ALGORYTMY

VOL. 1 • NO. 1 • 1962

INSTYTUT MASZYN MATEMATYCZNYCH PAN

Prace Instytutu Maszyn Matematycznych PAN - Ser. ALGORYTMY, vol. 1, Nr 1, 1962

E R R A T A

Str.	Wiersz		Jest	Powinno być
	od góry	od dołu		
10	8		.... $S(\varphi_i(p))\ \varphi_i(p)$	.... $S(\varphi_i(p))\ \varphi_i^2(p)$
11	13		.... $S(\varphi_i(p))\ \varphi_i(p)$	.... $S(\varphi_i(p))\ \varphi_i^2(p)$
12	5		and $T = K(S,\eta) \cap W$	and $T = K(S,\eta) \cap S(\xi_0,\lambda)$
12	6		.... $N = E^n - K(W,\eta) \cap \cap K(\xi_0,\lambda)$ is not ...	.... $N = S(\xi_0,\lambda) - T$ is not ...
12		5	$\mu_k \leqslant \mathcal{K}$	$\mu_k \leqslant \mathcal{K}(\varepsilon)$.
13	2		$\gamma = \mu_k \alpha_0 - \mu_k \Psi - \varphi$.	$\gamma \geqslant \mu_k \alpha_0 - \mu_k \Psi - \varphi$.
13		2	$R = \overline{K(x_0,\lambda) - T}$	$R = K(\xi_0,\lambda)$
14	1		$R \subset K(x_0,\varepsilon)$.	$R \subset K(\xi_0,\varepsilon)$
14	3,4,5,6,7,8, 9,10,11,12		całe skreślić!	and $\overline{N} \cup \overline{T} = Fr(R)$
40		11	. Sign of equality	= Sign of equality
53	11		.... C = (CAT / MOUSE) x x PENCE PENCE	.... C = (CAT / MOUSE)x x PENCE

A L G O R Y T M Y

Vol. I N°1 1962

P R A C E

Instytutu Maszyn Matematycznych

P o l s k i e j A k a d e m i i N a u k

Lata 1961 oraz 1962 mają zasadnicze znaczenie dla zastosowań maszyn matematycznych w Polsce. W tym okresie rozpoczęta została produkcja krajowa, jak również import maszyn cyfrowych, dzięki czemu poważnie wzrosła liczba ośrodków obliczeniowych wyposażonych w maszyny matematyczne. Istotnego znaczenia nabrała wymiana informacji i doświadczeń oraz zagadnienie publikowania odpowiednich prac naukowych. Ponieważ żadne z istniejących w Polsce czasopism nie posiada obecnie charakteru odpowiadającego tym potrzebom, postanowiliśmy, celem chociażby częściowego ich zaspokojenia, wydawać odrębną serię PRAC Instytutu Maszyn Matematycznych PAN o nazwie 'ALGORYTMY', ukazującą się w okresach kwartalnych.

Z najważniejszej tematyki zamierzamy uwzględnić w 'ALGORYTMACH':

- Metody obliczeń naukowych i technicznych. Z tego zakresu publikowane będą zarówno prace teoretyczne z metod numerycznych, jak również konkretne programy pisane w językach SAKO lub ALGOL.
- Metody programowania dla maszyn cyfrowych. Poruszane tu będą problemy dotyczące języków formalnych /w zastosowaniu do autokodów/, metody translacji oraz metody optymalizacji programów.
- Metody zastosowań administracyjno-gospodarczych. W tym zakresie publikowane będą prace teoretyczne, jak również przykładowe opisy konkretnych systemów.
- Metody modelowania cyfrowego.
- Prace nad matematyczną problematyką stosowania maszyn cyfrowych do sterowania procesami przemysłowymi.

W 'ALGORYTMACH' zamieszczane będą prace wykonane w Instytucie Maszyn Matematycznych oraz w innych Ośrodkach. Prace o bardziej teoretycznym i ogólnym charakterze będą publikowane w języku rosyjskim i angielskim, zaś materiały informacyjne, przeznaczone dla użytkowników krajowych - w języku polskim.

Prof. dr L. Łukaszewicz

Years 1961 and 1962 were of great significance for the use of mathematical machines in Poland. Our own production, as well as the import of digital computers from abroad began at that time. Due to this, the number of Polish Computing Centers provided with computers increased considerably. The exchange of information and experience as well as the publication of appropriate scientific papers became very important.

Since none of the Polish journals have met these needs, we decided to issue a new series of our publication 'Prace IMM PAN' which may partly cover the demands; it will appear quarterly, under the title 'ALGORYTMY' /Algorithms/. The main subjects we intend to include in 'ALGORYTMY' will be the following:

- Methods of theoretical and technical computations. This will include both papers on the theory of numerical methods and programs written in SAKO or in ALGOL.
- Methods of programming for digital computers. This will include problems concerning formal languages /in relation to autocodes/, methods of translation and methods of optimization of programs.
- Methods of business and economic application. This will comprise papers concerning theoretical problems as well as examples of real systems.
- Problems connected with the control of technological processes by means of digital computers.
- Methods of digital simulation.

Papers published in 'ALGORYTMY' may be both from our Institute and from other Centers. Papers on theory and of a more general character will be published in Russian or English and information material for home use - in Polish.

Prof. dr Leon Łukaszewicz

T R E Ś Ć
CONTENTS

NUMERICAL ANALYSES

Instytut Maszyn Matematycznych PAN
Algorytmy N° 1

518.12

ON A METHOD OF APPROXIMATIVE FINDING CONDITIONAL MAXIMUMS

by Tomasz PIETRZYKOWSKI

Received January 1962

The paper considers an approximative solution of the conditional maximum problem, with constrains in the form of the equality and inequality.

The purpose of the presented paper is to show that the problem of constrained maximum can be reduced under natural conditions to the problem of unconditional maximum of certain function.

Some partial results in this direction were previously obtained by the author in [2] and [3].

Let E^n denotes the n-dimensional euclidean spece and suppose that $W \subset E^n$ is defined as the set of points $p \in E^n$ satisfying the following conditions

$$\varphi_i(p) = 0 \qquad i = 1, \ldots, m$$
$$\varphi_i(p) \geqslant 0 \qquad i = m+1, \ldots, m+l \qquad /1/$$

where the real functions φ_i /i = 1, ..., m+1/ are defined and continuous on E^n.

Let us suppose further that f is a real continuous function on E^n and let us consider the following problem:

/P/ find a local strong maximum of f on the set W, which is the same, find a local strong conditional maximum f on E^n under the constrains /1/.

Let $\mu_1, \mu_2, \ldots,$ be a sequence of positive numbers converging to zero.

Now we define the sequence of function G_k such that

$$G_k(p) = \mu_k f(p) - \sum_{i=1}^{m} \varphi_i^2(p) + \sum_{i=m+1}^{m+l} S(\varphi_i(p))\varphi_i(p) \qquad p \in E^n, \quad k=1,2,\ldots$$

where $S(t)$ is the function defined on the real line by the formula

$$S(t) = \begin{cases} -1 & \text{for} \quad t < 0 \\ 0 & \text{for} \quad t \geqslant 0. \end{cases} \qquad /2/$$

It turns out that the problem /P/ is linked to the following problem:

$/P_k/$ find a local unconditional maximum of the function G_k on E^n /k = 1, 2, ../.

Namely the following theorem holds:

Theorem.

Let the point x_0 be a solution of the problem /P/. Then there exists such a sequence of points $x_1, x_2, \ldots,$ that

$$\lim_{k\to\infty} x_k = x_0$$

and x_k is a solution of the problem $/P_k/$ /k = 1, 2, .../

Proof.

First of all, we notice that if x_0 belongs to the interior of W, the function G_k is equal to the funct $\mu_k f$ in certain neighbourhood of x_0. Since $\mu_k > 0$, it is obvious that x_0 is the solution of $/P_k/$ for each k so the theore true in this case.

Let us suppose now that x_0 belongs to the boundry of W and ε be on arbitrary positive number. We shall prove that there exists such a $\varkappa_{(\varepsilon)}>0$ that

$$\|x_k - x_0\| < \varepsilon$$

where x_k is a solution of the problem $(P_{k(\varepsilon)})$ and $\mu_k < \varkappa(\varepsilon)$

Denote by Φ the function defined on E^n by the formula

$$\Phi(p) = -\sum_{i=1}^{m} \varphi_i^2(p) + \sum_{i=m+1}^{m+l} S(\varphi_i(p))\varphi_i(p) \qquad p \in E^n \qquad /3/$$

Let $\alpha_0 = f(x_0)$.

Since the point x_0 by the assumption is a strong local maximum of the function f on the set W there exists a number λ so that

$$0 < \lambda < \frac{\varepsilon}{2} \qquad /4/$$

and

$$\alpha < \alpha_0 \qquad /5/$$

where $\alpha = \sup_{p \in S} \{f(p)\}$ and $S = S(x_0, \lambda) \cap W$.*)

Since the set W is closed, x_0 is a boundry point of W it follows by compactness of S and in view of /5/ that there exists

---

* $S(x_0, \lambda) = \{p \in E^n, \rho(p, x_0) = \lambda\}$ denotes a sphere in E^n with the center in x_0 and radius λ.

a number η satisfying the following conditions /6/, /7/ and /8/ where

$$0 < \eta < \frac{\lambda}{2} \qquad /6/$$

$$\beta < \alpha_0 \qquad /7/$$

where $\beta = \sup_{p \in \overline{T}} \left\{ f(p) \right\}$ and $T = K(S, \eta) \cap W$ *)

the set $N = \left(E^n - K(W, \eta) \right) \cap K(x_0, \lambda)$ is not empty. /8/

Let

$$\varphi = \sup_{p \in \overline{N}} \left\{ \dot{\Phi}(p) \right\} \qquad /9/$$

and

$$\Psi = \sup_{p \in \overline{N}} \left\{ f(p) \right\} \qquad /10/$$

It can be readily veryfied in view of /2/ and /3/ that the function $\dot{\Phi}$ is equal to zero on the set W and is negative on $E^n - W$.

In view of /5/ and /7/ the intersection $\overline{N} \cap \overline{W}$ is empty, since the function $\dot{\Phi}$ is negative on N, this implies taking into account continuity of $\dot{\Phi}$ and /9/ that $\varphi < 0$.

Let

$$\varkappa(\varepsilon) = \frac{-\varphi}{2|\alpha_0 - \Psi|} \qquad /11/$$

evidently $\varkappa(\varepsilon) > 0$.

Let μ_k be a term of the sequence $\{\mu_k\}$ such that

$$\mu_k \leq \varkappa . \qquad /12/$$

Let us estimate the number

$$\delta = G_k(x_0) - \sup_{p \in \overline{N}} \left\{ G_k(p) \right\}$$

---

* $K(A, \eta) = \left\{ p \in E^n, \rho(A, p) \leq \eta \right\}$ denotes a closed η-neighbourhood of the set A in E^n. The set A can be reduced to one-point set.

From /3/, /9/ and /10/ it follows that

$$\gamma = \mu_k \alpha_0 - \mu_k \Psi - \varphi .$$

In view of /11/ and /12/ we obtain

$$\gamma = \mu_k \ (\alpha_0 - \Psi) - \varphi \geqslant - \mu_k \ |\alpha_0 - \Psi| \ - \varphi > \frac{\varphi}{2} - \varphi = - \frac{\varphi}{2} > 0.$$

Hence

$$G_k (x_0) > \sup_{p \in \bar{N}} \left\{ G_k (p) \right\}. \qquad /13/$$

Next let us consider the number

$$\delta = G_k (x_0) - \sup_{p \in \bar{T}} \left\{ G_k (p) \right\}.$$

From /3/ and /7/ we have

$$\delta = \mu_k \alpha_0 - \mu_k \beta - \sup_{r \in \bar{T}} \left\{ \Phi (p) \right\} .$$

Since the function Φ is non-positive on E^n, hence

$$\sup_{p \in \bar{T}} \left\{ \Phi (p) \right\} \leqslant 0.$$

Therefore

$$\delta \geqslant \mu_k \ (\alpha_0 - \beta).$$

From /7/, taking into acoount that $\mu_k > 0$ we have

$$G_k (x_0) > \sup_{p \in \bar{T}} \left\{ G_k (p) \right\}. \qquad /14/$$

It is easily seen, from /13/ and /14/ that

$$G_k (x_0) > \sup_{p \in \bar{N} \cup \bar{T}} \left\{ G_k (p) \right\}. \qquad /15/$$

Taking into aocount /6/ and /7/ it is obvious that the set

$$R = \overline{K(x_0, \lambda) - T}$$

is the closed neighbourhood of x_0.

Of course

$$R \subset K(x_0, \varepsilon). \qquad /16/$$

Now we shall prove the following inclusion:

$$\bar{N} \cup \bar{T} \supset Fr(R). \qquad /17/$$

Let $p \in Fr(R)$. We shall consider two cases. In the first $x \in S(x_0, \lambda) \cap (E^n - Int(T))$. Since $T \subset K(W, \eta)$ and in view of /7/ and /8/ we have

$$x \in \bar{N}.$$

In the second case $x \in Fr(T) \cap K(x_0, \lambda)$ since from $x \in Fr(T)$ it follows

$$x \in \bar{T}$$

so finally $x \in \bar{N} \cup \bar{T}$.

From /15/ and /17/ we obtain

$$G_k(x_0) > \sup_{p \in Fr(R)} \{G_k(p)\}. \qquad /18/$$

Since R is a closed neighbourhood of x_0 and the function G_k is continuous it follows from /18/ that there exists a local maximum x_k of G_k with $x_k \in Int(R)$. The proof of the statement is left to the reader.

Finally the inclusion /16/ implies

$$\|x_k - x_0\| < \varepsilon.$$

Let

$$\varepsilon_i = \frac{1}{i} \qquad /i = 1, 2, \ldots,/. \qquad /19/$$

From /18/ we know that for each ε_i there exists such a number $\varkappa_i$ that

$$\|x_k - x_0\| \leq \frac{1}{i} \qquad /i = 1, 2, \ldots,/ \qquad /20/$$

where x_k is a solution of the problem (P_k) for such a k that $\mu_k \leq \varkappa_i$.

Since the sequence $\{\mu_k\}$ is convergent to zero and all α_i are positive, hence for each natural i nearly all terms of the sequence $\{x_k\}$ satisfy the inequality /20/. Thus the proof is completed.

References:

1. FRISCH R.A.K.: The Logarithmic Potential Method of Convex Programming, Memorandum of 13 May 1955 from the University Institute of Economics, Oslo.

2. PIETRZYKOWSKI T.: Logarithmic Method for Finding Conditional Maximums, Prace ZAM, 1960: A10.

3. PIETRZYKOWSKI T.: Application of the Steepest Ascent Method to Linear Programming, Prace ZAM, 1961: A11.

Instytut Maszyn Matematycznych PAN
Algorytmy N° 1

681.142
518.12:517.392

AN EFFECTIVE METHOD OF COMPUTING GENERALIZED ROMAN FUNCTION

by Krzysztof MOSZYŃSKI
Karol PELDMAN

Received December 1961

The method of an effective calculation of the so-called generalized Roman Function is given in the present paper. This function is defined by an integral from zero to infinity involving Bessel Function of order zero. The suggested method is based on the solution of initial value problems for the system of ordinary differential equations. Information on the results obtained on the XYZ computer, using this method, as well as the program in ALGOL is given in the Appendix.

A function expressed as integral

$$P_n(k_1, k_2, r) = \frac{1}{r} + 2\int_0^{\infty} \frac{\left(k_1 e^{-2\lambda} + k_2 e^{-2\lambda n}\right) \cdot J_0(\lambda \cdot r)}{1 - k_1 e^{-2\lambda} - k_2 e^{-2\lambda n} + k_1 k_2 e^{-2\lambda(n-1)}}\, d\lambda \qquad /1/$$

where $|k_1|, |k_2| < 1$, $r > 0$, n is an integer, and $J_0(z)$ is a Bessel's function of order zero, is essential in geoelectric methods of geology [1].

The problem is to compute the tables of $P_n(k_1, k_2, r)$ for the given values of k_1, k_2, $0 < r_0 < r < 1000$ and of $1 < n < 32$.

The method proposed in [1] requires a great deal of work to find the complex zeros for a great number of algebraic polynomials the degree of which is n, $1 < n < 32$.

The authors want to avoid this difficulty and to give a more suitable method for a digital computer. This method is based on a direct computation of integral /1/.

First of all, a simple transformation $x = e^{-2\lambda}$ gives:

$$P_n(k_1, k_2, r) = \frac{1}{r} + \int_0^1 \frac{k_1 + k_2 x^{n-1}}{1 - k_1 x + k_1 k_2 x^{n-1} - k_2 x^n} J_0\left(-\frac{r}{2} \ln x\right) dx. \qquad /2/$$

The difficulties lie, therefore, in the function

$$u_0^r(x) = J_0\left(-\frac{r}{2} \ln x\right)$$

which, however bounded in the entire interval $[0, 1]$, gives an infinite number of 'oscillations' in the neighbourhood of zero.*) To make it more suitable for computation we introduce the new function

$$u_k^r(x) = x^k J_0\left(-\frac{r}{2} \ln x\right)$$

where $k > 1$ is an integer. This function is $O\left(\frac{x^k}{\sqrt{-\frac{r}{2} \ln x}}\right)$, when x tends to zero [2].

Let $P_k(x) = a_0 + a_1 x + \ldots + a_{k-1} x^{k-1}$ be any polynomial. If we denote the rational function in integral /2/ by $W_n(x)$, it will be:

$$\int_0^1 W_n(x) u_0^r(x) dx = \int_0^1 \left[W_n(x) - P_k(x)\right] u_0^r(x) dx + \int_0^1 P_k(x) u_0^r(x) dx$$

It is well known [3] that:

$$\int_0^1 P_k(x) u_0^r(x) dx = \sum_{l=0}^{k-1} \frac{a_l}{\sqrt{(l+1)^2 + \left(\frac{r}{2}\right)^2}}$$

---

* The values of k_1 and k_2 are chosen so that the polynomial $1 - k_1 x + k_1 k_2 x^{n-1} - k_2 x^n$ has no zeros in $[0, 1]$ /see for instance [1]/.

The polynomial $P_k(x)$ can be formed in such a way that all the members of a degree strictly less than k in the numerator of rational function

$$W_n(x) - P_k(x)$$

will vanish.

So

$$W_n(x) - P_k(x) = x^k V_n(x) = x^k \frac{a_k + a_{k+1}x + \ldots + a_{n+k-1}x^{n-1}}{1 - k_1 x + k_1 k_2 x^{n-1} - k_2 x^n} \qquad /3/$$

where $V_n(x)$ is the rational function of the same denominator as $W_n(x)$. The problem is therefore to compute the integral*)

$$Q = \int_0^1 V_n(x) u_k^r(x)\,dx. \qquad /4/$$

Let $0 < \varepsilon < 1$ and

$$\rho(\varepsilon) = \int_0^\varepsilon V_n(x) u_k^r(x)\,dx = \int_0^1 V_n(x) u_k^r(x)\,dx - \int_\varepsilon^1 V_n(x) u_k^r(x)\,dx.$$

We shall not compute, of course, the integral /4/ but the integral

$$\int_\varepsilon^1 V_n(x) u_k^r(x)\,dx.$$

The number ε must be chosen so that $|\rho(\varepsilon)| < \delta$ where δ is in the range of error admissible in computation.

Let us estimate $\rho(\varepsilon)$.

$$|\rho(\varepsilon)| \leq \int_0^\varepsilon |V_n(x) u_k^r(x)|\,dx; \qquad V_n(x) = \frac{A(x)}{1 - \xi(x)}$$

---

* The authors are grateful to doc. K. Bochenek for this idea.

where $A(x)$ is a polynomial, $|A(x)| < A$,

$$\xi(x) = k_1 x - k_1 \cdot k_2 x^{n-1} + k_2 x^n = x\left[k_1 - k_1 \cdot k_2 x^{n-2} + k_2 x^{n-1}\right].$$

It is clear that: $|\xi(x)| \leq Mx$, $M < 3$ in $[0, 1]$.

For $0 < x < \frac{1}{3}$, the inequality $0 < 1 - Mx < 1 - |\xi(x)| < |1 - \xi(x)|$ gives $\frac{1}{1 - Mx} > \frac{1}{1 - |\xi(x)|} > \frac{1}{|1 - \xi(x)|}$.

Therefore

$$\int_0^{\varepsilon} \left|V_n(x)\, u_k^r(x)\right| dx \leq A \cdot \int_0^{\varepsilon} \frac{x^k}{1 - Mx} \left|J_0\left(-\frac{r}{2} \ln x\right)\right| dx, \quad \text{for} \quad 0 < \varepsilon < \frac{1}{3}.$$

From the asymptotic formula of $J_0(z)$ we have the estimation [2]:

$$\left|J_0(z)\right| \leq \sqrt{\frac{2}{\pi z}}\left[1 + \frac{9}{128 z^2} + \frac{1}{8|z|}\right]$$

for $0 < x < \frac{1}{3}$, $z = -\frac{r}{2} \ln x$, $|z| \geq \frac{r_0}{2}\left|\ln \frac{1}{3}\right| \geq \frac{r_0}{2}$

and $\left|J_0(z)\right| \leq \frac{2}{\sqrt{\pi r}} \cdot \frac{B}{\sqrt{-\ln x}}$

where $B = 1 + \frac{9}{32 r_0^2} + \frac{1}{4 r_0}$.

Hence

$$\left|\rho(\varepsilon)\right| \leq \int_0^{\varepsilon} \left|V_n(x)\, u_k^r(x)\right| dx \leq \frac{2A \cdot B}{\sqrt{\pi r}} \int_0^{\varepsilon} \frac{x^k}{1 - Mx} \cdot \frac{dx}{\sqrt{-\ln x}}.$$

Using the 'formula for the mean value', we get:

$$\left|\rho(\varepsilon)\right| \leq \frac{2A \cdot B}{\sqrt{\pi r}} \cdot \frac{1}{\sqrt{-\ln \xi}} \int_0^{\varepsilon} \frac{x^k}{1 - Mx} dx \quad \text{when} \quad 0 < \xi \leq \varepsilon$$

$$|\rho(\varepsilon)| \leq \frac{2AB}{\sqrt{\pi r}} \cdot \frac{1}{\sqrt{-\ln \varepsilon}} \cdot \int_0^{\varepsilon} \frac{x^k}{1 - Mx}\, dx = \frac{2AB}{\sqrt{\pi r}} \cdot \frac{\varepsilon^{k+1}}{\sqrt{-\ln \varepsilon}} \cdot \left[\frac{1}{k+1} + \frac{M\varepsilon}{k+2} + \frac{(M\varepsilon)^2}{k+3} + \dots\right] \leq$$

$$\leq \frac{2AB}{\sqrt{\pi r}} \cdot \frac{1}{\sqrt{-\ln \varepsilon}} \cdot \left[\frac{1}{k+1} + \frac{M\varepsilon}{1} + \frac{(M\varepsilon)^2}{2} + \dots\right] = \frac{2AB}{\sqrt{\pi r}} \cdot \frac{\varepsilon^{k+1}}{\sqrt{-\ln \varepsilon}} \left[\frac{1}{k+1} - \ln(1 - M\varepsilon)\right].$$

Thus we have the estimation for $\rho(\varepsilon)$:

$$|\rho(\varepsilon)| \leq \frac{2AB}{\sqrt{\pi r}} \cdot \frac{\varepsilon^{k+1}}{\sqrt{-\ln \varepsilon}} \left[\frac{1}{k+1} - \ln(1 - M\varepsilon)\right] \qquad /5/$$

It is possible to chose suitable values for ε and k, so as to obtain a sufficiently small $|\rho(\varepsilon)|$. The number ε must be chosen not too close to zero.

Let us consider now some properties of the function $u_k^r(x)$ in the interval $[0, 1]$.

Since $u_k^r(x) = x^k J_0\left(-\frac{r}{2} \ln x\right)$, it is clear that

$$u_k^r(1) = 1 \qquad /6/$$

and

$$\lim_{x \to 0} u_k^r(x) = 0$$

for all values of $r > 0$ and $k = 0, 1, 2, 3, \dots$

$$\dot{u}_k^r(x) = kx^{k-1} J_0(z) + \frac{r}{2} x^{k-1} J_1(z) \qquad /7/$$

where

$$z = -\frac{r}{2} \ln x.$$

Hence

$$\dot{u}_k^r(1) = kJ_0(0) = k \qquad /8/$$

for all values of k.

Evidently

$$\lim_{x \to 0} \dot{u}_k^r(x) = 0 \qquad \text{for } k = 1, 2, 3, \ldots .$$

This result fails however for $k = 0$. The above follows from the asymptotic formulas

$$J_0(z) = \sqrt{\frac{2}{\pi z}}\left[\cos\left(z - \frac{\pi}{4}\right) - \sin\left(z - \frac{\pi}{4}\right) + O(z^{-1})\right]$$

$$J_1(z) = \sqrt{\frac{2}{\pi z}}\left[\cos\left(z - \frac{3}{4}\pi\right) - \sin\left(z - \frac{3}{4}\pi\right) + O(z^{-1})\right] .$$

For the second derivative we have:

$$\ddot{u}_k^r(x) = x^{k-2}\left\{J_0(z)\left[(k-1)k - \frac{r^2}{8}\right] + \frac{r}{2} J_1(z)(2k-1) + \frac{r^2}{8} J_2(z)\right\}. \qquad /9/$$

It is easy to see that:

$$\ddot{u}_k^r(1) = (k-1)k - \frac{r^2}{8}$$

and

$$\lim_{x \to 0} \ddot{u}_k^r(x) = 0 \quad \text{for} \quad k \geq 2. \qquad /10/$$

For $k = 0$ and 1 the limit does not exist.

The above results can be seen in the following table:

Table 1

x	$\dot{u}_k^r(x)$	$\dot{u}_k^r(x)$	$\ddot{u}_k^r(x)$
0	0	does not exist for $k = 0$	does not exist for $k = 0, 1$.
		0 for $k > 0$	0 for $k > 2$
1	1	k	$(k-1)k - \frac{r^2}{8}$

It is easy to prove that the funotion $u_k^r(x)$ oan be defined as the solution of the seoond order differential equation:

$$\ddot{u}_k^r(x) + \dot{u}_k^r(x)\left[1 - 2k - \frac{r}{2z}\right]x^{-1} + u_k^r(x)\left[\frac{kr}{2x^2}\left(2\frac{k}{r} + \frac{1}{z}\right) + \frac{r^2}{4}x^{-2}\right] = 0 \qquad /11/$$

where

$$z = -\frac{r}{2}\ln x.$$

Equation /11/ is not defined for $x = 1$; however, for $x = 1$, the 'limit equation' is of the form

$$\ddot{u}_k^r(1) = (k-1)k - \frac{r^2}{8}. \qquad /12/$$

Now it is possible to present the method of computation of /1/. The system of differential equations

$$\frac{dQ}{dx} = -V_n(x)u_k^r(x)$$

$$\frac{d^2u_k^r(x)}{dx^2} = -\frac{du_k^r(x)}{dx}\left[1 - 2k - \frac{r}{2z}\right]x^{-1} - u_k^r(x)\left[\frac{kr}{2x^2}\left(\frac{2k}{r} + \frac{1}{z}\right) + \frac{r^2}{4x^2}\right] \qquad /13/$$

$$\frac{dz}{dx} = -\frac{r}{2x}$$

with the initial conditions from table 1

$$\begin{aligned} Q(1) &= 0 \\ u_k^r(1) &= 1 \\ \dot{u}_k^r(1) &= k \\ z(1) &= 0 \end{aligned} \qquad /14/$$

gives the funotion $Q(x)$. Its value at the point ξ, $0 < \xi < 1$, defined from /5/ is a olose approximation of /4/.

For $x = 1$ one oan use equation /12/ instead of the seoond equation /13/.

The authors want to emphasize that the above method, when used on a digital computer, does not require any standard subroutine, except a program for solving a system of differential equations.

Computational experiences

The program for /13/ was completed in the Computing Center of ZAM in Warsaw and certain values of /1/ have been obtained on the XYZ computer.

The program in Algol is given in the appendix. The SAKO version of this program will appear elsewhere. The described method was adapted when $k = 5$ and $\varepsilon = \frac{1}{128}$. The well-known Runge-Gill algorithm was used for the integration of the system /13/ [4]. As it is easy to prove, the 5-th derivative of $u_k^r(x)$ is bounded for $k = 5$, on the interval $[0, 1]$. This is essential because of Runge-Gill's Method used here, which is of order 5. Good results /six decimal places exact/ were obtained using step $h = \frac{1}{256}$ for not very great values of r ($r \leq 32$), however, for small values of r, much a greater step h seems to be sufficient. The step must be decreased when r is augmented; it should be approximately reversely proportional to r. As it follows from /5/, the interval of integration of /13/ can decrease when r augments. In fact, the results thus obtained for intervals $\left[\frac{1}{128}, 1\right]$ and $\left[\frac{1}{32}, 1\right]$ differ on the last place only, even for the small values of r.

A p p e n d i x

THE PROGRAM IN ALGOL 60
for k = 5

```
array L[0:3];
    L[0]:= 0.5
    L[1]:= 0.29289322
    L[2]:= 1.70710678
    L[3]:= 0.16666666

begin prooedure COEFFICIENTS (W1, W2, W3) RESULTS TO: (W);
      value W1, W2, W3; integer W3; real array W;
      begin integer i;
      switch S := 1, 2, 3, 4, 5, 6, 7, 8;
          W[0] := W1
      if  W3>7  then go to 8 else go to S[W3+1];
1: go to 1;
2: go to 2;

3:              W[1]:= W1× W[0] + W2× (1 - W1× W[0]);
   for  i := 0  step  1  until  3  do
                W[i+2]:= W1 × W[i+1] + W2 × (W[i] - W1 × W[i+1]);
                W[6]:= W2 × W[4];
   for  i := 7  step  1  until  11  do  W[i]:= 0;  go to 9;
4:              W[1]:= W1× W[0];
                W[2]:= W1× W[1] + W2× (1 - W1× W[0]);
```

```
   for i := 0 step 1 until 2 do

            W[i+3]:= W1 × W[i+2] + W2×(W[i] - W1 × W[i+1]);
              W[6]:= W2×(W[3] - W1 × W[4]);
              W[7]:= W2 × W[4];

   for i := 8 step 1 until 11 do W[i]:= 0; go to 9;

5:          W[1]:= W1 × W[0];
            W[2]:= W1 × W[1];
            W[3]:= W1 × W[2] + W2×(1 - W1 × W[0]);
            W[4]:= W1 × W[3] + W2×(W[0] - W1 × W[1]);
            W[5]:= W1 × W[4] + W2×(W[1] - W1 × W[2]);
            W[6]:= W2 × (W[2] - W1 × W[3]);
            W[7]:= W2 × (W[3] - W1 × W[4]);
            W[8]:= W2 × W[4];

   for i := 9 step 1 until 11 do W[i]:= 0; go to 9;

6: for i := 0 step 1 until 2 do W[i+1] := W1× W[i];

            W[4]:= W1 × W[3] + W2×(1 - W1 × W[0]);
            W[5]:= W1 × W[4] + W2×(W[0] - W1 × W[1]);
            W[6]:= W2 × (W[1] - W1× W[2]);
            W[7]:= W2 × (W[2] - W1 × W[3]);
            W[8]:= W2 × (W[3] - W1 × W[4]);
            W[9]:= W2 × W[4];
            W[10]:= W[11]:= 0; go to 9;

7: for i := 0 step 1 until 3 do W[i+1] := W1 × W[i];

            W[5]:= W1 × W[4] + W2×(1 - W1 × W[0]);

   for i := 0 step 1 until 3 do W[i+6]:= W2×(W[i] - W1 × W[i+1]);

            W[10]:= W2 × W[4];
            W[11]:= 0; go to 9;

8: for i := 0 step 1 until 4 do W[i+1] := W1× W[i];

            W[6] := W2×(1 - W1×W[0]);
```

```
   for  i := 0  step 1 until 3 do W[i+7] := W2×(W[i] - W1×W[i+1]);
                W[11] := W2 × W[4];
9: end of coefficients;
prooedure  RG(F)FOR:(x, M); value M; real array x;
           integer M;
comment this procedure gives the solution of a system of differen-
        tial equations at the point  x+H,  when this solution for
        x  is known. The algorithm used here is a well-known
        Runge-Kutta method modified by Gill;
begin real  w, s, R1;  integer  j, i;
   for  j := 0 step 1 until 3 do begin F;
   for  i := 0 step 1 until 3×M-1 do
   begin if  j = 3  then begin  W := 2;  S := 3  end else
                          W := S := 1;
      R1:=L[j]×(x[i+M] - W × x[i + 2× M]);
    x[i]:=x[i] + R1
x[i+2×M]:=x[i+2×M] + 3×R1 - S×L[j]×x[i+M]  end;
                end; end of runge gill;
procedure  RHS(Y1, Y2, Y3, Y4, Y5, Y6, Y7);
   value Y1, Y2, Y3, Y4, Y5, Y6, Y7;  integer  Y5;
        real array  Y6, Y7;  real  Y1, Y2, Y3, Y4;
comment  this procedure oomputes the values of the right-hand
         sides of differential equations;
begin integer  S;  real  T1, T2, T3;
                   Y7[5]:= Y4;
                   Y7[6]:= -Y4×Y3/2;
                   Y7[7]:= Y4×Y7[4];
                   T1:=0;
```

```
for S := 11 step -1 until 6 do T1:=T1×Y7[0]+A[S];
                    T2:=Y7[0]↑(Y5 - 1);
if Y5 > 7 then T3:=X7[0]↑(Y5 - 6) else T3:=X7[0];
                    T2:=1-Y1×Y7[0]+Y2×(Y1-Y7[0])×T2;
                Y7[8]:=-Y4 × Y7[2]×(A[5]+T3×T1)/T2;
if Y7[0]=1 then Y7[9]:=Y4×(20-(Y3↑2)/8) else
begin T1:=((18/Y3)×Y7[4]-(Y7[2]/Y7[0]))×(Y3/2+50/Y3);
      T2:=(1/Y7[1])×(Y7[4]-(Y7[2]/Y7[0])×5);
   Y7[9]:=Y4×(Y3/(2×Y7[0]))×(T1+T2) end;
        end of RHS;
procedure READ(Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8, Z9);
comment this procedure is to be written in non Algol language. It
        reads into the storage the values of the parameters quoted
        in parentheses.
        - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        end of read;
procedure PRINT,
comment this procedure, in non Algol language, prints out from the
        memory the values P, K1, K2, R1, N, H, and E;
        - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        end of print;
  array A[0:11], RUNGE GILL [0:14]; real K1, K2, R, DELTA R,
  FINAL R, H, E, P, R1; integer N, F, j;
  1:  READ (K1, K2, R, DELTA R, FINAL R, H, EN, F);
comment The parameters quoted in parentheses define the variant.
        Variable F, must possess the value zero, except for
        the last variant to compute, when its value is one;
  COEFFICIENTS (K1, K2, N) RESULTS TO : (A);
```

```
comment The procedure COEFFICIENTS computes the coefficients of
        the polynomial P_k(x), as well as the coefficients of
        the denominator in formula /2/;

  for R1:=R step DELTA R until FINAL R do
  begin          RUNGE GILL [0]:= RUNGE GILL [2]:= 1;
                 RUNGE GILL [1]:= RUNGE GILL [3]:= 0;
                 RUNGE GILL [4]:= 5;

  for j := 5 step 1 until 14 do RUNGE GILL[j]:= 0;

comment the above sets the initial values for differential
        equations;

            P := 1/R1; for j := 0 step 1 until 4 do
            P := P + A[j] /sqrt ((j+1)↑2 + (R1/2)↑2);

2:    RG(RHS(K1, K2, R, H, N, A, RUNGE GILL)) FOR: (RUNGE GILL, 5);

  if E < RUNGE GILL [0] then go to 2 else
                  P := P + RUNGE GILL[3];

comment P is the required approximation to the funotion
  P(K1, K2, r);

        PRINT;

3: if F=0 then go to 1 else go to 3; end; end.
```

References

1. TEISSEYRE R.: New Method of Calculating Three-layer Curves for Geoelectric Methods, Acta Geophisica Polonica Vol. VI, N° 2.

2. LEBIEDIEW N.N.: Punkcje specjalne i ich zastosowania, PWN Warszawa 1957, p. 129.

3. Ibidem, p. 141.

4. WILKES M.V., WHEELER D.J., GILL S.: The Preparation of Programs for an Electronic Digital Computer, Cambridge 1957.

Instytut Maszyn Matematycznych PAN
Algorytmy N° 1
© 1962.11

519.271.01:518.12
681.142

COMPUTATION OF THE NORMAL CUMULATIVE DISTRIBUTION FUNCTION

by Elżbieta PLESZCZYŃSKA

Received January 1962

The paper contains effectively computed coefficients which allow convenient approximation of normal distribution function. The method for obtaining these coefficients is also given.

The normal cumulative distribution function with parameters 0 and 1 has the form:

$$F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt.$$

It may be approximated with accuracy to $\varepsilon = 3 \cdot 10^{-7}$ by means of the formula:

$$F^*(x) = \frac{1}{2}\left(\frac{1}{1 + a_1\left|\frac{x}{8}\right| + \ldots + a_6\left|\frac{x}{8}\right|^6}\right)^{16} \quad \text{for } x \leqslant 0 \qquad /1/$$

where

$$\begin{aligned} a_1 &= 0{,}3989387757 \\ a_2 &= 1{,}3530243935 \\ a_3 &= 1{,}6781446780 \\ a_4 &= 0{,}1556626432 \\ a_5 &= 1{,}6020483488 \\ a_6 &= 1{,}4111145983. \end{aligned}$$

For $X > 0$ the relation

$$F(x) = 1 - F(-x)$$

is used.

As $\varepsilon = F(-5)$, we accept $F(X) = 0$ for $X < -5$.

The formula (1) is suitable for programming for XYZ and ZAM-2 digital computers which have a fixed point and a long word consisting of 35 bits, as:

a. each coefficient a_i is comprised in one long word of the computer,

b. the expression $A = \dfrac{1}{1 + a_1 \left|\frac{x}{8}\right| + \ldots + a_6 \left|\frac{x}{8}\right|^6}$,

which should be raised to the power 16, is comprised within the interval $(0,1)$ for $X \neq 0$; it may therefore be given in a zero scale. If the numerator of this expression is replaced by $1 - 2^{-35}$, we always have then $0 < A < 1$, the correctness of the computation not being reduced.

The computation of a single $F(X)$ value in the XYZ computer, which is programmed by means of the above method, lasts about 0,12 sec.

The formula (1) is obtained by means of a transformation applied on the approximation formula for the function:

$$\Phi(X) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt,$$

given in [1].

The PROGRAM in ALGOL

```
procedure   NORMAL DISTRIBUTION (X) RESULT TO: (Y) ; value  x;
begin array A[0:6]; real Y, M, Z; integer i ; if x<-5 then Y :=0 else
      begin A[0]:= +1  ;
            A[1]:= +0.3989387757 ;
            A[2]:= +1.3530243935 ;
            A[3]:= +1.6781446780 ;
            A[4]:= +0.1556626432 ;
            A[5]:= +1.6020483488 ;
       M := A[6]:= +1.4111145983 ;
            Z := abs (x/8)
for  i := 5 step -1 until 0 do  M := M × Z + A[i] ;
      Y := +0.5 × (1/ M) ↑ 16 end ;
   if  x ⩽ 0  then go to END else
       Y := 1 - Y
END: end of the procedure;
```

References

1. HASTINGS C.: Approximations for Digital Computers, Princeton 1955, p. 185.

THEORY OF PROGRAMMING

Instytut Maszyn Matematycznych PAN
Algorytmy № 1

518.6

TRANSLATION OF ARITHMETIC FORMULAE IN SAKO

Jan BOROWIEC

Received January 1962

The paper presents the method for translating arithmetic formula, used in SAKO language translators for digital computers XYZ and ZAM-2. The local strongest operations /and functions/ are chosen in the given formula, then the elementary operations are constructed and sequentially programmed. Principles of the optimization of resulting program are also given.

Introduction

The translation of arithmetic formulae was many times elaborated because of its importance for automatic programming. The method accepted in FORTRAN 58 [1], [2], also the stack method [3], [4] should be distinguished. The method applied in FORTRAN is of a rather historical significance, and therefore it will not be considered in the present paper.

The stack method consists in chosing successive charakters of the formula and in constructing the resulting program with a simultaneous recurrent adjustment of working places.

In the case of relatively simple formulae, the above method permits the optimal use of working registers; analogous operations cannot be discovered as they are erased immediately after the segment of the resulting program has been written; the resulting program and its operation time become longer.

The translator, operating at stack principle, becomes notably complicated when indexed variables, arbitrary functions, arbitrary indexed expressions and function arguments are introduced into the arithmetic formula.

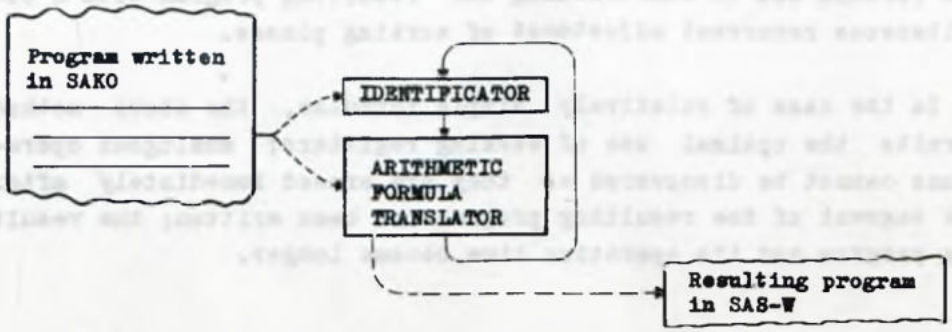
These disadvantages of the stack method induced the further described method to be accepted and used in arithmetic formulae translation for the SAKO language [5], [6], [8].

The present paper presents only the general idea of the method without considering technical details, as for instance: the application of ARITHMETIC FORMULA TRANSLATOR in a fixed point computer /which it was realized for/, the division of data into long and short ones, and also the capacity of internal storage of the computer is supposed to be not limited. All these limitations, due to technical parameters of a definite computer, do not cause qualitative changes in arithmetic formula translation:they only complicate or simplify it in details.

1. General principles

After meeting an arithmetic formula in a SAKO program, the program IDENTIFICATOR [7] transfers the control to the ARITHMETIC FORMULA TRANSLATOR program which is designedted to write the possibly optimal program realizing the given formula in SAS-W [7]. The ARITHMETIC FORMULA TRANSLATOR then transfers the control to the IDENTIFICATOR.

These dependences may be presented in the following graphic form:



Translation of arithmetic formulae is performed in three stages:

1. Formulae analysis and standardization.

2. Writing of the resulting program in SAS-W.

3. Optimization of the resulting program.

2. Formula analysis and standardization

When the ARITHMETIC FORMULAE TRANSLATOR intercepts the control, the formula appears in the internal storage of the computer, and each formula character is written in one word* /according to the way of reading information from the tape/. The part of the working storage designated to store the formula is called LINE Z. Individual characters or character groups in the formula may be classified as follows: variable, indexed variable, short number, long number, left brackets, right brackets, simple operation, language function, and defined function.

The analysis of the formula consists in the classification of individual characters or of their groups according to the above-mentioned way.

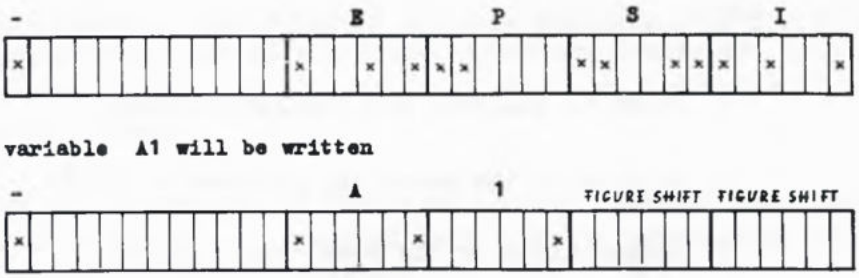
The standardization of the formula consists in a determined interpretation of the given character or of a group of characters and in copying the obtained information onto another part of the working storage called LINE LC.

The VARIABLE is replaced by a four-character group written in one long word** with the sign minus.
Examples:
Variable: EPSILON
will be written in LC in the form

---

* One short word = storage address = eighteen bits

** One long word = two storage addresses = thirtysix bits



OPERATION. LANGUAGES FUNCTION. DEFINED FUNCTION.

Simple operations, language functions and defined functions are called o p e r a t i o n s. A certain number, called the value of the given operation, is subordinated to each operation.

The table of values of an individual operation is given below.

SIGN	OPERATION	DIGITAL VALUE
,	Substitution	
.	Sign of equality	0
IND	Indexing	1
INP	Indexing in subroutines	2
+	Addition	3
-	Subtraction	4
/	Division	5
x	Multiplication	6
*	Raising to a power	7
	Language functions	8 - 31
	Defined functions	32 - 63

The operation number of a given operation is the digital value of the operation plus n.64 , where n is the difference between the number of left and right brackets preceeding this operation.

The operation number increases by 64 after the appearance of the left bracket on the left side of the operation, and it decreases by 64 after the appearance of the right bracket on the left side of the operation.

Example: In the expression

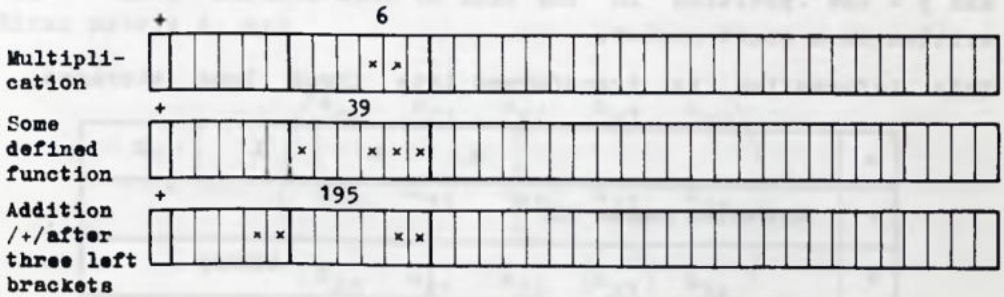
A + B = (C + D) + E ... ,

the operation numbers /written in brackets instead of the operation/ will be the following

A (3) B (6) C (67) D (3) E .... ;

Operations /i.e., simple operations, language functions and defined functions/ copied from LINE Z in LINE LC, are replaced by their binary numbers and written in one long word with the sign plus.

Example



LEFT BRACKET. RIGHT BRACKET.

Left bracket and right bracket are not copied on LINE LC. They only influence operation numbers appearing on the right side of the given brackets.
The SHORT NUMBER is converted to binary system and written on LC in one long word with the sign plus in binary scale 35.

Examples:

The LONG NUMBER is transmitted to the NUMBER TRANSLATING PROGRAM, which converts it to binary system /in an appropriate scale /, places it in the LIST OF LONG NUMBERS /called x v L D /, and gives the answer in the following form

+		x	v	L	D
+	.. ..		p		

where xvLD is written as a variable / but with the sign plus/, and p - the position in the LIST OF LONG NUMBERS xvLD - is written as a short number.

This information is transformed into three long storages:

-		x	v	L	D
+	operation number IND				
+		p - binary			

and written in LINE LC according to the above-described rules.

INDEXED VARIABLE

As it is known, the LIST OF INDEXED VARIABLES, called LW, is built on the basis of declarations: DIMENSION, BLOCK, STRUCTURE. Besides the name of the indexed variable, this list includes other information needed, for instance, for the realization of the algorithm of indexing, such as the number of the dimension of this variable and its number of words in each dimension. Let the n dimensional indexing variable

V A R I A B L E (I, J, K, L, ...,M)

have

i words in the first dimension
j words in the second dimension
k words in the third dimension
l words in the fourth dimension
.....;

the algorithm of indexing will then have the following form:

VARI (IND $\underbrace{((\ldots}_{\text{n brackets}}$ (I) x j + I) x k + K) x l + L)...+M))

Numbers j, k, l, ... and the amount of left brackets after the operation IND / equal to the dimension number of indexed variables/ is taken from the OF INDEXED VARIABLES.

Example:

Given matrix A, n=2

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} \end{pmatrix}$$

Let us find a_{23} by means of the algorithm of indexing:

i = 3 as it changes from 0 to 2,

j = 5 as it changes from 0 to 4,

I = 2,

J = 3.

A(2,3)= A(IND((2)x5 + 3)= A(IND(13)).

Indeed, a_{23} is the 13-th element of the matrix A if a_{00} is considered its zero element, and if it is counted in lines. INDEXED VARIABLE is copied from LINE Z on LINE LC in the following way:
The name of the INDEXED VARIABLE is written the same way as in the case of a simple Variable /see page 3/.

Operation IND, x and + are written as simple operations, left and right brackets being considered.

The values j, k, l, ..., being short numbers,are written as short numbers: those of them which are variables are written as variables.

Indexes I, J, K, ... may be arbitrary expressions; each expression value must therefore be copied according to corresponding rules.

3. Writing the resulting program in SAS-W

When the analysis and standardization of the formula are finished, variables, operations /simple operations, language functions, and defined functions/ and short words appear in long words on LINE LC. Each of the above values is written in one long word as given in 2.

Example: Let the following formula be on LINE Z:

F (X,Y) = A × X * 2 + (SUM (X,Y) × B) + C (ALFA) + Y * N

where

F is a defined function of value 45,
X, Y, B, ALFA, N are variables,
SUM is a defined function of value 46,
C is an indexed variable.

It is stated that variables and short numbers will be written below, in brackets, and operations in parentheses. Each value, written in brackets on LINE LC, occupies one long register. The given formula will then be copied on LINE LC in the following form:

(45)(64)[x](64)[Y](0)[A](6)[x](7)[2](3)(110)(128)[x](128)[y]
(70)[B](3)[C](65)[ALFA](3)[Y](7)[N]

On the basis of the given operation number and of its neighbouring operation numbers, it will be possible to state whether the RESULTING PROGRAM, realizing this operation, should be actually written, or whether a stronger*) operation should first be performed.

The given operation will be assumed evaluable if it is not one of the oited-below exceptions and if it fulfills the two following conditions:

a. its number**) is greater or equal to the nearest operation number on the right side,

b. its number is greater or equal to the nearest operation number on the left side.

If the given operation is on the utmost left /on the utmost right/ of the formula operation, fulfillment of condition a. /condition b./ will be sufficient to consider it as evaluable.

---

*For instance, in the expression A + B × C, multiplication must be done first, and then A may be added to the result B × C.

**Definition of the number is given on page 40.

Exceptions: Operations with the number N cannot be performed on the left side of the equality sign if this number fulfills the following condition:

$$0 \leq N \leq 66.$$

The operation of substituting arguments k. 64** in the function cannot be performed on the right side of the equality sign /operation 0/ if the function itself is one of this function arguments.

Examples:

LINE Z	F (X + Y, Z) = ...	
LINE LC	(F) (64) [X] (67) [Y] (64) [Z] (0) (67) is a performable operation	F - defined function with value (F); X,Y,Z - variables
LINE Z	TRY (ALFA, BETA) = ...	TRY - defined function with value (TRY); ALFA, BETA - variables
LINE LC	(TRY) (64) [ALFA] (64) [BETA] (0) .... (64) are not performable operations	
LINE Z	A = B + C + D × E	A, B, C, D, E - variables
LINE LC	[A] (0) [B] (3) [C] (3) [D] (6) [E] is a performable operation	
LINE Z	T = TRY (U,V,TRY (X,Y,Z))	TRY - defined function with value (TRY); U,V,X, Y,Z - variables
LINE LC	[T] (0) (TRY) (64) [U] (64) V (64) (64 + (TRY) (128) [X] (128) [Y] (128) [Z] not performable operations performable operations	

It is obvious from the above examples that the given formula may contain simultaneously several performable operations. In order to execute them successively, i.e., to write the resulting program realizing these operations, the following definition must be introduced:

The given operation will be called the local strongest one on LINE LC if it is the first performable operation from the left side.

The following definitions are inroduced:

[P] - the class of operation arguments, i.e., information units, on which this operation is realized

(0) the class of performable operations

[R] the class of working registers

} written in convention LC.

Let us consider the record in LC:

$$[P]^m \quad (0) \quad [P]^n$$

Notation $[P]^k$, where k is a natural number, indicates that parameters of the given operation appear k times /in neighbouring long registers/ on LINE LC.

For $m = n = 1$ there is a simple operation.

For instance:

[A] (3) [B] ;
[Z] (1) [5] ;
[L] (0) [TARA] ; and so on.

For $m > 1$, $n = 1$, we have to deal with the substitution $(k \cdot 64)$ /where k is a natural number/or with the equality sign (o) .

For instance:

(F)(64)[X](64)[Y](64)[Z] ; substituting of Z into the function F, $m = 5$

[A](65)[I](o)[P] sending P to the I-st place in A, $m = 3$

For $m = n = o$, we deal with the performance of a function in which arguments are already substituted.

Notation:

$$[P]^m \quad (0) \quad [P]^n$$

will be called - elementary operation.

Each elementary operation corresponds to a set of instructions SAS-W realizing this operation. This set is called the sequence of the given operation.

Examples of several sequences are given below.

For elementary operation $\underset{[RA]}{[A](3)[B]}$ the corresponding sequence is

Load Acc	A
Add	B
Store Acc	RA

For elementary operation $\underset{[R3]}{[ZAK](6)[STAR]}$ the corresponding sequence is

Load MpR	ZAK
Multiply	STAR
Store Acc	R3

For elementary operation $\underset{[R7]}{[X](5)[Y]}$ the corresponding sequence is

Load Acc	X
DIVIDE	Y
Store MpR	R7

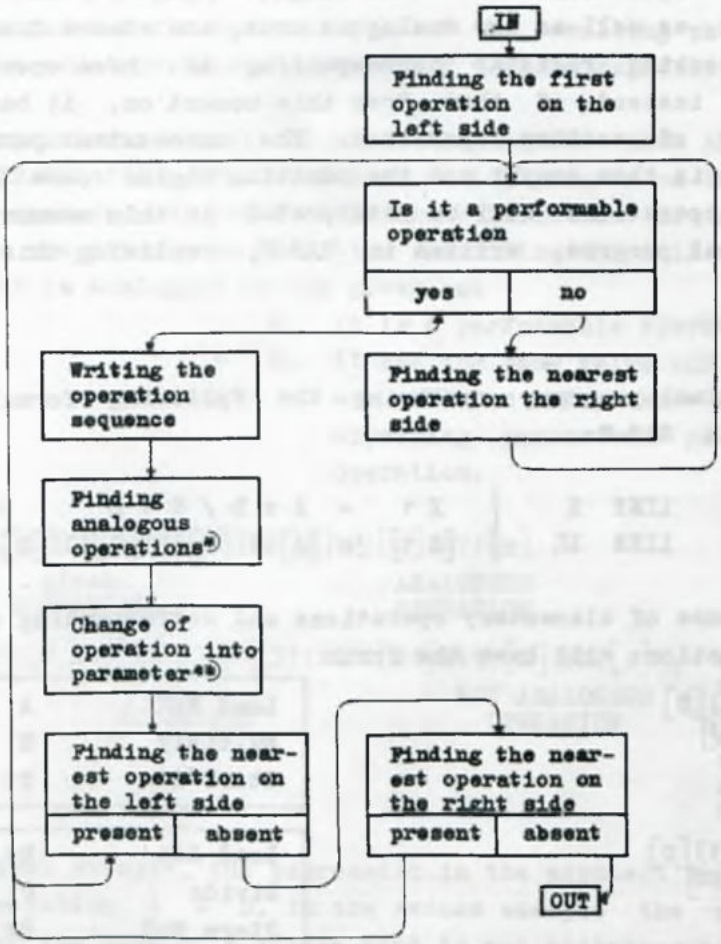
Let the defined function TOR have the value 38. Then for elementary operation $\underset{[R0]}{(38)}$ the corresponding sequence is

Load Acc	+ 0
Jump	* TOR
Store Acc	R0

and so on.

In this way the problem of programming arithmetic formulae is reduced to that of programming elementary operations.

The following flow-diagram presents the way chosen:



*See description of definition of analogous operations /see 4, p.51/

**After the operation sequence is written, the symbol of the working storage corresponding to this operation is recorded in its place. In all places of analogous operations, the symbol of the working storage is written and since this moment it becomes the parameter of another operation.

Thus the sequence is written for each performable operation. Analogous operations are then sought * /page 51 / and the given operation, as well as the analogous ones, are erased from LINE LC and a working register corresponding to these operations is recorded instead of them. From this moment on, it becomes the parameter of another operation. The successive performable operation is then sought and the handling begins once more. All formulae operations will be interpreted in this manner into the non-optimal program, written in SAS-W, realizing this formula.

Example:

A non-optimal program, realizing the following formula, is written in SAS-W:

LINE Z	X 1 = A x B / C + D
LINE LC	[X 1] (o)[A](6)[B](5)[C](3)[D]

The sequence of elementary operations and corresponding sequences of instructions will have the form:

[A](6)[B] [R1]	Load MpR A Multiply B Store Acc R1
[R1](5)[C] [R2]	Load Acc R1 Divide C Store MpR R2
[R2](3)[D] [R3]	Load Acc R2 Add D Store Acc R3
[X1](o)[R3]	Load R3 Store Acc X1

4. Optimization of resulting program

The purpose of the optimization of the resulting program is to shorten it and to reduce the number of working registers.

The optimization starts when the part of the ARITHMETIC FORMULA TRANSLATOR /writing the resulting program on the basis of LINE LC/ operates. It consists in the elimination of so-called analogous operations. An operation fulfilling the following two conditions is analogous to the given one

a. it is a performable operation,
b. it has the same value and parameters as well as the same sequence of appearing parameters as the given operation.

Examples:

[X](o)[A](3)[B](3)[C](5)[E](7)[A](67)[B] ,
given OPERATION — ANALOGOUS OPERATION

[Z](65)[P](o)[L](4)[K](3)(SQR)[X1](69)[L](68)[K] , where (SQR) is the operation number of taking a square root
given OPERATION — NOT ANALOGOUS OPERATION

In the first example, the expression in the exponent is analogous to the operation A + B. In the second example the operation L-K under the sign of a square root is not analogous to the given operation, as a not performable one.

The elimination of analogous operations consists in storing the working register in LINE LC in the place of the given operation, after the sequence-realizing the given operation has been written, and in the place of all operations-analogous to the given one.

Example:

Let us consider the formula:

LINE Z | Z = X * 2 + SIN (X * 2 + 1) / (X - (X * 2 - 1) x Y)
LINE LC | [Z](0)[X](7)[2](3)(SIN)[X](71)[2](67)[1](5)[X](68)[X](135)[2](132)[1](70)[Y]

where (SIN) is the operation taking sinus elementary operation being interpreted

[X](7)[2]
[R7]

After the program finds analogous operations to the given one, LINE LC has the form:

[Z](o)[R7](3)(sin)[R7](67)[1](5)[X](68)[R7](132)[1](70)[Y] ,

(SIN) is an operation of computing sinus, as above,

and only the following sequence will then be added to the RESULTING PROGRAM:

Load MpR	X
Multiply	X
Store Acc	R7

The further optimization is carried over the finished RESULTING PROGRAM, written in SAS-W.

This is the third stage of ARITHMETIC FORMULAE TRANSLATION. It consists in:

a. erasing needless instructions on the sequence contacts;
b. replacing some sequences of RESULTING PROGRAM instructions by shorter ones but realizing the same function:
c. erasing needless working storages and corresponding instructions of the RESULTING PROGRAM.

An example of erasing some needless instructions on sequence contacts is given:

Let us write the program for the formula:

LINE | Z Z = A + B + C
LINE | LC [Z](o)[A](3)[B](3)[C]

Let us write the sequence of elementary operations:

[A](3)[B] the corresponding sequence
[R1] SAS-W is

Load Acc	A
Add	B
Store Acc	R1

[R1](3)[C] the corresponding sequence
[R2] SAS-W is

Load Acc	R1
Add	C
Store Acc	R2

The instruction Load Acc R1 will be erased on the contact of sequences as non-effective.

Let us write the program for formula:

LINE Z C =(CAT / MOUSE) x FENCE FENCE
LINE LC [C](o)[CAT](69)[MOUSE](6)[FENCE]

Let us write the sequence of elementary operations and the corresponding sequences of SAS-W instructions:

[CAT](69)[MOUSE]
[RA]

Load Acc	CAT
Divide	MOUSE
Store MpR	RA

[RA](6)[FENCE]
[RB]

Load MpR	RA
Multiply	FENCE
Store Acc	RB

The instruction Load MpR RA will be erased on contacts of the sequences as a non-effective.

An example of changing the fragment of the RESULTING PROGRAM into a shorter fragment realizing the same function will now be given.

Let us write the program for the formula:

LINE Z T = Y + (A + B)
LINE LC T (o) Y (3) A (67) B .

Let us write the sequence of elementary operations and the corresponding sequences:

[A](67)[B] [R1]	Load Acc	A	may be changed	Load Acc	A
	Add	B	into the program	Add	B
	Store Acc	R1		Store Acc	R1
[Y](3)[R1] [R2]	Load Acc	Y		Add	Y
	Add	R1		Store Acc	R2
	Store Acc	R2			
[T](o)[R2]	Load Acc	R2			
	Store Acc	T		Store Acc	T

To reduce the number of working registers and instructions in the resulting program connected with the above mentioned working registers, the storing instructions from separate machine registers /the Accumulator and the Multiplying Register/ are considered.

If the working register, storing the content of separate registers of on arithmetic unit, does not appear in the RESULTING PROGRAM anywhere below, the instruction Store Acc or Store MpR may be erased from the program and the working register discharged.

Let us consider the last example:

In the program	Load Acc	A
	Add	B
	Store Acc	R1
	Add	Y
	Store Acc	R2
	Store Acc	T,

let us consider accumulator storing instructions Store Acc R1 and Store Acc R2 in the working storage. As the working storages R1 and R2 do not appear any more below the instructions may be erased and the program written in the following form:

```
Load Acc    A
Add         B
Add         Y
Store Acc   T,
```

which is obviously the optimal program for realizing the formula written.

Conclusion

It ought to be emphasized that the necessity of simplifying the translator induced some disadvantages of the discussed method. They are: the optimization of the resulting program not being fully realized /this problem seems to be not yet solved/; the limitation of the lenght of the formula which is due to the storage capacity ; a small number of identifying characters of variables /four/ and of functions /three/; the signalization of syntactic errors not being sufficiently extended.

However, a liberty of writing arithmetic formulae / indexed variable, language and defined functions, arbitrary superpositions of expressions and so on/, also a significant optimization of the lenght of the resulting program seem to prove rather serious advantages of the described method.

References

1. BACKUS J.W., BETER R.J. et al : The Fortran Automatic Coding System, Proc. of the Western Joint Comp. Conference, 1957:188.
2. IBM FORTRAN 58 Automatic Coding System, Reference Manual.
3. SAMELSON K., BAUER E.L.: Sequential Formulae Translation, Comm. ACM 1960:3,2,76.
4. DIJKSTRA E.W.: Recursive Programming, Num. Math. 1960:2,5,312.
5. ŁUKASZEWICZ L.: SAKO - An Automatic Coding System, Annual Rev. in Autom. Program, 1961:2.

6. MAZURKIEWICZ A.: Arithmetic Formulae and the Use of Subroutines in SAKO, Annual Rev. in Autom. Program, 1961:2.

7. SWIANIEWICZ J., SAWICKI S.: SAKO Translation, preprint. Presented at the Conf. on Automatic Programming Warsaw, Sept. 1961.

8. SZORC P.: Subroutines in SAKO, preprint. Presented at the Conf. on Automatic Programming. Warsaw, Sept. 1961.

Instytut Maszyn Matematycznych PAN
Algorytmy N° 1

518.6
681.142

O TRANSLACJI FORMUŁ ARYTMETYCZNYCH SAKO

Alfred SCHURMANN

Pracę złożono
w styczniu 1962 r.

Podano zastosowanie metody stosu do translacji formuł arytmetycznych SAKO, zawierających funkcje i zmienne indeksowane.

1. Wstęp

Praca niniejsza podaje zastosowanie metody stosu, znanej z pracy [1] do translacji formuł arytmetycznych SAKO zawierających funkcje i zmienne indeksowane.

Pierwsza część pracy obejmuje opis translacji formuł arytmetycznych SAKO metodą stosu ze szczególnym uwzględnieniem translacji funkcji i zmiennych indeksowanych. Część druga zawiera opis zastosowanej metody w odniesieniu do funkcji i zmiennych indeksowanych.

2. Translacja formuł arytmetycznych

2.1. Formuły arytmetyczne są zdefiniowane jak w języku SAKO [2], z tym, że wprowadzono odróżnienie między nawiasami funkcyjnymi / kwadratowe [....]/ i nawiasami zmiennych indeksowanych / trójkątne <...>/.

Definicja wyrażenia arytmetycznego w SAKO zawiera między innymi następujące elementy:

a. Jeśli G jest nazwą bloku, A, B, ..., Z są wyrażeniami przyjmującymi wartości całkowite i ilość tych wyrażeń jest równa ilości indeksów danej zmiennej indeksowanej G, wówczas

$$G\langle A,\ B,\ \ldots,\ Z\rangle$$

jest również wyrażeniem.

b. Jeśli F jest nazwą funkcji, A, B, ..., Z są wyrażeniami w ilości zgodnej z argumentami tej funkcji, wówczas

$$F[A,\ B,\ \ldots,\ Z]$$

jest również wyrażeniem.

Symbole użyte w formule arytmetycznej można podzielić na

a. zbiór znaków działań,

b. zbiór parametrów.

Parametrami są: liczby, zmienne proste, zmienne robocze, nazwy funkcji, nazwy bloków. Znakami działań są pozostałe znaki. Jeżeli nazwa funkcji lub nazwa bloku jest użyta jako argument funkcji lub zmiennej indeksowanej, wtedy stawia się przed nazwą odpowiednio znak ↑ lub →, np:

↑ FUN , → BLOK .

Blok jednowymiarowy będziemy nazywali linią. Zakładamy, że formuła zapisana jest na linii Z w sposób następujący: w jednym elemencie linii Z mieści się dokładnie jeden symbol formuły / parametr lub znak działań /, w Z (L) znajduje się, licząc od zera, L-ty symbol formuł. Przystępując do translacji formuły, będziemy tworzyć: linię parametrów LP, linię znaków działań LD oraz linię miejsc roboczych MR.

Translację formuł rozpoczynamy od początku linii Z następującą metodą:

jeżeli badany symbol na linii Z jest parametrem, to zapisujemy go do pierwszego wolnego miejsca za wierzchołkiem stosu* LP / inaczej: na wierzchołek stosu LP/ i przechodzimy do badania następnego znaku linii Z;

gdy badany symbol na linii Z jest działaniem, wtedy w zależności od działania, które znajduje się w wierzchołku stosu LD, wykonujemy czynności podane w Tablicy 1.

Tablica 1

Badany symbol formuł arytm. w (Z)L jest	Wierzchołek stosu LD - LD(S)*) zawiera	Wykonywane czynności
= ([,	zawartość w LD(S) nie jest badana	S = S + 1; przepisanie działania z Z(L) do LD(S); L = L + 1
+ -) ,] > ϕ	+ - / x *	wykonanie działania zawartego w LD(S); S = S - 1
+ -	, [< (	S = S + 1; przepisanie działania z Z(L) do LD(S); L = L + 1
x /	x * /	wykonanie działania zawartego w LD(S); S = S - 1
x / *	, ([< = + -	S = S + 1; przepisanie działania z Z(L) do LD(S); L = L + 1
*	*	wykonanie działania *; S = S - 1
/	/ x	S = S + 1; przepisanie działania z Z(L) do LD(S); L = L + 1
,	, [<	S = S + 1; przepisanie działania z Z(L) do LD(S); L = L + 1
]	,	wykonanie funkcji
>	<	wykonanie indeksowania zmiennej
)	(	S = S - 1; L = L + 1
→ ↑	zawartość w LD(S) nie jest badana	J = J + 1; przepisanie nazwy występującej za znakiem → lub ↑, do LP(J) i zaznaczenie, że w LP(J) jest nazwa wektora lub funkcji
ϕ	=	wykonanie znaku równości

Symbol ϕ oznacza koniec formuły arytmetycznej.

* Określenie zasady stosu znajduje się w pracy [1] s. 76-79.

Wykonanie translacji działań +, -, x, / oraz potęgowania opisane jest w pracy [1]. W związku z tym ograniczymy się do opisu translacji funkcji oraz zmiennych indeksowanych.

2.2. Z powyższego opisu wynika, że funkcję lub zmienną indeksowaną z obliczonymi argumentami, bezpośrednio przed kompilacją mamy zapisaną w sposób następujący *):

W stosie LD, poczynając od wierzchołka, zapisane są wszystkie przecinki, które oddzielały argumenty funkcji lub zmiennej indeksowanej. Bezpośrednio za tymi przecinkami znajduje się nawias /otwierający/ funkcji lub zmiennej indeksowanej. Ilość tych przecinków plus jeden równa się ilości argumentów danej funkcji lub zmiennej indeksowanej. Zmienne przyjmujące wartości tych argumentów zapisane są w stosie LP od wierzchołka w głąb stosu. Zmienne te będziemy dalej nazywali zmiennymi podstawiania.

Kolejność zmiennych jest następująca: zmienna pierwsza z wierzchołka stosu LP przyjmuje wartość ostatniego argumentu funkcji lub zmiennej indeksowanej, zmienna druga z wierzchołka - wartość przedostatniego argumentu funkcji lub zmiennej indeksowanej, itd. Bezpośrednio za zmiennymi podstawiania znajduje się nazwa funkcji lub nazwa zmiennej indeksowanej.

Wykonanie funkcji

Wartości zmiennych podstawiania z wierzchołka stosu LP podstawiamy do podprogramu o nazwie występującej w stosie LP za tymi zmiennymi. Wartościami zmiennych podstawiania mogą być również nazwy bloków i nazwy funkcji.

Stosy LD i LP obniżamy w zależności od ilości argumentów funkcji. Stos MR obniżamy w zależności od ilości zmiennych

---

* problem ten jest szerzej omówiony w 2.

roboczych, występujących wśród zmiennych podstawiania. Po tych czynnościach następuje wywołanie podprogramu obliczającego tę funkcję. Wynik obliczonej funkcji pamiętamy na stosie MR, a jego adres w stosie MR pamiętamy na stosie LP; przechodzimy do dalszego badania linii Z z następnym znakiem formuł arytmetycznych.

Wykonanie indeksowania zmiennej.

Otrzymane zmienne podstawiania są indeksami danej zmiennej indeksowanej. Jeżeli znamy zakresy tych indeksów, to indeksowanie tej zmiennej możemy wykonać przy pomocy tzw. algorytmu indeksowania.*)

Stosy LD, LP i MR obniża się tak, jak przy wykonywaniu funkcji. Nazwę zmiennej indeksowanej z obliczonym indeksem wpisuje się na wierzchołek stosu LP; jest ona odtąd traktowana jak zmienna prosta.

Dalsze badanie linii Z rozpoczynamy z następnym znakiem formuł arytmetycznych.

Niżej podany przykład ilustruje opisaną metodę translacji funkcji.

Niech na linii Z dana będzie formuła arytmetyczna

$$W = Z \times P\left[G\langle A + C,\ D\rangle,\ P\left[A,\ H[X]\right]\right]\phi$$

Każdy wiersz tablicy 2 zawiera: adres aktualnie badanego symbolu linii Z , aktualny stos LP, aktualny stos LD oraz aktualnie wykonywane działanie. Stany translatora, przedstawione w wierszach tablicy 2, powstają w wyniku analizy linii Z oraz stanu translatora, odpowiadającego poprzedniemu wierszowi tablicy 2. Symbol LP (3) oznacza adres 3-go elementu / licząc od zera/ na linii LP. Warto zwrócić uwagę na fakt, że w przykładzie tym korzysta się dwa razy z podprogramu obliczającego

---

* Algorytm ten podany jest w pracy [2] s. 30-31

funkcję F; za każdym razem podstawia się do niej inne argumenty.

Niech blok G ma wymiary 80, 100.

Tablica 2

L	Stos LP	Stos LD	Wykonanie działania
1	W	=	
3	W Z	= x	
5	W Z F	= x[	
7	W Z F G	= x[<	
9	W Z F G A	= x[<+	
10	W Z F G A C	= x[<+	
11	W Z F G	= x[<	MR/0/ = A + C
11	W Z F G MR/0/	= x[<,	
12	W Z F G MR/0/D	= x[<,	
13	W Z F G	= x[	MR/0/ = MR/0/x 100 + D
13	W Z F G	= x[	MR/0/ = LP/3/ + MR/0/ = G/MR/
14	W Z F MR/0/	= x[,	
16	W Z F MR/0/F	= x[,[	
18	W Z F MR/0/F A	= x[,[,	
21	W Z F MR/0/F A H X	= x[,[,[	
22	W Z F MR/0/F A	= x[,[,	MR/1/ = H[X]
22	W Z F MR/0/F A MR/1/	= x[,[,	
23	W Z F MR/0/	= x[,	MR/1/ = F[A, MR/1/]
23	W Z F MR/0/ MR/1/	= x[,	
24	W Z	= x	MR/0/ = F[MR/0/, MR/1/]
24	W Z MR/0/	= x	
25	W	=	MR/0/ = Z x MR/0/
25	W MR/0/	=	
25			W = MR/0/

3. Opis metody rozwiązywania funkcji i zmiennej indeksowanej

Metoda rozwiązywania zmiennej indeksowanej jest ta sama co funkcji.

3.1. Załóżmy, że rozwiązywaną funkcją jest funkcja F1. Jej argumentami mogą być dowolne wyrażenia arytmetyczne. Rozwiązywanie rozpoczyna się od zapisania nazwy F1 na wierzchołek stosu LP. Nawias zostaje zapisany na wierzchołek stosu LD.

3.2. Jeżeli argumentem funkcji jest liczba lub zmienna prosta, wówczas zostaje ona zapisana na wierzchołku stosu LP za nazwą F1. Jeżeli za argumentem występuje przecinek, to zostaje on zapisany na wierzchołek stosu LD; przechodzimy do badania następnego argumentu funkcji F1. Gdy za argumentem występuje nawias zamykający, wtedy funkcję F1 mamy zapisaną zgodnie z opisem w 2.2. i następuje wykonanie funkcji. Pokazuje to poniższy przykład:

Formułę

$$A = FU\left[T,\ Rx\ (F+B),\ G\left[FU\left[R,S\left[A+L\right],\ K\right]\right]\right]$$

zapisujemy następująco:

Linia Z
Rx (F+B), G[FU[R,S[A+L], K]]

Stos LP
A FU T

Stos LD
=[,

3.3. Jeżeli argumentem funkcji F1 jest proste wyrażenie arytmetyczne AF1, nie zawierające funkcji i zmiennej indeksowanej, to wyrażenie to rozwiązuje się znaną metodą stosu dla prostych formuł [1] , z tym że stos parametrów tego wyrażenia tworzy się na linii LP od wierzchołka stosu LP wzwyż; analogicznie tworzy się dla wyrażenia AF1 stos znaków działań na linii LD i stos miejsc roboczych na linii MR.

Objaśnia to przykład:

Linia Z

$$), G\left[FU\left[R\ ,S\left[A+L\right],\ K\right]\right]$$

Stos LP

A FU T R F B

Stos LD

=[, x (+

Wynik obliczonego wyrażenia AF1 zapamiętuje się na wierzchołku stosu MR, a adres tego wierzchołka /adres ten jest zmienną podstawiania/ zapisuje się na wierzchołek stosu LP. Dalej postępuje się według opisu w 3.2.

Ilustruje to przykład:

Linia Z

$$G\left[FU\left[R\ ,S\left[A+L\right],K\right]\right]$$

Stos LP

A FU T MR(0)

Stos LD

=[, ,

Stos MR

MR(0) = R x (F+B)

3.4. Niech argumentem funkcji F1 będzie dowolne wyrażenie arytmetyczne AF1. Z wyrażeniem tym postępuje się w następujący sposób:

Dopóki w wyrażeniu AF1 nie występuje funkcja, traktuje się je tak, jak proste wyrażenie arytmetyczne /zob.3.3./. Gdy w wyrażeniu AF1 pojawi się funkcja, wtedy wykonuje się te same

czynności, co przy rozpatrywaniu funkcji F1, tzn. zapis nazwy funkcji do LP, nawiasu [do LD, itd.

Według powyższego opisu w części 2 dochodzi się w sposób rekursywny do funkcji, której argumentami są proste wyrażenia arytmetyczne. Istnienie takiej funkcji jest oczywiste na podstawie definicji formuł arytmetycznych.

Niech FN będzie nazwą tej funkcji. Z funkcją tą wykonuje się czynności, które zostały opisane w 3.1., 3.2. i 3.3. Funkcja FN zostaje wykonana.

W przykładzie przedstawia się to następująco:

Linia Z

$$, K]]]$$

Stos LP

A FU T MR(0) G FU R MR(1)

Stos LD

= [, ,[[,

Stos MR

MR(0) = R x (F+B)

MR(1) = S [A+L]

Po wykonaniu, funkcja FN została zredukowana do zmiennej prostej. Postępując dalej zgodnie z powyższym opisem, redukujemy funkcje typu FN do zmiennych prostych, a funkcje o argumentach zawierających funkcje typu FN redukuje się do funkcji typu FN. W ten rekursywny sposób dochodzi się do obliczenia wyrażenia AF1 oraz funkcji F1.

4. Uwagi końcowe

Metoda stosu niczym nie ustępuje metodzie dotychczas stosowa-

nej do translacji formuł arytmetycznych.

W obu metodach możliwa jest optymalizacja programu wynikowego przez wykreślenie z formuły tzw. analogicznych operacji.

Zalety opisanej metody:

1. znaczne skrócenie programu tłumaczącego formułę arytmetyczną na język maszyny,
2. skrócenie czasu translacji formuły,
3. optymalne wykorzystanie pamięci roboczej.

Literatura

1. SAMELSON K., BAUER F.L.: Sequential Formula Translation, Comm. ACM 3, 1960, s. 76-83.
2. Praca zbiorowa, System Automatycznego Kodowania SAKO, Prace ZAM, Ser. C2, 1961, s. 36-38, 63.

ON TRANSLATION OF ARITHMETIC FORMULAE

Summary

The translation of arithmetic formulae SAKO using stack method is described. If the arithmetic expression contains functions or indexed variables the arguments of the functions and indices are the arithmetic expressions.Symbols used in the formula are divided into parameters and operation signs. Numbers, simple variables,function names and names of indexed variables are parameters. The remaining symbols are operation signs.

The scheme of arithmetic formulae translation is presented as follows:

Let LP be the, so-called, parameter line, and LD - operation signs line. If the examined formula symbol is a parameter one writes it on the top of the staok LP and one begins to examine the next sign of this formula. If the examined formula symbol is an operation sign, then depending on the operation sign placed on the top of the stack LP, the action is performed according to table 1, e.g. the examined symbol is rewritten on the top of the stack LD, or the operation is performed from the top of the stack LD.

After having executed the operation, further examination of arithmetic formulae are to be performed.

THEORY OF COMPUTERS

Instytut Maszyn Matematycznych PAN
Algorytmy N° 1

518.6
681.142

SOME INFORMATION ABOUT AN ADDRESSLESS COMPUTER

by Włodzimierz OSTALSKI

Received January 1962

The paper proposes the formal language for general-purpose addressless digital computer. The proposed language does not represent a complete computer instruction set. It only determines instruction groups which should appear in it. Attention is drawn towards certain requirements in regard to the organization of such a computer, for instance to facilitate computing with double-length numbers.

Z. Pawlak in [1] and [2] describes the methods of designing an addressless computer, which may realize arithmetic expressions with substitutions and repetitions. Such a computer may constitute the basis for the development of an all-purpose or specialized digital computer, the instructions of which would be not ordinary ones but similar to those of some autocode systems.

Such a computer would be more convenient for mathematicians than a classical one.

A general concept of language for an addressless computer is given below. Symbols, formulas and current language expressions form the internal language of such a computer. The program is transfered from the input device into the memory directly, without being translated, the characters of no importance to the computer are neglected /i.e. space, line and so on/. The length of the instructions is not fixed but, of course, the number of /alpha-numerical/ characters is limited in each instruction.

The end of the instruction is denoted by a 'point' /the character CR may be this 'point' in a teletype code/.

Formal structure of language.

1. All words are written in capital letters, the entire expression being in inverted commas.

Examples:

'STOP'
'GO TO'
'IF' ... 'GO TO'
'READ'
'PRINT' ...'SIZE'
'PRINT TEXT'

2. Numbers are denoted by small letters. A single number is denoted by a letter or by a letter and number from 0 to 9. Some letters are distinguished as names of indexes /for instance - from i to n/.

Examples:

a, b, b1

Sets of numbers are denoted by small letters with indexes; indexes are written in square brackets.

Examples:

$a[i,j]$ $i = 1\ (1)\ t$ $j = 1\ (1)\ t$

signifies a quadratic matrix of dimension t.

$a[s,j]$ $j = 1\ (1)\ t$

signifies the s-th row of the matrix.

$b[i]$ $i = 1\ (1)\ p$ is a vector.

$a[s,t]$ is an element of the matrix $a[i,j]$ where $i = s$, $j = t$.

The shifting of a number a to the left or to the right is denoted by $a.10^p$.

3. Expressions and formulas may be numbered by natural numbers, their successiveness not being required.

Arithmetics.

Numbres are represented in binary-coded decimals. Arithmetical operations are carried out as well in fixed as in floating points. Only fifteen decimal digits are used.

One or two double-length registers, having the property of an accumulator with regard to multiplication and the possibility of adding to its first and second part /in fixed point operations only/, and also as an overflow position, must be distinguished from the memory in order to facilitate operations with double precision.

This being so, the subroutine of every arithmetic operation with double precision may be written as one instruction.

For instance:

Let the name of the above distinguished place be A, and two numbers of double-length: $\hat{a} = /a1, a2/$, $b = /b1, b2/$.

Then:

the sum is $A = /a2 + b2/ \cdot 10^{-15} + a1 + b1$

the product is $A = /a1 \cdot b2 + a2 \cdot b1/ \cdot 10^{-15} + a1 \cdot b1$

The subroutine for division is more complicated.

These instructions are standard formulas in the sense of [1].

Note

Z. Pawlak suggests a parenthesis-free notation of arithmetic formulas more convenient from the technical point of view. This does not matter in our case as we use the parenthesis notation for simplification.

Subroutines.

Short subroutines which may be included in one formula are denoted by single capital letters and called out as in [2]. Long subroutines are called out by the instruction:

'GO TO SUBR' XYZ .

This instruction induces the name /number/ of the next instruction to be located in the first free 'return register'. The last subroutine instruction is

'RETURN' ,

which causes a jump to the place noted in the 'return register' used last. This return register is simultaneously cleared out. Working places of every subroutine or standard formula are constant. Subroutines can be fixed in dead memory or be included in the program.

It is important that the main, operational and auxiliary storages /except when magnetic tape is present/ be treated as a whole.

References

1. PAWLAK Z.: Realization of the Rule of Substitution in the Addressless Computer, Bull. Acad. Polon. Sci, Sèr. Sci. Techn. 1961:9, 531-534.

2. PAWLAK Z.: Realization of the Rule of Substitution in the Addressless Computer without Working Memory, ibid. 1961:9 , 579-580.

Instytut Maszyn Matematycznych PAN
Algorytmy N° 1

517.11

ON NORMAL EQUIVALENTS OF TRUTH FUNCTIONS

by Stanisław WALIGÓRSKI
Received October 1962

The subject of the present paper is the extension of the problem of simplifying normal equivalents of truth functions formulated in [15]. The function defined on a set of normal expressions with values from a partly ordered set is introduced. The problem under consideration is to find normal equivalents of truth functions such that a value of γ be minimal. Partial orderings of normal expressions connected with simplifying of expressions in different ways are defined. Algorithms for solving this problem for functions isotone with respect to these orderings are included.

Introduction

Many papers have dealt with simplifying normal formulae of truth functions, and there exist quite a number of algorithms for solving this problem /refer for example to [2], [5] [7-17], [19], [20], [21] /. In most papers the simplicity of a formula is estimated according to a number of literals that this formula contains. When designing switching circuits a number of literals in formulae describing them can, in many cases, be regarded as sufficient estimation of the complexity of those circuits. It happens, however, that such estimation reflects neither the actual worth of these circuits nor their complexity. In such cases it is convenient to assign a number to every formula describing the switching circuit; this number would represent the worth of the circuit, a number of the elements included, etc. By this means we obtain a certain real function defined on the set of formulae, and designing will then be reduced to finding minimums of that function on sets of normal equivalents of truth functions.

The real function determined on the set of formulae maps it onto an ordered set of numbers, and in this way the worth of two arbitrary formulae can be compared. In some practical cases, however, comparison of the worth of certain formulae is rendered difficult or even proves to be useless. In such cases it is convenient to use the function that maps the set of formulae onto a certain partly ordered set instead of a real function.

Logical design of switching circuits that can be constructed of k different kinds of elements will be a good example of the practical application of the problem under consideration. In comparing different circuits we have to consider a number of elements of each kind that each circuit includes. To each circuit /hence to each formula as well/ is then assigned k- element sequence of natural numbers. A set of these sequences can be only partly ordered.

In such a case the problem of designing the circuit consists in finding normal expressions for which values of a function determined on the set of expressions are minimal.

Such extension of the problem of simplifying normal equivalents of truth functions is the objective of the present paper. The results given may be used in a particular case for finding equivalents with a minimal number of literals /i.e., of variables or negated variables/.

1. According to the terminology used by other authors finite alternation of conjunctions of a finite number of variables or negated variables will be termed alternative normal formula; it should be noted that conjunctions do not appear repeatedly in that alternation and in each conjunction does not appear twice in the same variable or negated variable. Since this paper deals with alternative formulae only, the word 'alternative' will be omitted for simplicity of wording. A set of all N-element combinations of zeros and ones will be denoted by B^N. This set is partly ordered in the following way: if $x = x_1, x_2, \ldots, x_N \in B^N$ and

$$y = y_1, y_2, \ldots, y_N \in B^N$$

then $x \leqslant y$ if and only if $x_i \leqslant y_i$ for $i = 1, 2, \ldots, N$, assuming that $0 \leqslant 0$, $0 \leqslant 1$, $1 \leqslant 1$, and not $1 \leqslant 0$.

It often happens in practice that the function of N-variables realized by a designed network for certain values of the sequence $x = x_1, x_2, \ldots, x_N$ can take the arbitrary value 0 or 1 ; in such cases the function is not defined. Such values are often named 'don't-care conditions' . For simplicity of wording we assume, if no particular reservations are made, that every normal formula has a determined value for all possible values of variables.

If the function $f(x)$ is defined on the set $P \subset B^N$ and maps P into B then every normal formula equal to f on P will be termed normal equivalent of $f(x)$. The function $f(x)$ determined on $P \subset B^N$ is called isotone if for arbitrary $u \in P$ and $v \in P$ from $u \leqslant v$ follows $f(u) \leqslant f(v)$. If for any pair $u, v \in P$ does not hold $u \leqslant v$ then every function determined on P is isotone.

2. Every isotone function has its normal equivalent that does not contain negated variables. Actually, if $f(x_1, x_2, \ldots, x_N)$ is an isotone function determined on B^N then the following formula is true

$$f(x_1, x_2, \ldots, x_N) = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_N) + \quad /1/$$

$$+ x_i f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_N). \quad i = 1, 2, \ldots, N$$

Hence

$$f(x_1, x_2, \ldots, x_N) = \sum_{j \in B^N} \prod_{i=1}^{N} (x_i + \bar{j}_i)\, f(j) \qquad /2/$$

where — denotes a negation, + and $\sum$ denote an alternation, and $\prod$ denotes a conjunction.

In the formula on the right side of equality /2/ no variable x_i is negated.

Denote

$$p_j(x) = \prod_{i=1}^{N} (x_i + \bar{j}_i) \qquad /3/$$

where $j \in B^N$, $x \in B^N$.

If $j \neq 00 \ldots 0$, then $p_j(x)$ is a conjunction of all variables x_i with the indices i such that $j_i = 1$. If $j = 00 \ldots 0$, then $p_j(x) = 1$ for all $x \in B^N$.

The formula /2/ can be now expressed as follows

$$f(x) = \sum_{j \in B^N} p_j(x) f(j) = \sum_{j \in \underline{f}} p_j(x) \qquad /4/$$

where $\underline{f}$ is a set of all $j \in B^N$ such that $f(j) = 1$.

If the function $g(x)$ is determined and isotone on the set $P \subset B^N$, $P \neq B^N$ then exists the isotone function $f(x)$ determined on B^N such that $f(x) = g(x)$ for $x \in P$. It implies the existence of a normal equivalent of $g(x)$ that does not contain negated variables. Any normal formula that does not contain negated variables can be expressed as follows:

$$\sum_{j \in A} p_j(x) \qquad /5/$$

A can be an arbitrary non-empty subset of B^N. Since the assignment of formulae to the subsets of B^N is a one-one correspondence we can consider instead of formulae the sets corresponding to them; likewise, instead of conjunctions, the appropriate elements of B^N can be considered.

The problem of simplifying normal formulae containing negated variables can be reduced to the analogous problem dealing with formulae not containing negated variables. On the other hand, the problem of finding normal equivalents of an arbitrary function can be reduced to the same problem for isotone functions. For that purpose it will be satisfactory to treat all literals, i.e., variables and negated variables, as independent values. Strictly speaking, we can introduce mapping v of the set B^N into B^{2N}: if $x = x_1, \ldots, x_N \in B^N$ then

$$\upsilon(x_1, \ldots, x_N) = x_1, x_2, \ldots, x_N, x_{N+1}, \ldots, x_{2N}$$

where

$$\upsilon(x_i) = x_i, \upsilon(\bar{x}_i) = x_{i+N} \qquad \text{for } i = 1, \ldots, n$$

For every function f defined on the set $P \in B^N$ we can introduce a function f_υ defined on $\upsilon(P)$ such that for every $x \in P$ there is $f_\upsilon(\upsilon(x)) = f(x)$. The sequence x is substituted by a pair: the sequence x and its negation $\bar{x}$.

To every normal w of N-variables we assign the formula w_υ in which each variable x_i is substituted by $\upsilon_i(x)$ while $\bar{x}_i$ by $\upsilon_{i+N}(x)$.

Any normal formula of N-variables can be expressed as follows:

$$w(x) = \sum_{k \in C \subset B^{2N}} \Pi_k(x) \qquad /6/$$

where

$$\Pi_k(x) = \prod_{i=1}^{N} (x_i + \bar{k}_i)(\bar{x}_i + \bar{k}_{i+N}) ;$$

$\Pi_k(x)$ for $x \in B^N$ and $k \in B^{2N}$ is a conjunction of all variables x_i such that $k_i = 1$, and of negated variables $\bar{x}_i$ such that $k_{i+N} = 1$. In particular, if i exists such that $k_i\, k_{i+N} = 1$, then $\Pi_k(x) = 0$. If $k_i = \bar{k}_{i+N}$ for $i = 1, \ldots, N$ then $\Pi_k(x)$ is a complete product. If $k_i = 0$ for $i = 1, 2, \ldots, 2N$ then $\Pi_k(x) = 1$.

Substituting variables $x, \bar{x} \leftarrow \upsilon(x)$ in w we obtain

$$w(x) = \sum_{k \in C} \Pi_k(x) = \sum_{k \in C} \prod_{i=1}^{N} (x_i + \bar{k}_i)(\bar{x}_i + \bar{k}_{i+N}) = \sum_{k \in C} \prod_{i=1}^{N} (\upsilon_i(x) + \bar{k}_i)(\upsilon_{i+N}(x) + \bar{k}_{i+N}) =$$

$$= \sum_{k \in C} \prod_{i=1}^{2N} (\upsilon_i(x) + \bar{k}_i) = \sum_{k \in C} p_k(x) = w_\upsilon(\upsilon(x)) .$$

We see that the discussed mapping maintains equality of functions in the following sense: if $f(x) = w(x)$ on the set $P \subset B^N$ then $f_\upsilon(\upsilon(x)) = w_\upsilon(\upsilon(x))$ on $\upsilon(P) \subset B^{2N}$. With this in mind, our considerations will be confined to the problem of finding equivalents of isotone functions, since passing from any function to the isotone one and from isotone formulae to the ones with negated variables is always easy.

3. The conjunction $p_j(x)$ equals 1 on the set of x such that $x \geqslant j$. Hence $\sum_{j \in A} p_j(x) = 1$ if and only if $j \in A$ exists such that $j \leqslant x$. If such a formula is to be an equivalent of a certain isotone function then it must be equal to that function on points in which the function is determined; on other points the formula can be of some value, provided that the function, after being expanded, remains isotone.

Hence follows:

Theorem 1

Necessary and sufficient conditions so that the function $f(x)$ determined and isotone on $P \subset B^N$ be equal on P to the normal formula $\sum_{j \in A} p_j(x)$, are:

f1. For every $x \in P$ if $f(x) = 1$ then a $j \in A$ exists such that $j \leqslant x$.

f2. For every $x \in P$ if $f(x) = 0$ then for every $j \in A$, $j \leqslant x$ does not hold.

Let $\mathcal{A}(f)$ denote a family of all sets A that satisfy the conditions f1, f2 for the given function $f(x)$ defined on $P \subset B^N$, and thus assign all normal equivalents of $f(x)$. The family $\mathcal{A}(f)$ is additive. The greatest set A_{max} of the family $\mathcal{A}(f)$ is the set of all $j \in B^N$ satisfying the condition f2, and since $\underline{f} \subset A_{max}$ then this set satisfies also the condition f1.

According to [1], by a minimal element of arbitrary partly ordered set K we mean an element $x \in K$ such that for every $y \in K$, $y \leqslant x$ implies $y = x$. A set of minimal elements of the partly ordered set K is denoted by Min K.

For assigning A_{max} it is sufficient to find the set Min A_{max} of its minimal elements. Min A_{max} corresponds to the set of prime implicants and can be found by applying any method by means of which assigning all prime implicants of a given function is possible. Some methods for simplifying normal formulae /for example the one of McCluskey [11]/ render possible direct assignment of

the whole A_{max} although it is expressed in another form. When the McCluskey method is applied the following set corresponds to A_{max}: it consists of values of variables for which the function equals 1, don t-care conditions, prime implicants and all partly simplified entries obtained during computation of prime implicants.

X is a minimal set of the family $\mathcal{A}(f)$ if and only if

a1. $X \subset A_{max}$
a2. X satisfies the condition f1
a3. no set smaller than X satisfies the condition f1.

Min $\mathcal{A}(f)$ can be assigned in the following way. A table is to be formed, to each column of which we assign one element of A_{max}, and one element of $\underline{f}$ to each row. At the intersection point of the column corresponding to $j \in A_{max}$ with the row corresponding to $x \in \underline{f}$ we enter 0 if $j \leq x$, and 1 in the opposite case. The subset of A_{max} satisfies the conditions a2 and a3 if in the appropriate subset of columns there is at least one zero in each row, and if after any column is deleted there appears a row with ones only. The algorithm for assigning such subsets is described in [22] and it is reduced to finding prime implicants of a certain auxiliary function. Therefore, when an algorithm for assigning Min A_{max} to an arbitrary function is known, then we can assign Min $\mathcal{A}(f)$. In an analogous way, when we have the algorithm by means of which direct assignment of A_{max} is feasible /like the mentioned algorithm of McCluskey/, then we can directly assign $\mathcal{A}(f)$ with the aid of the described table.

$\mathcal{A}(f)$ can be assigned in another way too, i.e., by adding all possible subsets of A_{max} to the sets belonging to Min $\mathcal{A}(f)$.

4. Normal equivalents of a truth function, which have minimal worth can be assigned by finding the set $\mathcal{A}(f)$ and rejecting formulae with non-minimal worth. This method proved, however, to be very inconvenient in practice, as it requires a great number of data to operate and a lot of computations to perform. We shall discuss further some cases in which that procedure can be simplified. For that purpose we shall investigate more closely certain

partial orderings of formulae or subsets of B^N corresponding to that formulae, what gives the same effect.

Subsets of B^N are partly ordered by the inclusion relation of sets $\subset$. We introduce the relation $\subset_1$, the definition of which is as follows.

Definition: $X \subset_1 Y$ if and only if there exists mapping φ of the set Y on X such that for every $y \in Y$ there is $\varphi(y) \leqslant y$.

The relation $\subset_1$ is, of course, reflexive and transitive; moreover, it is antisymmetric. Indeed, assume that $X \subset B^N$ and $Y \subset B^N$, where N is a natural number arbitrarily fixed, and $X \subset_1 Y \subset_1 X$. It means that there exist one-one correspondences φ_1 and φ_2, defined on X and Y respectively, such that $x \geqslant \varphi_1(x)$ and $y \geqslant \varphi_2(y)$. $\varphi_2 \varphi_1$ is one-one mapping of X on X, and for every $x \in X$ there is $x \geqslant \varphi_2 \varphi_1(x)$. If for a certain $x \in X$ there is $x > \varphi_2 \varphi_1(x)$, and since $\varphi_2 \varphi_1$ is its one-one correspondence then there must be $\varphi_2 \varphi_1(x) > (\varphi_2 \varphi_1)^2(x)$ etc; in this way, we could make an infinite sequence of different elements of X. It is impossible, however, since X is finite, and hence for every $x \in X$ we have $x = \varphi_2 \varphi_1(x)$, i.e., $x = \varphi_1(x)$ and therefore $X = Y$.

We have shown then that $\subset_1$ is a partial ordering. If in the formula expressed as $\sum_{j \in X} p_j(x)$, where $x \in B^N$, $X \subset B^N$, a certain number of literals will be deleted so as to leave in each conjunction at least one literal, and if the repeated conjunctions will be rejected, we obtain then the expression $\sum_{j \in Y} p_j(x)$ such that $Y \subset_1 X$.

The sum of relations $\subset$ and $\subset_1$ is not a partial ordering as it is not transitive. Let us investigate a transitive closure of this sum in the sense of [18].

Definition: $X \overset{1}{\subset} Y$ if and only if there exists a finite sequence of subsets of B^N, namely $X = X_0, X_1, \ldots, X_n = Y$ such that for every i in $1 \leqslant i \leqslant n$ we have $X_{i-1} \subset X_i$ or $X_{i-1} \subset_1 X_i$.

The relation $\overset{1}{\subset}$ is transitive and reflexive; let us check it for antisymmetry. If $X \overset{1}{\subset} Y$ then a number of elements of X is not greater than a number of elements of Y. Hence, if $X \overset{1}{\subset} Y$

and $Y \overset{1}{\subset} X$, then X and Y have the same number of elements. From the definition follows the existence of the sequences

$$X = X_0, X_1, \ldots, X_n = Y = Y_0, Y_1 \ldots, Y_m = X$$

such that between their successive elements occurs the inequality $\subset$ or $\subset_1$. Extreme elements of these sequences, however, have the same number of elements, and consequently a number of elements of all sets in a sequence are equal; all the inequalities $\subset$ can then be substituted by $=$. Hence $X \subset_1 Y \subset_1 X$, i.e., $X = Y$. The relation $\overset{1}{\subset}$ is therefore a partial ordering.

If in the normal formula $\sum_{j \in X} p_j(x)$ literals and conjunctions will be arbitrarily deleted so as to obtain finally the normal formula $\sum_{j \in Y} p_j(x)$, then $Y \overset{1}{\subset} X$.

Lemma 1

Let $K \subset B^N$, $S \subset B^N$ and let T be a family of all subsets of S satisfying the following condition: if $X \in T$, then for every $a \in K$ there exists $b \in X$ such that $b \leqslant a$.

Then

$$\mathrm{Min}_1\, T = T \cdot 2^{\mathrm{Min}\, S} \qquad /8/$$

where 2^I denotes a family of all subsets of I
S denotes a sum of sets of family T
Min_1 denotes a set of minimal elements with respect to the relation $\subset_1$.

Proof

Let us assume that $X \subset S$ and $X \not\subset \mathrm{Min}\, S$. There exist then $x \in X - \mathrm{Min}\, S$ and $y \in \mathrm{Min}\, S$ such that $y < x$. Let $\varphi(x) = y$ and $\varphi(z) = z$ for $z \in X$ and $z \neq x$. It is readily seen that $\varphi(X) \subset_1 X$ and $\varphi(X) \neq X$.

From the assumed property of the family T it follows then that

$Y = \varphi(X) \in T$ and therefore $X \notin \mathrm{Min}_1\, T$. Hence, if $X \in \mathrm{Min}_1\, T$, then $X \subset \mathrm{Min}\, S$.

Let us suppose that $X \subset \operatorname{Min} S$, $Y \subset S$ and $Y \subsetneq_1 X$. There exists a mapping φ of X onto Y such that $\varphi(x) \leqslant x$ for every $x \in X$. Since $X \subset \operatorname{Min} S$, φ must be an identity function and then $Y \subsetneq_1 X$ implies $Y = X$.
Hence, $T \cdot 2^{\operatorname{Min} S} \subset \operatorname{Min}_1 T$ and thus $T \cdot 2^{\operatorname{Min} S} = \operatorname{Min}_1 T$.

Lemma 2

If $X, Y \subset B^N$ and $X \subset^1 Y$ then exists a $Z \subset B^N$ such that $X \subsetneq_1 Z \subset Y$.

Proof

It will be satisfactory to show that if $X \subset K \subsetneq_1 Y$ then an $L \subset B^N$ exists such that $X \subsetneq_1 L \subset Y$; the relations $\subset$ and $\subsetneq_1$ being transitive imply the Theorem. If $K \subsetneq_1 Y$ then a function φ exists such that $K = \varphi(Y)$ and $\varphi(x) \leqslant x$ for $x \in Y$. Let $L = Y\varphi^{-1}(X)$; hence $L \subset Y$ and $X = \varphi(L)$, and $\varphi(u) \leqslant u$ for $u \in L$; therefore, $X \subsetneq_1 L \subset Z$.

Lemma 3

Let T be a family of sets satisfying the conditions of Lemma 1. Then

$$\operatorname{Min}^1 T = \operatorname{Min} \operatorname{Min}_1 T = \operatorname{Min} (T \cdot 2^{\operatorname{Min} S}) \qquad /9/$$

where Min^1 denotes a set of minimal elements with respect to relation $\overset{1}{\subset}$.

Proof

From the definition of $\overset{1}{\subset}$ it follows that if $X \in \operatorname{Min}^1 T$ and $Y \in T$, $X \neq Y$ then $Y \not\subset X$ and $Y \not\subsetneq_1 X$. Therefore, $\operatorname{Min}^1 T \subset \subset \operatorname{Min}_1 T \cdot \operatorname{Min} T$.

If $A \in \operatorname{Min} T \cdot \operatorname{Min}_1 T$, then for every $B \in \operatorname{Min}_1 T$, $B \neq A$ does not hold $B \subset A$; hence $A \in \operatorname{Min} \operatorname{Min}_1 T$ and therefore

$$\operatorname{Min}^1 T \subset \operatorname{Min}_1 T \quad \operatorname{Min} T \subset \operatorname{Min} \operatorname{Min}_1 T.$$

Now we shall prove the inverse inclusion.

Let $X \in \operatorname{Min} \operatorname{Min}_1 T$; thus for an arbitrary $Y \in T$ from $Y \neq X$ follows $Y \not\subset_1 X$. Let us suppose that the inclusion does not hold. In such a case, $Y \in T$ and $Y \subset^1 X$ and $Y \neq X$; there is a Z, $Z \subset B^N$ such that $Y \subsetneq_1 Z \subset X$. From Lemma 1 follows $X \subset \operatorname{Min} S$ and therefore $Z \subset \operatorname{Min} S$. If $Y \in T$ and $Y \subsetneq_1 Z$, then $Y \subset \operatorname{Min} S$; hence $Y = Z$. Therefore, from $Y \subset^1 X$ follows $Y \subsetneq_1 X$ which, as shown earlier, is impossible. Then the inclusion $\operatorname{Min} \operatorname{Min}_1 T \subset \operatorname{Min}^1 T$ is true.

5. Let V be a finite partly ordered set, and let γ be a function that maps V onto another partly ordered set. Ordering relations in both sets will be denoted by symbol $\leqslant$.

Lemma 4

If for an arbitrary $X \in V$, $Y \in V$ from $X < Y$ follows $\gamma(X) \leqslant \gamma(Y)$ then for every $K \in \operatorname{Min} \gamma(V)$ exists a $M \in \operatorname{Min} V$ such that $\gamma(M) = K$.

Proof

Let $K \in \operatorname{Min} \gamma(V)$. Let us assume that there exists a $L \in V$ such that $\gamma(L) = K$ and $L \notin \operatorname{Min} V$. Then a $M \in \operatorname{Min} V$ also exists such that $M < L$. However, $\gamma(L) \in \operatorname{Min} \gamma(V)$ and $\gamma(M) \leqslant \gamma(L)$; hence $\gamma(M) = \gamma(L) = K$.

Lemma 5

If for an arbitrary $X \in V$, $Y \in V$ from $X < Y$ follows $\gamma(X) < \gamma(Y)$ then $\operatorname{Min} \gamma(V) \subset \gamma(\operatorname{Min} V)$.

Proof

Let $K \in \operatorname{Min} \gamma(V)$. Consider $L \in V$ such that $\gamma(L) = K$. If $K \in \operatorname{Min} \gamma(V)$ then for every $T \in \gamma(V)$ we have $T \not< K$, and so for every $M \in V$ there is $\gamma(M) \not< \gamma(L) = K$; thus $M \not< L$. Therefore $L \in \operatorname{Min} V$ and $K \in \gamma(\operatorname{Min} V)$.

Lemma 6

If for an arbitrary $X \in V$, $Y \in V$ is $X < Y$ if and only if $\gamma(X) < \gamma(Y)$, then $\operatorname{Min} \gamma(V) = \gamma(\operatorname{Min} V)$.

Proof

Let $K \in \gamma(\text{Min } V)$. Consider $L \in \text{Min } V$ such that $\gamma(L) = K$. For every $M \in V$ we have $M \not< L$, i.e., for every $T \in \gamma(V)$ there is $T \not< K = \gamma(L)$; thus $K \in \text{Min } \gamma(V)$. We have therefore shown that if $X < Y$ follows from $\gamma(X) < \gamma(Y)$, then $\gamma(\text{Min } V) \subset \text{Min } \gamma(V)$. From the above and from Lemma 5 follows $\gamma(\text{Min } V) = \text{Min } \gamma(V)$.

6. In what follows we assume that γ is a function determined on a set of normal formulae and values of that function are on a certain partly ordered set W. To simplify the wording we assume that γ is also defined on sets corresponding to these formulae and the following equality is true

$$\gamma(A) = \gamma\Big(\sum_{j \in A} p_j(x)\Big) . \qquad /10/$$

Theorem 2

If for arbitrary normal formulae such that w is to be obtained from v by deleting its certain conjunctions there is

$$\gamma(w) \leqslant \gamma(v),$$

then all γ-minimal normal equivalents of a given isotone truth function f can be found in the following way:

1. assign A_{max}
2. make a table as described above in 3 and assign $\text{Min } \mathcal{A}(f)$ according to it
3. find all γ-minimal sets in $\text{Min } \mathcal{A}(f)$
4. for each set B obtained in this way find all sets C such that $B \subset C$ and $\gamma(B) = \gamma(C)$
5. write normal formulae corresponding to the sets obtained in 3 and 4.

Proof

If $\sum_{j \in X} p_j(x)$ follows from $\sum_{j \in Y} p_j(x)$ by deleting certain conjunctions then $X \subset Y$ and $X \neq Y$. The set of

γ-minimal elements in $\mathcal{A}(f)$ can be denoted by $\operatorname{Min}_{\gamma}(\mathcal{A}(f))$. From Lemma 4 it follows that for every $K \in \operatorname{Min}_{\gamma}(\mathcal{A}(f))$ there exists a $M \in \operatorname{Min}(\mathcal{A}(f))$ such that $\gamma(M) = K$. Hence, already in point 3 of the algorithm we obtain at least one formula with a minimal value of γ. It is clear then that all remaining elements of $\operatorname{Min}_{\gamma}(\mathcal{A}(f))$ are to be found in point 4 of that algorithm. To find these elements the following proves helpful: if $B \in \operatorname{Min}_{\gamma}(\mathcal{A}(f))$, $B \subset C_1 \subset C_2 \subset \ldots \subset C_n$ and $\gamma(B) = \gamma(C_n)$, then $\gamma(B) = \gamma(C_1) = \gamma(C_2) = \ldots = \gamma(C_n)$, and if $D \notin \operatorname{Min}_{\gamma}(\mathcal{A}(f))$ and $D \subset E$, then $E \notin \operatorname{Min}_{\gamma}(\mathcal{A}(f))$.

Theorem 3

If we accept the assumptions of Theorem 2 substituting the inequality /11/ by the inequality

$$\gamma(w) < \gamma(v) \qquad /12/$$

then all γ-minimal normal equivalents of the given isotone function f can be found by means of the algorithm of Theorem 2 rejecting its point 4.

Proof

The function γ satisfies the conditions of Lemma 5 and therefore

$$\operatorname{Min}_{\gamma}(\mathcal{A}(f)) \subset \gamma(\operatorname{Min}\mathcal{A}(f)).$$

Theorem 4

If for arbitrary formulae $w = \sum_{j \in X} p_j(x)$, $v = \sum_{j \in Y} p_j(x)$ we obtain $\gamma(w) < \gamma(v)$ if and only if w follows from v by deleting certain conjunctions, then all γ-minimal normal equivalents of a given isotone function f can be found by means of the algorithm of Theorem 2 rejecting its points 3 and 4.

Proof

The function γ satisfies the conditions of Lemma 6 and therefore

$$\operatorname{Min}_{\gamma}(\mathcal{A}(f)) = \gamma(\operatorname{Min}\mathcal{A}(f)).$$

Theorem 5

If for arbitrary formulae $w = \sum_{j \in X} p_j(x)$, $v = \sum_{j \in Y} p_j(x)$ such that w is obtainable from v by deleting its certain literals /however, in each conjunction at least one literal is left/, and by rejecting repeated products, there is

$$\gamma(w) \leqslant \gamma(v), \qquad /11/$$

then all γ-minimal normal equivalents of a given isotone function f can be found in the following way:

1. assign $\operatorname{Min} A_{max}$
2. form a table analogous to that described in section 3, and assign all subsets of $\operatorname{Min} A_{max}$ belonging to $\mathcal{A}(f)$ according to it; note, however, that in the columns of that table there should be only elements of $\operatorname{Min} A_{max}$ instead of A_{max}
3. from the results obtained take all γ-minimal sets
4. for each set B obtained in this way find all sets C such that $B \subset_1 C$ and $\gamma(B) = \gamma(C)$
5. write normal formulae corresponding to the sets obtained in 3 and 4.

Proof

If $\sum_{j \in X} p_j(x)$ follows from $\sum_{j \in Y} p_j(x)$ in the above way /by deleting literals and rejecting repeated conjunctions/ then $X \subset_1 Y$ and $X \neq Y$. The set $\mathcal{A}(f)$ satisfies the condition of Lemma 1, if we take for K, from the assumption of the Lemma, the set of all elements x such that $x \in P$ and $f(x) = 1$ and for S the set A_{max} /refer to Theorem 1/. Then from Lemma 1 follows $\operatorname{Min}_1 \mathcal{A}(f) = \mathcal{A}(f) \cdot 2^{\operatorname{Min} A_{max}}$. The proof goes further than the proof of Theorem 2 but all $\subset$ and Min are to be substituted by $\subset_1$ and Min_1, respectively.

Theorem 6

If we accept the assumptions of Theorem 5 substituting the inequality /11/ by

$$\gamma(w) < \gamma(v), \qquad /12/$$

then all γ-minimal normal equivalents of f can be found by means of the algorithm of Theorem 5 rejecting its point 4.

Proof

The function γ satisfies the conditions of Lemma 5 and therefore

$$\text{Min}\ \gamma(\mathcal{A}(f)) \subset \gamma(\text{Min}_1\ \mathcal{A}(f)) = \gamma(\mathcal{A}(f) \cdot 2^{\text{Min } A_{max}}) .$$

Theorem 7

If for every pair of formulae $w = \sum_{j \in X} p_j(x)$, $v = \sum_{j \in Y} p_j(x)$ we have $\gamma(w) < \gamma(v)$ if and only if w follows from v by deleting literals in the way given above, then all γ-minimal normal equivalents of a given funotion f can be found by means of the algorithm of Theorem 5 rejecting its points 3 and 4.

Proof

The function γ satisfies the conditions of Lemma 6, and therefore

$$\text{Min}\ \gamma(\mathcal{A}(f)) = \gamma(\text{Min}_1\ \mathcal{A}(f)) = \gamma(\mathcal{A}(f) \cdot 2^{\text{Min } A_{max}}) .$$

Theorem 8

If for two arbitrary normal formulae $w = \sum_{j \in X} p_j(x)$, $v = \sum_{j \in Y} p_j(x)$ such that w can be obtained from v by deleting its certain literals and conjunctions, there is

$$\gamma(w) \leqslant \gamma(v), \qquad /11/$$

then all γ-minimal equivalents of the given function f can be found by means of the following algorithm:

1. assign $\text{Min } A_{max}$
2. make a table as in point 2 of Theorem 5 and assign all minimal subsets of $\text{Min } A_{max}$ belonging to $\mathcal{A}(f)$ according to it
3. from the obtained results take all γ-minimal sets
4. for each set B obtained in this way find all sets C such

that $B \overset{1}{\subset} C$ and $\gamma(B) = \gamma(C)$

5. write normal formulae corresponding to the sets obtained in the above points 3 and 4.

Proof

If $\sum_{j \in X} p_j(x)$ follows from $\sum_{j \in Y} p_j(x)$ by deleting its certain literals and conjunctions, then $X \subset Y$ and $X \neq Y$. From Lemma 3 follows $\mathrm{Min}^{1} \mathcal{A}(f) \subset \mathrm{Min}\, \mathrm{Min}_{1} \mathcal{A}(f) = \mathrm{Min}\, \mathcal{A}(f) \cdot 2^{\mathrm{Min}\, A_{max}}$. From Lemma 4 it follows that for every $K \in \mathrm{Min}\, \gamma(\mathcal{A}(f))$ there exists a $M \in \mathrm{Min}\, \mathrm{Min}_{1} \mathcal{A}(f)$ such that $\gamma(M) = K$. The proof goes further as the proof of Theorem 2 provided that all symbols $\subset$ are to be substituted by $\overset{1}{\subset}$.

Theorem 9

If we accept the assumptions of Theorem 8 substituting the inequality /11/ by the inequality

$$\gamma(w) < \gamma(v), \qquad /12/$$

then all γ-minimal normal equivalents of a given isotone function can be found by means of the algorithm of Theorem 8 rejecting its point 4.

Proof

The function γ satisfies the conditions of Lemma 5 and therefore by virtue of Lemma 3 we have

$$\mathrm{Min}\, \gamma(\mathcal{A}(f)) \subset \gamma(\mathrm{Min}^{1} \mathcal{A}(f)) = \gamma(\mathrm{Min}\, \mathcal{A}(f) \cdot 2^{\mathrm{Min}\, A_{max}}).$$

Theorem 10

If for arbitrary formulae $w = \sum_{j \in X} p_j(x)$, $v = \sum_{j \in Y} p_j(x)$ we obtain $\gamma(w) < \gamma(v)$ if and only if w follows from v by deleting literals and conjunctions, then all γ-minimal normal equivalents of the given isotone function f can be found by means of the algorithm of Theorem 8 rejecting its points 3 and 4.

Proof

The function γ satisfies the conditions of Lemma 6 and therefore $\operatorname{Min} \gamma(\mathcal{A}(f)) = \gamma(\operatorname{Min}^1 \mathcal{A}(f)) = \gamma(\operatorname{Min} \mathcal{A} \cdot 2^{\operatorname{Min} A_{max}})$. The set $\operatorname{Min} A \cdot 2^{\operatorname{Min} A_{max}}$ was obtained in point 2 of the algorithm.

7. Theorems 2-10 include cases in which the function γ is isotone with respect to $\subset$, $\subset_1$ or $\subset^1$. The algorithms given are also for the case when $\gamma(w)$ is a number of literals in a formula w /Theorem 9/.

In a way similar to that in which Theorems 2-10 have been derived we can obtain algorithms for finding γ-minimal normal equivalents when γ is antitone with respect to $\subset$, $\subset_1$ or $\subset^1$. In such cases, if we confine ourselves to considering equivalents in which appear only variables that are arguments of a given truth function then, for solving a problem, it would be satisfactory to find certain maximal sets with respect to the appropriate ordering relations. We are not, however, going to handle all such cases in detail as suitable algorithms for solving the problem can be obtained by a reasoning analogous to that by means of which we arrived at the above theorems.

As mentioned earlier computations specified in algorithms of Theorems 2-10 can be performed if we know any effective method of finding $\operatorname{Min} A_{max}$ and A_{max} /the latter directly or from $\operatorname{Min} A_{max}$/ for arbitrary truth functions; a method for finding all prime implicants of arbitrary functions would then be satisfactory. In this sense the present paper is a generalization of papers giving algorithms for finding normal equivalents with a minimal number of literals.

E X A M P L E S

Example 1

Find γ-minimal normal equivalents for the function determined in the table.

$\gamma(w)$ is the greatest number of identical letters appearing in the expression w /variables and negated variables are treated as different letters/.

x_3	x_2	x_1	f(x)
0	1	0	1
0	1	1	1
1	0	1	0
1	1	0	0
1	1	1	1

For the remaining values of x the function is not determined.

When in the expression w letters and products are canceled, $\gamma(w)$ remains constant or it decreases; therefore, the function satisfies the assumptions of the Theorem 8.

Negated variables are being written in the table:

$\bar{x}_3$	$\bar{x}_2$	$\bar{x}_1$	x_3	x_2	x_1	$f_v(x)$
1	0	1	0	1	0	1
1	0	0	0	1	1	1
0	1	0	1	0	1	0
0	0	1	1	1	0	0
0	0	0	1	1	1	1

Using any method, one obtains prime implicants of the function:

$$\text{Min } A_{max} = \{ 000011, 010010, 100000, 001001, 011000 \}$$

Quine's table

						x_1	x_2	x_3	$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$
						1	0	0	1	0	
						1	1	0	0	0	
						0	0	0	0	0	
						0	0	0	1	1	
						0	1	0	0	1	
						0	0	1	0	0	
$\bar{x}_3$	$\bar{x}_2$	$\bar{x}_1$	x_3	x_2	x_1						
1	0	1	0	1	0			+			
1	0	0	0	1	1	+		+			
0	0	0	1	1	1	+					

From the above table one obtains the expression

$$w = \bar{x}_3 + x_1 x_2 \qquad \gamma(w) = 1$$

The value of γ does not change if the literals x_3, $\bar{x}_1$ or $\bar{x}_2$ are introduced to w.
Since the sequence corresponding to $\bar{x}_1 \bar{x}_2$ belongs to Min A_{max}, then $\bar{x}_1 \bar{x}_2$ or $\bar{x}_1 \bar{x}_2 x_3$ may be added to w.

If in the expression w $\bar{x}_1 \bar{x}_3$ substitutes $\bar{x}_3$, or $x_1 x_2 x_3$ substitutes $x_1 x_2$ /but not simultaneously/, expressions obtained will be equivalents of the function f.
Finally, the following expressions are solutions of the problem:

$$\bar{x}_3 + x_1 x_2$$
$$\bar{x}_3 + x_1 x_2 + \bar{x}_1 \bar{x}_2$$
$$\bar{x}_3 + x_1 x_2 + \bar{x}_1 \bar{x}_2 x_3$$
$$\bar{x}_1 \bar{x}_3 + x_1 x_2$$
$$\bar{x}_3 + x_1 x_2 x_3$$
$$\bar{x}_3 + x_1 x_2 x_3 + \bar{x}_1 \bar{x}_2$$

Example 2

Find γ-minimal normal equivalents of the function determined in the table

$$\gamma(w) = \frac{\text{the number of letters in the expression } w}{1 + \text{the number of conjunctions with one letter}}$$

x_3	x_2	x_1	$f(x)$
0	0	0	0
0	1	0	0
1	1	0	0
1	0	0	1
1	0	1	1

The function satisfies the assumptions of Theorem 6: the value of γ decreases after canceling letters, but it may increase while canceling conjunctions.

Table of the function f after writing negations :

$\bar{x}_3$	$\bar{x}_2$	$\bar{x}_1$	x_3	x_2	x_1	$f_v(x)$
1	1	1	0	0	0	0
1	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	1	1	0	0	1
0	1	0	1	0	1	1

$$\text{Min } A_{max} = \{000001, 010010, 010100, 100100\}$$

Quine's table

	x_1	x_2	x_3	$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$
$\bar{x}_3 \bar{x}_2 \bar{x}_1 x_3 x_2 x_1$	1	0	0	0	0	0
0 1 1 1 0 0						+
0 1 0 1 0 1	+					+

All subsets of Min A containing 010100, belong to $\mathcal{A}(f)$.

The value of γ may be decreased when a conjunction is added to the expression, only if this conjunction includes one letter. Therefore, it is sufficient to examine the value of γ for the following subsets of Min A_{max} .

$\{010100\}$	$w = \bar{x}_2 x_3$	$\gamma(w) = 2$
$\{010100, 000001\}$	$w = \bar{x}_2 x_3 + x_1$	$\gamma(w) = \frac{3}{2} = 1.5$

The second of the above expressions is the solution.

Example 3

Find normal equivalents with the smallest number of conjunctions for the function determined in the table.

x_3	x_2	x_1	$f(x)$
0	0	0	0
0	0	1	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

The table of the function after writing negations.

$\bar{x}_3$	$\bar{x}_2$	$\bar{x}_1$	x_3	x_2	x_1	$f_v(x)$
1	1	1	0	0	0	0
1	1	0	0	0	1	1
1	0	0	0	1	1	1
0	1	1	1	0	0	0
0	1	0	1	0	1	0
0	0	1	1	1	0	1
0	0	0	1	1	1	1

Min $A_{max} = \{000010, 001001, 100001, 100100\}$

The set A_{max} contains 48 elements, but it suffices to write to the Quine's table only elements less than one or more sequences for which f_ϑ has the value 1.

	x_1	x_2	x_3	$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$
	0	1	0	0	0	1
	1	1	0	0	1	1
	0	1	0	1	1	0
	0	1	0	1	1	1
$\bar{x}_3$ $\bar{x}_2$ $\bar{x}_1$ x_3 x_2 x_1						
1 1 0 0 0 1						+		+
1 0 0 0 1 1	+	+				+	+	+	
0 0 1 1 1 0	+		+		+	+			
0 0 0 1 1 1	+	+	+	+					

Computing the Quine's table we obtain the following expressions with the least number of conjunctions:

$$x_2 + x_1 \bar{x}_3$$
$$x_2 + x_1 \bar{x}_2 \bar{x}_3$$
$$x_2 x_3 + x_1 \bar{x}_3$$

Conclusion

Operating with combinations of zeros and ones instead of conjunctions and with sets of these combinations instead of alternative normal formulae is very convenient when the discussed problems are to be solved on binary digital computers. Also, expressing a value of every variable by means of a pair of bits, i.e., the value of a variable and its negation /transformation ϑ/ is a technique used in different completed programmes of simplifying normal formulae. Such a way of expressing the values of variables has been used in programmes of this kind worked out in our Institute.

Acknowledgement

The author wishes to thank Dr. A. Ehrenfeucht and Dr. A.W. Mostowski for their valuable remarks on reading the manuscript.

References.

1. BIRKHOFF G.: Lattice Theory, New York 1948.

2. BUTLER K.J.Jr., WARFIELD J.N.: A Digital Computer Program for Reducing Logical Statements to a Minimal Form, Proc. Natl. Electronics Conf. 1959:15, 456-466.

3. COPI I.: Symbolic Logic, New York 1948.

4. DUNHAM B., FRIDSHAL R.: The Problem of Simplifying Logical Expressions, J. Symbolic Logic 1959:24, 17-19.

5. GAVRILOV M.: Minimizacija bulevyoh funkcij chrakterizujuščich relejnye cepi, Avtomatika i Telemechanika 1959:20, 1217-1238.

6. GRZEGORCZYK A.: Zarys Logiki Matematycznej, Warszawa 1961.

7. HARRIS B.: An Algorithm for Determining Minimal Representations of a Logic Function, IRE Trans. 1957:EC-6, 103-108.

8. Harvard Comp. Lab., Annals of the Computation Laboratory in Synthesis of Electronic Computing and Control Circuits, Harvard University Press 1951:27.

9. JABLONSKIJ S.V.: Funkcional'nye postroenija v k-značnoj logike, Trudy Mat. Instituta im. Steklova 1958:51, 5-142.

10. KARNAUGH M.: The Map Method for Synthesis of Combinational Logic Circuits Commun. and Electronics, Trans AIEE pt, I, 72.

11. McCLUSKEY E.J.: Minimization of Boolean Functions, Bell System Techn. Journal 1953:35, 1417-1444.

12. MOTT T.H.Jr.: Determination of the Irredundant Normal Forms of a Truth Function by Iterated Consensus of the Prime Implicants, IRE Trans. 1960:EC-9, 254-252.

13. NELSON R.J.: Simplest Normal Truth Functions, J. Symbolic Logic 1955:20, 105-108.

14. NELSON R.J.: Weak Simplest Normal Truth Functions, J. Symbolic Logic 1955:20, 232-234.

15. QUINE W.V.: The Problem of Simplifying Truth Functions, Amer. Math. Monthly 1952:59, 521-531.

16. QUINE W.V.: A Way to Simplify Truth Functions, Amer. Math. Monthly 1955:62, 627-631.

17. QUINE W.V.: On Cores and Prime Implicants of Truth Functions Amer. Math. Monthly 1959:66, 755-760.

18. RIGUET J.: Relations binaires, fermetures, correspondances de Galois, Bull. de la Societé Mathématique de France 1948:76, 114-155.

19. URBANO R.H., MUELLER R.K.: A Topological Method for the Determining of the Minimal Forms of a Boolean Function, IRE Trans. 1956:EC-5, 126-132.

20. VEITCH E.W.: A Chart Method for Simplifying Truth Functions, Proc. of the Assoc. for Comp. Mach. Mellon Inst. May 1952: 127-133.

21. VOJSVILLO E.K.: Metod uproščenija form vyraženija funkcij istinnosti, Naučnye Doklady Vysšej Školy Filosofskie Nauki 1958:2, 120-135.

22. WALIGÓRSKI S.: Calculation of the Quine's Table for Truth Functions, Prace ZAM PAN 1961, 2, A15.