IMPLEMENTATION AND TRANSFORMATION OF ALGORITHMS BASED ON AUTOMATA

PART II: SYNTHESIS OF EVALUATION PROGRAMS

by A. THAYSE

Abstract

This paper considers the realization of algorithms by programs composed of conditional instructions (binary programs). A mathematical tool, the algebra of P-functions, is introduced for the analysis and synthesis of these programs. P-functions present the following advantages with respect to other known methods: they detect automatically the reconvergent instructions and are able to synthesize a class of programs (non-simple programs) non reachable by other methods.

1. Introduction

The purpose of the present paper is to introduce the concept of P-function which will appear to be the adequate mathematical tool for synthesizing, transforming and hence optimizing evaluation programs.

Section 2 may be considered as a large introductory chapter which simultaneously states the problems at hand and describes in a formal way the types of solutions that will be proposed.

In section 3 one develops the algebra of P-functions which will be used in section 4 for synthesizing programs and in part III of this work for optimizing these evaluation programs.

2. Introduction to conditional program transformation

2.1. Problem statement

Glushkov\(^1,2\) has proposed to describe systems performing computations, a model of cooperation between two subsystems, called operational automaton and control automaton, respectively (see ref. 3). The control automaton is the actual implementation of the computation algorithm; it accepts instructions of the following type.

\[
\begin{align*}
N & \quad f_0(x) \quad \sigma_0 \quad N_0 \\
N & \quad f_1(x) \quad \sigma_1 \quad N_1 \\
\quad & \quad \vdots \quad \quad \vdots \quad \quad \vdots \\
N & \quad f_{m-1}(x) \quad \sigma_{m-1} \quad N_{m-1} \\
\end{align*}
\]

(1)
The interpretation of this instruction may be found in ref. 3; one of the results of ref. 3 is to show that an instruction of the type (1) may always be partitioned into two subprograms: an evaluation program and an execution program. In the course of any computation the conditional instructions, i.e. instructions of the form

\[ N \text{ if } x_i = 0 \text{ then go to } N_0 \text{ else go to } N_1, \]

precede the execution instructions, i.e. instructions of the form

\[ N \text{ do } \sigma_0 \text{ and go to } N_0. \]

In ref. 3 one shows that the optimization of (1) (the optimization criterions will be recalled herebelow) reduces to a separate optimization of its conditional program and of its execution program. The optimization of the execution program is quite an elementary problem and has been considered in ref. 3. The purpose of the present work is to present program transformation techniques for optimizing the evaluation program.

Any evaluation program \( N \) may be described by means of a flow-chart schematized in fig. 1a. The evaluation program of fig. 1a has one input (or initial instruction) labelled \( N \) and \( q \) outputs (or final instructions) labelled \( P_i, 0 \leq i \leq q - 1 \). It is made up of a loop-free interconnection of instructions of the type (2), each of which is represented by its usual flow-chart symbol. Besides to being loop-free, the only restriction to which the interconnection of instructions is constrained is that any instruction output is connected to one and only one instruction input: this constraint prevents any possibility of parallelism and of non determinism in the program.

There are clearly a high number of programs which may describe a computation: these programs are e.g. dependent on the successive choices of the condition variables \( x_i \) and on the possibility of having reconvergent instructions. In the scheme of fig. 1a the instruction \( M_c \) is called reconvergent since it may be reached through the intermediate of at least two (in this case \( M_a \) and \( M_b \)) instructions. There exists thus an optimization problem as long as the synthesis of programs is concerned. The main optimization criterions that will be used further on are the following ones.

(a) The maximal duration of a computation (time criterion).

This duration is reflected in the scheme of figure 1a by the length of the longest path between \( N \) and any \( P_i, 0 \leq i \leq q - 1 \) (the length of a path is defined as the number of its edges).

(b) The number of instructions (cost criterion).

Our main purpose will be to find efficient algorithms for generating programs having either the shortest maximal computation time or the mini-
mal number of instructions. The following intermediate criterion will appear to be of particular interest further on.

(c) Between all the programs having the shortest maximal computation time find those having a minimum number of instructions (*relative cost criterion*). The importance of this last criterion derives from the fact that algorithms for deriving programs having the shortest maximal computation time are the easiest to derive; moreover it appears that the minimal number of instructions in these programs is very near to the absolute minimal number of instructions in any program.

![Diagram](image)

Fig. 1a. Evaluation program, b. Execution program.

2.2. *Introduction to the synthesis tools*

Our purpose will now be to introduce in a very rough way the mathematical
Implementation and transformation of algorithms based on automata

Consider the flow-chart of fig. 1a representing an evaluation program realizing the instruction labelled $N$. To the instruction $N$ we associate a function

$$ f: \{0, 1\}^n \rightarrow \{P_0, P_1, \ldots, P_{q-1}\}, \quad (4) $$

where the $P_i$ are labels associated with the outputs of $N$. Define moreover the Boolean functions $g_i(x)$, $0 \leq i \leq q - 1$ as follows.

$$ g_i(x) = 1 \text{ if and only if } f(x) = P_i, $$

$$ g_i(x) = 0 \text{ otherwise.} $$

This allows us to represent the instruction $N$ by means of the following formal Boolean expression:

$$ f(x) = \bigvee_{i=0}^{q-1} g_i(x) P_i. \quad (5) $$

The $g_i(x)$ are $q$ Boolean functions satisfying the conditions

$$ \bigvee_{i=0}^{q-1} g_i(x) = 1, \quad (6) $$

$$ g_i(x) \land g_j(x) = 0 \quad \forall i, j \in \{0, 1, \ldots, q - 1\}. \quad (7) $$

The orthogonality conditions (7) guarantee that no parallelism is present in the program while the condition (6) implies that the function $f$ is completely specified. While the conditions (7) will always be satisfied, the condition (6) will in some cases be dropped. This occurs e.g. when dealing with incompletely specified programs, i.e. programs where the next instruction is unspecified for a given value of the condition variables $x = (x_{n-1}, \ldots, x_1, x_0)$.

Now any of the internal instructions in the program (e.g. $M_c$ in fig. 1a), may be considered as an initial instruction to a subprogram; hence $M_c$ may be represented by a formal Boolean expression of the same type as (5), i.e.

$$ h_c(x) = \bigvee_{i} g_i^c(x) P_i. \quad (8) $$

The set of Boolean functions $\{g_i^c(x)\}$ satisfies the conditions (6) and (7).

The synthesis and optimization of evaluation programs will be stated by introducing the concept of a $P$-function associated with an instruction.

Consider first the evaluation program of fig. 1a as a black-box with one input terminal $N$ and $q$ output terminals $P_0, P_1, \ldots, P_{q-1}$. To the input terminal $N$ one associates the pair of functions $\langle 1, f(x) \rangle$; to each of the output terminals $P_i$ one associates the pairs of functions $\langle g_i(x), P_i \rangle$, respectively. These functions will be called the $P$-functions associated with the corresponding instruction label. One verifies that any one of the $P$-functions $\langle g, h \rangle$...
A. Thayse

associated with a terminal satisfies the relation

\[ f(x) g(x) = h(x) g(x). \]  

(9)

This relation is interpreted as follows: in the domain of the \( n \)-cube characterized by the relation

\[ g(x) = 1 \]

the function \( f(x) \) reduces to the function \( h(x) \) (\( f(x) \) and \( h(x) \) are in fact formal Boolean expressions of the form (5)); hence the Boolean function \( g(x) \) will be called the domain function while the function \( h(x) \) will be called the co-domain function of the \( P \)-function \( \langle g, h \rangle \) respectively. To any internal instruction \( M_e \) represented by the formal expression (8) one associates the \( P \)-function \( \langle g_e, h_e \rangle \) with \( g_e \) a Boolean function satisfying the relation (9), i.e.

\[ f(x) g_e(x) = h_e(x) g_e(x). \]  

(10)

The interest of the concept of \( P \)-function lies in the following observations (see also fig. 2).

(a) The instruction labelled \( M \) with \( x \) as condition variable and to which is associated the \( P \)-function \( \langle h, g \rangle \) is transformed into two instructions labelled \( M_0 \) and \( M_1 \) and having \( \langle \overline{x} g, h(x = 0) \rangle \) and \( \langle x g, h(x = 1) \rangle \) as \( P \)-functions respectively (see fig. 2a);

(b) The instructions \( M_0 \) and \( M_1 \) having \( \langle g_0, h_0 \rangle \) and \( \langle g_1, h_1 \rangle \) as \( P \)-functions respectively are obtained from an instruction \( M \) having \( \langle \overline{x} g_0 \lor x g_1, \overline{x} h_0 \lor x h_1 \rangle \) as \( P \)-function (see fig. 2b).

One sees thus that there exist transformation laws allowing us: either knowing the \( P \)-function of an instruction to obtain the \( P \)-functions of its two following instructions, or knowing the \( P \)-functions of these two last instructions to find the \( P \)-function of the instruction from which they were derived. Observe
already now that to any instruction \( M \) is associated one and only one co-
domain function \( h \): this function evaluates the subprogram to which the label
\( M \) has been given. To any instruction \( M \) may generally be associated several
functions \( g \) as reflected by the fact that several functions \( g \) may satisfy the
relation (9): this function describes indeed a domain in which \( f \) reduces to \( h \).
Except in the case where this domain reduces to a unique vertex of the \( n \)-cube
there exist several domains in which \( f \) reduces to \( h \).

Summarizing: to any instruction \( M \) one associates the \( P \)-function \( \langle g, h \rangle \),
\( h \) evaluating the subprogram realized by \( M \) and the solutions of \( g(x) = 1 \)
characterizing the domain in which this subprogram is realized.

The synthesis of an evaluation program may then be stated in one of the
two following forms.

(a) Starting from the instructions \( \{ P_i \} \), represented by their \( P \)-functions
\( \{ \langle g_i, P_i \rangle \} \), find a composition law \( T \) which, when acting on a pair of \( P \-
functions, generates a new \( P \)-function, i.e.

\[
\langle g_0, h_0 \rangle T \langle g_1, h_1 \rangle = \langle g, h \rangle. \tag{11}
\]

It is requested that an iterative use of this law \( T \) produces finally the \( P \-
function \( \langle 1, f \rangle \) and that the intermediate \( P \)-functions are in one-to-one
correspondence with the intermediate instructions.

(b) Starting from the instruction \( N \), represented by its \( P \)-function \( \langle 1, f \rangle \), find
a decomposition law \( \Delta \) which, when acting on a \( P \)-function, generates
two new \( P \)-functions, i.e.

\[
\Delta(\langle g, h \rangle) = \{ \langle g_0, h_0 \rangle, \langle g_1, h_1 \rangle \}. \tag{12}
\]

It is requested that an iterative use of this law \( \Delta \) produces finally the \( P \-
functions \{ \langle g_i, P_i \rangle \} \) and that the intermediate \( P \)-functions are in one-to-
one correspondence with the intermediate instructions.

A synthesis method of the type (a) will be called a Leave-to-Root algorithm
(or L–R algorithm) since it starts from the leaves \( P_i \) of the program and comes
to its root \( N \). Similarly, a synthesis method of the type (b) will be called a
Root-to-Leave algorithm (or R–L algorithm).

2.3. Introduction to optimization tools

As reflected by the contents of sec. 2.2, the synthesis of an evaluation pro-
gram reduces to iterative rules for generating \( P \)-functions. To any function
\( f(x) \), with \( x = (x_{n-1}, ..., x_1, x_0) \) may potentially be associated \((2^{2^n} - 1) \) \( P \-
functions: there are indeed \( 2^{2^n} - 1 \) non-empty subdomains of the \( n \)-cube and
thus the same number of domain functions \( g(x) \). It turns out that the selection
of \( P \)-functions that are really useful for the optimization of evaluation pro-
A. Thayse

grams is an essential task in any optimization process. The optimization process will thus be performed by choosing three fundamental parameters, namely:

(a) *the initial P-functions* which are the P-functions associated to the instruction(s) from which one starts;

(b) *a composition law* $T$ or a decomposition law $\Delta$, these laws applied to the initial instruction(s) generate in an iterative way a set of instructions which constitute the evaluation program;

(c) *a selection criterion* which has basically two purposes. A first purpose of this criterion is to minimize, as much as possible, the number of P-functions that are to be computed for evaluating an optimal program; a second purpose is to select the minimization criterion.

Let us already point out now some particular values of the above parameters that will be of special interest further on in this text.

**Initial P-functions**

(a) $\langle 1, f(x) = \bigvee_i g_i(x) P_i \rangle$;

(b) $\langle \langle g_i(x), P_i \rangle \rangle$;

(c) $\langle \langle p_{ij}(x), P_i \rangle \rangle$,

with $\bigvee_j p_{ij}(x) = g_i(x)$, $p_{ij}$ is a prime implicant of $g_i$ (one verifies that $\langle p_{ij}(x), P_i \rangle$ is a P-function of $f$ since it satisfies (9)).

**Composition or decomposition laws**

(b.1) Composition law $T^0$ with respect to $x_0$:

$\langle g_j, h_j \rangle T^0 \langle g_k, h_k \rangle = \langle \bar{x}_i g_j \lor x_i g_k, \bar{x}_i h_j \lor x_i h_k \rangle$;

(b.2) Composition law $T^1$ with respect to $x_i$:

$\langle g_j, h_j \rangle T^1 \langle g_k, h_k \rangle = \langle g_j(x_i = 0) g_k(x_i = 1), \bar{x}_i h_j \lor x_i h_k \rangle$;

(b.3) Decomposition law $\Delta$ with respect to $x_i$:

$\Delta(\langle g_j, h_j \rangle) = \{\langle \bar{x}_i g_j, h_j(x_i = 0) \rangle, \langle x_i g_j, h_j(x_i = 1) \rangle\}$.

To state selection criterions in a rigorous form before establishing the algebra of P-functions is more difficult. Three classes of selection criterions will roughly be described below. These criterions will rigorously be stated in part III of this work after the algebra of P-functions has been developed in in sec. 3.

**Selection criterions**

(c.i) Selection criterions grounded on the comparison of the size of the domain
functions; these selection criterions are used together with a composition law \( T \).

(c\(_2\)) Selection criterions grounded on the properties of the normal disjunctive form of the co-domain functions; these selection criterions are used together with a decomposition law \( \Delta \).

(c\(_3\)) Selection criterions grounded on the functional properties of both domain and co-domain functions; these selection criterions are used with both composition and decomposition laws.

It is classical (see e.g. ref. 4) to distinguish between several types of evaluation programs.

An evaluation program is said to be a (evaluation) tree if it does not contain any reconvergent instruction.

An evaluation program is said to be a simple program if any variable \( x_i \) may be tested at most once during a computation.

The most general type of evaluation program is the non-simple program where it is allowed that a same variable may be tested several times during a given computation.

When it is not necessary to distinguish between these three types of programs, the calling evaluation program will be used.

Moreover it will always be assumed further on that the programs dealt with are irredundant programs: an evaluation program is said to be irredundant if and only if

(a) each part of the program may be reached during at least one computation;

(b) it does not contain instructions of the form

\[ N_j \text{ if } x_i = 0 \text{ then go to } N_0 \text{ else go to } N_0. \]  

The optimization of one of these three types of evaluation programs, i.e. trees, simple programs and non-simple programs is performed by means of an appropriate choice of the three optimization parameters which are: the initial \( P \)-functions, the composition law and the selection criterion. The choice that has to be made between these parameters in order to perform optimization has been written down in table I herebelow; the entries of this table will be detailed in part III of this work.

2.4. Hardware interpretation of programs

As a conclusion of this sec. 2 one presents briefly some relations that exist between the software realization (as depicted in fig. 1a) of an evaluation program and its hardware realization. Evaluation programs may be realized by means of networks made up of demultiplexers and OR-gates and by means of
networks made up of multiplexers.

### TABLE I

<table>
<thead>
<tr>
<th>Optimization criterion</th>
<th>Type of evaluation program</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tree</td>
<td>simple program</td>
<td>non-simple program</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>L–R</td>
<td>$a_3, b_1, c_1$</td>
<td>$a_2, b_1, c_1$</td>
<td>$a_2, b_2, c_1$</td>
</tr>
<tr>
<td></td>
<td>R–L</td>
<td>$a_1, b_3, c_2$</td>
<td>$a_1, b_3, c_2$</td>
<td>$a_1, b_2, c_3$</td>
</tr>
<tr>
<td>relative cost</td>
<td>L–R</td>
<td>$a_3, b_1, c'_1$</td>
<td>$a_2, b_1, c'_1$</td>
<td>$a_2, b_2, c'_1$</td>
</tr>
<tr>
<td></td>
<td>R–L</td>
<td>$a_1, b_3, c'_2$</td>
<td>$a_1, b_3, c'_2$</td>
<td>$a_1, b_2, c'_3$</td>
</tr>
<tr>
<td>cost</td>
<td>L–R</td>
<td>$a_3, b_1, c''_1$</td>
<td>$a_2, b_2, c''_1$</td>
<td>$a_2, b_2, c''_1$</td>
</tr>
<tr>
<td></td>
<td>R–L</td>
<td>$a_1, b_3, c''_2$</td>
<td>$a_1, b_3, c''_2$</td>
<td>$a_1, b_2, c''_3$</td>
</tr>
</tbody>
</table>

**Fig. 3.** Evaluation program.
In figure 3 one gives a more detailed configuration of the evaluation program of fig. 1a; in particular it is recalled that to each instruction may be associated a $P$-function. These $P$-functions have been written below the corresponding instruction in fig. 3.

A demultiplexer of one command variable $x$ is a logic gate with one input terminal $y$ and two output terminals $z_0$ and $z_1$ realizing the functions

$$z_0 = \bar{x}y \quad \text{and} \quad z_1 = x y.$$  \hfill (14)

The graphical representation of a demultiplexer is shown in the net of fig. 4.

---

**Proposition 1** (see also figs 3 and 4). The logical network deduced from the evaluation program of fig. 3 in the following way:
(a) to each instruction $M$ of the program corresponds a system: demultiplexer-OR-gate; the demultiplexer input terminal is connected to the OR-gate output terminal;
(b) the next instructions to $M$ corresponding to the values 0 and 1 of the command variable $x_i$ are the outputs of the demultiplexer respectively; the inputs to the OR-gate correspond to the instructions preceding $M$ (evidently if there is a single instruction preceding $M$, the OR-gate may be dropped and this instruction is connected to the demultiplexer input);
(c) the input of the initial demultiplexer (corresponding to the instruction $(1, f)$) is connected to the Boolean constant 1;

realizes the switching functions $[g_0, g_1, ..., g_{q-1}]$. Moreover the OR-gate corresponding to the instruction $M$ having $(g, h)$ as $P$-function realizes a switching function $g' \subseteq g$ so that the demultiplexer connected to this OR-gate realizes the switching functions $x_i g'$ and $\bar{x}_i g'$.

Proof. The proof of this proposition is quite obvious if one observes that the system: demultiplexer-OR-gate realizes the domain functions generated by the decomposition law $\Delta$ (see also fig. 2a).

A multiplexer of one command variable $x$ is a logic gate with two input terminals $y_0$ and $y_1$ and one output terminal $z$ realizing the function

$$z = \bar{x}y_0 \lor xy_1.$$  \hspace{1cm} (15)

Proposition 2 (see also figs 3 and 5). The logical network deduced from the evaluation program of fig. 3 in the following way:
(a) to each instruction $M$ of the program corresponds a multiplexer;
(b) the next instructions to $M$ corresponding to the values 0 and 1 of the command variable $x_i$ are connected to the inputs of the multiplexer; the output of the multiplexer are connected to the instructions preceding $M$;
(c) the inputs of the initial multiplexers (corresponding to the instructions whose $P$-functions are $\langle g_i, P_i \rangle$) are connected to the instruction labels $P_i$;

realizes the function $f$. Moreover the multiplexer corresponding to the instruction $M$ having $\langle g, h \rangle$ as $P$-function realizes the function $h$.

Proof. The proof immediately derives from the fact that the output functions of the $P$-functions are generated by the composition law

$$h = x_i h_0 \lor \bar{x}_i h_1$$

which corresponds to the output function (15) of the multiplexer.
3. The algebra of $P$-functions

3.1. Introductory concepts

Let $L$ be a finite non-empty set; we consider functions

$$f : [0, 1]^n \rightarrow L.$$  \hspace{1cm} (16)

One will generally assume that the set $L$ is formed by a list of instruction labels, i.e. $L = \{P_0, P_1, \ldots, P_{q-1}\}$; the function $f$ may then be represented by means of a formal Boolean expression of the form (5). One will also consider for $L$ the set of integers: $\{0, 1, \ldots, q - 1\}$ and its particular important case.
The function \( f \) reduces then to the well-known *pseudo-logic* and *Boolean functions* respectively\(^5\). One will moreover assume that the function \( f \) may be incompletely defined.

Let \( f(x) \) and \( h(x) \) be two functions of the form (16) and let \( g(x) \) be a Boolean function with \( x = (x_{n-1}, \ldots, x_1, x_0) \); the pair of functions \( \langle g, h \rangle \) will be called a *P*-function of \( f \) (and one will write \( \langle g, h \rangle \lor f \)) if and only if

\[
fg = hg.
\]

The condition \( g = 1 \) defines a domain where \( f = h \) (see sec. 2.2); the *P*-function \( \langle g, h \rangle \) constitutes a *partial description* of \( f \) in the domain characterized by \( g = 1 \). The set of \( P \)-functions

\[
\{ \langle g_i, h_i \rangle | \langle g_0, h_0 \rangle \lor f, \ i \in I \}
\]

constitutes a *total description* of \( f \) if and only if the solutions of

\[
\bigvee_{i \in I} g_i(x) = 1
\]

characterize the domain where \( f \) is defined; in particular if \( f \) is a completely defined function the condition (19) becomes

\[
\bigvee_{i \in I} g_i(x) = 1
\]

Clearly \( (1, f) \) and \( \{ \langle g_i(x), P_i \rangle \} \) constitute two total descriptions of

\[
f(x) = \bigvee_{i=0, q-1} g_i(x) P_i.
\]

For \( f(x) \) a Boolean function \( \{ \langle f, 1 \rangle, \langle \bar{f}, 0 \rangle \} \) constitutes another total description.

Further on no distinction will be made between completely and incompletely defined functions; this unified treatment may be reached by modifying slightly the definition of the Boolean function \( g_i(x) \) as follows: the solutions of the equation \( g_i(x) = 1 \) will characterize the *maximal domain* in which \( f \) may take the label \( P_i \). Since a non-specification may be viewed as the possibility for \( f \) to take several labels \( P_i \) on some sub-domains of the \( n \)-cube, the \( g_i \) do not satisfy the orthogonality conditions when \( f \) is incompletely defined.

Let us prove some elementary properties of the algebra of \( P \)-functions.

**LEMMA 1**

(a) \( \langle g, h \rangle \lor f \) and \( g' \leq g \Rightarrow \langle g', h \rangle \lor f \).

(b) \( \langle g_0, h \rangle \lor f \) and \( \langle g_1, h \rangle \lor f = \langle g_0 g_1, h \rangle \lor f \) and \( \langle g_0 \lor g_1, h \rangle \lor f \)
Implementation and transformation of algorithms based on automata

Proof.
(a) \((g, h) \vee f\) and \(g' \leq g \Rightarrow fg = hg\) and \(gg' = g' = fg' = hgg'\) and \(fg' = hg' = f(g', h) \vee f.\)
(b) \((g_0, h) \vee f\) and \((g_1, h) \vee f = f(g_0, h) = hg_1 = f(g_0g_1) = hh(g_0g_1) = \langle g_0g_1, h \rangle \vee f\) and \((g_0 \vee g_1, h) \vee f.\)

From the preceding lemma one deduces that the set
\[ \{g | (g, h) \vee f\} \]
is closed for the operations of conjunction and of disjunction. It constitutes thus a sublattice of the lattice of Boolean functions and has consequently a maximum element denoted \([f, h]\). The relation
\[ [f, h] = 1 \]
is thus satisfied on each vertex where \(f = h\) and on these vertices only. Let us denote by \(f \oplus h\) a Boolean function equal to 0 on the vertices where \(f = h\) and equal to 1 otherwise. One has thus
\[ [f, h] = f \oplus h. \] (22)

The theorems that will be proven below are valid for formal Boolean expressions of the form (21). The proof will, however, be given for Boolean functions since the form of \([f, h]\) in this last case allows us to use the usual Boolean formalism. For example, one proves that
\[ \langle [f, h], h \rangle \vee f \]
by observing that
\[ f(f \oplus h) = fh = h(f \oplus h) \]

3.2. Theorems on composition laws

The formalism of proof for the theorems below and for \(f\) a formal Boolean expression is left to the reader.

THEOREM 1 (Theorem on the composition law \(T^0\)).
\[ \langle g_0, h_0 \rangle \vee f \quad \text{and} \quad \langle g_1, h_1 \rangle \vee f = \langle g_0, h_0 \rangle T^0_\chi \langle g_1, h_1 \rangle = \langle g_0 \chi \vee g_1 x, h_0 \chi \vee h_1 x \rangle \vee f. \]
Moreover \(g_0 = [f, h_0]\) and \(g_1 = [f, h_1] = \bar{x}g_0 \vee xg_1 = [f, \bar{x}h_0 \vee xh_1].\)

Proof. One has immediately
\[ f(\bar{x}g_0 \vee xg_1) = \bar{x}f\bar{g}_0 \vee xf\bar{g}_1 \quad \text{e.g.} \quad (\bar{x}h_0 \vee xh_1)(\bar{x}g_0 \vee xg_1) = \bar{x}h_0g_0 \vee xh_1g_1 \]
The first part of the proposition follows then from the relations

\[ f_{g_0} = h_0 g_0 \] and \[ f g_1 = h_1 g_1. \]

Let us now prove the preservation of the maximality character of the domain function \( x g_0 \lor x g_1 \). One has successively

\[
\begin{align*}
\overline{f} \oplus (\overline{x h_0} \lor x h_1) &= \overline{f} \oplus (\overline{x h_0} \oplus x h_1) \\
&= (\overline{f} \oplus x f) \oplus (\overline{x h_0} \oplus x h_1) \\
&= \overline{x (f \oplus h_0) \lor x (f \oplus h_1)} = \overline{x g_0 \lor x g_1}
\end{align*}
\]

\[ \square \]

**THEOREM 2.** (Theorem on the composition law \( T^1 \).)

\[ \langle g_0, h_0 \rangle \lor f \] and \[ \langle g_1, h_1 \rangle \lor f \]

\[ \langle g_0, h_0 \rangle T^1_1 \langle g_1, h_1 \rangle = \langle g_0(x = 0) g_1(x = 1), h_0 \overline{x} \lor h_1 x \rangle \lor f. \]

Moreover, if \( g_0 = [f, h_0] \) and \( g_1 = [f, h_1] \), \( g_0(x = 0) g_1(x = 1) \) is the greatest function independent of \( x \) and contained in \( [f, (\overline{x h_0} \lor x h_1)] \).

**Proof.** Taking into account the fact that

\[ \overline{x g_0(x)} \lor x g_1(x) = \overline{x g_0(0)} \lor x g_1(1) \]

and applying theorem 1, one obtains

\[ \langle \overline{x g_0(0)} \lor x g_1(1), \overline{x h_0} \lor x h_1 \rangle \lor f. \]

The property then results from the fact that

\[ g_0(0) g_1(1) \leq \overline{x g_0(0)} \lor x g_1(1) \]

and from the application of lemma 1(a). Moreover one knows that if \( g_0 \) and \( g_1 \) are maximal functions \( \overline{x g_0} \lor x g_1 \) is also maximal domain function. Since \( g_0(x = 0) g_1(x = 1) \) is the meet difference (see ref. 5) of this last function, it is the greatest function degenerated in \( x \) and satisfying the inequality (23). \[ \square \]

Theorems 1 and 2 presented properties of two laws that were called composition laws. Recall (see sec. 2.2) that composition laws were introduced in order to obtain the total description \( \langle 1, f \rangle \) of a program given by its total description \( \langle g, P_f \rangle \). Let us define the composition laws in a more rigorous way.

A composition law \( T^1 \) is a law acting on pairs of P-functions and satisfying the following properties.

(a) To the instruction labels \( M_0, M_1 \) and \( M \) of

\[ M \] if \( x = 0 \) then go to \( M_0 \) else go to \( M_1 \)

one associates the P-functions \( \langle g_0, h_0 \rangle, \langle g_1, h_1 \rangle \) and \( \langle g_0, h \rangle \) respectively; then
Implementation and transformation of algorithms based on automata

\[ \langle g_0, h_0 \rangle T^l_x \langle g_1, h_1 \rangle = \langle g_l(g_0, g_1, x), h_0 \bar{x} \lor h_1 x \rangle, \]
\[ = \langle g_l, h \rangle. \quad (24) \]

(b) An iterated use of the law \( T^l_x \), \( x_i \in x \), acting on \( \langle g_{jk}, P_j \rangle \) with \( g_{jk} \), Boolean functions satisfying

\[ \lor_k g_{jk} = g_j, \quad j = 0, 1, \ldots, q - 1 \]

produces in at least one way a \( P \)-function having 1 as domain function.

**Comments**

(a) A composition law produces a unique co-domain function i.e. \( h_0 \bar{x} \lor h_1 x \) since this function is uniquely determined by the interconnection of instructions as shown by the fig. 2b; a composition law may produce several domain functions since the function \( f \) may reduce to \( h_0 \bar{x} \lor h_1 x \) in several subdomains of the \( n \)-cube. One requests that the law \( T^l_x \) must be able to produce a domain function equal to 1 since a total description of \( f \) implies that the complete \( n \)-cube must be covered by a unique domain function.

(b) If \( f \) is a completely defined function, to the domain function 1 is necessarily associated the co-domain function \( f \), i.e. the final \( P \)-function obtained is \( \langle 1, f \rangle \). If \( f \) is incompletely defined, the final \( P \)-functions are of the form \( \langle 1, f_j \rangle \) where the \( f_j \) are completely defined functions compatible with \( f \).

The above definition of composition law \( T^l_x \) allows us to state the following general theorem on composition laws.

**THEOREM 3** (Theorem on composition law \( T^l_x \).) A composition law \( T^l_x \) associates to a pair of \( P \)-functions \( \langle g_0, h_0 \rangle \) and \( \langle g_1, h_1 \rangle \) a \( P \)-function \( \langle g_l, h \rangle \) with

\[ \langle g_l, h \rangle = \langle g_0(x = 0) g_1(x = 1), g_l \leq \bar{x}g_0 \lor xg_1, \bar{x}h_0 \lor xh_1 \rangle. \quad (26) \]

**Proof.** The domain function \( g_l \) must be smaller than its maximal element, i.e. \( \bar{x}g_0 \lor xg_1 \) (see theorem 1); moreover, any composition law should be able to produce a domain function equal to 1. The law \( T^l_x \) must be such that

\[ \langle \bar{x}a_0 \lor x\beta_0 \lor y_0, h_0 \rangle T^l_x \langle \bar{x}a_1 \lor x\beta_1 \lor y_1, h_1 \rangle = \langle 1, \bar{x}h_0 \lor xh_1 \rangle \]

if and only if \( a_0 \beta_1 = 1 \). One verifies that the smallest composition law acting on \( x \) and which effectively produces the term \( a_0 \beta_1 \) is \( T^l_x \).

3.3. **Theorem on decomposition and on merging laws**

Recall (see sec. 2.2) that a decomposition law was introduced in order to obtain the total description \( \{ \langle g_l, P_j \rangle \} \) of a program given by its total descrip-
tion \(1, f\). Hence a decomposition law \(\Delta_x\) is a law acting on the set of \(P\)-functions and satisfying the following properties.

(a) To the instruction labels \(M_0, M_1\) and \(M\) of

\[
\begin{align*}
M &\text{ if } x = 0 \text{ then go to } M_0 \text{ else go to } M_1
\end{align*}
\]

one associates the \(P\)-functions \(\langle g_0^l, h_0 \rangle, \langle g_1^l, h_1 \rangle\) and \(\langle g, h \rangle\) respectively; then

\[
\Delta_x\langle g, h \rangle = \{\langle g_0^l, h(x = 0) \rangle, \langle g_1^l, h(x = 1) \rangle\} = \{\langle g_0^l, h_0 \rangle, \langle g_1^l, h_1 \rangle\}.
\]

(b) An iterated use of the law \(\Delta_x\), \(x_i \in x\), acting on \(1, f\) produces in at least one way the \(P\)-functions \(\{\langle g_{jk}, P_j \rangle\}\) with \(g_{jk}\) Boolean functions satisfying the relations (25).

**THEOREM 4.** (Theorem on decomposition law \(\Delta^l\)). A decomposition law \(\Delta_x^l\) associates to a \(P\)-function \(\langle g, h \rangle\) the pair of \(P\)-functions \(\langle g_0^l, h_0 \rangle, \langle g_1^l, h_1 \rangle\) with

\[
\Delta_x^l\langle g, h \rangle = \{\langle xg \leq g_0^l \leq f \oplus h(x = 0), h(x = 0) \rangle, \langle xg \leq g_1^l \leq f \oplus h(x = 1), h(x = 1) \rangle\}
\]

(28)

**Proof.** The proof derives immediately from theorem 1. □

Denote by \(\Delta_x\) the law

\[
\Delta_x\langle g, h \rangle = \{\langle xg, h(x = 0) \rangle, \langle xg, h(x = 1) \rangle\};
\]

(29)

it is the only decomposition law that (presently) seems of practical use.

Let us now introduce the concept of merging of instructions. In an evaluation program \(N\), two instructions \(M_a\) and \(M_b\) are said to be mergeable in a single instruction \(M\) if and only if the two following subprograms of \(N\) are equivalent.

\[
\begin{align*}
N, \ldots, M_a, M_b, \ldots, \\
M_a &\text{ if } x = 0 \text{ then go to } N_a \text{ else go to } N_a' \\
M_b &\text{ if } x = 0 \text{ then go to } N_b \text{ else go to } N_b'
\end{align*}
\]

\[
\begin{align*}
N, \ldots, M, M, \ldots, \\
M &\text{ if } x = 0 \text{ then go to } P \text{ else go to } Q
\end{align*}
\]

Let us now show how the merging of the instruction \(M_a\) and \(M_b\) into the single instructions \(M\) may be interpreted in terms of transformations of their respective \(P\)-functions.

Let
Implementation and transformation of algorithms based on automata

\[ \langle g_a, h_a \rangle \] the P-function associated with \( M_a \),

\[ \langle g_b, h_b \rangle \] the P-function associated with \( M_b \),

\[ \langle g, h \rangle \] the P-function associated with \( M \).

Since instructions \( M_a \) and \( M_b \) cover the sub-domains of the \( n \)-cube characterized by the equations \( g_a = 1 \) and \( g_b = 1 \) respectively, the P-function of the instruction \( M \) has a domain function

\[ g = g_a \lor g_b. \]  \hspace{1cm} (30)

Since the instructions \( M_a \) and \( M_b \) may be considered as final instructions of sub-programs computing \( h_a \) and \( h_b \) in \( g_a \) and \( g_b \) respectively, a condition for \( M_a \) and \( M_b \) to be mergeable is that \( h_a \geq h_b \) in the domain characterized by \( g_a = 1 \) and that \( h_b \geq h_a \) in the domain characterized by \( g_b = 1 \). Since \( h \) is moreover assumed to be a co-domain function of \( f \), the following constraints must be satisfied by the domain and co-domain functions.

\[ h_a g_a \geq h_b g_a, \quad h_b g_b \geq h_a g_b, \]  \hspace{1cm} (31)

\[ f(g_a \lor g_b) = h(g_a \lor g_b). \]  \hspace{1cm} (32)

Since \( f g_a = h_a g_a \) and \( f g_b = h_b g_b \), and in view of the inequalities (31), the relation (32) may successively be written

\[ h(g_a \lor g_b) = h_a g_a \lor h_b g_b, \]

\[ = h_a g_a \lor h_b g_b \lor h_a g_b \lor h_b g_a, \]

\[ = (h_a \lor h_b)(g_a \lor g_b). \]  \hspace{1cm} (33)

It follows that \( h_a \lor h_b \) may be taken as co-domain function for the instruction \( M \). The above considerations on merging of instructions may be gathered in the following theorem.

THEOREM 5 (Theorem on merging law.) Two instructions \( M_a \) and \( M_b \) with \( x \) as condition variables and to which are associated the P-functions \( \langle g_a, h_a \rangle \) and \( \langle g_b, h_b \rangle \) respectively may be merged into a single instruction \( M \) with \( x \) as condition variable and \( \langle g_a \lor g_b, h_a \lor h_b \rangle \) as P-function if and only if

\[ h_a g_a \geq h_b g_a \quad \text{and} \quad h_b g_b \geq h_a g_b. \]

One will then write: \( M = M_0 \cup M_1 \).

4. Synthesis of evaluation programs: examples

This section will entirely be devoted to the application of the algebra of P-functions to some examples of synthesis of evaluation programs and of Boolean functions (it is recalled that the synthesis of a Boolean function \( f \) is
A. Thayse

identical to the synthesis of an evaluation program $N$ with two output labels $P_0$ and $P_1$, i.e. $N = \overline{fP_0} \lor fP_1$.

For sake of conciseness, a conditional instruction

$$N \text{ if } x = 0 \text{ go to } N_0 \text{ else go to } N_1,$$

will be written

$$NxN_0N_1.$$  \hspace{1cm} (31)

The composition laws $T^i_{xj}$ and $\Delta_{xj}$ will be denoted $T_i$ and $\Delta_j$ respectively; the $P$-functions will be labelled by the labelling of the instructions to which they are associated respectively. If a $P$-function $C$ may be obtained in several ways by composing e.g. different $P$-functions $A_j$ and $B_j$ with respect to different condition variables, one will write

$$P = A_j T^i_j B_j = A_k T^i_k B_k \ldots \text{ etc.}$$  \hspace{1cm} (35)

This means that the instruction $P$ may be realized in several ways by choosing different condition variables and different next instructions respectively. Clearly, in the process of building a program, when a relation like (35) is encountered, one has to choose one of the possibilities (e.g. $A_j T^i_j B_j$) for realizing $P$; the choice of a given relation corresponds to the choice of a particular program $P$ between several possibilities.

**Example** 1. Consider the Boolean function $f$, described either by its truth table II or by its formal expression (36) herebelow.

<table>
<thead>
<tr>
<th>TABLE II</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f = x_0 \lor x_2(x_1 \lor \overline{x}_3)$.</td>
</tr>
</tbody>
</table>

Consider the following total description of $f$:

$$A_0 = \langle \overline{x}_0(\overline{x}_2 \lor \overline{x}_1 x_3), 0 \rangle, \quad A_1 = \langle x_0 \lor x_2(x_1 \lor \overline{x}_3), 1 \rangle.$$
An iterative application of the composition law $T^i_i$ (it is recalled that the index “$i$” is the index of the variable $x_i$ on which the law $T^1$ acts) allows us to obtain the following $P$-functions.

$B_0 = A_0 T^1_1 A_1 = \langle \bar{x}_2 \lor \bar{x}_1, x_2, x_0 \rangle$

$B_1 = A_0 T^1_2 A_1 = \langle \bar{x}_0 \lor \bar{x}_2, x_2 \rangle$

$B_2 = A_0 T^1_1 A_1 = \langle \bar{x}_0 x_2 x_3, x_1 \rangle$

$B_3 = A_1 T^1_1 A_0 = \langle \bar{x}_0 \lor x_1, x_2, \bar{x}_3 \rangle$

$C_0 = B_0 T^1_2 A_1 = \langle x_0 \lor x_1 \lor x_3, x_0 \lor x_2 \rangle$

$C_1 = B_1 T^1_3 A_0 = \langle \bar{x}_0 \lor \bar{x}_1, \bar{x}_3 x_2 \rangle$

$C_2 = B_0 T^1_1 A_1 = \langle \bar{x}_0 x_0 \lor x_0 x_3 \lor x_2 x_3, x_0 \lor x_1 \rangle$

$C_3 = A_1 T^1_3 B_2 = B_3 T^1_1 A_1 = B_1 T^1_3 B_2 = B_0 T^1_1 B_1 = \langle \bar{x}_0 x_2, \bar{x}_3 \lor x_1 \rangle$

$C_4 = A_1 T^1_3 B_0 = \langle x_0 x_1 \lor \bar{x}_3, \bar{x}_2 x_1, \bar{x}_3 \lor x_0 \rangle$

$C_5 = B_0 T^1_1 B_1 = \langle \bar{x}_2 \lor x_3, x_0 \lor \bar{x}_3 \lor x_2 \rangle$

$D_0 = B_0 T^1_2 C_0 = C_5 T^1_0 A_1 = C_0 T^1_1 C_2 = \langle \bar{x}_2 \lor x_3, x_0 \lor x_1 \rangle$

$D_1 = A_1 T^1_3 C_2 = C_4 T^1_1 C_0 = C_4 T^1_1 A_1 = \langle x_0 \lor x_2 \lor x_0 \lor x_1 \lor \bar{x}_3 \rangle$

$D_2 = C_0 T^1_1 B_0 = C_1 T^1_3 A_1 = C_1 T^1_3 C_4 = B_0 T^1_2 C_4 = \langle \bar{x}_2 \lor \bar{x}_1, x_0 \lor \bar{x}_3 \lor x_2 \rangle$

$D_3 = C_1 T^1_3 B_1 = A_0 T^1_2 C_3 = C_1 T^1_0 C_0 = B_1 T^1_2 C_5 = \langle x_0, x_3 (\bar{x}_3 \lor x_1) \rangle$

$E = D_3 T^1_0 A_1 = D_2 T^1_1 C_0 = B_0 T^1_2 D_1 = C_0 T^1_3 D_0 = \langle 1, f \rangle$.

**TABLE III**

<table>
<thead>
<tr>
<th>program (a)</th>
<th>program (b)</th>
<th>program (c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E \ x_0 \ D_3 \ A_1$</td>
<td>$E \ x_0 \ D_3 \ A_1$</td>
<td>$E \ x_1 \ D_2 \ C_0$</td>
</tr>
<tr>
<td>$D_3 \ x_1 \ B_1 \ C_1$</td>
<td>$D_3 \ x_2 \ C_3 \ A_0$</td>
<td>$D_2 \ x_3 \ C_0 \ B_0$</td>
</tr>
<tr>
<td>$C_1 \ x_3 \ B_1 \ A_0$</td>
<td>$C_3 \ x_1 \ A_1 \ B_3$</td>
<td>$C_0 \ x_2 \ B_0 \ A_1$</td>
</tr>
<tr>
<td>$B_1 \ x_2 \ A_0 \ A_1$</td>
<td>$B_3 \ x_3 \ A_1 \ A_0$</td>
<td>$B_0 \ x_0 \ A_0 \ A_1$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>program (d)</th>
<th>program (e)</th>
<th>program (f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E \ x_2 \ B_0 \ D_1$</td>
<td>$E \ x_3 \ C_0 \ D_0$</td>
<td>$E \ x_0 \ D_3 \ A_1$</td>
</tr>
<tr>
<td>$D_1 \ x_3 \ A_1 \ C_2$</td>
<td>$D_0 \ x_1 \ B_0 \ C_0$</td>
<td>$D_3 \ x_2 \ C_3 \ A_0$</td>
</tr>
<tr>
<td>$C_2 \ x_1 \ B_0 \ A_1$</td>
<td>$C_0 \ x_2 \ B_0 \ A_1$</td>
<td>$C_3 \ x_3 \ A_1 \ B_2$</td>
</tr>
<tr>
<td>$B_0 \ x_0 \ A_0 \ A_1$</td>
<td>$B_0 \ x_0 \ A_0 \ A_1$</td>
<td>$B_2 \ x_1 \ A_0 \ A_1$</td>
</tr>
</tbody>
</table>
Some programs that can be deduced from the above $P$-functions are written down in table III (programs (a)-(f)); the corresponding flow-charts of these programs are given in fig. 6.

Example 2. Consider the function

\[ f: \{0, 1\}^4 \rightarrow \{a, b, c, d, e, g\} \]

given by means of its truth table IV.
The following set of P-functions is a total description of $f$.

$$
A_0 = \langle \bar{x}_1 \bar{x}_2, a \rangle, \quad A_1 = \langle x_0 x_2, b \rangle, \quad A_2 = \langle \bar{x}_0 x_3 (x_1 \lor x_2), e \rangle
$$

$$
A_3 = \langle x_1 \bar{x}_0 (\bar{x}_0 \lor \bar{x}_2), d \rangle, \quad A_4 = \langle \bar{x}_0 \bar{x}_1 \bar{x}_3, c \rangle, \quad A_5 = \langle x_0 x_1 \bar{x}_2 x_3, g \rangle.
$$

By using the composition law $T^1$, one obtains successively

$$
B_0 = A_4 \, T_1^1 \, A_3 = \langle \bar{x}_0 x_2 \bar{x}_3, c \bar{x}_1 \lor dx_1 \rangle
$$

$$
B_1 = A_2 \, T_0^1 \, A_5 = \langle x_1 \bar{x}_2 x_3, e \bar{x}_0 \lor gx_0 \rangle
$$

$$
C_0 = B_0 \, T_3^1 \, A_2 = \langle \bar{x}_0 x_2, ex_3 \lor \bar{x}_3 (c \bar{x}_1 \lor dx_1) \rangle
$$

$$
C_1 = A_3 \, T_3^1 \, B_1 = \langle x_1 \bar{x}_2, x_3 (e \bar{x}_0 \lor gx_0) \lor \bar{x}_3 d \rangle
$$

$$
D_0 = C_0 \, T_0^1 \, A_1 = \langle x_2, \bar{x}_1 [ex_3 \lor \bar{x}_3 (c \bar{x}_1 \lor dx_1)] \lor x_0 b \rangle
$$

$$
D_1 = A_0 \, T_1^1 \, C_1 = \langle \bar{x}_2, \bar{x}_1 a \lor x_1 [x_3 (e \bar{x}_0 \lor gx_0) \lor \bar{x}_3 d] \rangle
$$

$$
E = D_1 \, T_1^1 \, D_0 = \langle 1, f \rangle.
$$

By starting from the same set of functions $A_i$ and by using the composition law $T^0$ one obtains successively

$$
B'_0 = A_4 \, T_1^1 \, A_3 = \langle \bar{x}_3 (x_1 \bar{x}_0 \lor x_1 \bar{x}_2 \lor \bar{x}_0 x_2), c \bar{x}_1 \lor dx_1 \rangle
$$

$$
B'_1 = A_2 \, T_0^1 \, A_5 = \langle x_3 (x_1 \bar{x}_0 \lor x_1 \bar{x}_2 \lor \bar{x}_0 x_2), e \bar{x}_0 \lor gx_0 \rangle
$$

$$
C' = B'_0 \, T_3^1 \, B'_1 = \langle x_1 \bar{x}_0 \lor x_1 \bar{x}_2 \lor \bar{x}_0 x_2, \bar{x}_3 (c \bar{x}_1 \lor dx_1) \lor x_3 (e \bar{x}_0 \lor gx_0) \rangle
$$

$$
D'_0 = C' \, T_0^1 \, A_1 = \langle x_2 \lor \bar{x}_0 x_1, \bar{x}_0 [x_3 (c \bar{x}_1 \lor dx_1) \lor x_3 e] \lor bx_0 \rangle
$$

$$
D'_1 = A_0 \, T_1^1 \, C' = \langle \bar{x}_2 \lor \bar{x}_0 x_1, x_1 [x_3 d \lor x_3 (e \bar{x}_0 \lor gx_0)] \lor a \bar{x}_1 \rangle
$$

$$
E' = D'_1 \, T_1^1 \, D'_0 = \langle 1, f \rangle.
$$
A. Thayse

The programs resulting from the use of the laws $T^1$ and $T^0$ are written down in tables V (a) and (b) respectively; their corresponding flowchart is given in fig. 7a and 7b respectively. Observe that (a) is a simple program while (b) is a non-simple program.

| TABLE V |
|---|---|
| **program (a)** | **program (b)** |
| $E$ | $E$ |
| $x_2$ | $x_2$ |
| $D_1$ | $D_1'$ |
| $D_0$ | $D_0'$ |
| $x_0$ | $x_0$ |
| $C_0$ | $C$ |
| $A_1$ | $A_0$ |
| $D_1'$ | $x_1$ |
| $A_0$ | $B_0'$ |
| $C_1$ | $C$ |
| $A_3$ | $A_4$ |
| $B_1$ | $A_3$ |
| $B_0$ | $B_1'$ |
| $x_1$ | $x_0$ |
| $A_4$ | $A_2$ |
| $A_3$ | |
| $B_1$ | $A_6$ |

It is interesting to note that, by putting $c = g = 1$ and $a = b = d = e = 0$ in table IV, one obtains the function

$$f = \bar{x}_3 x_2 \bar{x}_1 \bar{x}_0 \lor x_3 \bar{x}_2 x_1 x_0,$$

which is the most elementary Boolean function having a non-simple program as optimal program; the flow-chart of this optimal non-simple program is given in fig. 8.

The program of fig. 8 was obtained by using the composition law $T^0$ and the initial conditions $\langle f, 0 \rangle$ and $\langle f, 1 \rangle$; it may also be obtained by using the decomposition law $\Delta$ together with the merging law $\cup$ (of theorem 5) and the initial conditions $\langle 1, f \rangle$.

$$E = \langle 1, \bar{x}_3 x_2 x_1 x_0 \lor x_3 \bar{x}_2 x_1 x_0 \rangle$$

$$\Delta_2 (E) = \{\langle \bar{x}_2, x_3 x_1 x_0 \rangle, \langle x_2, \bar{x}_3 \bar{x}_1 \bar{x}_0 \rangle\} = \{D_1', D_0'\}$$

$$\Delta_1 (D_1') = \{\langle \bar{x}_2, \bar{x}_1, 0 \rangle, \langle \bar{x}_2 x_1, x_3 x_0 \rangle\} = \{A_1^0, C_1\}$$

$$\Delta_0 (D_0') = \{\langle x_2, x_0, \bar{x}_3 \bar{x}_1 \rangle, \langle x_2 x_0, 0 \rangle\} = \{C_0, A_3^0\}$$

$$C_1 \cup C_0 = \{\langle x_2 \bar{x}_0 \lor \bar{x}_2 x_1, \bar{x}_3 \bar{x}_1 \lor \bar{x}_3 x_0 \rangle\} = C'$$

$$\Delta_3 (C') = \{\langle \bar{x}_3 (x_2 x_0 \lor \bar{x}_2 x_1), \bar{x}_1 \rangle, \langle x_3 (x_2 x_0 \lor \bar{x}_2 x_1), x_0 \rangle\} = \{B_0, B_1'\}$$

$$\Delta_1 (B_0) = \{\langle \bar{x}_3 \bar{x}_1 (x_2 \bar{x}_0 \lor \bar{x}_2 x_1), 1 \rangle, \langle x_1 \bar{x}_3 (x_2 \bar{x}_0 \lor \bar{x}_2 x_1), 0 \rangle\} = \{A_1, A_8^0\}$$

$$\Delta_0 (B_1') = \{\langle \bar{x}_0 x_3 (x_2 \bar{x}_1 \lor \bar{x}_2 x_1), 0 \rangle, \langle x_0 x_3 (x_2 \bar{x}_0 \lor \bar{x}_2 x_1), 1 \rangle\} = \{A_1^2, A_6^0\}$$
Implementation and transformation of algorithms based on automata

\[ A_0 = A_0^0 \cup A_0^2 \cup A_0^3 \cup A_0^4 = \langle \bar{f}, 0 \rangle \]
\[ A_1 = A_1^1 \cup A_1^2 = \langle f, 1 \rangle. \]

(a)

(b)

Fig. 7. Programs for example 2. (a) Simple program. (b) Non-simple program.

Example 3. Consider the incompletely defined function given by means of its truth table VI.

The following set of \(P\)-functions is a total description of \(f\).

\[ A_0 = \langle \bar{x}_0 \lor \bar{x}_1 \bar{x}_2, 0 \rangle, \quad A_1 = \langle x_1 \lor x_0 x_2, 1 \rangle. \]

By using the composition law \(T^1\) one obtains successively

\[ B_0 = A_0 T_0^1 A_1 = \langle x_1 \lor x_2, x_0 \rangle \]
\[ B_1 = A_0 T_1^1 A_1 = \langle \bar{x}_0 \lor \bar{x}_2, x_1 \rangle \]
\[ B_2 = A_0 T_2^1 A_1 = \langle x_1 \bar{x}_0 \lor \bar{x}_1 x_0, x_2 \rangle \]
A. Thayse

\[ C_0 = B_1 T_2^1 B_0 = \langle 1, x_0 x_2 \lor x_1 \bar{x}_2 \rangle \]
\[ C_1 = A_0 T_0^1 B_2 = \langle \bar{x}_1, x_2 x_0 \rangle \]
\[ C_2 = B_2 T_1^1 A_1 = \langle x_0, x_2 \lor x_1 \rangle \]
\[ D_0 = A_0 T_0^1 C_2 = \langle 1, x_0 (x_1 \lor x_2) \rangle \]
\[ D_1 = C_1 T_1^1 A_1 = \langle 1, x_2 x_0 \lor x_1 \rangle \].

**TABLE VI**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>×</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Fig. 8. Program for realizing a Boolean function.**

From the set of \(P\)-functions one deduces three programs: \(C_0\), \(D_0\) and \(D_1\) which compute three Boolean functions compatible with the entries of table VI, i.e.: \(x_0 x_2 \lor x_1 \bar{x}_2\), \(x_0 (x_1 \lor x_2)\) and \(x_1 \lor x_2 x_0\) respectively. These programs are written down in table VII and are depicted by their corresponding flow-charts of fig. 9.

These programs may also be obtained by applying the decomposition law \(\Delta_x\) to the total description of \(f\)
One obtains successively

\[ \Delta_0 (A) = \{ \langle x_0, P_0 \vee x_1 P_1 \rangle, \langle x_0, x_1 x_2 P_0 \vee (x_1 \vee x_2) P_1 \rangle \} = \{ B_0, B'_1 \} \]

\[ \Delta_1 (A) = \{ \langle x_1, (x_0 \vee x_2) P_0 \vee x_0 x_2 P_1 \rangle, \langle x_1, x_0 P_1 \rangle \} = \{ B'_2, B'_3 \} \]

\[ \Delta_2 (A) = \{ \langle x_2, (x_0 \vee x_1) P_0 \vee x_1 P_1 \rangle, \langle x_2, x_0 P_0 \vee x_0 P_1 \rangle \} = \{ B'_4, B'_5 \} \]

\[ \Delta_2 (B'_1) = \{ \langle x_2 x_0, x_1 P_0 \vee x_1 P_1 \rangle, \langle x_2 x_0, P_1 \rangle \} = \{ C'_0, C'_1 \} \]

\[ \Delta_2 (B'_2) = \{ \langle x_2, x_1, x_0 P_0 \vee x_0 P_1 \rangle \} = \{ C'_2, C'_3 \} \]

\[ \Delta_2 (B'_3) = \{ \langle x_2 x_0, x_1, x_0 P_0 \vee P_1 \rangle \} = \{ C'_4, C'_5 \} \]

\[ \Delta_2 (B'_4) = \{ \langle x_2 x_0, x_0, x_0 P_0 \vee x_1 P_1 \rangle \} = \{ C'_6, C'_7 \} \]

\[ \Delta_2 (B'_5) = \{ \langle x_2 x_1 x_0, P_0 \rangle, \langle x_2 x_1 x_0, P_1 \rangle \} = \{ D'_0, D'_1 \} \]

\[ \Delta_2 (B'_6) = \{ \langle x_2 x_1 x_0, x_0 P_0 \rangle, \langle x_2 x_1 x_0, P_1 \rangle \} = \{ D'_2, D'_3 \} \].

**TABLE VII**

<table>
<thead>
<tr>
<th>program (a)</th>
<th>program (b)</th>
<th>program (c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_0 x_2 B_1 B_0</td>
<td>D_0 x_0 A_0 C_2</td>
<td>D_1 x_1 C_1 A_1</td>
</tr>
<tr>
<td>B_1 x_1 A_0 A_1</td>
<td>C_2 x_1 B_2 A_1</td>
<td>C_1 x_0 A_0 B_2</td>
</tr>
<tr>
<td>B_0 x_0 A_0 A_1</td>
<td>B_2 x_2 A_0 A_1</td>
<td>B_2 x_2 A_0 A_1</td>
</tr>
</tbody>
</table>

**Fig. 9. Example of program for an incompletely specified function.**

**5. Conclusions**

The problems related to the synthesis and optimization of evaluation pro-
grams have been stated in the introductory sec. 2. In this section one stated that synthesis and optimization might be viewed as functions of three parameters, namely the initial conditions of the problem, the choice of a composition law and of a selection criterion. The basic mathematical tool for transformation of conditional functions, namely the algebra of $P$-functions, has been introduced in sec. 2 and thoroughly studied in sec. 3. One showed in sec. 4, by dealing with some illustrative examples, how the choice of both initial conditions and composition laws led to the synthesis of evaluation programs. The choice of selection criteria in order to optimize programs will be the subject of part III of this work.

Acknowledgements

I am much indebted to Dr M. Davio for fruitful discussions and for a significant simplification in the presentation of the algebra of $P$-functions. I thanks also Prof. D. Mange and E. Sanchez, both at the university of Lausanne, for their interesting comments.

Philips Research Laboratory

References