42488

PROF. ING. CORRADO BÖHM
VIA S. CRESCENZIANO, 20 - TEL. ~~815.597~~ 838-0412
I 00199 ROMA

Roma, March,22 1968

Professor

Donald E. Knuth

California Institute
of Technology
PASADENA, California

Dear Prof. Knuth,

I hope you received some documentation about my activity before 1952. Iwill enclose here photocopies of some cards prepared inNovember 1950, which perhaps will have some historical interest.

I received your paper about semantics of context-free languages. I found it very stimulating; I would like to thank you for sending it to me. As you remarked my main interest is some mathematical thinking about the programming activity. For example , a question which is important to me is the following: It is possible to characterize from a mathematical-logical point of view a compiler versus an interpreter doing the same job? The question seems very naive, but if you consider that the existence of an interpreter is strongly related to the semantics of a programming language, you will perhaps admit the need for better definitions in this domain. I would be interested to know your own ideas about this preliminary question to the construction of a compiler-compiler in CUCH.

Let me turn now to a new topic. A friend of mine, Giorgio Sacerdoti, until recently director of the Research Laboratory on Electronics of the Olivetti- General Electric, now at Olivetti Society, is charged to organise a three-day International Memorial Meeting(not a board meeting) in Italy for next October. You will find the motivation in the enclosed document.

Since some time you have being on the list of potential speakers. I related to my friend some of our personal exchanges of view in Pasadena. As a matter of fact you are considered one of the key speakers of the Convention. We realize how busy you are; but we would be more than delighted if you were to accept this invitation. The financial support for travel expenses, etc. will be entirely paid by Olivetti Society.

Let me know, please, if you are not in Pasadena in the next 10-12 days (or where you will be) because my friend Sacerdoti is undertaking to travel in USA to organize the Symposium, and he wishes contact you.

Yours sincerely

Corrado Böhm

March 26, 1968

Professor Ing. Corrado Böhm
Via S. Crescenziano, 20
I00199 Roma, Italy

Dear Corrado:

Thanks for the interesting historical information you have sent me. Please tell me what previous experience in computer programming you had had before you got the idea for your original programming language, and try to tell me (as well as you can remember) what were the primary influences which led to this work. What (if any) ideas of other people inspired you?

Regarding your question of compilers and interpreters, I must say I don't believe there is any clear distinction. Every computer program can be regarded as an interpretive routine (interpreting its data), and as a syntax-directed compiler(analyzing its data). The only differences seem to be the extent to which the data governs the flow of the program; many programs carry out their computations in roughly the same order, regardless of the form of the data, and such programs don't have much of the character of interpretive or syntax-directed routines. The difference in "flavor" between interpretive routines and syntax-directed routines is essentially that the former deal primarily with data that has a simple linear structure, while the latter deal primarily with data having a nested structure.

The planned Olivetti symposium sounds like a very fine undertaking, and I am very flattered to be considered as a potential "key speaker". The chance to visit Italy again with all expenses paid is very attractive. But I must decline this generous invitation, since I will be working in a full time job at the Institute for Defense Analyses this fall, and my work will not allow me to travel. I have already cancelled plans to see the Olympic Games in Mexico and the IFIP Congress in Edinburgh. I hope there will be future opportunities to visit Italy after a year or two; I must pass up the opportunity this time.

Cordially,

DEK:kh

Donald E. Knuth

P.S. Perhaps Kasami (Japan) and Dijkstra (Netherlands) could be persuaded to speak of their recent interesting work at the Olivetti symposium.

518.61 CODIFICAZIONE INTRINSECA

26 Nov 50

Codificazione intrinseca - Grafi - Esempi

Una delle più gravi difficoltà che si incontrano nella preparazione di un programma per una macchina calcolatrice attuale è costituita dal fatto che nella preparazione deve essere rispettato un ordine che è in relazione collo sviluppo delle operazioni nel tempo. Ogni codificatore possiede una norma di prudenza che consiste nell'intercalare fra problemi parziali delle caselle di memoria vuote in cui più tardi, quando cioè viene acquistata una visione d'assieme del problema, si possano riempire con ordini di collegamento tra le varie parti del problema. Col sistema che noi descritto tale misura preventiva risulta superflua in quanto ogni problema parziale può essere scritto e programmato indipendentemente dal seguire temporale con gli altri problemi parziali. Poiché ci si rende in certo senso liberi dai valori numerici degli indirizzi e dalla loro successione, è giustificato il termine di codificazione intrinseca dato a tale metodo.

Per meglio illustrare la successione e le alternative tra le differenti operazioni di un problema facciamo uso di grafi. Ogni punto del grafo rappresenta una catena di operazioni ~~[illegible]~~ che simboleggiamo con una lettera dell'alfabeto. Se esiste una variante del problema dove alla catena A succede immediatamente la catena B allora si conviene di rappresentare questa possibilità col segmento orientato

$P_{A,B}$ $\qquad A \longrightarrow B \qquad p_{AB}=1$

Analogamente per le altre catene.

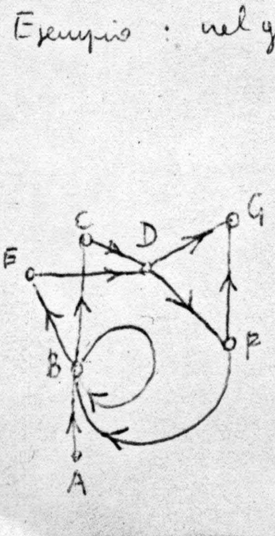
Il grafo si può anche definire per mezzo della matrice d'incidenza p_{ik} in cui p es. $p_{AB}=1$. Ogni problema ha una prima operazione (sorgente) ed un ultima operazione (pozzo)

Esempio: nel grafo sottostante la sorgente è A ed il pozzo è G

la matrice d'incidenza è:

	A	B	C	D	E	F	G
A	0	1	0	0	0	0	0
B	0	1	1	0	1	0	0
C	0	0	0	1	0	0	0
D	0	0	0	0	0	1	1
E	0	0	0	1	0	0	0
F	0	1	0	0	0	0	1
G	0	0	0	0	0	0	0

La sorgente è caratterizzata dal fatto che la colonna è formata tutta da zeri. ~~Analogamente~~ Dualmente il posso è caratterizzato da una riga formata da zeri.



Supponiamo che ogni catena C_i consti di tre parti ordinate come segue:

C_i {
1. all'inizio un indicazione del tipo: ~~Qui comin~~ La prossima operazione è la prima della catena C_i
2. ~~la~~ la catena delle operazioni che costituiscono C_i
3. ~~[illegible]~~ la determinazione della catena ~~successiva~~ immediatamente successiva a C_i.

È chiaro che se ogni catena ha la costituzione sopraccennata, l'ordine con cui le catene si susseguono nella codificazione è indifferente purché il legame logico e temporale fra le diverse catene viene ugualmente rispettato. Ecco per esempio la costituzione della catena B nel problema rappresentato dal grafo testé disegnato

1)	$\pi \to B$
2)	$\dots \to \dots$
	$\dots \to \dots$
	$\dots \to \dots$
3)	$\dots \to p_{BE}$
	$\dots \to p_{BC}$
	$\dots \to p_{BE}$
	$B\cdot p_{BB} + C\cdot p_{BC} + E\cdot p_{BE} \to \pi$

Mentre tutti i comandi corrispondenti alle parti 2) e 3) vengono regolarmente codificati, il comando 1) $\pi \to B$ viene soltanto eseguito, cioè l'indirizzo della prima operazione in 2) ~~[illegible]~~ (cioè l'indirizzo delle operazioni non ancora codificate) viene portato ~~all~~ all'indirizzo B. Dimodoché tutte le catene che possono avere come successiva ~~[illegible]~~ la catena B possono scegliere il giusto indirizzo.

Il vantaggio di tale sistema è che l'operatore non ha bisogno di conoscere il contenuto della cellula B. Tale sistema acquista naturalmente ancora + valore abbinato alla codificazione automatica applicata allo svolgimento di parentesi e di polinomi in forma normale (vedi schede relative). Ecco lo schema del programma corrispondente al grafo portato come esempio:

prima operazione $A \to \pi$

Una qualsiasi permutazione nell'ordine delle catene. Le catene G terminano con $0 \to \pi$

27 Nov. 50 CODIFICAZIONE AUTOMATICA DI OPERAZIONI ELEMENTARI

Codificazione automatica delle operazioni elementari

(I TEORIA)

Per operazione elementare si intende un operazione formalizzata così

1) $V_1 \, Op_2 V_3 \rightarrow V_4$ anche

2) $Op_2 V_3 \rightarrow V_4$ prove

3) $V_3 \rightarrow V_4$

Dove V rappresenta una Variabile cioè uno dei seguenti simboli

a, b ... z ; A, B ... Z ; ↓A ↓B .. ↓Z ; π, π' ; ~~...~~, 0, 1, ?

(Vedi formalismo – scheda)

Per operazione Op s'intende una delle seguenti: +, −, ·, :, mod, ∪, ∩, (a due operandi) e $\nu(\cdot)$, $(-1)^{\cdot\cdot}$, pos(...), segno(..), valore assol. |.|

ecc.

La codificazione automatica fa corrispondere biunivocamente ad ogni operazione un numero N e un indirizzo i_N

$$\left[X \; op \; Y \rightarrow Z \right] \longleftarrow N = C(X)\cdot c_1 + C(Op)\cdot c_2 + C(Y)\cdot c_3 + C(Z)\cdot c_5$$

c_1, c_2, c_3, c_4 sono costanti positive intere caratteristiche per ogni codificazione e C(X) rappres. il numero intero corrispondente al simbolo X. Analogamente per C(Op), ecc.

Al numero N viene attribuito un indirizzo, generalmente a due comandi elementari susseguentisi corrispondo due numeri in due indirizzi susseguentisi.

Riduzione di 2) e 3) a 1 Per comodità maggiore nel formulare il programma supporremo di cost. si passare dall'operazione ~~...~~ 2 al numero N e i . supponiamo che esista una Variabile ~~...~~ ed un operazione in "bianco" $\boxed{V}$ e $\boxed{Op_2}$ per cui 2) e 3 si scriveranno

1)' $\boxed{V} \, Op_2 V_3 \rightarrow V_4$ dove $C(\boxed{V}) = 0$ e $C(\boxed{Op_2}) = 0$

1)'' $\boxed{V} \, \boxed{Op_2} V_3 \rightarrow V_4$

Ciò significa che se ad ogni segno (variabile od operazione) corrisponde nella macchina un tasto, occorre fornire la macchina dei tasti $\boxed{V}$ e $\boxed{Op_2}$. Così si ottiene che ogni comando elementare ha la forma 1) cioè è una successione di 5 segni. Consideriamo ora una successione di comandi elementari e designamo le variabili con φ, le operazioni con ω le → con σ. Allora si avrà indicando con n il numero d'ordine di ogni segno

1+n =	1	2	3	4	5	6	7	8	9	10	11	12	13
4)	φ	ω	φ	σ	φ	φ	ω	φ	σ	φ	φ	ω	φ
	φ	ω	φ	σ	φ	φ	ω	φ	σ	φ	φ	ω	φ
n =	0	1	2	3	4	5	6	7	8	9	10	11	12

Si noterà che le variabili φ si trovano a quei posti n per cui vale

$\varphi)$ $(n \bmod 5) \bmod 2 . = 0$

le operazioni ω si troveranno a quei posti n per cui vale

$\omega)$ $(n \bmod 5 - 1 = 0$

ed infine le → σ si troveranno a quei posti per cui vale

$\sigma)$ $n \bmod 5 - 3 = 0$

Ciò deve permettere un controllo per ~~...~~ potere segnalare (con l'arresto della macchina p. es.) il caso in cui una successione di segni perde il suo significato.

<u>Caso eccezionale</u> in cui V_1 o $V_2 = \pi$ e $V_4 \neq \pi$. In tale caso, come già notato altrove (vedi scheda codificazione intrinseca) il comando non deve essere codificato ma eseguito subito. In tale caso e solo in tale caso si ha, chiamando con $P_{\varphi(n)}$ quella funzione di $\varphi(n)$ che è uguale a 0 se φ è π e $=1$ ~~altrove~~ se $\varphi^{(n)}$ non è π

$P_{\varphi(n_1)}$ e $P_{\varphi(n+2)}$

$$\boxed{P_{\varphi(n)} \cdot P_{\varphi(n+2)} + \delta_{\varphi(n+4)} - 1 = 0}$$

dove inoltre esse	$n \bmod 5 = 0$

Il programma per la codificazione automatica comporterà dunque:

1 Una corrispondenza biunivoca K(...) tra ogni ~~...~~, singola φ, ω e σ (chiamata perciò funzione ed ottenibile per costruzione della tastiera) ed un numero intero K(φ), K(ω) dove per esempio K(A) ≠ K(↓A) e K(+) ≠ K(−). Tale corrispondenza è prestabilita. Allo schiacciare del tasto i corrisponde l'entrata di K(i) nella cellula K della macchina.

2. Un controllo della successione 4) dove la macchina si ferma non appena un termine della successione non è corretto

3. La costruzione del numero N e dell'indirizzo i_N. L'indirizzo i_{N_1} del primo comando codificato si supporrà contenuto nella cellula A fin dall'inizio del calcolo

Nel caso eccezionale di cui sopra deve essere provveduto all'esecuzione del comando invece che alla sua codificazione

FORMALISMO per la CODIFICAZIONE AUTOMATICA

Immaginiamo di avere una macchina a programma che disponga delle operazioni Op: $\pm$, $\cdot$, $:$, $\cup$, $\cap$ cioè della somma (+) differenza (−) prodotto ($\cdot$) divisione (:) massimo ($\cup$) e minimo ($\cap$) fra due numeri interi relativi.

Per ognuna di queste operazioni sia riservato un tasto. Inoltre vi sia una tastiera numerica per le entrate, una tastiera letterale con lettere minuscole (indicanti indirizzi delle memorie contenenti ordini) ed una con lettere maiuscole (indicanti indirizzi contenenti numeri) e oltre due simboli 0 e 1 che valgono zero ed uno rispettivamente.

Sono inoltre definiti i seguenti codici di cui diamo esempi.

$-123 \to a$ Il numero −123 è da portare nella cellula o indirizzo a (Commutazione...)

$a \to b$ il contenuto dell'indirizzo a è da portare all'indirizzo b

$a \cdot b \to c$ il prodotto dei numeri trovantesi negli indirizzi a e b è da portare all'indirizzo c

$a \cup b \to c$ il maggiore dei numeri trovantesi " " " " " "

Analogamente per le altre operazioni. Da notare che l'operazione generica

$a \; Op \; b \to a$ si interpreta dicendo: il risultato dell'operazione tra i numeri trovantesi in a e b sostituisce il numero che si trovava in a.

Il simbolo $\downarrow$ in unione con gli indirizzi acquista un significato di doppio rinvio. Esempi.

$a \; Op \; b \to \downarrow c$ Il risultato dell'Op tra i numeri trovantisi in a e b è da portare all'indirizzo coincidente col numero contenuto in c

$\downarrow a \; Op \; b \to c$ Il ~~[illegible]~~ risultato dell'Op tra il numero il cui indirizzo trovasi nella cella a e il numero il cui indirizzo è b è da portare all'indirizzo c.

Il simbolo Π indica la cellula specializzata il cui contenuto è (ad ogni momento dell'esecuzione di un calcolo) l'indirizzo del prossimo comando elementare da effettuare. Nella codificazione automatica l'importanza di tale cellula è resa evidente dalle seguenti superistruzioni:

Sia α il risultato numerico di un calcolo (numero positivo intero) ~~[illegible]~~:

$\alpha \to \Pi$ è il supercomando che si legge: ~~il prossimo~~ l'indirizzo del prossimo comando elementare è α. Se $\alpha = 0$ si ha il seguente supercomando:

$0 \to \Pi$ da leggere l'indirizzo del prossimo comando non esiste la macchina deve fermarsi e non calcolare più (STOP)

E' da notare che se un supercomando non ha luogo l'indirizzo contenuto in Π è ad ogni momento del calcolo quello successivo all'indirizzo del comando attualmente in via d'esecuzione. E' ovvio chiaro che un super comando del tipo $\Pi \to \Pi$ non modificherà per nulla l'andamento dei calcoli ed è completamente superfluo.

Oltre alle operazioni a due operandi indicate all'inizio sarebbe bene che la macchina possedesse tasti per le seguenti operazioni ad un operando (però tale esigenza è di natura essenzialmente pratica non concettuale)

$$\operatorname{sign} a = \begin{cases} +1 & \text{se } a>0 \\ 0 & a=0 \\ -1 & a<0 \end{cases} \qquad |a| = \begin{cases} a & \text{se } a\geq 0 \\ -a & \text{se } a\leq 0 \end{cases} \qquad \nu(a) = \begin{cases} 0 & \text{se } a\neq 0 \\ 1 & \text{se } a=0 \end{cases}$$

$$\operatorname{pos}(a) = \begin{cases} a & \text{se } a>0 \\ 0 & \text{se } a\leq 0 \end{cases}$$

Dato che a è un numero relativo tali operazioni si riducono a combinazioni di già note

$$\operatorname{sign} a = (|a+1| - |a-1|):2 = 1 + a\cup(-1) - a\cup 1$$

$$|a| = a\cup(-a)$$

$$\nu(a) = 1 - 1\cap|a|$$

$$\operatorname{pos}(a) = 0\cup a$$

Per interpretare ~~i vecchi co~~ il comando sopra ricordato in uso presso alcune macchine è anche utile introdurre la funzione

$$f(a) = \begin{cases} 1 & \text{se } a>0 \\ 0 & \text{se } a\leq 0 \end{cases} = 1\cap a - 0\cap a$$

Da notare che

$$1 - f(a) = 1\cup a - 0\cup a$$

Esempio di condizionato. Esegui il comando che si trova all'indirizzo A solo se il contenuto della cellula b è positivo

$$[1\cap b - 0\cap b]A + [1\cup b - 0\cup b]\Pi \to \Pi$$

Tale risultato si può anche ottenere per calcolo sulle proposizioni (vedi scheda: applicazioni del calcolo delle proposizioni) È chiaro che un qualsiasi comando condizionato, alternativo, ecc può facilmente mettersi sotto questa forma.

Collegamenti della macchina con l'esterno e distinzione fra fase di codificazione e fase di calcolo.

$a \to ?$ il contenuto di a viene reso visibile all'esterno (stampato o su quadrante luminoso)

$? \to a$ la macchina si è fermata in attesa che un dato si trasmetta alla cellula a.

Tale comando è molto utile per la codificazione automatica

Nella fase di codificazione la cellula Π è trattata come una cellula qualsiasi* ed il suo ruolo viene preso dalla cellula Π'. In tale fase dunque il comando $\Pi \to A$ (è analogo al comando $-12.5 \to a$) deve leggersi così: il contenuto della cellula Π deve essere portato nella cellula A.

Tutti gli altri comandi di cui sopra acquistano invece un significato differente che permette appunto la codificazione automatica. (vedi sezione speciale)

(* perde cioè l'attributo di super cellula)

Codificazione automatica delle operazioni elementari (II: Realizzazione)

Sia p esempio da codificare il programma per estrarre la radice quadrata di un numero ξ (nella cellula X). Il programma è notoriamente, conoscendo un valore approssimato z (nella cellula t) della radice

La precisione richiesta con numeri interi sia per esempio di ± 1

$\pi \to A$
$t \to Z$
$(Z + 1 : z) : 2 \to t$
$\{1 \cap [0 \cup [(t-z)^2 - 1]]\} \cdot A \to \pi$

il programma per esteso diventa:

1+n = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 = 1+n ... n-2 n-1 n n+1

S = □ □ π → A □ □ t → Z 1 : Z → a a + Z → b b : 2 → t ecc. ... z · A → π

φ ω φ σ φ φ ω φ σ φ φ ω φ σ φ φ ω φ σ φ φ ω φ σ φ ... φ ω φ σ φ

n mod 5: 1 2 3 4 5 1 2 3 4 5 = n mod 5 + 1 ... 0 0 1 2 3 4 = n mod 5

z(S) = 0 1 0 2 0 0 1 0 2 0

Consideriamo ora la seguente pseudoficazione di un segno generico S

$$(4) \quad K(S) = 3 \cdot C(S) + z(S)$$

dove $C(S)$ rappresenta la codificazione prestabilita nella macchina dove si ha

$C(\to) = C(\boxed{V}) = C(\boxed{D}) = 0$ e definiamo $z(S)$ con le relazioni

$z(\sigma) = 2$
$z(\varphi) = 0$
$z(\omega) = 1$

p. es. $\begin{cases} z(A) = 0 & z(?) = 0 \\ z(\to) = 2;\ z(+) = 1;\ z(\cdot\!\!\cdot) = 1;\ z(:) = 1 \\ z(0) = 0 & z(1) = 0 \text{ ecc.} \end{cases}$

tenendo conto delle relazioni $\varphi)$ $\omega)$ e $\sigma)$ è facile vedere che per il controllo si deve avere

$$(5) \quad (n \bmod 5) \bmod 2 + D(n \bmod 5 - 3) = z(S)$$

D'altra parte si ha da (4)

$$(6) \quad z(S) = K(S) \bmod 3$$

$$K(\to) = 2$$

$$(7) \quad C(S) = K(S) : 3$$

chiamiamo $C(\pi) = C_\pi$

allora la grandezza $p_{\varphi(n)}$ è definita dal

$$(8) \quad p_{\varphi(n)} = \left(z(\varphi) - z(S)\right)\left(C_\pi - C(S)\right) \cap 1$$

$$p_\varphi = \{z(S) \cdot [C_\pi - C(S)]\} \cap 1$$

all'inizio di ogni comando elementare si trovi nella cellula P il numero 1

sia $n \bmod 5 = m = (0,1,2,3,4)$

La grandezza p_m sia definita come p_4 allora il criterio per l'esecuzione ~~...~~ di un comando invece che la sua tabulazione è

$$\cancel{p_1 p_2 p_3 p_4 + p_5 - 1 = 0}$$

$$p_0 p_1 p_2 p_3 + p_4 - 1 = 0$$

cellula P cellula p

Per ottenere il programma occorre siano soddisfatti i seguenti preliminari ~~la cellula successiva alla cellula~~ H sia h lo indirizzo numerico della cellula H e la cellula $h+1$ contenga il ordine codificato $(N \to \pi')$

π' rappresenta il prossimo comando (attivo durante la codificazione).

Le quattro costanti siano così distribuite nelle seguenti cellule

$c_1 \to t$

$c_2 \to t+1$ inoltre si $t \to F$

$c_3 \to t+2$ primo indirizzo dei comandi da distribuire si trovi nella cellula J

$c_5 \to t+4$ 1 inizio $\to J$

Il primo comando eseguibile soltanto col principio di una codificazione automatica è

I comando $Q - 1 \to n$

$\pi \to N$

$n+1 \to n$

$0 \to H$

$1 \to P$

$\pi \to Q$

$? \to K$

$K : 3 \to C$

$K \bmod 3 \to z$

$\nu[\nu(n \bmod 5 - 3) + (n \bmod 5) \bmod 2 - z] \cdot R \to \pi$

$\pi \to R$

$F + n \bmod 5 \to G$

$C \cdot (\downarrow G) + H \to H$

$\nu^2(n \bmod 5 - 4) \cdot S + \nu(\bmod 5 - 4) \cdot T \to \pi$

$\pi \to S$

$\nu^2[z \cdot (C_\pi - C)] \to p$

$P \cdot p \to P$

$Q \to \pi$

$\pi \to T$

$\nu^2[z \cdot (C_\pi - C)] \to p$

$\nu(p + P - 1) \cdot h + \nu^2(p + P - 1) \cdot V \to \pi$

$\pi \to V$

$H \to \downarrow J$

$J + 1 \to J$

$N \to \pi$

Descrizione del Formalismo

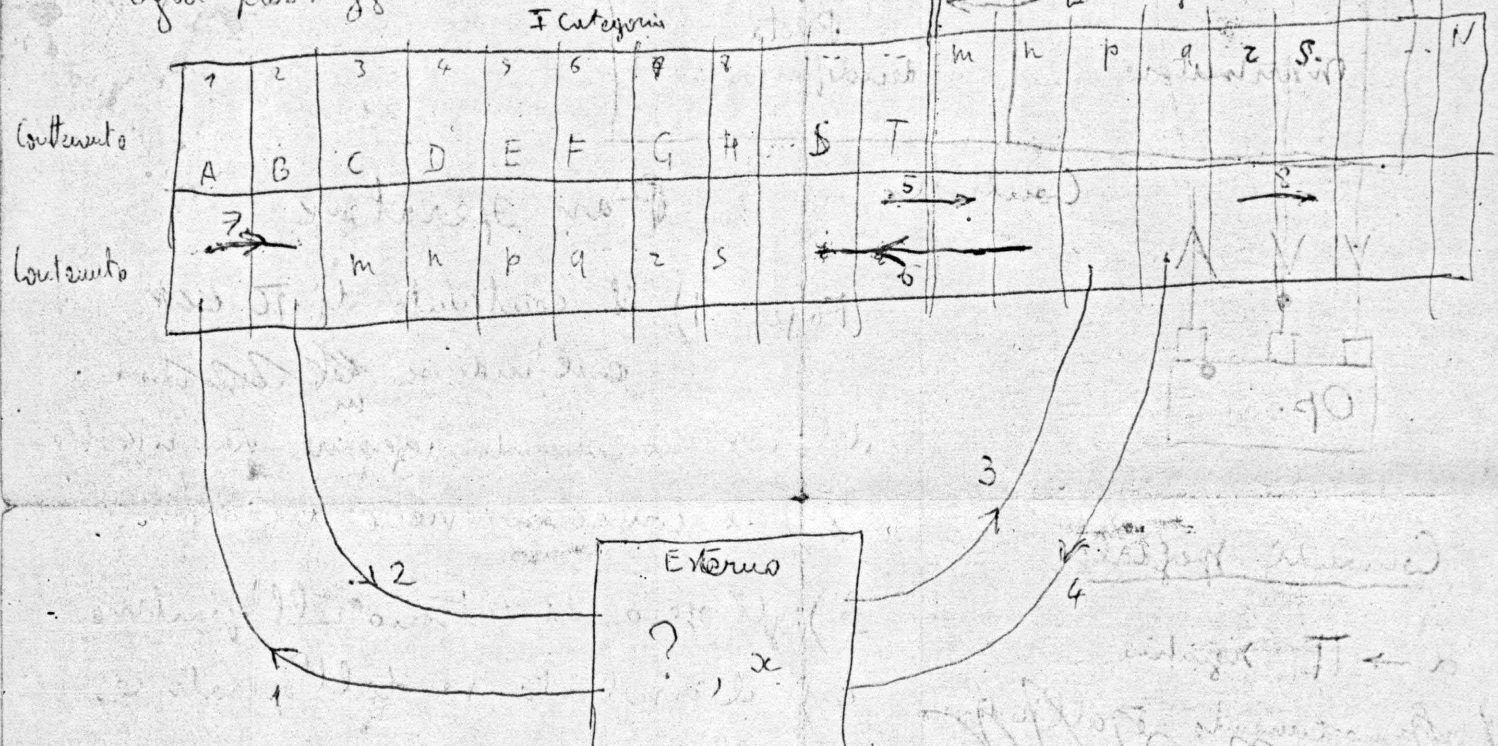
Consideriamo una macchina avente

1 N cellule di cui ~~alcune possono~~ numerate da 1 a N di cui però le prime L sono caratterizzate da lettere alfabetiche (maiuscole, greche ecc.) A, B ... Z.

2 Ogni cellula può contenere 0 oppure un numero positivo intero $\leq N$ (capacità ~~limitata~~) massimo delle cellule = numero di cellule!)

(3 I numeri negativi con complemento)

Cerchiamo di rappresentare con un formalismo adatto ogni passaggio di numeri dall'esterno all'interno ecc.



1) $? \to A$
$x \to A$ (part)

2) $B \to ?$

3) $? \to \downarrow E$ oppure $? \to p$
$x \to \downarrow E$ (part) $x \to p$

4) $\downarrow F \to ?$

5) $T \to \downarrow C$ oppure $T \to m$

6) $\downarrow D \to S$

7) $A \to B$

8) $\downarrow G \to \downarrow H$ oppure $\downarrow G \to s$

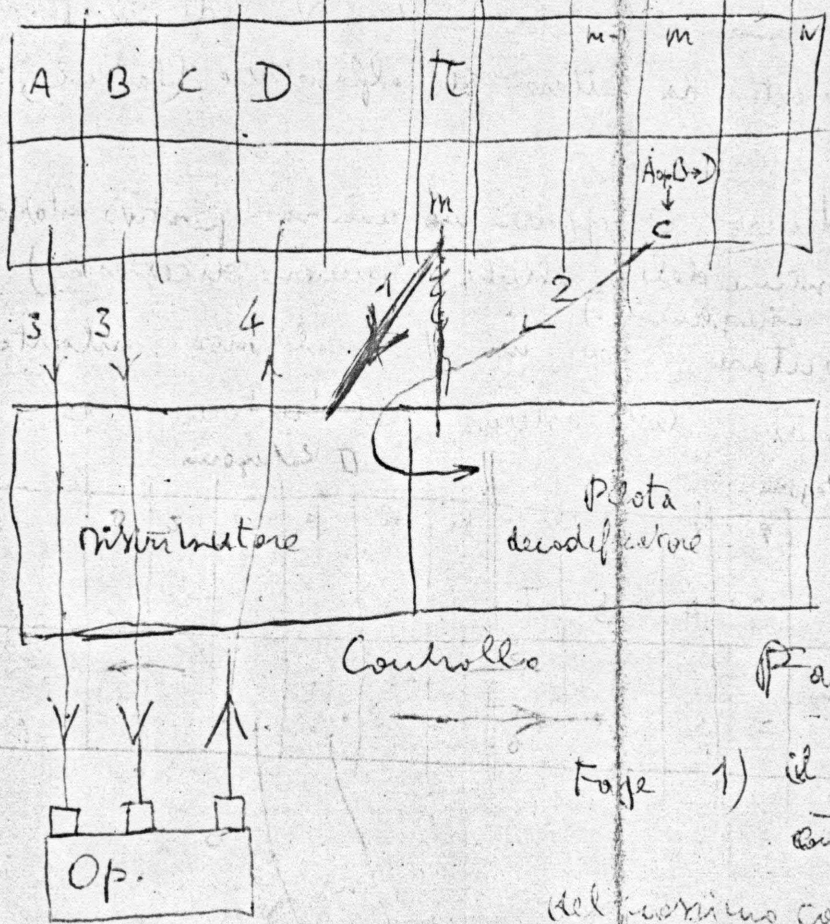
Una cellula speciale detta π contiene sempre l'indirizzo del prossimo comando da effettuare.

Si ottiene con questo metodo che ciò che sta a sinistra di → è un numero, rappresentato spesso col contenente; mentre ciò che sta a destra è ~~un contenente~~ sempre un indirizzo di cellula sempre o contenente.

Disegnazione del funzionamento relativo a una macchina 30 Nov 50

5/8.51

Operazioni tipiche



il numero C rappresenti il comando A op B → D

Fasi operative

Fase 1) il contenuto di π ~~...~~ cioè l'indirizzo ~~...~~ m del prossimo comando appare ~~...~~

2) il comando viene ...

3) gli operandi entrano nell'operatrice

4) il risultato va dall'operatrice alla memoria

5) (Non disegnato) ~~il prossimo indirizzo~~ l'indirizzo del prossimo comando entra nella cellula π.

Comandi speciali

α → π significa

~~il prossimo comando è all'indirizzo~~

~~α ... α → π ... qualora → π significa che il numero ...~~

α → π

l'indirizzo del prossimo comando è α o il contenuto di α (se α è una cellula)

π → A il ~~prossimo com~~ numero contenuto in π va nella cellula A od anche ciò che segue si dice in A.

Convenzione speciale

0 → π è il comando di stop della macchina in quanto non esiste nessuna cellula ~~che è numerata con~~ 0. il cui indirizzo è 0.

PROGRAMMI TIPICI

25 nov. 50

PROGRAMMI TIPICI
ESPRESSI NEL FORMALISMO ↓ →

1. Si sia da formulare il programma per il calcolo della funzione

$$y_i = f(x_i) \qquad (i = 1, 2 \cdots m)$$

fissato il valore di x_1 nella cellula 147, di x_2 nella cellula 148 ecc.
si voglia avere y_1 nella cellula 353, y_2 nella cellula 354 ecc.
Simboleggiamo il programma per calcolare f con lo stesso simbolo f. Il programma è analogo anche nel caso di una funzione di due variabili.

PROGRAMMA 1

$147 \rightarrow X$
$353 \rightarrow Y$
$0 \rightarrow N$
$m \rightarrow M$

$\pi \rightarrow A$
$f(\downarrow X) \rightarrow \downarrow Y$
$X+1 \rightarrow X$
$Y+1 \rightarrow Y$
$N+1 \rightarrow N$
$\{1 \cap [0 \cup (M-N)]\} \cdot A \rightarrow \pi$

2. Si sia da formulare il programma per il prodotto scalare di due vettori a m componenti

$$Z = \sum_{i=1}^{i=m} x_i y_i$$

Tale programma si può anche formulare con formula ricorrente più adatta alla codificazione

ponendo $Z = \sum_{i=1}^{i=m} z_i$, $z_i = x_i \cdot y_i$, $r = (0, 1, \ldots m-1)$

e scrivendo $t_r + z_{r+1} = t_{r+1}$

si ottiene $t_m = Z$

Si trovi la prima componente di $\vec{X}$ alla cellula 25, la seconda (26) ecc.
e la prima componente di $\vec{Y}$ " " 125 " " (126) ecc.
Si desidera il risultato alla cellula 200.

PROGRAMMA 2

$0 \rightarrow$ cellula 200
$200 \rightarrow Z$
$25 \rightarrow X$
$125 \rightarrow Y$
$0 \rightarrow N$
$m \rightarrow M$
$\pi \rightarrow B$
$(\downarrow X)\cdot(\downarrow Y) + \downarrow Z \rightarrow \downarrow Z$
$X+1 \rightarrow X$
$Y+1 \rightarrow Y$
$N+1 \rightarrow N$
$\{1 \cap [0 \cup (M-N)]\} \cdot B \rightarrow \pi$

3. Prodotto di due matrici di ordine m

l'elemento generico a_{il} (i = riga k = colonna) della prima matrice si trovi alla cellula $(a-1)+l+(i-1)m$; analogamente l'elemento b_{lk} della seconda matrice si trovi nella cellula $b-1+k+(l-1)m$. L'elemento prodotto c_{ik} si troverà nella cellula $c-1+k+(i-1)$ Inoltre all'inizio tutte le cellule $c-1+k+(i-1)\cdot m$ ($0 \leq i, k \leq$ contengano lo zero.

Si chiede il programma per

$$c_{ik} = \sum_{l=1}^{l=m} a_{il}\, b_{lk}$$

Tale programma è una generalizzazione del programma per il prodotto scalare di due vettori. Nel seguito si simboleggia ~~(R) l'espressione~~ per brevità

$$1 \cap (0 \cup R) \leftarrow (R)$$

Programma 3

$0 \to c-1+k+(i-1)\cdot m \quad [1 \le i,\, k \le m]$

$1 \to i$

$\pi \to T$

$1 \to k$

$\pi \to S$

$1 \to l$

$\pi \to P$

$a-1+l+(i-1)\cdot m \to X$

$b-1+k+(l-1)\cdot m \to Y$

$c-1+k+(i-1)\cdot m \to Z$

$(\downarrow X)\cdot(\downarrow Y)+\downarrow Z \to \downarrow Z$

$l+1 \to l$

$(m+1-l)\cdot P + [1-(m+1-l)]\pi \to \pi$

$k+1 \to k$

$(m+1-k)\cdot S + [1-(m+1-k)]\cdot\pi \to \pi$

$i+1 \to i$

$(m+1-i)\cdot T \to \pi$

GRAFO.

30 NOV 50 Completamento di programmi tipici

1 Sia da formulare il programma per

$$y_i = f(x_i) \qquad i = 1, 2 \dots m$$

sia il valore di x_1 nella cellula p, x_2 nella cellula $p+1$ ecc. ed il valore di y_1 si voglia avere nella cellula q, y_2 nella $q+1$ ecc.

$m \to M$
$0 \to N$
$p \to X$
$q \to Y$
$\pi \to A$
$? \to \downarrow X$
$f(\downarrow X) \to \downarrow Y$
$X+1 \to X$
$Y+1 \to Y$
$N+1 \to N$
$\{1 \cap [0 \cup (M-N)]\} \cdot A \to \pi$

2 Programma per il prodotto scalare di due vettori a m componenti.

$$Z = \sum_{i=1}^{i=m} x_i y_i$$

Tale programma si può formulare con ricorrenza (più adatta) ponendo

$$Z = \sum_{i=1}^{i=m} z_i \qquad z_i = x_i y_i$$

e scrivendo

$$t_r + z_{r+1} = t_{r+1} \qquad r = (0, 1 \dots m-1)$$

dove $t_0 = 0$. Allora si ha

$$t_m = Z$$

La prima componente di $\vec{X}$ si trovi nella cellula p, la seconda $p+1$ ecc.
" " " di $\vec{Y}$ " " " q " " $q+1$ "
Si desidera il risultato nella cellula Z.

$0 \to Z$
$p \to X$
$q \to Y$
$0 \to N$
$m \to M$
$\pi \to B$
$? \to \downarrow X$
$? \to \downarrow Y$
$(\downarrow X) \cdot (\downarrow Y) + Z \to Z$
$X+1 \to X$
$Y+1 \to Y$
$N+1 \to N$
$\{1 \cap [0 \cup (M-N)]\} \cdot B \to \pi$

Prodotto di due matrici di ordine m

L'elemento a_{11} si trovi nella cellula a, a_{12} nella cellula $a+1$, a_{1m} nella cellula $a+(m-1)$, l'elemento a_{21} nella cellula $a+m$ ecc. cioè l'elemento

a_{il} si troverà nella cellula

$$a-1 + l + (i-1)m$$

analogamente l'elemento b_{lk} nella cellula

$$b-1 + k + (l-1)m$$

l'elemento prodotto c_{ik} si voglia trovare nella cellula $c-1 + k + (i-1)$. Tutte queste cellule all'inizio contengono lo zero.

Si chiede il programma per

$$c_{ik} = \sum_{l=1}^{l=m} a_{il}\, b_{lk}$$

Tale programma è una generalizzazione del programma per il prodotto scalare di due vettori – Nel seguito simboleggeremo per brevità con (R) l'espressione

$$1 \cap (O \cup R)$$

~~alza~~

$m \to M$
$o \to N$
$c \to Z$ C
$z \to X$
$a \to X$
$x \to A$
$b \to Y$
$b \to B$
$\pi \to R$
$0 \to \downarrow Z$
$? \to \downarrow X$
$? \to \downarrow Y$
$N+1 \to N$
$Z+1 \to Z$
$X+1 \to X$
$Y+1 \to Y$

$[1-(M^2-N)]U + (M^2-N)\cdot R \to \pi$

preparazione ed entrata

Esecuzione

$\pi \to U$
$1 \to i$
$\pi \to T$
$1 \to K$
$\pi \to S$
$1 \to l$
$\pi \to P$

$A-1+l+(i-1)M \to X$
$B-1+K+(l-1)M \to Y$
$C-1+K+(i-1)M \to Z$
$(\downarrow X)\cdot(\downarrow Y)+\downarrow Z \to \downarrow Z$
$l+1 \to l$
$(M+1-l)\cdot P + [1-(M+1-l)]\,\pi \to \pi$
$K+1 \to K$
$(M+1-K)\cdot S + [1-(M+1-K)]\,\pi \to \pi$
$i+1 \to i$
$(M+1-i)\cdot T \to \pi$

~~$m \to M$~~
~~$o \to N$~~
~~$c \to Z$~~
~~$\pi \to R$~~
~~$0 \to \downarrow Z$~~
~~$N+1 \to N$~~
~~$Z+1 \to Z$~~
~~$[1-(M^2-N)]U + (M^2-N)\cdot R \to \pi$~~
~~$\pi \to U$~~
~~$1 \to i$~~
~~$\pi \to T$~~
~~$1 \to K$~~
~~$\pi \to S$~~
~~$1 \to l$~~
~~$\pi \to P$~~

S 1 ÷ S 21

S 3
S 4

S U P P O R T O   A L L A   D I S C U S S I O N E

4183
78e

SEZIONE I Logica e teoria della programmazione

Lavoro n. 3 " Calculatrices digitales. Du déchiffrage des formules...(1954)
" n. 4 " Sulla programmazione mediante formule (1954)

Motivazione , originalità, cenni storici

Documentazione

Vedi dissertatione 3 nota pag.

La lettura di un lavoro di Zuse nel 1949 ha suggerito a Böhm la possibilità di analizzare meccanicamente una qualsiasi formula con parentesi e di fare un pannello per schede Bull che abilitasse una tabulatrice a rispondere se tale formula fosse sintatticamente accettabile o no. Più tardi egli propose al prof. Stiefel(di cui era assistente presso l' ETH di Zurigo) di essere relatore di una tesi di dottorato sulla programmazione automatica (oggi si direbbe sulla costruzione di un compilatore) e per giustificare tale titolo descrisse verbalmente un algoritmo per tradurre una formula con parentesi in successione di istruzioni di una data macchina. Nel 1951 una prima stesura della dissertazione era pronta ma non fu accettata da Stiefel adducendo come causa la forma imperfetta. Qualche mese dopo appariva un articolo di Rutishauser (altro assistente di Stiefel che Böhm appunto rimpiazzava in quanto era nel 1950 in America) ove veniva descritto il medesimo procedimento senza citare la fonte (Ecco perchè,nella citazione di Samelson e Bauer, Rutishauser risulta il primo ad occuparsi di simili problemi). Böhm fu obbligato a cambiare procedimento ed ebbe la fortuna e l'abilità di trovare un metodo più generale e più interessante, che presentò nella stesura finale nel 1952 e che fu approvata da Stiefel, che tuttavia non autorizzò la stampa nelle "Mitteilungen" dell'Istituto che egli dirigeva. Böhm allora non sapendo quando la dissertazione avrebbe potuto essere stampata depositò come brevetto italiano le idee essenziali ivi contenute.

given; ready; Some months later appeared; replaced; therein

x 3/2

t 3-1 x 3

d (Minister)

Importanza dei lavori n.3 e 4.

La problematica trattata da Böhm ebbe uno sviluppo enorme nella costruzione del Fortran (1954-55). Più tardi Rutishauser, divenuto nel frattempo professore ordinario, fu tra i promotori del linguaggio Algol (1958) che seguiva e sviluppava le medesime linee di pensiero iniziate da Böhm. Dei lavori di Böhm si parlò poco e solo nel 1960 nell'articolo di Samelson e Bauer fu riconosciuta, almeno in parte la sua importanza. Tuttavia la citazione bibliografica rimase incompleta, in apparenza inspiegabilmente. Una spiegazione, non precisamente

x 3

t 3-2

S3
S4

2

(seguito importanza dei lavori n. 3, e 4)

Documentazione

bonaria , può però essere fatta osservando la grande somiglianza fra il brevetto tedesco richiesto dagli autori Bauer e Samelson nel 1958 e quello italiano ottenuto nel 1952 da Böhm.

Un riflesso di questo poco chiaro stato di cose si può notare osservando un inciso scritto dagli autori del libro "Algol 60 implementation", i quali casualmente vennero a conoscenza del lavoro di Böhm al momento della correzione delle bozze: " Questo lavoro, che molti ritengono meritasse più attenzione di quanta in origine ne abbia ricevuta, ...".

L3 - 3÷76

a4 (Ministero)

Il lavoro n. 3 è stato recensito (examined) dall'analista numerico americano Goldstine e ha avuto una lunghissima recensione da parte del logico Tamari , molto lusinghiera.

R3-4

Böhm, Corrado: Calculatrices digitales. Du déchiffrage de formules logico-mathématiques par la machine même dans la conception du programme. Ann. Mat. pura appl., IV. Ser. **37**, 175—217 (1954).

Für die Herstellung eines Rechenprogramms in seiner endgültigen, durch die Maschine ausführbaren Form ist die monotone, aber viel Sorgfalt erfordernde Arbeit der Explizierung und Übersetzung des mathematischen Programms in eine der Maschine „verständliche", d. h. entwurfsgemäß vorgesehene konventionelle Sprache, den „Code", bezeichnend. Um dies überflüssig zu machen und diese wichtige Quelle von Irrtümern und damit verbundenem Zeitverlust zu beseitigen, konstruiert man entweder besondere Hilfsautomaten (H. H. Aiken, K. Zuse) oder benutzt die in Cambridge (England) entwickelte Methode der „library of sub-routines" (M. V. Wilkes, Report on the preparation of programmes for EDSAC etc., Cambridge 1950). Die in dieser Arbeit (eine schon 1952 der ETH Zürich vorgelegte Diss.) vorgeschlagene Methode der automatischen Kodifikation besteht in folgendem: Man schreibt das Programm in einer der traditionellen mathematischen Schreibweise ziemlich nahen, allerdings geeignet standardisierten Form als eine Folge elementarer Formeln nieder; nach einem dementsprechend festgelegten Rahmenprogramm — das man sinngemäß Superroutine nennen könnte — liest und versteht es die Maschine, d. h. interpretiert es automatisch als ein im engeren Sinn detailliertes Programm, das es beantworten kann. Dies geht über die in dieser Richtung liegenden Arbeiten von K. Zuse (dies. Zbl. 32, 360) und H. Rutishauser (dies. Zbl. 49, 212) hinaus [und übertrifft an Allgemeinheit auch eine von A. E. Glennie (ca. 1952—3) in Manchester ausprobierte und anscheinend sich wohlbewährende ähnliche Methode. Anm. d. Ref.] Es ist natürlich nicht überraschend, daß die sogenannten Universalmaschinen („all purpose"), deren schon viele (besonders in USA und England) gebaut worden sind, ipso facto eine so weitgehende Selbstautomatisierung gestatten. Nichtsdestoweniger ist die wirkliche Durchführung interessant. Die für idealisierte Maschinen geltenden Erkenntnisse von A. M. Turing (dies. Zbl. 16, 97) liefern dem Verf. zwei Arbeitshypothesen: die logische Äquivalenz aller Universalmaschinen und die Identifizierbarkeit der Definition einer Maschine (d. h. ihres Funktionierens) mit der einer numerischen Rechenmethode. Die vorgeschlagene Methode ist insofern invariant, als sie im Prinzip für jede Universalmaschine gilt; sie wird jedoch im einzelnen an einer besonderen dezimalen „Drei-Adressen"-Universalmaschine, die wirklich gebauten Maschinen ähnelt, erläutert. Die praktische Durchführung der automatischen Kodifikation benötigt zusätzlich nur einen geeigneten Fernschreiber, der die algebraisch-logischen Symbole — 1. Klammern; 2. → „wird" (Substitutions- und Transfersymbol); 3. Buchstaben, d. h. Variablen und Konstanten einschl. der syntaktischen Symbole des Rahmenprogramms; 4. binäre, z. T. entbehrliche Operationssymbole — in eineindeutig umkehrbarer Weise in höchstens 4-stellige Zahlen verwandelt, aus denen das erwähnte Rahmenprogramm kodifizierte Instruktionen in Form 14-stelliger Zahlen aufbaut und sie, je nachdem, ausführt, oder im Gedächtnis stapelt, usw. Die gröbere Struktur dieses automatischen Kodifikationsprogramms läßt sich mit Hilfe eines „Flow"-Diagramms nach H. H. Goldstine und J. von Neumann (Planning and coding of problems for an electronic computing instrument, part II, vol. I, Princeton 1947) graphisch übersehen. Vom Verf. überwundene Schwierigkeiten und daraus folgende Fotschritte in der Kunst der Programmherstellung sind: 1. Die methodisch wichtige abstrakte Formulierung eines allgemeinen zyklischen Grundprogramms; 2. die einfache Programmation einer verallgemeinerten „bedingten" Instruktion („wenn die an der Adresse a sich befindende Zahl $C(a) > 0$ ist, so soll die an der Adresse x, anderenfalls die an der Adresse y sich befindende Instruktion ausgeführt werden"); 3. die abstrakte Einführung beliebig iterierter Substitutions- und Erkundungsoperationen, die Adressen von Adressen usw. benutzen (ersetzt die sogenannte „B-tube"); 4. Indizierung und Adressenkalkül (weitergehend als das „floating address"-System von Wilkes, dies. Zbl. 50, 133. Anm. d. Ref.); 5. automatische Kodifizierung beliebig verschachtelter Klammerausdrücke; 6. im Programm vorgesehene sofortige Stoppung der Maschine, wenn sinnlose Formeln erscheinen. Zum Schluß wird noch die wohlbekannte Überschüssigkeit gewisser arithmetischer und logischer Operationen diskutiert. — Der Ref. macht auf die weniger bekannte, auch dem Verf. entgangene, Überschüssigkeit der üblichen Klammernbezeichnung aufmerksam: Man kann z. B. alle Schließungsklammern vernachlässigen, wenn man, wie der Verf. es vorschreibt, die Assoziativität ignoriert. Überhaupt dürfte die Anwendung der Lukasiewiczschen Symbolik für binäre wie auch andere Operationen und damit verbundene arithmetische Formulierungen weitgehende Vereinfachung und Verallgemeinerung gestatten. Zum Problem sinnvoller Formeln sind auch die „linguistischen" Ideen von E. L. Post belangreich (für ein hierher gehörendes arithmetisches Kriterium siehe P. Rosenbloom, The elements of mathematical logic, dies. Zbl. 41, 148, p. 154, Theorem 1). Die automatische Kodifikation macht natürlich die bewährte Methode der Subroutinenbibliothek nicht überflüssig; vielmehr gestattet sie, die letztere nach geeigneter Modifikation auf weniger elementare Teile der Programmherstellung zu konzentrieren. Wohl infolge des Mangels geeigneter Maschinen im kontinentalen Europa vor 1952 spricht die Arbeit nicht von der praktischen und ökonomischen Bewährung der Methode, insbesondere auch nicht über die für die automatische Kodifikation benötigte zusätzliche Maschinenzeit und Gedächtniskapazität, die sicher nicht unerheblich sind. Häufiger Bezeichnungswechsel und einige geringfügige Druckfehler oder Versehen (in solchen Arbeiten fast unvermeidlich) stören kaum die gute Lesbarkeit des Textes.

D. Tamari.

Zbl. für Math. Vol 57 (1956) pag 107-108

Vedi traduzione doc. a 3.

Böhm, Corrado. Calculatrices digitales. Du déchiffrage de formules logico-mathématiques par la machine même dans la conception du programme. Ann. Mat. Pura Appl. (4) **37**, 175–217 (1954).

The author proposes some engineering and coding methods toward automatic programming for computers. A basic three-address computer is augmented by a facility under which each address x of an order is accompanied by a digit specifying that the information-transfer acts on the location x, as customary, if $\epsilon=0$, but on a memory position specified in the location x, if $\epsilon=1$. This provides a means for handling variable addresses. A double use of a symbol for both address and content of a location permits literal equations such as $a\cdot b\rightarrow r$, $A\rightarrow\pi$ (the latter effecting a control-transfer) to be key-punched directly, a small input code being still needed. A program is then presented [different from that of H. Rutishauser, Mitt. Inst. Angew. Math. Zürich no. 3 (1952); MR 15, 64] for translating a certain class of parenthetical expressions into machine code.

H. H. Goldstine (Princeton, N. J.).

MATH. REV Vol 16 n. 9 pag 963 (1955)

2867:

Böhm, Corrado. Sulla programmazione mediante formule. Consiglio Naz. Ricerche. Pubbl. Ist. Appl. Calcolo no. 423 (1955), 9 pp.

Brief description of a problem-oriented language.

A. S. Householder (Oak Ridge, Tenn.)