On a minimization algorithm for Boolean functions

By F. MÓRICZ

1. Logical design of circuits with a single output, using solid-state integrated circuits, as primitive elements, leads to several non-traditional optimization problems which require to find, for any given Boolean function, a formula (or all formulas), composed from fixed Boolean functions as primitive elements, representing the given function and minimal with respect to a given objective function.

In this note our purpose is to present an algorithm which (provided the objective function satisfies a simple restriction) obviously leads to the exact solution of the problem and, moreover, a limited number of its steps which can be implemented on a digital computer delivers a fairly good approximative solution. (Of course, the approximation will be the better the more steps are performed and thus a larger computer may provide a better approximation.)

In view of the general nature of the problem, the algorithm will be formulated here in a very general and comprehensive way which, for each practical application, must be specified in accordance with the given particular primitive elements and objective function.

2. Assume $\Theta = \{\vartheta_1, \vartheta_2, \ldots\}$ is a functionally complete system[1] of a finite number of Boolean functions, or in other words, of logical operations in a general sense where the number of operands of each operation, i.e. the number of arguments of each function ϑ_i, can be arbitrary. Let $X = \{x_1, x_2, \ldots\}$ be the (countable) set of the available Boolean variables. (In an actual realization of the algorithm to be formulated we have, of course, to limit ourselves to a finite set of variables.)

The *formulas* considered here are all those composed of the constants 0 (falsity) and 1 (truth) and of the given Boolean variables by means of operations belonging to Θ.[2] We say that two formulas, F and G, are *identically equal* or that the equality $F = G$ is an *identity* if for every valuation their values coincide; here by *valuation* we mean a mapping which makes correspond to each of the variables belonging to X one of the constans 0 and 1.

A *substitution instance* of a formula F is, by definition, any formula that can be obtained by replacing all occurrences in F of some variables, say $x_{i_1}, \ldots, x_{i_r}$ ($i_1, \ldots, i_r$ are different positive integers), by an equal number of formulas, say

[1] See, e.g., A. Ádám [1], Chapter 4.

[2] In other words, the following symbol strings are called formulas: (i) 0 and 1; (ii) any element of X; (iii) $\vartheta(F_1, \ldots F_r)$ where $\vartheta \in \Theta$ is a Boolean function with r variables and $F_1, \ldots, F_r$ are formulas; (iv) nothing else.

$H_1, \ldots, H_r$, respectively; for the formula thus obtained we introduce the notation:

$$F(x_{i_1} := H_1; \ldots; x_{i_r} := H_r).$$

Here the variables $x_{i_1}, \ldots, x_{i_r}$ and formulas $H_1, \ldots, H_r$ are called *substituends* and *substituents*, respectively. Analogously, by a substitution instance of an identity $F=G$ we mean any equality of the form

$$F(x_{i_1} := H_1; \ldots; x_{i_r} := H_r) = G(x_{i_1} := H_1; \ldots; x_{i_r} := H_r),$$

which is easily shown to be also an identity.

Consider now a formula F and an identity $G=H$ the left-hand side G of which is a (proper or non-proper) subformula of F. For each occurrence of G in F decide independently whether it is to be left unchanged or replaced by H and proceed accordingly. Any of the formulas that can be obtained in this way are said to arise from F by a *direct application* of the identity $G=H$; the number of such formulas is 2^s, where s denotes the number of occurrences of G in F. If the formula F is left untouched we speak of a *trivial* direct application. If G is not a subformula of F then the only possible direct application is the trivial one.

The formula F' is said to be obtained from F by an *application* of the identity $G=H$ if F' arises from F by a direct application of some substitution instance of the identity $G=H$. Analogously as above, calling an application *trivial* if it leaves F untouched, in case G has no substitution instance that is a subformula of F the only possible application of the identity $G=H$ is the trivial one.

We note that the minimization problem mentioned above has been studied in detail so far mainly in the case of the classical propositional calculus, i.e. when $\Theta=\{\vartheta_1, \vartheta_2, \vartheta_3\}$, where $\vartheta_1(x_1, x_2) = x_1 \wedge x_2$, $\vartheta_2(x_1, x_2) = x_1 \vee x_2$ and $\vartheta_3(x_1) = \bar{x}_1$ (negation). For practical applications also important is the case when consists of Sheffer's alternative or Peirce's joint denial only[3], or of some of their generalizations for several variables, known as NAND and NOR elements.

3. Let $c(F)$ be a mapping from formulas to real numbers. We call the number $c(F)$ the *weight* of the formula F. An identity $G=H$ is said to be *weight-reducing* if:

$$c(G) > c(H).$$

We shall assume that $c(F)$ satisfies the following requirement, which in most cases of practical application does indeed hold:

(*) If the formula F' is obtained from F by a non-trivial direct application of a weight-reducing identity then we have

$$c(F') < c(F).$$

This condition ensures that the direct application to a formula of a weight-reducing identity is always efficient in the sense that it reduces the weight of this formula. Some care must be taken, however, in connection with non-direct applica-

[3] See the classical papers of C. S. Peirce [2] and H. M. Sheffer [3].

tions since the analogous assertion is not necessarily true for them even in the most simple cases occurring in practice.[4]

In most cases that are practically important, the meaning of the weight function $c(F)$, playing the role of an objective function to be minimized, is either the length of the formula F,[5] or the cost involved in its technical realization under specified circumstances.

4. After these preliminaries the minimization problem can be formulated precisely as follows: Assume that we are given a functionally complete system $\Theta = \{\vartheta_1, \vartheta_2, \ldots\}$ of a finite number of Boolean functions, a countable set $X = \{x_1, x_2, \ldots\}$ of Boolean variables, and an objective function $c(F)$, satisfying property $(*)$, defined for all formulas that can be composed by means of the given primitive elements. For a given Boolean function, represented by a formula F, consider the set $\mathscr{F} = \mathscr{F}(F) = \{F_1, F_2, \ldots\}$ of all formulas identical to F. Any formula F_{i_0}, belonging to $\mathscr{F}$, such that

$$c(F_{i_0}) \leqq c(F_i)$$

holds for all formulas $F_i \in \mathscr{F}$, is said to be a *minimal representation* of F. (In general, there exist several such formulas F_{i_0}.) An algorithm which, for any given formula F, selects a minimal representation of F is called a *minimization procedure.*

5. Now we have reached the stage where we can outline the ideas on which our minimization procedure is based.

(1) Enter the formula F given as input datum, possibly in a converted form suitable for the computer, on a list called the "*list of formulas to be minimized*".

(2) For any formula G newly entered on the list of formulas to be minimized, form all its subformulas, and then all those formulas H that have at least one substitution instance which is a subformula of G, and, finally, enter on a list called the "*list of the left-hand sides of applicable identities*" all those of these formulas H that do not yet occur there.

(3) For any formula H newly entered on the list of the left-hand sides of applicable identities, generate all formulas K having a weight less than H has. For each of these formulas K check whether it is identically equal to H; if yes then enter the identity $H = K$ on a list called the "*list of applicable identities*".

(4) Apply directly to every formula occurring on the list of formulas to be minimized all (weight-reducing) identities newly recorded on the list of applicable identities and also all those substitution instances of these identities that are weight-

[4] E.g. in case of the classical propositional calculus, taking the total number of occurrences of variables in the formula F as $c(F)$,

$$x_1 \wedge x_1 \wedge x_1 \wedge x_2 = x_1 \wedge x_2 \wedge x_2$$

is obviously a weight-reducing identity, but its substitution instance

$$x_1 \wedge x_1 \wedge x_1 \wedge (x_3 \wedge x_4 \wedge x_5) = x_1 \wedge (x_3 \wedge x_4 \wedge x_5) \wedge (x_3 \wedge x_4 \wedge x_5)$$

is not.

[5] There are many different weight functions called the lenght of a formula, e.g. those defined as the number of occurrences of variables or as the number of occurrences of variables and function symbols, etc. in the formula in question.

reducing.[6] Add those of the resulting formulas which are not yet contained in the list of formulas to be minimized to this list.

If the list of formulas to be minimized is not enlarged in step (4) then the algorithm is concluded by printing out one of the formulas with minimal weight occurring on this list; otherwise it continues at (2) again.

6. It is easy to see that our algorithm finally leads to an exact solution of the minimization problem formulated above. Indeed, if M is a minimal representation of the formula F given as input datum then $F=M$ is an identity.

If M has smaller weight than F has, i.e. the identity $F=M$ is weight-reducing, then it will sooner or later occur on the list of applicable identities, for F, as a subformula of itself, is to be found on the list of the left-hand sides of applicable identities. Then, by a direct application of the identity $F=M$ to F, we obtain M as a formula to be added to the list of formulas to be minimized. Hence, finally, either M or another formula of the same weight will be printed.

If, however, the weight of M equals that of F then F is already itself of minimal weight. Each of the generated weight-reducing identities can be applied to F only trivially, and thus the algorithm concludes after the first performance of step (4), and the only formula on the list of formulas to be minimized will be F, as a minimal representation of itself.

We emphasize that each of the formulas on the list of formulas to be minimized (among others F itself) has to stay on this list even if a formula identically equal to it of smaller weight is added to this list. Otherwise the application of a weight-reducing identity might impede later, possibly more advantageous, application of another such identity.

In practice, storage capacity or available running time limitations might prevent the continuation of the algorithm until its conclusion. If one is forced to interrupt the algorithm, we propose to print out one of the formulas with minimal weight from the list of formulas to be minimized as an approximative solution.

7. It is expedient to give the input formula of the algorithm in the so-called *Łukasiewicz bracket-free notation* (shortly Ł-notation; also known as Polish notation), or to convert it into that form by a supplementary algorithm.[7] The Ł-notation considerably simplifies the performing the algorithm.

Among others, if the formulas are written in Ł-notation, it is relatively easy to construct, by making use of the so-called *push down store*,[8] the sub-algorithms for the following tasks:

[6] In view of condition (*), a non-trivial direct application of a weight-reducing identity always reduces the weight of the formula in question, but in case of a non-direct application, as we already noted, it might happen that some substitution instance of a weight-reducing identity is not weight-reducing (see footnote[4]).

In principle, a direct application of all identities newly recorded on the list of applicable identities to every formula occurring on the list of formulas to be minimized would suffice. However, disregarding the weight-reducing substitution instances of these identities would lenghten our algorithm to such an extent that it were not practically feasible any more.

[7] See J. Łukasiewicz and A. Tarski [4], pp. 30—50. As for a simple proof of the unambiguous character of this notational system see, e.g., L. Kalmár [5], pp. 11—15.

[8] See F. L. Bauer and K. Samelson [6].

(i) Elimination of the Boolean constants from a given formula;

(ii) Production, for a given formula G, of all formulas that have at least one substitution instance which is a subformula of G;

(iii) Determination of the truth-value of a given formula for a given valuation of the variables occurring in it and, by repetitions of this sub-algorithm, the decision of the question whether two formulas are identically equal or not;

(iv) Application of an identity to a given formula.

8. As for the implementation of the algorithm on a computer, the most delicate part is (3), since it requires producing, given a formula H, all formulas K that have a smaller weight than H has. This part can perhaps most easily be realized in practice by splitting it into two steps:

(v) Production of all *formula types* of given weight. A formula type associated with a given formula containing no Boolean constants can be obtained, by definition, by replacing all Boolean variables occurring in this formula by a common one, x (without subscript), say.

(vi) Production of all the formulas of a given type containing variables from a given set only, e.g. that of the variables occurring in the input formula.

Empirical evidence (in cases that are practically most important) shows that weight-reducing identities with a smaller weight on the left-hand side, when applied, are more efficient than those with a left-hand side having a greater weight. Therefore it is advisable to apply the former ones first.

Hence, it is appropriate to generate and store the formulas in order of increasing weight. Thereby, the algorithm, even if it is interrupted, delivers a well approximating solution.

9. In the above version of the algorithm its run is controlled by the formula F to be minimized, at least in the sense that only those weight-reducing identities are produced which are non-trivially directly applicable to F or to another formula, arisen from F, occurring on the list of formulas to be minimized. In this of the way a great deal of computing time and storage room may be spared if we have only one formula to minimize.

If, however, we want to minimize several formulas, the above way might be disadvantageous. Indeed, in this case the algorithm produces, separately for each of the formulas to be minimized, all formulas, built up from the available stock of variables, that have smaller weights than those occurring on the list of the left-hand sides of applicable identities have. This might result in a very redundant repetition in the production of formulas.

An alternative version of the algorithm consists, e.g. in case only positive integers occur as weights, in generating and tabulating a "*complete* system of *independent* weight-reducing identities" up to a given ceiling for their left-hand side. In more detail, this version produces a set $\mathfrak{M}$ of weight-reducing identities such that

(α) Any weight-reducing identity such that the weight of its left-hand side does not exceed the given ceiling can be obtained, and therefore its direct applications can be replaced, by a finite number of direct applications of identities which either belong to $\mathfrak{M}$ or are weight-reducing substitution instances of identities belonging to $\mathfrak{M}$;

(β) $\mathfrak{M}$ is minimal in the sense that no identity belonging to $\mathfrak{M}$ can be obtained

by a finite number of direct applications of either other identities in $\mathfrak{M}$ or such substitution instances of these as are weight-reducing.

This variant of the algorithm is advantageous if we have to minimize a large number of formulas, since it requires to draw up the above chart of weight-reducing identities together with all their appropriate substitution instances only once, and then we have only to attempt to apply directly the identities in this chart to the formulas to be minimized. Nevertheless, we have to take into consideration that generating and tabulating the chart in question might require an enormous storage capacity.

10. Another variant of our algorithm consists in that besides the weight-reducing identities we admit such ones as leave the weights of their left-hand sides unchanged. More precisely, for every formula on the list of the left-hand sides of applicable identities we generate all the identities $H = K$ such that the weight of K does not exceed that of H; and then we apply any such substitution instance of each of these as are not weigh-augmenting to all formulas on the list of formulas to be minimized. (See especially steps (3) and (4) of the algorithm described above.)

For example, in case of classical propositional calculus with the weight of a formula meaning its length (see footnote[5]), this variant of our algorithm enables us to make use of the associative and commutative laws of conjunction and disjunction; these identities obviously do not change the length of a formula, but they may prepare for the application of another, strictly weight-reducing, identity[9].

Using this version of the algorithm we may, possibly, arrive at a minimal representation of the starting formula much quicker, though the price of this may be a much larger storage capacity used up. In yet another possible variant of our algorithm, for which the remarks made just now apply still more strongly, we may allow the application of certain weight-augmenting identities as well; e.g., in case of the classical propositional calculus the use of distributive law in the direction $(x_1 \vee x_2) \wedge x_3 = (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$ may sometimes prove useful by preparing the way for the application of a powerful weight-reducing identity.[10]

RESEARCH GROUP ON MATHEMATICAL LOGIC
AND THEORY OF AUTOMATA OF THE
HUNGARIAN ACADEMY OF SCIENCES,
SOMOGYI BÉLA U. 7,
SZEGED, HUNGARY.

[9] The situation is illustrated by the following simple example, for which the author is indebted to an oral communication of G. Specker:

$$(\dots((x_1 \vee x_2) \vee x_3) \dots \vee x_n) \vee x_1 = (\dots(((x_1 \vee x_1) \vee x_2) \vee x_3) \dots \vee x_{n-1}) \vee x_n =$$
$$= (\dots((x_1 \vee x_2) \vee x_3) \dots \vee x_{n-1}) \vee x_n.$$

[10] The following example may serve as an illustration:

$$((x_1 \vee x_2) \wedge x_3) \vee x_2 = (x_1 \wedge x_3) \vee (x_2 \wedge x_3) \vee x_2 = (x_1 \wedge x_3) \vee x_2.$$

References

[1] ÁDÁM, A., *Truth functions and the problem of their realization by two-terminal graphs*, Akadémiai Kiadó, Budapest, 1968.
[2] PEIRCE, C. S., A Boolian algebra with one constant (c. 1880), *Collected papers* IV., pp. 13—18.
[3] SHEFFER, H. M., A set of five independent postulates for Boolean algebras with application to logical constants, *Trans. Amer. Math. Soc.*, v. 14, 1913, pp. 481—488.
[4] ŁUKASIEWICZ J. & A. TARSKI, Untersuchungen über den Aussagenkalkül, Sprawozdania zposiedzén Towarzystwa Naukowego Warszawskiego, Wydz. III, 23, 1930.
[5] KALMÁR, L., Another proof of the Markov-Post theorem, *Acta Math. Acad. Sci. Hung.*, v. 3, 1952, pp. 1—27.
[6] BAUER, F. L. & K. SAMELSON, Sequential formula translation, *Comm. ACM*, v. 3, 1960, pp. 76—83; addendum on p. 351.

(Received March 31, 1970; Section 10 added May 8, 1970)