND	OR JMP	CNTD
DIR IND	ADD	CNTC
DIR	JMP	CNTC

Note that the product terms are pair-wise output-disjoint. Therefore, this representation is output-disjoint and is compatible with any set R of partial order relations on S^O, and in particular the empty set. The following table is another symbolic cover of the function specified in Example 6:

INDEX	AND OR ADD JMP	CNTA
DIR IND	AND OR	CNTB
DIR IND	ADD JMP	CNTC
IND	JMP OR	CNTD

Here, $R = \{(\text{CNTD}, \text{CNTB}); (\text{CNTD}, \text{CNTC})\}$. Note that the fourth product term has an intersection with the second and third one and these products are not output-disjoint. By choosing this particular partial order, the cover cardinality is reduced by two. Moreover, note that the first implicant can be removed by assuming that CNTA is the default output, as pointed out in Example 5.

B. Symbolic Minimization

Symbolic minimization is a procedure that attempts to determine a symbolic cover of a symbolic function in a minimum number of product terms. Finding a minimum symbolic cover is a difficult task. An analysis of the computational complexity of the problem has not been done yet. However, we conjecture that any method to find a minimum cover should involve the solution of a covering problem, which is a NP-complete problem [8]. Therefore, heuristic algorithms are used to determine a minimal (local minimum) solution. It is important to remark that recent progress in heuristic logic minimization has led to techniques which very often yield minimum solutions in the binary [11], [2] and multiple-valued [18] case. We assume that the reader is familiar with heuristic logic minimization [11], [2]. Since most of the routines in the symbolic minimization algorithm are based on logic minimizer ESPRESSO-II, we refer the reader to [2] for details.

Symbolic minimization is achieved by an iterative improvement of the initial cover $C^0(P^0, R^0)$. The symbolic function is described as input by the set of products P^0, while R^0 is an empty set of ordering relations because no order relation is meaningful *a priori* among the elements of a symbolic description. Therefore, P^0 is always a set of output-disjoint products. The main idea of symbolic minimization is to generate the order R during the minimization process. The symbolic minimizer detects partial order relations that are necessary to define sums of product terms which would decrease the symbolic cover cardinality. As a result, the order relations are determined *a posteriori* by the minimizer. The output of the minimizer is a minimal cover $C(P, R)$.

Symbolic minimization is a very complex technique. To help the reader in understanding this method and the underlying principle, we first consider symbolic functions with one symbolic output only and we present a simplified version of the main loop of the symbolic minimization algorithm. Then the complete algorithm is described.

1) A Simplified Symbolic Minimization Loop: Symbolic minimization can be achieved by an iterative loop that uses a multiple-valued-input, binary-valued-output minimization procedure. This procedure can be regarded as a black box that takes as input the representation of a multiple-valued-input function and its *don't care* set and returns a minimal representation. Computer programs ESPRESSO-II [2], ESPRESSO-MV [18], and MINI [11] can be used in this regard.

It is convenient to represent the partial order R by a directed acyclic graph $G(V, A)$, where the vertex set V is in one-to-one correspondence with S^O. The edge set A is initialized empty and is constructed during the minimization process. An edge between two vertices defines a order relation between the corresponding elements of S^O. Therefore the sum of two distinct elements of S^O is well defined if there is a directed path between the corresponding vertices.

Let us arbitrarily label the elements of the range: $S^O = \{s_i^O, i = 1, 2, \cdots, q\}$. Let ON_i, $i = 1, 2, \cdots, q$, be the subset of the initial set of product terms P^0 consisting of the product terms whose output part $\tau = s_i^O$. Note that the set ON_i does not intersect the set ON_j if $i \neq j$ because P^0 is output-disjoint. Each set ON_i specifies a symbolic single-output single-valued function. The original symbolic function can be seen as a collection of q multiple-valued-input, binary-valued-output functions whose *on* set corresponds to the points of the domain mapped into s_i^O, whose *off* set corresponds to those points mapped into s_j^O, $j \neq i$, and whose *don't care* set corresponds to the unspecified points [2]. A representation of each set ON_i

by a minimal number of product terms, denoted here by M_i, can be obtained $\forall i = 1, 2, \cdots, q$ by using a multiple-valued-input, binary-valued-output minimization technique. In principle, by performing q minimization in this way, a minimal cover of the original function can be computed as $P = \cup_{i=1}^{q} M_i$, i.e., as a collection of the q minimal covers M_i. It is shown in [2] how to perform the q minimizations simultaneously. This procedure is called here **disjoint minimization** because the minimal cover P of any completely specified function is output-disjoint.[2] Disjoint minimization does not exploit the benefit of choosing an order R to minimize the cover, and therefore is a weak optimization technique.[3]

The main idea of symbolic minimization is that the cover P is not constrained to be output-disjoint by introducing appropriate order relations among the elements of S^O. For example, suppose that $(s_j^O, s_i^O) \in R$. Then any point of the domain represented by ON_j can be used to reduce the cardinality of ON_i in the minimization process. In the minimal symbolic representation, such point is still mapped into s_j^O because $(s_j^O, s_i^O) \in R$. In other words, the subset of the domain represented by ON_j is a part of the *don't care* set while minimizing ON_i. We represent the *don't care* set by the set of product terms DC_i. In this case, $ON_j \subseteq DC_i$.

Example 11: Consider the first cover of Example 10. Let s_i^O = CNTB and s_j^O = CNTD. Then, ON_i is

DIR AND OR CNTB
IND AND CNTB

Suppose (CNTD, CNTB) $\in R$. Then DC_i includes

IND OR JMP CNTD.

Therefore, the point of the domain s^I = IND OR can be used to reduce the cardinality of ON_i that can be represented as

DIR IND AND OR CNTB.

To minimize ON_i, an explicit representation of the corresponding *don't care* set is needed. Equivalently, the *off* set can be specified and the *don't care* set obtained by complementation of the *on* and *off* sets. To take advantage of the order relations, we use a definition of the *off* set different from that used in [2] and mentioned before. For our purposes, the *off* set corresponding to ON_i is the subset of s^I that is mapped by the function f to a value different than s_i^O and covered by s_i^O, because $\forall s^I \in s^I$ s.t. $f(s^I) \neq s_i^O$ and $f(s^I)$ is covered by s_i^O, $p(s^I, s_i^O)$ is not an implicant of the function. If $G(V, A)$ represents the partial order, then the *off* set can be defined as a set of product terms: $OFF_i = \cup_J ON_j$; $J = \{ j$ s.t.$\exists$ a path from v_i to v_j in $G(V, A)\}$.

[2]If the original function is incompletely specified, the minimal cover P is still output-disjoint if we restrict the definition of intersection among implicants by considering S as the *care* set of the function.

[3]In a previous paper [5], the author used the concept of symbolic minimization to deal with the optimal state assignment problem. Since no symbolic minimization technique was available at that time, disjoint minimization was used instead.

At each iteration of the symbolic minimization loop, M_i is obtained by minimizing ON_i, using a routine that performs multiple-valued-input binary-valued-output minimization. We invoke the minimization routine with the pair (ON_i, OFF_i) so that the corresponding *don't care* set DC_i, computed by the minimizer by complementation, includes by construction of the sets ON_j for which no path exists in $G(V, A)$ from v_i to v_j. As a result, minimization may be very efficient in reducing the cardinality of ON_i because of the particularly advantageous *don't care* set. If M_i intersects ON_j, the relation (s_j^O, s_i^O) is recorded by adding (v_j, v_i) to the edge set of the graph.

SYMBOLIC MINIMIZATION LOOP
Data ON_i, $i = 1, 2, \cdots, q$;
Data $G(V, A)$;
$A = \phi$; $P = \phi$;
for $(k = 1$ to $q)\{$
    $i =$ **select** (k);
    $OFF_i = \cup_J ON_j$; $J = \{ j|\exists$ a path from v_i to v_j in $G(V, A)\}$;
    $M_i =$ **minimize** (ON_i, OFF_i);
    $A = A \cup \{(v_j, v_i)$s.t.$M_i \cap ON_j \neq \phi\}$;
    $P = P \cup M_i$;
$\}$;

Procedure **select** sorts the sets ON_i according to a heuristic criterion. Procedure **minimize** is a call to a multiple-valued-input binary-valued-output minimizer. The algorithm generates a set of symbolic products $P = \cup_{i=1}^{q} M_i$ and the directed graph $G(V, A)$.

Theorem 1: The graph $G(V, A)$ generated by the symbolic minimization loop is acyclic.

Proof: By construction. Initially, the graph is acyclic because the edge set A is empty. Suppose that at iteration k the graph has no cycles. Then at iteration $k + 1$ the graph has no cycles according to the following argument. Let i be the index returned by **select** at iteration $k + 1$. Let $\bar{J} = \{ j$ s.t.$M_i \cap ON_j \neq \phi\}$. Since the edges (v_j, v_i); $j \in \bar{J}$ are those and only those added to the edge set at iteration $k + 1$, then any cycle must include a vertex v_j, $j \in \bar{J}$ and a directed path must exist between v_i and that vertex. Let $J = \{ j|\exists$ a path from v_i to v_j in $G(V, A)\}$. Then $\bar{J} \cap J = \phi$, because $M_i \cap OFF_i = \phi$, i.e., the minimal cover of the *on* set cannot intersect the *off* set. Then no cycle can be introduced at step $k + 1$. Therefore, the final graph $G(V, A)$ is acyclic. ■

Since $G(V, A)$ is acyclic, R represents a partial order relation on S^O. It is now important to show that $C(P, R)$ is a cover of the symbolic function specified by the initial cover $C^0(P^0, R^0)$. We assume the correctness of procedure **minimize** in returning the minimal covers M_i of the covers ON_i, $i = 1, 2, \cdots, q$.

Theorem 2: $C(P, R)$ is a minimal cover of the original symbolic function represented by $C^0(P^0, R^0)$ and $|P| \leq |P^0|$.

Proof: Consider each symbolic output value s_i^O, $i = 1, 2, \cdots, q$. Since the elements of the symbolic function domain mapped by f into s_i^O are represented by ON_i in P^0

and M_i is a minimal representation of ON_i, then for each element s^I in the domain mapped by f into s_i^O and $\forall i$, $i = 1, 2, \cdots, q$, there exists at least one symbolic implicant $p \in P$ whose output part is s_i^O and s.t. $p(s^I, s_i^O) = f(s^I)$. For a generic element s^I of the domain, let $P(s^I) = \{p \in P \text{ s.t. } p(s^I, \tau) \neq \phi\}$. Since $C(P, R)$ is not necessarily output-disjoint, the output parts of the product terms in $P(s^I)$ may be conflicting. However, since $M_i \cap ON_j \neq \phi$ implies $(v_j, v_i) \in A$ $\forall i, j$, then the sum of the product terms in $P(s^I)$ in $f(s^I)$. Moreover, $C(P, R)$ is minimal because the covers M_i, $i = 1, 2, \cdots, q$ are minimal and $P = \cup_{i=1}^{q} M_i$. Eventually, since $|M_i| \leq |ON_i|$, $i = 1, 2, \cdots, q$, then $|P| = \Sigma_{i=1}^{q} |M_i| \leq \Sigma_{i=1}^{q} |ON_i| = |P^0|$. ■

The order R depends on the heuristic sorting of the sets ON_i, done by procedure **select**. As a result, the cardinality of the cover $C(P, R)$ generated by the algorithm strongly depends on this routine. Several heuristics have been tried. Note that as more edges are added to the graph, it is more likely that the *off* sets become large and the *don't care* sets are small. An effective heuristic is to sort the sets ON_i in descending order of cardinality, so that the largest sets will benefit from large *don't care* sets. A key ingredient for an effective reduction of the cover minimality is that the graph should be kept as sparse as possible by introducing only the ordering relations needed to reduce the cover cardinality. Keeping the graph sparse corresponds to keep many degrees of freedom to order S^O in the later iterations of the algorithm. Note that if **minimize** is a "standard" minimization algorithm, it aims at reducing both the product of literal cardinality in each ON_i. Therefore, the local optimization of the number of literals in the **minimize** procedure may introduce new edges in the graph and reduce the likelihood of reducing the symbolic cover cardinality at a later step. For these reasons, the symbolic minimization loop has to be modified for efficiency by tuning the **minimize** routine to the symbolic minimization problem.

2) The Symbolic Minimization Algorithm: We consider here symbolic functions with multiple outputs (i.e., $m \geq 1$): one output variable is a symbolic q-valued variable; the other output variables can take values in the ordered set of symbols $\{0, 1\}$, i.e., are binary-valued variables. In this case, the symbolic implicant (product term) output part τ has m scalar components τ_l, $l = 1, 2, \cdots, m$; τ_1 can take any of the q values in $S_1^O = \{s_{1,1}^O, s_{1,2}^O, \cdots, s_{1,q}^O\}$; τ_l, $l = 2, 3, \cdots, m$ can be either 0 or 1; in addition $\tau_1 = \phi$ if the implicant does not carry any information regarding the q-valued symbolic output. This special case is considered because it applies to finite-state machines whose next-state function is specified by symbols and whose primary outputs are binary-valued functions. The general case of m symbolic valued outputs will be mentioned later.

To obtain an efficient algorithm for symbolic minimization, it is necessary to "open the black box" and examine more carefully the operations that the procedure **minimize** performs. We call **optimize** the modified procedure for multiple-valued-input minimization used in the algorithm. The symbolic minimization algorithm is as follows:

```
SYMBOLIC MINIMIZATION
Data C(P^0, R^0);
Data G(V, A);
A = φ; P = φ;
for (k = 1 to q)
      ON_k^0 = slice (P^0, s_{1,k}^O);
OFF_b = get_off_set (P^0);
P^1 = disjoint_minimize (P^0);
for (k = 1 to q)
      ON_k^1 = slice (P^1, s_{1,k}^O);
P^φ = slice (P^1, φ);
for (k = 1 to q){
      i = select (k);
      OFF_i = OFF_b ∪ ∪_J ON_j^0; J = {j | ∃ a path from v_i
                                       to v_j in G(V, A)};
      M_i = optimize (ON_i^1, OFF_i);
      P = P ∪ M_i;
};
P = merge (P, P^φ);
```

The first improvement over the simplified symbolic minimization loop can be explained in terms of preprocessing of the input data. As mentioned before, the ordering relations R need only be introduced when they reduce the cardinality of P. Therefore, it may be useful to perform a disjoint minimization before entering the symbolic minimization loop. Disjoint minimization is done by procedure **disjoint_minimize**, which invokes a multiple-valued-input minimizer, such as ESPRESSO-II. The minimized cover P^1 is a minimal representation of the m scalar components of the function. Now we call ON_i^1, $i = 1, 2, \cdots, q$, the sets of product terms in P^1 with $\tau_1 = s_{1,i}^O$. The procedure that returns the sets ON_i^1 from P^1 is called **slice**. The sets ON_i^1 contain all the points of the domain mapped into $s_{1,i}^O$ and possibly some points of the *don't care* set. Note that there may be a nonempty subset of products in P^1 with $\tau_1 = \phi$; these product terms do not carry any information related to the first scalar component and do not affect the symbolic minimization loop: they are stored in set P^ϕ. Similarly, while the *off* sets related to each symbolic value of the first scalar component are defined in the loop, the *off* sets related to the other components do not change. We call OFF_b the union of the *off* sets corresponding to the binary-valued components of the function, i.e., components $l = 2, 3, \cdots, m$. Procedure **get_off_set** extracts the *off* set OFF_b from the original representation P^0. In the main loop, OFF_i is the union of two subsets: OFF_b and $\cup_J ON_j^0$, which represents the *off* set of the value of the first component being considered. Note that the sets ON_j^0 are a "slice" of the original representation $C^0(P^0, R^0)$, so that they correspond to the points of the domain, and only those points, that have to be mapped by f into $s_{1,i}^O$, $i = 1, 2, \cdots, q$.

In the symbolic minimization loop, procedure **select** sorts the sets ON_i^1 in descending order of cardinality. Then the sets M_i are obtained from ON_i^1 by procedure **optimize**.

Procedure **optimize** sets $M_i = ON_i^1$ and then performs operations on M_i to reduce its cardinality. Procedure **optimize** differs from procedure **minimize** (described in the previous subsection) because the intersections between M_i and ON_j^0, $j \neq i$, are monitored during the minimization process. In particular, if at any given point $M_i \cap ON_j^0 = \phi$ and a point of the domain corresponding to ON_j^0 is useful as a *don't care* point to reduce the number of implicants of M_i, this point is used and the order relation (j, i) is recorded by adding (v_j, v_i) to the edge set of the graph. If $M_i \cap ON_j^0 \neq \phi$, then the points of ON_j^0 can be used unconditionally as *don't care* points. By doing this, we introduce a new order relation only when the cardinality of M_i is reduced by deleting one (or more) implicants. Note that by monitoring the intersections $M_i \cap ON_j^0$, we avoid the set intersections after the minimization step, as done in the simplified loop. Procedure **optimize** is implemented by a modified version of program ESPRESSO-II.[4] At the exit of the main loop, procedure **merge** appends to P the implicants of P^ϕ that are not covered by (or that cannot be merged with) implicants of P.

The primary goal of symbolic minimization is to reduce the cardinality of the cover. An interesting question is to explore the role of the symbolic literals and how the secondary goal of symbolic minimization can be related to the literals. A clue can be obtained by relating symbolic minimization to the binary encoding problems sketched in Section II. The minimal symbolic cover is an intermediate step in the process of computing a minimal Boolean cover. Therefore, the symbolic literals in a symbolic cover should be chosen to ease the encoding of the minimal symbolic cover into a Boolean cover as much as possible. Note first that a literal consisting of all admissible values for the corresponding variable is equivalent to a *don't care* condition on that variable. Such a literal is called a **full** literal. Therefore, it is always convenient to attempt to expand each literal to full. This is the equivalent to attempting to minimize input literals in standard Boolean minimization. When it is not possible to expand a literal to full, it is questionable which is the optimal cardinality for each literal, i.e., the optimal number of words in each literal. The strategy used in symbolic minimization is the following. For each implicant of the minimal cover, we compute an **expanded implicant**, whose literals have maximal cardinality. The expansion process increases the literal cardinality of a product term, while retaining it as an implicant of the function. Then we compute a **reduced** implicant whose literals have minimal cardinality. The reduction process decreases the literal cardinality of a product term, while making sure that the reduced product term and the remaining ones are still a cover of the function [2]. By comparing each expanded implicant with the corresponding reduced implicant, we can detect the *don't care* words in each literal. Such words represent input conditions that do not affect the value of the minimal cover and that can be used effectively to ease the encoding problem, as shown in Section IV. Procedure **optimize** implements the literal optimization strategy.

We show now that $C(P, R)$ is a cover of the symbolic function specified by $C^0(P^0, R^0)$. We assume the correctness of procedures **disjoint-minimize** and **optimize** in returning the corresponding minimal covers. We note first that the graph $G(V, A)$ constructed by the algorithm is acyclic because the structure of the main loop and the steps that construct the graph are the same as in the symbolic minimization loop of the previous subsection and, therefore, Theorem 1 applies.

Theorem 3: $C(P, R)$ is a minimal cover of the original symbolic function represented by $C^0(P^0, R^0)$ and $|P| \leq |P^1| \leq |P^0|$.

Proof: Let $C_1(P, R)$ be the cover of the first component of the function generated by the algorithm. Let $C_1^0(P^0, R^0)$ be the original cover of the first component. As far as the first component is concerned, the symbolic minimization algorithm is equivalent to applying the simplified loop on the sets ON_i^1, $i = 1, 2, \cdots, q$. Since all the points of the domain mapped into $s_{1,i}^O$ are represented by ON_i^1, $i = 1, 2, \cdots, q$, by Theorem 2, $C_1(P, R)$ is a minimal cover of the first component of the function specified by $C_1(P^0, R^0)$. Let now P_l, $l = 2, 3, \cdots, m$, be the cover of component l computed by the algorithm, P_l^1, $l = 2, 3, \cdots, m$, be the cover of component l after disjoint minimization, and P_l^0, $l = 2, 3, \cdots, m$, be the original cover of component l. Let $ONE_l = \{s^l \in S^l \text{s.t. } f_l(s^l) = 1\}$ and $ZERO_l = \{s^l \in S^l \text{s.t. } f_l(s^l) = 0\}$, $l = 2, 3, \cdots, m$. Then, for each component $l = 2, 3, \cdots, m$, P_l^1 represents all the points in ONE_l and none of the points in $ZERO_l$ by the assumption of correctness of procedure **disjoint_minimize**. Similarly, P_l represents at least all the points represented by P_l^1 and none of the points in $ZERO_l$ because P is constructed as a union of P^ϕ and the sets M_i, and no product term in M_i, $i = 1, 2, \cdots, q$, with $\tau_l = 1$ represents any point in $ZERO_l$ because M_i is obtained by **optimize** (ON_i^1, OFF_i) and $OFF_b \subseteq OFF_i$ represents the *off* set of components $l = 2, 3, \cdots, m$. Therefore, $C(P, R)$ is a cover of the original symbolic function represented by $C^0(P^0, R^0)$. The cover $C(P, R)$ is minimal because the covers M_i, $i = 1, 2, \cdots, q$, are minimal and only the products P^ϕ that are not covered by $\cup_{i=1}^q M_i$ are appended to it at the exit of the main loop. Moreover, since $|M_i| \leq |ON_i^1|$, $i = 1, 2, \cdots, q$, and $|ON_i^1| \leq |ON_i^0|$, $i = 1, 2, \cdots, q$, then $|P| = |P^\phi| + \Sigma_{i=1}^q |M_i| \leq |P^\phi| + \Sigma_{i=1}^q |ON_i^1| = |P^1| \leq \Sigma_{i=1}^q |ON_i^0| = |P^0|$. ■

The symbolic minimization algorithm invokes q times procedure **optimize**, whose computational complexity is similar to that of minimizer ESPRESSO-II. Therfore, the total computational complexity grows linearly with the number of elements in S_1^O. The minimization procedure is

[4]Procedure EXPAND of program ESPRESSO-II is modified as follows. By using the terminology of [2], let EXPAND1 be the routine that takes an implicant c and a cover and returns an expanded implicant c^+ and the set W of implicants covered by c^+. Just after the call to EXPAND1, an additional routine checks the intersections $c^+ \cap ON_i^O$, $i \neq j$. If $\exists j$ such that $c^+ \cap ON_i^O \neq \phi$ and $(v_j, v_i) \notin A$ and $W = \phi$, then c^+ is replaced by the original cube c.

TABLE I
RESULTS OF SYMBOLIC MINIMIZATION

Example	Original cover cardinality	Minimal cover cardinality	Minimal disjoint-cover cardinality
EX1	24	9	16
EX2	91	49	57
EX3	170	78	78
EX4	11	5	8
EX5	25	10	11
EX6	24	17	24
EX7	115	89	94
EX8	107	57	92
EX9	184	106	115
EX10	16	14	15
EX11	166	102	111
EX12	49	11	12
EX13	25	10	11
EX14	20	8	13
EX15	56	23	24
EX16	32	15	16
EX17	108	46	55
EX18	32	17	18
EX19	14	9	10
EX20	30	22	23

a heuristic procedure: no theoretical bounds on the computational complexity have been proven for heuristic minimization; however, experimental results have shown that it is practical to minimize a wide range of logic functions [11], [2] and, therefore, the symbolic minimization algorithm can be used in this perspective.

As a final remark, symbolic minimization can be extended to multiple-output functions ($m > 1$) with m symbolic outputs by extending the definitions and operations appropriately. In this case, m partial orders on the sets S_i^O, $i = 1, 2, \cdots, m$, have to be recorded. The symbolic minimization algorithm can be extended to cope with this case. However, since each symbolic implicant may imply more than one output condition, it is very complex to determine the best sequence to apply the **optimize** procedure. For this reasons, further investigation is still needed to solve the general case.

C. Implementation and Results

The symbolic minimization algorithm has been implemented in a computer program called CAPPUCCINO because it is based on the logic minimizer ESPRESSO-II. CAPPUCCINO is written in APL and incorporates a modified version of the ESPRESSO-II original program [2]. CAPPUCCINO implements the symbolic minimization algorithm described in Section III-B-2. The simplified loop was presented in Section III-B-1 to ease the understanding of symbolic minimization. It was implemented only in an early experimental stage and then superseded by the complete algorithm. CAPPUCCINO has been tested on several examples. Table I summarizes the results.

The first two numeric columns show the original and final cover cardinality. The last column shows the final cardinality obtained by disjoint-minimization, by using program ESPRESSO-II. In some cases (EX1, EX2, EX8, EX17, $\cdots$), CAPPUCCINO does significantly better than ESPRESSO-II in reducing the cover cardinality. In some others, the advantage of introducing covering relations among the output symbols is not a major factor in reducing the cover cardinality. These comparisons have to be understood with caution because we are comparing different minimization techniques and not program performances.

Computing time ranges from a few seconds to about 20 minutes for the largest example on an IBM 3081 computer. Note that APL is interpreted and, therefore, the execution is much slower with comparison to compiled code programs. Today, CAPPUCCINO is limited to covers of about 2000 symbolic product terms due to memory limitations of the APL workspace and computing time. A compiled code implementation of the algorithm based on the data structure and the routines of program ESPRESSO-MV [18] (in place of the APL version of ESPRESSO-II) would definitely increase the capability and the performance of the program.

IV. ENCODING PROBLEMS AND ALGORITHMS

A. Encoding Problems

Symbolic minimization is used as an intermediate step in solving problems P1–P4 of Section I. Since the final result must be a binary-valued logic circuit implementation, the symbolic representation has to be translated into a binary-valued (Boolean) one. If a multiple-valued circuit implementation technology were available (including logic gates implementing the literal function [17]), then the (minimal) symbolic representation could be mapped into a multiple-valued representation with the same cardinality by interchanging the words s with $r(s)$, where $r(\cdot)$ is an appropriate enumeration. If a partial-order relation exists on a set of words, then the enumeration must be consistent with it.

Let us consider first problems P1–P3. The goal of the following encoding technique is to find a binary-valued *sum of products* representation of the switching function with as many product terms as the (minimal) symbolic representation.[5] To construct such a Boolean cover, it is sufficient to determine: 1) an encoding of the words related to each symbolic input variable such that each symbolic implicant can be represented by one Boolean implicant; 2) an encoding of the words related to each symbolic output variable that preserves the covering relations; i.e., such that the encoding of the sum of any subset of symbolic products is the sum of the corresponding Boolean products. For this reason, we consider two encoding problems: the former is related to the encoding of the

[5]Unfortunately, it is not possible to state the minimality of the Boolean cover because a Boolean implicant may be covered by the sum of two or more implicants. Symbolic minimization detects pair-wise covering relations and neglects (in the version presented here) one-to-many relations that cannot be expressed by the partial order (e.g., a word may be made equivalent to the simultaneous assertion of two or more of words). An extension of symbolic minimization to cope with this situation would be of great theoretical interest, but would probably also complicate the problem of finding a Boolean encoding. In practice, the Boolean covers obtained by the present technique are often minimal or close to minimal.

symbols representing the input variables; the latter to the encoding of the output variables.

Let us consider first the encoding of the input variables. If a variable can take at most two symbolic values, it has a trivial Boolean encoding. In the general case, a variable that can take more than two symbolic values is represented by more than one binary-valued variable. Then, to achieve the goal of encoding each symbolic implicant by one Boolean implicant, we must represent each symbolic literal by one product of Boolean literals. A product of Boolean literals is called **cube** or **face** because it is a subspace of the Boolean hyperspace that can be represented by a hypercube. Therefore, the encoding of the words must be such that each symbolic literal can be represented by a face (Boolean cube) that is a subspace of the Boolean space that contains the encoding of all and only the symbols in the literal [5]. If *don't care* words are specified in that literal, then it is indifferent whether the face representing the literal contains the encoding of *don't care* words or not.

The problem of encoding the words related to the output variables is different because the output part of the symbolic implicants corresponding to an output variable consists of one word only (and not of a symbolic literal with possibly more than one word, as in the case of the input variables). However, the encoding of the words related to the output variables must be such that the covering relations are preserved while transforming the symbolic cover into a Boolean cover. Therefore, the encoding must be such that, for any two words joined by an order relation, the corresponding encoding are linked by a covering relation, i.e., the first word covers bit-wise the second word.

The encoding problem derived from problem P4 has an additional constraint. In this case, there is one set (or more sets) of words corresponding to both input and output variables. The encoding of this set of words must satisfy the requirements for the encoding of the input variables and the output variables simultaneously.

We now formally present the encoding problems. Let S be a set of words to be encoded and let $n_s = |S|$. Let n_p be the cardinality of the (minimal) symbolic cover. Let n_b, the encoding length, i.e., the number of Boolean variables used to represent S. The encoding problem is studied using matrix notation. Some matrices we consider have pseudo-Boolean entries from the set: $\{0, 1, *, \phi\}$ where * represents the *don't care* condition (i.e., either 1 or 0) and ϕ represents the empty value (i.e., neither 1 nor 0). Logical product and sum on pseudo-Boolean variables is defined as follows:

Λ	0	1	*	ϕ
0	0	ϕ	0	ϕ
1	ϕ	1	1	ϕ
*	0	1	*	ϕ
ϕ	ϕ	ϕ	ϕ	ϕ

V	0	1	*	ϕ
0	0	*	*	0
1	*	1	*	1
*	*	*	*	*
ϕ	0	1	*	ϕ

Let S be the set of values taken by a symbolic input variable. Let us consider the set of literals in the (minimal) symbolic cover related to that variable. The **word-literal incidence matrix** A (or incidence matrix in short) is a matrix: $A \in \{0, 1, *\}^{n_p \times n_s}$

$$A = \begin{bmatrix} a_{1\cdot} \\ a_{2\cdot} \\ \cdots \\ a_{n_p\cdot} \end{bmatrix} = [a_{\cdot 1} | a_{\cdot 2} | \cdots | a_{\cdot n_s}] = \{a_{ij}\}$$

where: $a_{ij} =$ 1 if word j belongs to literal i; * if word j is a *don't care* word in literal i; 0 else.

Example 12: Let S be the set of operation codes in the symbolic function specified in Examples 1 and 10, i.e., $S = \{\text{AND, OR, ADD, JMP}\}$. Consider the minimal symbolic cover of Example 10. Then

$$A = \begin{bmatrix} 1111 \\ 1100 \\ 0011 \\ 0101 \end{bmatrix}.$$

Now let S be the set of values taken by a symbolic output variable. The **partial order adjacency matrix** $B \in \{0, 1\}^{n_S \times n_S}$ (or adjacency matrix in short) is the adjacency matrix of the graph representing the transitive closure of the partial order R. If word i covers word j, then $b_{ij} = 1$. If word i covers word j and word j covers word k, then $b_{ij} = 1$, $b_{jk} = 1$ and $b_{ik} = 1$. Since covering is a transitive relation, we represent directly all the implied covering relations by matrix B. Moreover, since it is trivial that each word covers itself, we choose not to represent it by convention, i.e., $b_{ii} = 0$, $i = 1, 2, \cdots, n_s$.

Example 13: Let S be the set of controls in the symbolic function specified in Examples 1 and 10. Consider the minimal symbolic cover of Example 10. Then

$$B = \begin{bmatrix} 0000 \\ 0000 \\ 0000 \\ 0110 \end{bmatrix}.$$

The **encoding matrix** E is a matrix $E \in \{0, 1\}^{n_S \times n_b}$

$$E = \begin{bmatrix} e_{1\cdot} \\ e_{2\cdot} \\ \cdots \\ e_{n_s\cdot} \end{bmatrix} = [e_{\cdot 1} | e_{\cdot 2} | \cdots | e_{\cdot n_b}]$$

whose rows are the encoding of the words.

Definition 1: Let $a \in \{0, 1, *, \phi\}$ and $x \in \{0, 1, *, \phi\}$. The **selection** of x according to a is

$$a \cdot x = \begin{cases} x & \text{if } a = 1 \\ \phi & \text{else.} \end{cases}$$

Selection can be extended to two-dimensional arrays and is similar to matrix multiplication.

Definition 2: Let $A \in \{0, 1, *, \phi\}^{p \times q}$ and $X \in \{0, 1, *, \phi\}^{q \times r}$. The **matrix pseudo-Boolean selection** is

$$A \cdot X = C = \{c_{ij}\}^{p \times r}$$

where $c_{ij} \equiv V_{k=1}^{q} a_{ik} \cdot x_{kj}$ or equivalently $c_{ij} \equiv a_{i1} \cdot x_{1j} V a_{i2} \cdot x_{2j} V \cdots V a_{iq} \cdot x_{qj}$.

Let us consider the problem of encoding the symbolic input variables first. This problem was presented in [5] for the first time. We report here the most relevant results in a more general formulation. We represent the encoding of the symbolic literals by the **face** matrix $F \in \{0, 1, *, \phi\}^{n_p \times n_b}$

$$\bar{F}^3 \Lambda F = (\bar{a}_{\cdot 3} \cdot e_{3\cdot}) \Lambda (A \cdot E) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \cdot [10] \quad \Lambda \quad \begin{bmatrix} 1111 \\ 1100 \\ 0011 \\ 0101 \end{bmatrix} \cdot \begin{bmatrix} 01 \\ 00 \\ 10 \\ 11 \end{bmatrix}$$

$$= \begin{bmatrix} \phi\phi \\ 10 \\ \phi\phi \\ 10 \end{bmatrix} \Lambda \begin{bmatrix} ** \\ 0* \\ 1* \\ ** \end{bmatrix} = \begin{bmatrix} \phi\phi \\ \phi 0 \\ \phi\phi \\ 10 \end{bmatrix} \neq \Phi.$$

$$F = \begin{bmatrix} f_{1\cdot} \\ f_{2\cdot} \\ \cdots \\ f_{n_p\cdot} \end{bmatrix}.$$

Each row of F is a face of the n_b-dimensional Boolean hypercube and corresponds to the face that encodes the symbolic literal. The face matrix can be obtained by performing the matrix pseudo-Boolean selection of E according to an incidence matrix A

$$F = A \cdot E.$$

Example 14: Consider the incidence matrix of Example 12 and the encoding of Example 4. Then

$$E = \begin{bmatrix} 00 \\ 01 \\ 10 \\ 11 \end{bmatrix} \qquad F = A \cdot E = \begin{bmatrix} ** \\ 0* \\ 1* \\ *1 \end{bmatrix}.$$

Now let $\bar{A} = \{\bar{a}_{ij}\}$, where $\bar{a}_{ij} = 1$ if $a_{ij} = 0$; else $\bar{a}_{ij} = 0$. Then $\bar{F}^i \equiv \bar{a}_{\cdot i} \cdot e_{i\cdot}$ is a matrix whose rows are 1) the encoding of word i, if word i neither belongs to the symbolic literal j nor is a *don't care* word; 2) empty values. An encoding matrix E is said to satisfy the **input constraint relation** for a given incidence matrix A if

$$\bar{F}^i \Lambda F \equiv \begin{bmatrix} \bar{f}_{1\cdot}^i \Lambda f_{1\cdot} \\ \bar{f}_{2\cdot}^i \Lambda f_{2\cdot} \\ \cdots \\ \bar{f}_{n_p\cdot}^i \Lambda f_{n_p\cdot} \end{bmatrix} = \Phi, \qquad \forall i = 1, 2, \cdots, n_s$$

where Φ is the empty matrix, i.e., a matrix whose rows have at least one ϕ entry and therefore representing no point in the Boolean space.

Example 15: The encoding matrix of Example 14 satisfies the input constraint relation. However, if we swap the first two rows of E, the input constraint relation is no longer satisfied because the encoding of the third word ADD intersects the fourth face, or equivalently

The problem of encoding the values taken by a symbolic input variable is equivalent to finding an encoding matrix satisfying the input constraint relation. An optimal solution is one of minimal encoding length. Therefore, we can state

> **Encoding problem E1:** Given an incidence matrix A, find an encoding matrix E with minimal number of columns that satisfies the input constraint relation.

Let us consider now the problem of encoding the output variables.

Definition 3: Let $A \in \{0, 1\}^{p \times q}$ and $X \in \{0, 1\}^{q \times r}$. The **matrix Boolean selection** is

$$A \circ X = C = \{c_{ij}\}^{p \times r}$$

where $c_{ij} \equiv V_{k=1}^{q} a_{ik} \Lambda x_{kj}$ and the sum (V) and product (Λ) operators on Boolean variables have the usual meaning:

V	0	1
0	0	1
1	1	1

Λ	0	1
0	0	0
1	0	1

Let now $G = B \circ E$. Row i, $i = 1, 2, \cdots, n_s$, of matrix G is the logical sum of the encoding of the words that must be covered by the encoding of word i. Therefore, we say that a matrix E satisfies the **output constraint relation** for a given adjacency matrix B if E covers G or equivalently

$$\bar{E} \wedge G = O$$

where $\bar{E}$ is the Boolean complement of E and O is the matrix of 0 entries.

In this case, the problem of encoding the values taken by a symbolic output variable is equivalent to finding an encoding matrix satisfying the output constraint relation. An optimal solution is one of minimal encoding length. Therefore, we can state

Encoding problem E2: Given an adjacency matrix B representing to a partial order, find an encoding matrix E with minimal number of columns that satisfies the output constraint relation.

The solution of problem P1 (P2 or P3) requires the solution of encoding problem E1 (E2 or both) for each symbolic variable, after symbolic minimization. The solution of problem P4 requires the encoding of one set (or more sets) or words corresponding to both input and output variables, after symbolic minimization.

Encoding problem E3: Given an incidence matrix A and an adjacency matrix B representing a partial order, find an encoding matrix E with minimal number of columns that satisfies both the input and the output constraint relation.

We explore now the existence of solutions to the encoding problems E1, E2, and E3. It was shown in [5] that 1-hot encoding satisfies always any input constraint relation.

Theorem 4: The identity encoding matrix $E = I \in \{0, 1\}^{n_s \times n_s}$ satisfies the input constraint relation for any given incidence matrix A.

Proof: It is reported in [5]. ■

Theorem 5: Given any incidence matrix A, let $\tilde{A}$ be any Boolean matrix obtained from A by replacing any * entry by 1 or 0. Then $E = \tilde{A}^T$ satisfies the input constraint relation.

Proof: It can be derived from the proof of a similar theorem [5] in a straightforward way. ■

Theorems 4 and 5 show that there always exist an encoding that satisfies the input constraint relation, but the length of the encoding suggested by the theorems are often far from the minimal length.

For the output variable encoding problem, there exists a trivial solution corresponding to the 1-hot encoding solution to the input encoding problem. In particular, if we enumerate the words consistently with the partial order (i.e., if we reorder the words so that word i never covers word j if $j > i$), then a strictly upper triangular matrix of 1 entries is a valid solution. Moreover, note that the first column of this encoding matrix can be dropped, because all entries are 0.

Theorem 6: The encoding matrix $E = PU$ satisfies the output constraint relation for any given adjacency matrix B representing a partial order, where $U \in \{0, 1\}^{n_s \times n_s - 1}$; $U = \{u_{ij}\}$; $u_{ij} = 1$ if and only if $j \geq i$ and P is a permutation matrix such that $P^{-1}BP$ is a strictly upper triangular matrix.

Proof: Note first that since B represents a partial order, there always exists a symmetric permutation such that $P^{-1}BP$ is a strictly upper triangular matrix. Now $G = B \circ E = B \circ PU = BP \circ U$. Then $\bar{E} \wedge G = P\bar{U} \wedge BP \circ U = P(\bar{U} \wedge W)$, where $W = P^{-1}BP \circ U$. Since $P^{-1}BP$ is a strictly upper triangular matrix, then $w_{ij} = 0$ if $j \leq i$. However, $\bar{u}_{ij} = 0$ if $j \geq i$. Then $\bar{E} \wedge G = O$, where O is a matrix of 0 entries and the output constraint relation is satisfied. ■

Theorem 7: Given any adjacency matrix B representing a partial order, $E = \bar{B}^T$ satisfies the output constraint relation.

Proof: Let $G = B \circ \bar{B}^T$. Then $g_{ij} = \bigvee_{k=1}^{n_s} b_{ik} \wedge \bar{b}_{ik} = 0$, $\forall i, j = 1, 2, \cdots, n_s$. Therefore, $\bar{E} \wedge G = O$ and the output constraint relation is satisfied. ■

Theorems 6 and 7 show that there always exist an encoding that satisfies the output constraint relation. As in the previous case, the length of these encodings is often far from the minimal length.

Unfortunately, it is not possible to state the unconditional existence of an encoding that satisfies both the input and the output constraint relations simultaneously.

Theorem 8: Given any incidence matrix A and any adjacency matrix B representing a partial order, a necessary and sufficient condition for the existence of an encoding that satisfies both the input and the output constraint relations is that for each triple of words $r, s, t \in S$ such that $b_{rs} = 1$ and $b_{st} = 1$, $\exists k$ s.t. $a_{kr} = 1$; $a_{ks} = 0$; $a_{kt} = 1$.

Proof: Necessity. For the sake of contradiction, suppose there exists an encoding matrix E satisfying both the input and the output constraint relation and suppose that $b_{rs} = b_{st} = 1$ and for some k, $1 \leq k \leq n_s$, $a_{kr} = 1$, $a_{ks} = 0$, and $a_{kt} = 1$. Let $J_{rt} = \{j | e_{rj} = 1 \text{ and } e_{tj} = 0\}$ and $J_{st} = \{j | e_{sj} = 1 \text{ and } e_{tj} = 0\}$. Since by assumption the encoding of r covers the encoding of s which covers the encoding of t, then $J_{rt} \subset J_{st}$. Then face $f_{k\cdot}$ is such that $f_{kj} = *$ if $j \in J_{rt}$ and either $f_{kj} = *$ or $f_{kj} = e_{tj}$ if $j \notin J_{rt}$. Since $e_{sj} = e_{tj}$ $\forall j \notin J_{st}$ and $f_{kj} = *$ $\forall j \in J_{st}$ then $f_{k\cdot} \wedge e_{s\cdot} \neq \Phi$, the input constraint is not satisfied and we have a contradiction.

Sufficiency. Let $\tilde{A}$ be a matrix obtained from A by replacing the * entries by 1 or 0. Then, $\tilde{E} = \tilde{A}^T$ satisfies the input constraint relation. Let E be a matrix constructed as follows. For each column k of $\tilde{E}$, let

$$J_k^0 = \{j | \tilde{e}_{jk} = 0 \quad \text{and} \quad \exists x \text{ s.t. } \tilde{e}_{xk} = 1 \quad \text{and} \quad b_{xj} = 1\}$$

$$J_k^1 = \{j | \tilde{e}_{jk} = 0 \quad \text{and} \quad \exists x \text{ s.t. } \tilde{e}_{xk} = 1 \quad \text{and} \quad b_{jx} = 1\}.$$

The set $J_k^0(J_k^1)$ is the set of words with a 0 encoding in $\tilde{e}_{\cdot k}$ and required to be covered by (to cover) some words with a 1 encoding in $\tilde{e}_{\cdot k}$. Note that by assumption

$J_k^0 \cap J_k^1 = \phi$. Now let

$$e_{\cdot k} = \begin{cases} \tilde{e}_{\cdot k} & \text{if } J_k^1 = \phi \\ \bar{\tilde{e}}_{\cdot k} & \text{if } J_k^1 \neq \phi \text{ and } J_k^0 = \phi \\ t_{\cdot k} & \text{else} \end{cases}$$

where $t_{\cdot k}$ is a column vector whose entries are bit-pairs

$$t_{jk} = \begin{cases} 01 & \forall j \text{ s.t. } \tilde{e}_{jk} = 1 \\ 11 & \forall j \in J_k^1 \\ 00 & \text{else.} \end{cases}$$

Then $e_{\cdot k}$ satisfies the output constraint relation. Since the encoding matrix E is obtained from $\tilde{E}$ by replacing some columns by their complement or by a two-bit encoding of their entries, then the encoding matrix E satisfies the input constraint relation. Moreover, since the covering relations are satisfied for each column by construction, then also the entire encoding matrix E satisfies the output constraint relation. ■

B. Encoding Algorithms

A solution to the encoding problems E1, E2, or E3 is an encoding of minimal length that satisfies the input, output, or both constraint relations. It can be shown easily by example that, for many instances of these problems, the encoding matrices of Theorems 4–8 do not have minimal numbers of columns. Therefore, it is interesting to devise algorithms which construct a solution to problems E1, E2, and E3. Unfortunately, these are computationally complex problems of combinatorial optimization and it is not known whether an optimal solution can be computed by nonenumerative procedures. Since the growth of computation time as the size of the problem increases is a practical limitation to computer-aided design systems, we consider here heuristic algorithms for the solution of the above problems. Experimental results show that the encodings constructed by these algorithms have reasonably short length, and often equal to the minimum length solution when this is known.

The heuristic techniques presented below solves the encoding problems using greedy strategies. An encoding matrix E is grown from an initial seed matrix by concatenating rows and/or columns. At each step, the best local concatenation is computed, while keeping the previously computed encoding matrix as such. As a result, the encoding matrix grows in size, until all the rows are encodings for the words in S that satisfy the constraint relations. The heuristic selections that drive the algorithm attempt to minimize the steps that increase the number of columns to obtain a solution, and therefore guarantee a weak optimality.

There are two major approaches to constructing matrix E: a **row-based** method and a **column-based** one. In the former method, the encoding matrix is constructed row by row, i.e., by computing the encoding of the words one at a time [5]. In the latter approach, the encoding matrix is constructed column by column, i.e., by computing one bit of the encoding of all the words at each pass. This idea was introduced first by Dolotta [7] to solve the optimal state assignment problem, though the encoding problem was stated differently.

A row-based encoding method was presented in [5] to solve problem E1. This technique is also effective for solving problem E2. The algorithm can be sketched as follows.

STEP 1: Select a word not yet encoded.
STEP 2: Determine the encodings for that word satisfying the constraint relation restricted to that word and to the previously encoded words.
STEP 3: If no encoding is found, increase the code dimension by adding a column to E and go to STEP 2.
STEP 4: Assign an encoding to the selected word by adding a row to E.
STEP 5: If all words have been encoded, stop. Else go to STEP 1.

We refer the reader to [5] for the details of the algorithm when applied to problem E1. We describe here in brief how this algorithm can be used to solve problem E2. The words are selected iteratively at STEP 1 by choosing nodes with no outgoing edges in the directed acyclic graph representing the partial-order relation. The node corresponding to the selected word is then deleted from the graph. Let n_b be the current encoding length. At the beginning, let $n_b = \text{ceiling } (\log_2 n_s)$. In the first pass of the algorithm, the encoding of the first selected word is a row vector of dimension n_b with all 0 entries. In the subsequent passes at STEP 2, the encodings of length n_b that are not rows of E are determined by Boolean complementation. Encodings that do not satisfy the output constraint relation restricted to the selected and already encoded words are discarded. At STEP 3, if no valid encoding is left, a column of 0 entries is appended to matrix E. Otherwise, an encoding is selected at STEP 4 that minimizes the number of 1 entries.

The row-based encoding algorithms generate short encodings (see [5] for experimental results) but fail to be effective for large examples. In particular, the candidate encodings generated at STEP 2 may increase exponentially with the encoding length n_b. Therefore, when the encoding length increases, the computer implementation of the algorithm slows down considerably. Moreover, it is complex to handle problem E3 with this approach because the effectiveness of the algorithm depends heavily on the heuristic ordering of words at STEP 1, and for both input and output constraints there may exist conflicting orderings.

The need to handle both constraints simultaneously, as well as a desired computational-time complexity linear in n_b, has lead to the development of a column-based algorithm. In a column-based algorithm, a single-bit encoding of all the words is introduced at each step. While there always exist single-bit encodings of all the words that satisfy the output constraint relation, it is unlikely that such an encoding satisfies the input constraint relation. Therefore, we say that, given an incidence matrix A, an encod-

ing matrix E **partially satisfies** the input constraint relation if E satisfies the input constraint relation for A', where A' is a subset of the rows of A. The **satisfaction ratio** is $n_{p'}/n_p$, where $n_{p'}$ is the row cardinality of A'. Then, the column-based encoding algorithm can be sketched as follows.

STEP 1: Select a column vector in $\{0, 1\}^{n_s}$ that satisfies the output constraint relation and corresponding to a maximal satisfaction ratio.
STEP 2: If at the first pass, let E be the selected vector. Else, append the selected column vector to E.
STEP 3: If E satisfies the input constraint relation, stop. Else go to STEP 1.

Before describing in detail the column-based encoding algorithm, we mention some properties of the encoding problems that are relevant to this method. If each column of E satisfies the output constraint relation, so does the entire matrix E and *vice versa*. This is not true for the input constraint relation. In particular, if E satisfies the input constraint relation, then a subset of columns of E may not satisfy it. However, we can prove that by appending a column to E, the satisfaction ratio cannot decrease.

Theorem 9: If E satisfies the input constraint relation for a given A, then $E' = [E|T]$ satisfies the input constraint relation, where T is any $\{0, 1\}$ matrix with n_s rows.

Proof: It is reported in [5]. ■

Therfore, our strategy is to select columns that increase the satisfaction ratio until it reaches unity and the algorithm terminates.

Theorem 10: If E satisfies the input constraint relation for a given A, then $E' = [E|\alpha^T]$ satisfies the input constraint relation for $\left[\frac{A}{\alpha}\right]$.

Proof: Since E satisfies the input constraint relation for A, by Theorem 9 $[E|\alpha^T]$ satisfies the input constraint relation for A. We just need to prove that $[E|\alpha^T]$ satisfies the input constraint relation for α. Since $[\alpha^T]$ satisfies the input constraint for $[\alpha]$ by Theorem 5, then $[E|\alpha^T]$ satisfies the input constraint relation for α, again by Theorem 9. ■

If the algorithm is used to solve problem E1, by selecting as columns the transpose of the rows of A, we construct as a solution $E = A^T$, which is a valid solution if we replace the * entries by 0's or 1's (Theorem 5). On the other hand, if the algorithm is used to solve problem E3, there exists also a column selection that increases the satisfaction ratio by at least $1/2 \times n_p$, i.e., at most two columns will be needed to satisfy the input constraint set by each row of A, while satisfying any admissible output constraint. This is shown by the proof of sufficiency of Theorem 8. However, since the optimality of the solution is measured by the number of columns of E, we need column selections that maximize the increase of the satisfaction ratio. The column selection procedure is described in the detail in the sequel.

For computational efficiency, it is important to reduce the number of rows of A to the minimal number that represents an equivalent constraint on the encoding. It is trivial that duplicate rows can be deleted, as well as rows without 0 entries or with only one 1 entry. The former set does not represent a real constraint (all encodings must be contained in the Boolean space) and the latter set requires that some encodings must contain themselves (which is a tautology). The number of rows of A can be compressed further by using the result of the following theorem.

Theorem 11: Let $\tilde{A}$ be the subset of the rows of A with no * entries. An encoding matrix E satisfies the input constraint relation for $\tilde{A}$ if and only if E satisfies the constraint relation for A', where A' is the subset of rows of $\tilde{A}$ that are not Boolean products of two or more rows of $\tilde{A}$.

Proof: The proof is reported in [5]. ■

The number of rows of A can be reduced according to these arguments. If the number of rows after performing the reduction is *ceiling* $(\log_2 n_s)$, then $\tilde{A}^T$ is an optimal solution to E1, where A is obtained from A by replacing the * entries by 1's or 0's. In general, there may be duplicate columns in $\tilde{A}$ which would lead to duplicate encoding of the words. This is easily resolved by appending appropriate columns to E to distinguish the duplicate encodings. The problem of duplicate encodings would not exist if we consider the original incidence matrix A obtained from a minimal symbolic cover. In fact, two columns with identical entries would imply that the corresponding words are incident to the same set of literals, and therefore are indistinguishable with respect to the switching function.

The column-based encoding algorithm is based on the ideas sketched above. The input to the algorithm is the incidence matrix A, the adjacency matrix B, and the encoding options, including possibly an upper bound n_{b_max} on the numbers of columns of the encoding matrix E. Let *FI* and *FO* be two logical flags: *FI* is TRUE when solving problems E1 or E3 and *FO* is TRUE when solving problems E2 or E3.

```
COLUMN-BASED ENCODING ALGORITHM
Data A, B;
Data FI, FO, n_b_max;
n_b = 0;
if (FI) A = clean (A);
if (FI) A = compress (A);
if (FI Λ FO) A, B = verify_constraints (A, B);
do {
    e = column_select;
    if (n_b = 0)
        E = e;
    else
        E = [E|e];
    n_b = column cardinality of E;
    if (FI) A = reduce_constraints (A);
};
while (termination criterion not satisfied);
```

Procedure **clean** records first the multiplicity of each row of A in a weight vector. Then, duplicate rows are

deleted, as well as rows without 0 entries or with only one 1 entry. Procedure **compress** reorders first the rows of A as $\left[\frac{\tilde{A}}{A^*}\right]$, where $\tilde{A}$ has no * entries. Then **compress** returns $A = \left[\frac{A'}{A^*}\right]$, where A' is obtained by deleting the rows of $\tilde{A}$ that are Boolean products of two or more rows of $\tilde{A}$, as justified by Theorem 11.

Procedure **verify__constraints** is invoked if both flags *FI* and *FO* are TRUE, that is, when problem E3 is being solved. The necessary and sufficient condition for the existence of an encoding for the given constraints (Theorem 8) is checked before beginning the encoding. If conflicting constraints are found, then some constraints must be released so that it is possible to find a solution. Unfortunately, a consequence of the release of a constraint is that the encoded Boolean cover might require more implicants that the minimal symbolic cover. An optimization sub-problem is to find the set of constraints whose release would minimize the impact on the encoded cover, i.e., that would minimize the possible increase of implicants. We have found experimentally that it is effective to keep the matrix A unaltered and release the covering constraints by lowering some 1 entries of B to 0 until the modified B matrix and the matrix A satisfy the conditions for existence of an encoding.

Procedure **reduce__constraints** aims at reducing the computational burden of selecting a column by updating the matrix A at each pass of the algorithm. The rationale of the update is as follows. Let E be the encoding matrix partially constructed at a given pass of the algorithm. A partial face matrix $F = A \cdot E$ corresponds to this encoding. Then it is possible that the encodings of some words do not belong to some Boolean subspaces specified by F. The encoding of these words may be originally required not to intersect these faces and therefore this requirement is already satisfied by the partial encoding. In this case, the corresponding entries in the A matrix, that were originally 0s, can be modified to *. If a row of A does not contain any more 0 entries, then the corresponding constraint is satisfied. Equivalently, E satisfies the input constraint relation for that row of A. Since appending additional columns to E will leave the constraint satisfied, that row can be dropped from A.

Example 16: Suppose we are solving problem E1. Let

$$A = \begin{bmatrix} 0100011 \\ 1001000 \\ 0001001 \end{bmatrix}.$$

Let $E = [0100011]^T$. Then $F = A \cdot E = [10^*]^T$. Then the constraint represented by the first row of A is satisfied because the first face is 1 and the encoding of the words (in positions 1, 3, 4, and 5) which must not intersect this face are all 0's. Therefore, the first row of A can be dropped from further consideration. Moreover, the second face is 0, and the words whose encoding must not intersect this face are in positions 2, 3, 5, 6, and 7. The encoding of the word in positions 2, 6, and 7 is 1 and cannot intersect the second face. Therefore, the constraints that remain to be satisfied can be represented by matrix

$$A = \begin{bmatrix} 1^*010^{**} \\ 0001001 \end{bmatrix}.$$

Procedure **reduce__constraints** can be summarized as follows:

reduce__constraints
$F = A \cdot E$;
for (i = 1 to n_p) {
  for (j = 1 to n_s) {
    if ($a_{ij} = 0$ and $f_{\cdot i} \cap e_{\cdot j} \neq \Phi$) $a_{ij} = *$;
  };
};
A = **clean** (A);

The termination criterion of the column-based encoding algorithm depends on the options which are specified. In any case, the algorithm continues to append columns to E if some encodings are equal to some others. If the flag *FI* is set, then the algorithm terminates when E satisfies the input constraint relation. In view of the reduction of the constraint matrix done by procedure **reduce__constraints**, the algorithm terminates when matrix A has no rows left. If an upper bound is specified on the encoding length, then a sufficient condition for termination is reaching that bound. Note that, in this case, the encoding will not necessarily satisfy the constraint relations. However, care is taken in the column selection procedure so that the constructed encoding matrix E with bounded column cardinality has all rows different from each other.

We can now describe the **column__select** procedure, which is the heart of the algorithm. We consider first the cases in which no upper bound on the encoding length is specified. Let us assume that only flag *FI* is TRUE. The procedure aims at minimizing the encoding length, and therefore we look for a vector in $\{0, 1\}^{n_s}$, that maximizes the satisfaction ratio at each pass of the algorithm. To achieve this goal, we consider first all the pairs of rows of A. Two rows of A, $a_{1\cdot}$ and $a_{2\cdot}$ are said to be **compatible** if either $a_{1j} = a_{2j}$ $\forall j$ s.t. $a_{1j} \neq *$ and $a_{2j} \neq *$ or $a_{1j} = \bar{a}_{2j}$ $\forall j$ s.t. $a_{1j} \neq *$ and $a_{2j} \neq *$. It is clear that, for each pair of compatible rows, there exist a 0-1 assignment to the * entries of either one that is an encoding vector that satisfies the input constraint relation for the pair of rows.

Example 17: Let

$$A = \begin{bmatrix} *11001 \\ 00011* \\ 11110* \end{bmatrix}.$$

The first two rows of A are compatible. The encoding $[111001]^T$ satisfies the input constraint relation for these two rows.

Given a maximal set of pair-wise compatible rows,

there exists a 0-1 assignment to any of them that is an encoding maximizing the satisfaction ratio. Therefore, procedure **column_select** determines first a set of pairwise compatible rows of A, denoted by A', maximizing a figure of merit that takes into account the set cardinality and the row multiplicity (stored in the weight vector). Let J be the set of indices of the columns of A' whose entries are not all *. Clearly, only the positions in the encoding column vector specified by J are important to satisfy the constraint relation specified by A'. Procedure **column_select** takes one row in A' and assigns 0 or 1 to the * entries in the positions specified by J in such a way to maintain compatibility. (Procedure **select_i** returns a column vector given a matrix A according to this strategy.) Then, the remaining * entries can be assigned according some tie rules. In the case that only flag *FI* is set, the * entries are assigned first as to minimize the size of the faces (dimension of the Boolean subspaces) represented by the corresponding face matrix. In case of further ties, the assignment is done to maximize the number of 0 entries in the A matrix that will be modified to * by procedure **reduce_constraints**.

Let us now turn to the case in which both flags *FI* and *FO* are TRUE (problem E3). The column selection involves first the detection of a vector which maximizes the satisfaction ratio, as in the previous case. We call $\tilde{e}$ this vector. Then, this vector has to be transformed to satisfy the output constraint relation as well. Let $J^0 = \{ j | \tilde{e}_j = 0$ and $\exists x$ s.t. $\tilde{e}_x = 1$ and $b_{xj} = 1\}$ be the set of words with a 0 encoding in $\tilde{e}$ and required *to be covered* by some words with a 1 encoding in $\tilde{e}$. Let $J^1 = \{ j | \tilde{e}_j = 0$ and $\exists x$ s.t. $\tilde{e}_x = 1$ and $b_{jx} = 1\}$ be the set of words with a 0 encoding in $\tilde{e}$ and required to *cover* some words with a 1 encoding in $\tilde{e}$. Note that these two sets do not intersect because the matrices A, B satisfy the assumptions of Theorem 8, after having been possibly modified by procedure **verify_ constraints**. There are now four possibilities:

i) $J^1 = \phi$ and $J^0 = \phi$;
ii) $J^1 = \phi$ and $J^0 \neq \phi$;
iii) $J^1 \neq \phi$ and $J^0 = \phi$;
iv) $J^1 \neq \phi$ and $J^0 \neq \phi$.

In the first two cases, there exists a 0-1 assignment of the * entries of vector $\tilde{e}$ that satisfies the output constraint relation and we set $e = \tilde{e}$. In the third case, there exists a 0-1 assignment of the * entries of the complement of vector $\tilde{e}$ that satisfies the output constraint relation and we set $e = \bar{\tilde{e}}$. In the last case, we set $e = t(\tilde{e})$, where $t(\tilde{e})$ is a column vector whose entries are bit-pairs:

$$t_j = \begin{cases} 01 & \forall j \text{ s.t. } \tilde{e}_j = 1 \\ 11 & \forall j \in J^1 \\ 00 & \forall j \in j^0 \\ e_j | e_j & \text{else} \end{cases}.$$

There exists now a 0-1 assignment to the * entries of each column of e that satisfies the output constraint relation. Moreover, any 0-1 assignment to the * entries of e satisfies the input constraint relation for the same subset of rows of A as $\tilde{e}$. The assignment of the * entries is done according to tie rules, as mentioned before.

If the algorithm is used to solve problem E2 (only flag *FO* is true), then the column selection is done by determining directly a vector that leads to a short encoding among those that satisfy the output constraint relation. The strategy used to select a column, having as a primary goal the determination of a short-length encoding, is similar to that used in the case an upper bound is imposed on the encoding length.

If an upper bound $n_{b\_\max}$ on the encoding length is specified, then it is imperative that the rows of E are different from each other after $n_{b\_\max}$ column assignments. Therefore, there may be groups of identical rows in E with at most cardinality $2^{(n_{b\_\max} - n_b)}$, after having assigned n_b columns. For this reason, the column selection is done such that, if there are groups of identical rows in $[E|e]$, their cardinality is most $2^{(n_{b\_\max} - n_{b'})}$, where $n_{b'}$ is the column cardinality of $[E|e]$. When flag *FI* is TRUE (or when both flags *FI* and *FO* are true), this requirement restricts the selection of the column vector e (or $\tilde{e}$). In particular, we have to consider a reduced set of rows of A in which we search for a maximal set of compatible rows. (Procedure **drop** drops the rows of A incompatible with the code length goal from further consideration in the **column_select** routine.) The assignment of the * entries is done to satisfy first the encoding-length constraint by trying to minimize the cardinality of the groups of identical rows in $[E|e]$. In case of tie, the previous rules are used.

In the case that both flags are FALSE, (or in the case that *FI* is TRUE, E satisfies the input constraint relation and some rows of E are identical) then **column_select** returns a column which would minimize the cardinality of the groups of identical rows in $[E|e]$. (Procedure **select_0**.) If, in addition, flag *FO* is TRUE, the search is limited to the column vectors satisfying the output constraint relation. (Procedure **select_o**.)

In summary, procedure **column_select** can be represented as follows:

```
column_select
if (bound on n_b) A = drop (A);
if (FI and satisfaction ratio < 1){
      e = ẽ = select_i (A);
      if (FO){
            if (J^1 ≠ φ and J^0 = φ) e = ẽ̄;
            if (J^1 ≠ φ and J^0 ≠ φ)e = t(ẽ);
      };
};
else {
      if (FO)
            e = selecto_o;
      else
            e = selecto_0;
};
```

$e = e$ with 0-1 assignment of the * entries according to the tie rules;

We summarize now the properties of the column-based encoding algorithm in a qualitative way. The algorithm constructs an encoding matrix E. If problem E2 or E3 have to be solved, each column is chosen as to satisfy the output constraint relation. Therefore, so does the entire matrix E. If problem E1 or E3 have to be solved, each column is chosen as to satisfy partially the input constraint relation, or equivalently to satisfy the input constraint relation for a subset of the rows of A. These rows are then discarded from further consideration. After a finite number of steps, matrix E satisfies the input constraint relation. The heuristic selection of the column attempts to increase maximally the satisfaction ratio at each step, and therefore guarantees a weak minimality of the column cardinality of E. If an upper bound on the encoding length is specified, then the encoding may satisfy only partially the constraint relations, but attempts to satisfy most of the constraints.

The construction of matrix E with bounded column cardinality allows to tradeoff the minimality of the cover cardinality for that of the encoding length. In particular, given an encoding of length n_b, it is possible to determine the cover cardinality (or a bound on the cover cardinality n_p) on the basis of the satisfied constraints. Therefore, for a particular switching function, it is possible to determine a set of pairs of parameters (n_b, n_p) that relate to the size of the implementation.

The worst-case computational cost of the algorithm grows cubically with n_s because the **column_select** routine involves $0(n_s^2)$ operations and is invoked $n_b = 0(n_s)$ times. It grows quadratically with n_p because of the pairwise comparisons in routines **column_select** and **reduce_constraints**. It grows linearly with n_b. Note that n_b is not an input datum parameter, but the linear growth shows that the amount of computation per column is constant.

As a final remark, we would like to compare the column-based encoding algorithm with the one proposed by Dolotta [7] and later perfected by Weiner [23], Torng [22], and Story [20]. These algorithms addressed the state assignment problem for finite-state machines, and the mechanism of the encoding algorithm is similar to the one presented here. However, the selection of the columns was based on a heuristic criterion; in particular, a scoring function was used to select a column on the basis of the likelihood that logic minimization, which would have followed the encoding, could reduce the two-level cover cardinality. (Story's method optimized the number of and–or inputs for each column choice.) In our approach, we relate each column assignment to the satisfaction of some input constraints and therefore to a known reduction of the cover cardinality. Therefore, column selection is related to cover minimality in a deterministic way. However, our encoding technique is still a heuristic one because the greedy strategy considers and assigns only one column at a time.

TABLE II
OVERALL RESULTS

Example	n_i	n_s	n_o	n_p	$n_{p'}$	n_b
EX1	2	6	2	24	11	4
EX2	7	16	7	91	49	8
EX3	1	9	2	170	78	12
EX4	2	4	1	11	6	2
EX5	2	9	1	25	10	5
EX6	1	12	1	24	17	7
EX7	7	48	19	115	89	10
EX8	8	20	6	107	68	7
EX9	11	32	9	184	107	9
EX10	1	8	1	16	14	4
EX11	9	30	10	166	103	12
EX12	4	4	4	49	11	3
EX13	2	11	1	25	10	6
EX14	4	5	1	20	8	4
EX15	8	7	5	56	23	5
EX16	8	4	5	32	15	4
EX17	4	27	3	108	49	11
EX18	4	8	3	32	17	4
EX19	2	7	2	14	9	3
EX20	4	15	3	30	22	7

C. Implementation and Results

The column-based encoding algorithm has been implemented in program CREAM. CREAM is written in APL and consists of about 25 functions. CREAM is designed to be used with CAPPUCCINO: it takes the representation of a minimal symbolic cover and generates an encoding that can replace the symbolic entries. The final result is a Boolean cover that can be implemented as a PLA (after having been folded or partitioned, if desired), or used as a starting point for multiple-level synthesis, as done by the YLE program in the Yorktown Silicon Compiler [3].

CREAM solves the encoding problem E1, E2, or E3 at the user's request. It accepts an upper bound on the number of columns to be used in the encoding. For a given bound and the corresponding encoding of all the symbolic fields, it is possible to estimate the area taken by a PLA implementation. Therefore, it is possible to estimate the area as well as the aspect ratio. This computation can be done for different bounds on the encoding, and therefore a designer can choose among several implementations with different areas and aspect ratios.

CAPPUCCINO and CREAM have been tested on several examples. Some results are reported in Table II.

The examples are sequential circuits, i.e., we are solving problem P4 by symbolic minimization first and by solving encoding problem E3 after. The symbolic representations have four fields corresponding to the primary inputs/outputs (which are represented by Booleans variables) and the present/next states (which are represented by symbolic variables). The first four numeric columns represent the parameters of the function: n_i is the number of primary inputs, n_s is the number of states, n_o is the number of primary outputs, and n_p is the cardinality of the initial symbolic cover. The last two columns show the final cardinality of the Boolean cover and the state encoding length. Note that for a few examples (As EX1, EX4, $\cdots$), the Boolean cover cardinality is larger than the minimal symbolic cardinality (reported in the third column of Table I) because it was not possible to satisfy all

TABLE III
ENCODING-LENGTH COMPARISONS

Example	n_s	$\log_2 n_s$	n_b	$n_{s'}$	$n_{s''}$
EX1	6	3	4	3	3
EX2	16	4	8	7	7
EX3	9	4	12	12	12
EX4	4	2	2	2	2
EX5	9	4	5	4	4
EX6	12	4	7	4	6
EX7	48	6	10	8	8
EX8	20	5	7	6	7
EX9	32	5	9	7	7
EX10	8	3	4	4	3
EX11	30	5	12	9	9
EX12	4	2	3	3	3
EX13	11	4	6	4	4
EX14	5	3	4	3	3
EX15	7	3	5	5	5
EX16	4	2	4	4	4
EX17	27	5	11	9	7
EX18	8	3	4	4	4
EX19	7	3	3	3	3
EX20	15	4	7	7	5

constraints in the encoding. Note also that a further reduction in cardinality may be obtained by minimizing again the Boolean covers that are not minimal. The computing time is in the order of few seconds for all these examples, on an IBM 3081 computer.

In Table III, we try to evaluate the optimality of the encoding, in terms of encoding length. The second column represents the number of words to be encoded n_s; the third represents the minimal number of bits needed to encode the words regardless of any constraint on their encoding (i.e., *ceiling* ($\log_2 n_s$)); the fourth column represents the encoding length as constructed by CREAM as a solution to problem E3. These three columns show that the encoding length computed by CREAM is close enough to the minimum length; for most of the examples *ceiling* ($\log_2 n_s$) $\leq n_b \leq 2 \times$ *ceiling* ($\log_2 n_s$). For some examples (as EX14, EX16, EX18, EX19, $\cdots$), it is possible to provide that no encoding of length inferior to n_b can be a solution of the encoding problem. The results of CREAM can be compared with those obtained by program KISS [5], which uses a row-based encoding algorithm. The length of a solution to problem E1 computed by CREAM is given in the fifth column of the table and the length of an encoding constructed by KISS in the last column. (The algorithm of KISS can solve only problem E1.) Note that, for some examples, the row-based algorithm gives a shorter encoding. However, the corresponding implementation requires more product-terms, as shown by the fourth column of Table I.

As a final remark, it would be interesting to rate the effectiveness of the symbolic design methodology by relating the experimental results to a measure of the difficulty of the examples, as in the case of channel routing. Unfortunately, it is difficult to classify the examples on an absolute scale. The examples chosen here are derived from finite state machine (FSM) tables. The present methodology appears to be effective especially for FSM's having large and sparse transition graphs.

V. Concluding Remarks and Future Work

Combinational and sequential circuits can be optimized using the symbolic design methodology. Symbolic functions are represented by tables of symbols, which may be direct translations of hardware description language programs. Symbolic minimization allows encoding-independent optimization of switching functions, and the encoding algorithms construct a binary representation of the symbols that translate the minimal symbolic cover into a compatible Boolean cover.

The target technology of the symbolic design method is a two-level *sum of products* circuit implementation, such as a programmable logic array. However, this technique can be used in conjunciton with other implementation methodologies, such as those supported by the Yorktown Silicon Compiler [3], by mapping the optimal two-level representation into a multiple-level logic representation which fits the implementation technology.

Future work will address the extension of symbolic design to multiple-level implementations. In this case, the objective function of the optimization will consider directly multiple-level implementations without resorting to the *sum of product* model.

It is important to note that the algorithms we presented are heuristic and can still be improved. Other schemes for symbolic minimization can be tried. The symbolic minimization loop could be replaced by a multiple-valued expansion of the output parts of the symbolic implicants that takes into account the order relations. In the case of problems P4 and E3, the symbolic minimization algorithm could include directly restrictions on the operations on the symbolic cover such that the computed cover can always be encoded into a Boolean cover with the same cardinality. The encoding algorithm could be improved by combining row-based and column-based encoding techniques or by iterating or backtracking in the encoding procedure to achieve shorter encodings. Ideally, the encoding algorithm should be merged with the minimization procedure so that the entire optimization method could use the silicon area of the physical implementation as the objective function.

Acknowledgment

The author would like to thank R. Brayton, R. Rudell, and A. Sangiovanni-Vincentelli for many helpful discussions and D. Ostapko and J. White for reading the manuscript.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. New York: Addison Wesley, 1974.

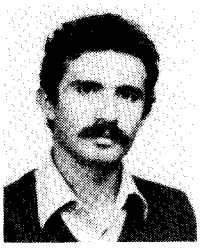
[2] R. Brayton, G. D. Hachtel, C. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.

[3] R. Brayton, N. Brenner, C. Chen, G. De Micheli, C. McMullen, and R. Otten "The Yorktown Silicon Compiler," in *Proc. Int. Symp. on Circuit and Systems* (Kioto, Japan), June 1985, pp. 391-394.

[4] G. De Micheli, R. Brayton, and A. L. Sangiovanni-Vincentelli, "KISS: A program for optimal state assignment of finite state machines," in *Proc. ICCAD* (Santa Clara), Nov. 1984.

[5] G. De Micheli, R. Brayton, and A. L. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, no. 3, pp. 269–284, July 1985.
[6] G. De Micheli, "Symbolic minimization of logic functions," in *Int. Conf. on Comp. Aid. Des.*, (Santa Clara), Nov. 1985, pp. 293–295.
[7] T. A. Dolotta and E. J. McCluskey, "The coding of internal states of sequential machines," *IEEE Trans. Electron Comput.*, vol. EC-13, pp. 549–562, Oct. 1964.
[8] M. R. Garey and D. S. Johnson, *Computers and Intractability*. San Francisco: W. H. Freeman, 1978.
[9] J. Hartmanis and R. E. Stearns, *Algebraic Structure Theory of Sequential Machines*. Englewood Cliffs, NJ: Prentice Hall, 1966.
[10] F. Hill and G. Peterson, *Introduction to Swtiching Theory and Logical Design*. New York: Wiley, 1981.
[11] S. J. Hong, R. G. Cain and D. L. Ostapko, "MINI: A heuristic approach for logic minimization," *IBM J. Res. Develop.*, vol. 18, pp. 443–458, Sept. 1974.
[12] S. K. Hurst, "Multiple-valued logic—Its status and its future," *IEEE Trans. Comput.*, vol. C-33, pp. 1160–1179, Dec. 1984.
[13] A. Nichols and A. Bernstein, "State assignment in combinational networks," *IEEE Trans. Electron Comput.*, vol. EC-14, pp. 343–349, June 1965.
[14] E. J. McCluskey, "Minimization of Boolean functions," *Bell. Lab. Tech. J.*, vol. 35, p. 1417–1444, Apr. 1956.
[15] E. L. Post, "Introduction to a general theory of elementary propositions," *Amer. J. Math.*, vol. 43, pp. 163–185, 1921.
[16] F. Preparata and R. Yeh, *Introduction to Discrete Structures*. Addison Weseley, 1973.
[17] D. Rine, *Computer Science and Multiple-Valued Logic*. New York: North Holland, 1977.
[18] R. Rudell and A. Sangiovanni-Vincentelli, "ESPRESSO-MV: Algorithms for Multivalued Logic Minimization," in *Proc. Custom Int. Circ. Conf.*, (Portland, OR), May 1985.
[19] T. Sasao, "Input variable assignment and output phase assignment of PLA's," *IBM Res. Rep.*, no. 1003, June 1983.
[20] J. R. Story, H. J. Harrison, and E. A. Reinhard, "Optimum state assignment for synchronous sequential circuits," *IEEE Trans. Comput.*, vol. C-21, pp. 1365–1373, Dec. 1972.
[21] S. Y. H. Su and P. T. Cheung, "Computer minimization of multivalued switching functions," *IEEE Trans. Comput.*, vol. C-21, pp. 995–1003, 1972.
[22] H. C. Torng, "An algorithm for finding secondary assignments of synchronous sequential circuits," *IEEE Trans. Comput.*, vol. C-17, pp. 416–469, May 1968.
[23] P. Weiner and E. J. Smith, "Optimization of reduced dependencies for synchronous sequential machines," *IEEE Trans. Electron Comput.*, vol. EC-16, pp. 835–847, Dec. 1967.

*



Giovanni De Micheli (S'79–M'83) was born in Milano, Italy, in 1955. He received the Dr.Eng. degree (summa cum laude) in nuclear engineering from the Politecnico di Milano, Italy, in 1979, and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley, in 1980 and 1983, respectively.

He spent the Fall of 1981 as Consultant in Residence at Harris Semiconductor, Melbourne, FL. In 1983, he was Assistant Professor at the Department of Electronics of the Politecnico di Milano. In 1984, he joined the technical staff of IBM T.J. Watson Research Center Yorktown Heights, NY, where he is currently Project Leader of the Design Automation Workstation Group. He has been Codirector of the NATO Advanced Study Institute on logic synthesis and silicon compilation, held in L'Aquila, Italy, in 1986.

His research interests include several aspects of the computer-aided design of integrated circuits with particular emphasis on silicon compilation, logic synthesis, optimization, and verification of VLSI circuits.

Dr. De Micheli was granted a Fulbright Scholarship in 1980, a Rotary International Fellowship in 1981, and an IBM Fellowship for VLSI in 1982 and 1983. He received a Best Paper Award at the 20th Design Automation Conference, in June 1983.