



Linguistic History of Software Engineering

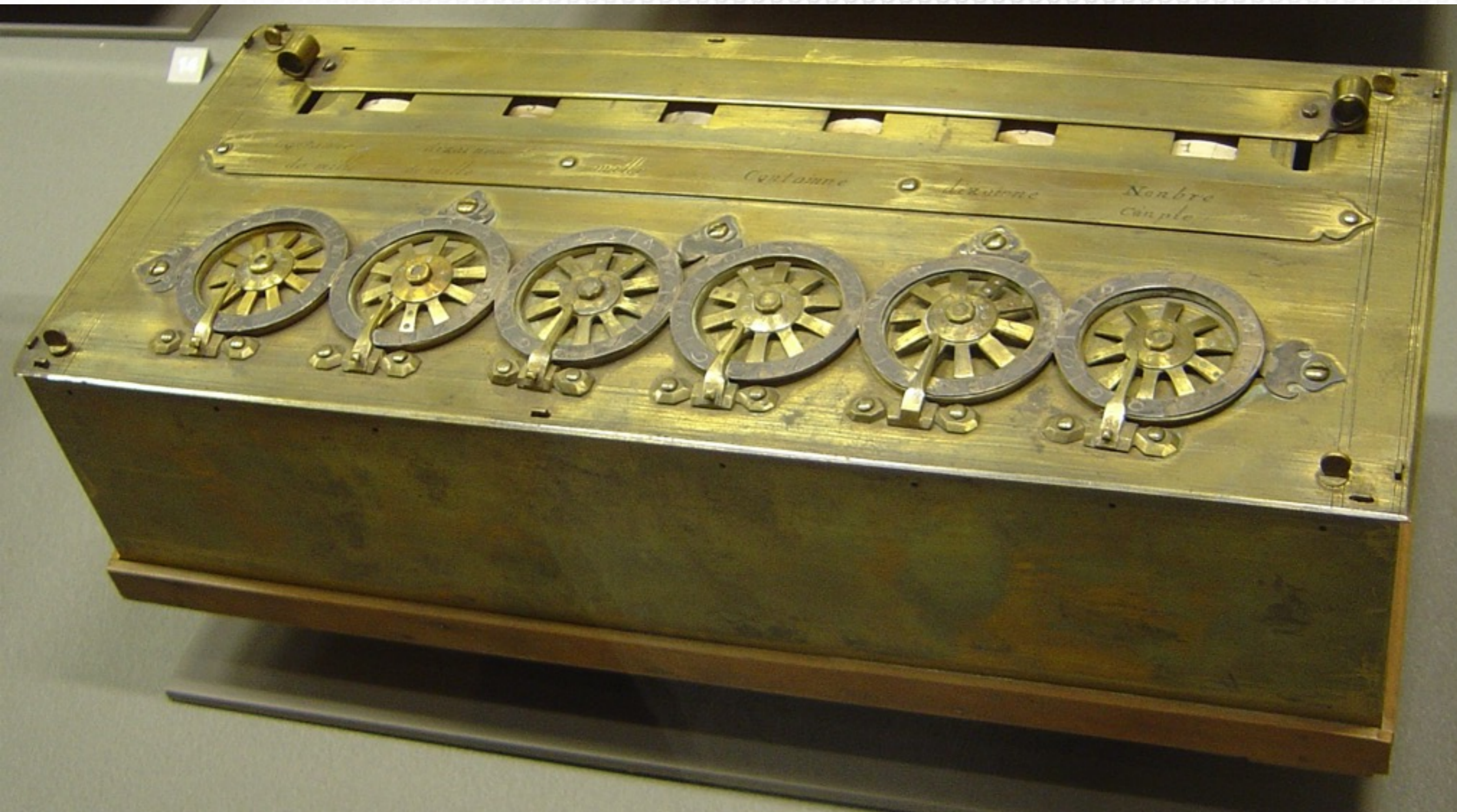
Vadim Zaytsev aka @grammarware,
Universiteit van Amsterdam, 2014

CC-BY-SA

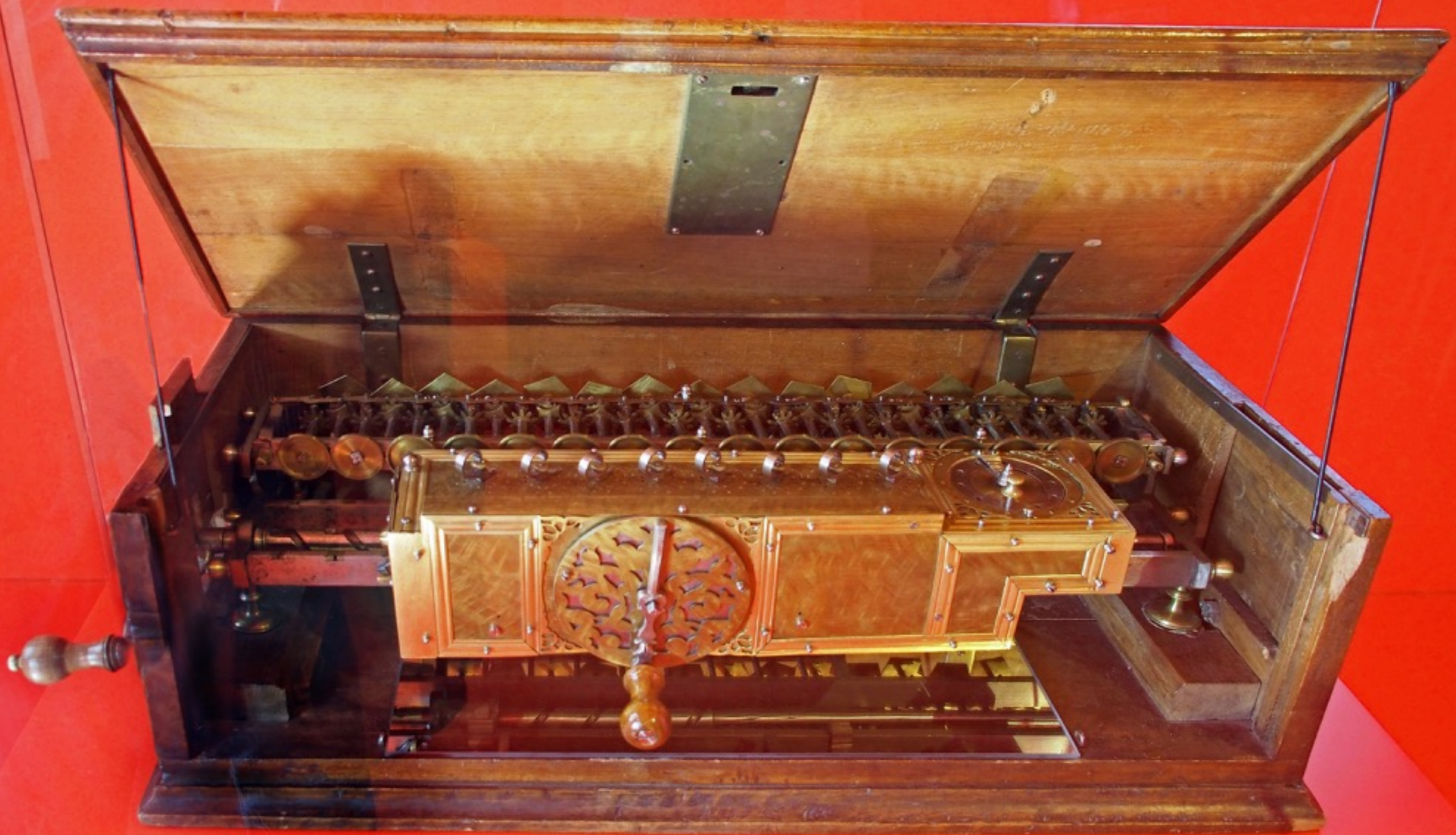
Vadim Zaytsev



- Universiteit van Amsterdam (2013–2016?)
- Centrum Wiskunde & Informatica (2010–2013)
- Universität Koblenz-Landau (2008–2010)
- Vrije Universiteit Amsterdam (2004–2008)
- Universiteit Twente (2002–2004)
- Rostov State Transport University (1999–2008)
- Rostov State University (1998–2003)

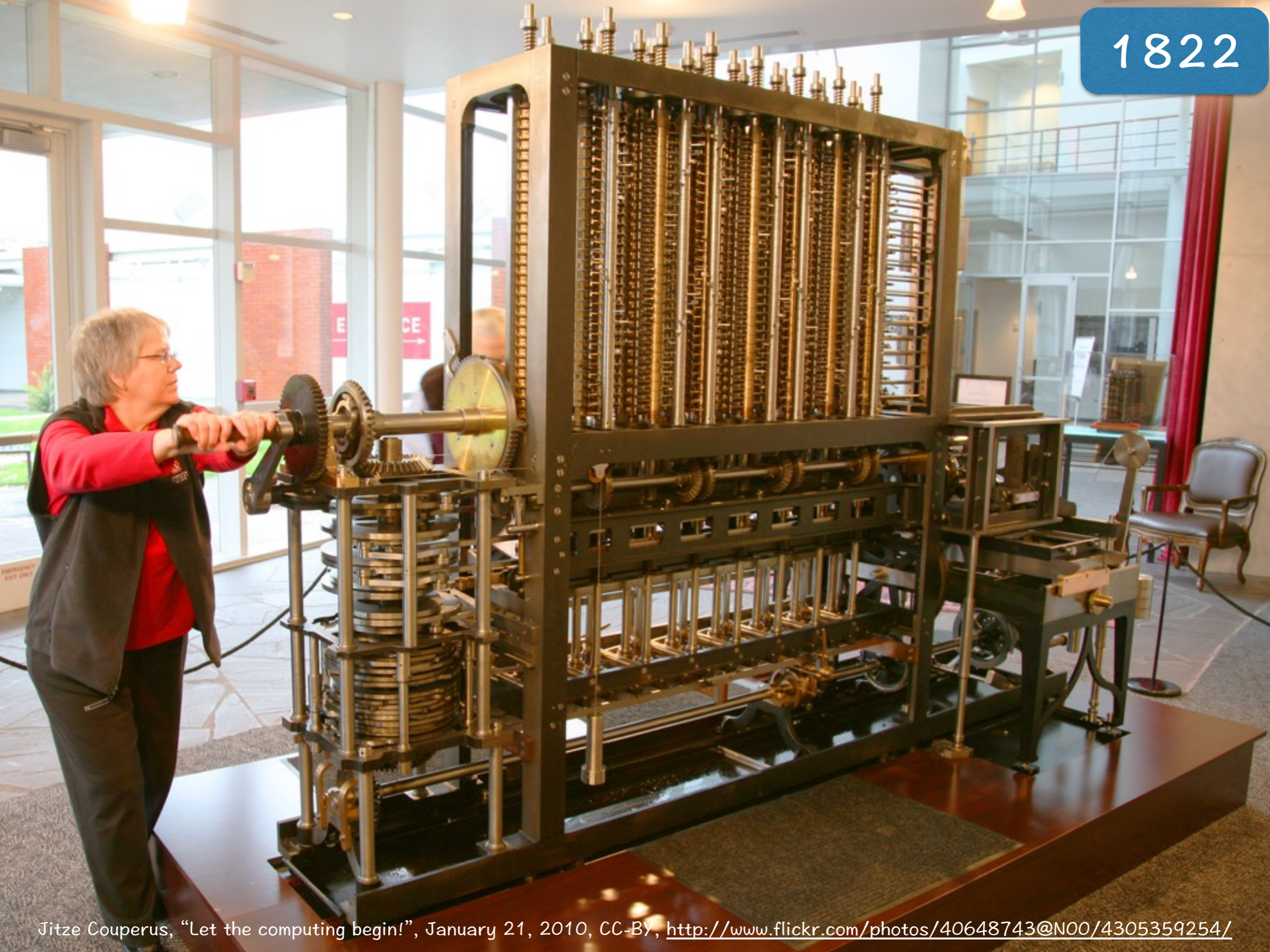


David Monniaux, "Machine à calculer de Blaise Pascal sans sous ni deniers",
[https://commons.wikimedia.org/wiki/File:Arts et Metiers Pascaline dsc03869.jpg](https://commons.wikimedia.org/wiki/File:Arts_et_Metiers_Pascaline_dsc03869.jpg), CC-BY-SA

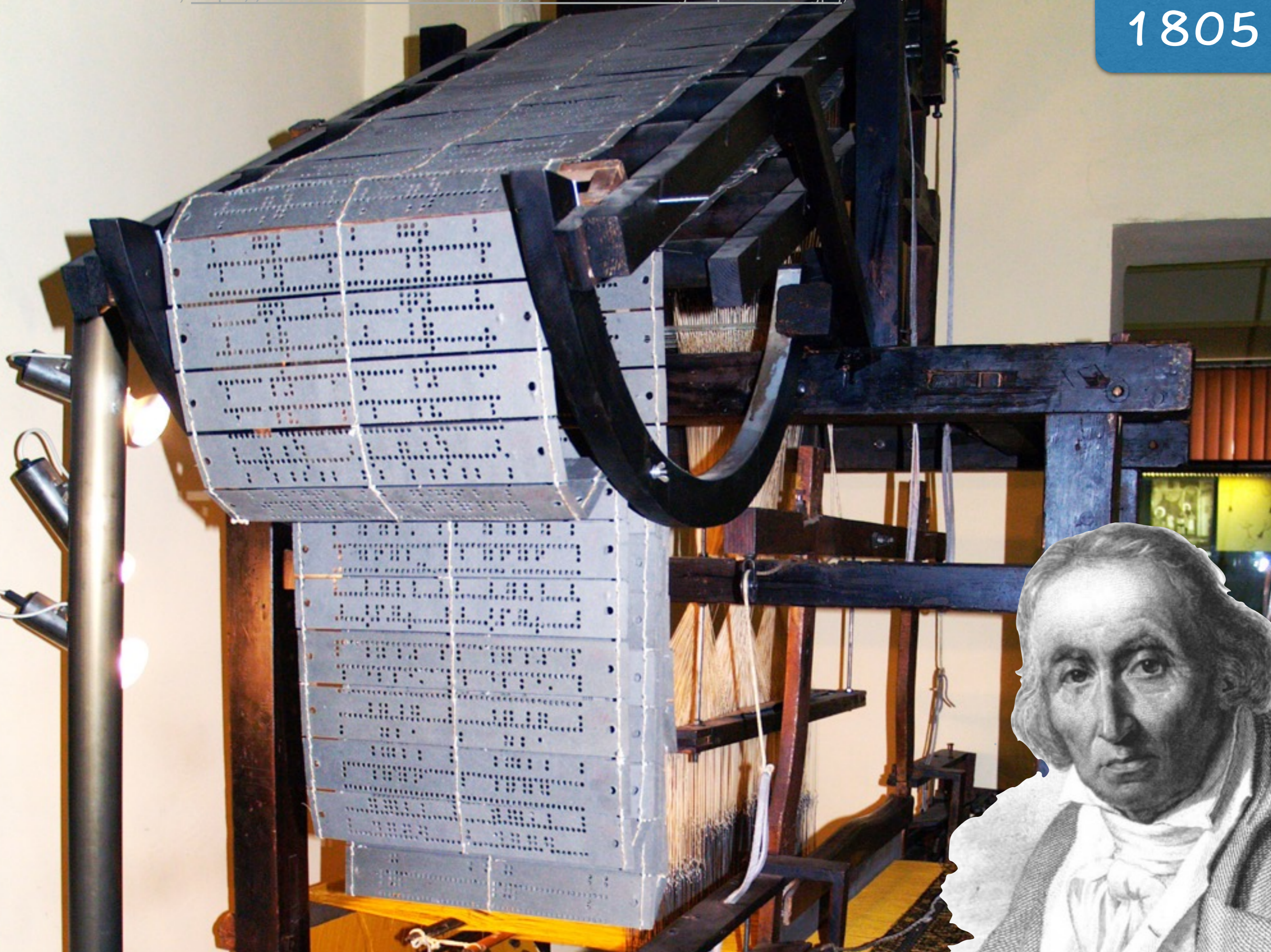


Hajotthu, "Leibniz' Vier-Spezies-Rechenmaschine",
[https://commons.wikimedia.org/wiki/File:Leibniz Rechenmaschine \(1690\).jpg](https://commons.wikimedia.org/wiki/File:Leibniz_Rechenmaschine_(1690).jpg), CC-BY

1822



1805



These cards, however, have nothing to do with the regulation of the particular *numerical* data. They merely determine the *operations** to be effected, which operations may of course be performed on an infinite variety of particular numerical values, and do not bring out any definite numerical results unless the numerical data of the problem have been impressed on the requisite portions of the train of mechanism. In the above example, the first essential step towards an arithmetical result, would be the substitution of specific numbers for n , and for the other primitive quantities which enter into the function.

Again, let us suppose that for F we put two complete equations of the fourth degree between x and y . We must then express on the cards the law of elimination for such equations. The engine would follow out those laws, and would ultimately give the equation of one variable which results from such elimination. Various *modes* of elimination might be selected; and of course the cards must be made out accordingly. The following is one mode that might be adopted. The engine is able to multiply together any two functions of the form

$$a + bx + cx^2 + \dots px^n.$$

This granted, the two equations may be arranged according to the powers of y , and the coefficients of the powers of y may be arranged according to powers of x . The elimination of y will result from the successive multiplications and subtractions of several such functions. In this, and in all other instances, as was explained above, the particular *numerical* data and the *numerical* results are determined by means and by portions of the mechanism which act quite independently of those that regulate the *operations*.

In studying the action of the Analytical Engine, we find that the peculiar and independent nature of the considerations which in all mathematical analysis belong to *operations*, as distinguished from *the objects operated upon* and from the *results* of the operations performed upon those objects, is very strikingly defined and separated.

It is well to draw attention to this point, not only because its full appreciation is essential to the attainment of any very just and adequate general comprehension of the powers and mode of action of the Analytical Engine, but also because it is one which is perhaps too little kept in view in the study of mathematical science in general. It is, however, impossible to confound it with other considerations, either when we trace the manner in which that engine attains its results, or when we prepare the data for its attainment of those results. It were much to be desired, that when mathematical processes pass through the human brain instead of through the medium of inanimate mechanism, it were equally a necessity of things that the reasonings connected with *operations* should hold the same just place as a clear and well-defined branch of the subject of analysis, a fundamental but yet independent

* We do not mean to imply that the *only* use made of the Jacquard cards is that of regulating the algebraical *operations*. But we mean to explain that *those* cards and portions of mechanism which regulate these *operations*, are wholly independent of those which are used for other purposes. M. Menabrea explains that there are *three* classes of cards used in the engine for three distinct sets of objects, viz. *Cards of the Operations*, *Cards of the Variables*, and certain *Cards of Numbers*. (See pages 678 and 687.)



Ada Lovelace (1815–1852)

1954



8

1954



9

1952

7

5



6

1952

5



4



Adriaan J. van Wijngaarden (1916–1987)

photo credit: <http://www.kennislink.nl/publicaties/rekenmeisjes-en-rekentuig>



| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|-----|-----|-----|-----|-----|-----|-----|-----|------|
| 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 |
| 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 |
| 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 |
| 128 | 129 | 130 | 131 | 132 | 133 | 134 | 135 | 136 |
| 137 | 138 | 139 | 140 | 141 | 142 | 143 | 144 | 145 |
| 146 | 147 | 148 | 149 | 150 | 151 | 152 | 153 | 154 |
| 155 | 156 | 157 | 158 | 159 | 160 | 161 | 162 | 163 |
| 164 | 165 | 166 | 167 | 168 | 169 | 170 | 171 | 172 |
| 173 | 174 | 175 | 176 | 177 | 178 | 179 | 180 | 181 |
| 182 | 183 | 184 | 185 | 186 | 187 | 188 | 189 | 190 |
| 191 | 192 | 193 | 194 | 195 | 196 | 197 | 198 | 199 |
| 200 | 201 | 202 | 203 | 204 | 205 | 206 | 207 | 208 |
| 209 | 210 | 211 | 212 | 213 | 214 | 215 | 216 | 217 |
| 218 | 219 | 220 | 221 | 222 | 223 | 224 | 225 | 226 |
| 227 | 228 | 229 | 230 | 231 | 232 | 233 | 234 | 235 |
| 236 | 237 | 238 | 239 | 240 | 241 | 242 | 243 | 244 |
| 245 | 246 | 247 | 248 | 249 | 250 | 251 | 252 | 253 |
| 254 | 255 | 256 | 257 | 258 | 259 | 260 | 261 | 262 |
| 263 | 264 | 265 | 266 | 267 | 268 | 269 | 270 | 271 |
| 272 | 273 | 274 | 275 | 276 | 277 | 278 | 279 | 280 |
| 281 | 282 | 283 | 284 | 285 | 286 | 287 | 288 | 289 |
| 290 | 291 | 292 | 293 | 294 | 295 | 296 | 297 | 298 |
| 299 | 300 | 301 | 302 | 303 | 304 | 305 | 306 | 307 |
| 308 | 309 | 310 | 311 | 312 | 313 | 314 | 315 | 316 |
| 317 | 318 | 319 | 320 | 321 | 322 | 323 | 324 | 325 |
| 326 | 327 | 328 | 329 | 330 | 331 | 332 | 333 | 334 |
| 335 | 336 | 337 | 338 | 339 | 340 | 341 | 342 | 343 |
| 344 | 345 | 346 | 347 | 348 | 349 | 350 | 351 | 352 |
| 353 | 354 | 355 | 356 | 357 | 358 | 359 | 360 | 361 |
| 362 | 363 | 364 | 365 | 366 | 367 | 368 | 369 | 370 |
| 371 | 372 | 373 | 374 | 375 | 376 | 377 | 378 | 379 |
| 380 | 381 | 382 | 383 | 384 | 385 | 386 | 387 | 388 |
| 389 | 390 | 391 | 392 | 393 | 394 | 395 | 396 | 397 |
| 398 | 399 | 400 | 401 | 402 | 403 | 404 | 405 | 406 |
| 407 | 408 | 409 | 410 | 411 | 412 | 413 | 414 | 415 |
| 416 | 417 | 418 | 419 | 420 | 421 | 422 | 423 | 424 |
| 425 | 426 | 427 | 428 | 429 | 430 | 431 | 432 | 433 |
| 434 | 435 | 436 | 437 | 438 | 439 | 440 | 441 | 442 |
| 443 | 444 | 445 | 446 | 447 | 448 | 449 | 450 | 451 |
| 452 | 453 | 454 | 455 | 456 | 457 | 458 | 459 | 460 |
| 461 | 462 | 463 | 464 | 465 | 466 | 467 | 468 | 469 |
| 470 | 471 | 472 | 473 | 474 | 475 | 476 | 477 | 478 |
| 479 | 480 | 481 | 482 | 483 | 484 | 485 | 486 | 487 |
| 488 | 489 | 490 | 491 | 492 | 493 | 494 | 495 | 496 |
| 497 | 498 | 499 | 500 | 501 | 502 | 503 | 504 | 505 |
| 506 | 507 | 508 | 509 | 510 | 511 | 512 | 513 | 514 |
| 515 | 516 | 517 | 518 | 519 | 520 | 521 | 522 | 523 |
| 524 | 525 | 526 | 527 | 528 | 529 | 530 | 531 | 532 |
| 533 | 534 | 535 | 536 | 537 | 538 | 539 | 540 | 541 |
| 542 | 543 | 544 | 545 | 546 | 547 | 548 | 549 | 550 |
| 551 | 552 | 553 | 554 | 555 | 556 | 557 | 558 | 559 |
| 560 | 561 | 562 | 563 | 564 | 565 | 566 | 567 | 568 |
| 569 | 570 | 571 | 572 | 573 | 574 | 575 | 576 | 577 |
| 578 | 579 | 580 | 581 | 582 | 583 | 584 | 585 | 586 |
| 587 | 588 | 589 | 590 | 591 | 592 | 593 | 594 | 595 |
| 596 | 597 | 598 | 599 | 600 | 601 | 602 | 603 | 604 |
| 605 | 606 | 607 | 608 | 609 | 610 | 611 | 612 | 613 |
| 614 | 615 | 616 | 617 | 618 | 619 | 620 | 621 | 622 |
| 623 | 624 | 625 | 626 | 627 | 628 | 629 | 630 | 631 |
| 632 | 633 | 634 | 635 | 636 | 637 | 638 | 639 | 640 |
| 641 | 642 | 643 | 644 | 645 | 646 | 647 | 648 | 649 |
| 650 | 651 | 652 | 653 | 654 | 655 | 656 | 657 | 658 |
| 659 | 660 | 661 | 662 | 663 | 664 | 665 | 666 | 667 |
| 668 | 669 | 670 | 671 | 672 | 673 | 674 | 675 | 676 |
| 677 | 678 | 679 | 680 | 681 | 682 | 683 | 684 | 685 |
| 686 | 687 | 688 | 689 | 690 | 691 | 692 | 693 | 694 |
| 695 | 696 | 697 | 698 | 699 | 700 | 701 | 702 | 703 |
| 704 | 705 | 706 | 707 | 708 | 709 | 710 | 711 | 712 |
| 713 | 714 | 715 | 716 | 717 | 718 | 719 | 720 | 721 |
| 722 | 723 | 724 | 725 | 726 | 727 | 728 | 729 | 730 |
| 731 | 732 | 733 | 734 | 735 | 736 | 737 | 738 | 739 |
| 740 | 741 | 742 | 743 | 744 | 745 | 746 | 747 | 748 |
| 749 | 750 | 751 | 752 | 753 | 754 | 755 | 756 | 757 |
| 758 | 759 | 760 | 761 | 762 | 763 | 764 | 765 | 766 |
| 767 | 768 | 769 | 770 | 771 | 772 | 773 | 774 | 775 |
| 776 | 777 | 778 | 779 | 780 | 781 | 782 | 783 | 784 |
| 785 | 786 | 787 | 788 | 789 | 790 | 791 | 792 | 793 |
| 794 | 795 | 796 | 797 | 798 | 799 | 800 | 801 | 802 |
| 803 | 804 | 805 | 806 | 807 | 808 | 809 | 810 | 811 |
| 812 | 813 | 814 | 815 | 816 | 817 | 818 | 819 | 820 |
| 821 | 822 | 823 | 824 | 825 | 826 | 827 | 828 | 829 |
| 830 | 831 | 832 | 833 | 834 | 835 | 836 | 837 | 838 |
| 839 | 840 | 841 | 842 | 843 | 844 | 845 | 846 | 847 |
| 848 | 849 | 850 | 851 | 852 | 853 | 854 | 855 | 856 |
| 857 | 858 | 859 | 860 | 861 | 862 | 863 | 864 | 865 |
| 866 | 867 | 868 | 869 | 870 | 871 | 872 | 873 | 874 |
| 875 | 876 | 877 | 878 | 879 | 880 | 881 | 882 | 883 |
| 884 | 885 | 886 | 887 | 888 | 889 | 890 | 891 | 892 |
| 893 | 894 | 895 | 896 | 897 | 898 | 899 | 900 | 901 |
| 902 | 903 | 904 | 905 | 906 | 907 | 908 | 909 | 910 |
| 911 | 912 | 913 | 914 | 915 | 916 | 917 | 918 | 919 |
| 920 | 921 | 922 | 923 | 924 | 925 | 926 | 927 | 928 |
| 929 | 930 | 931 | 932 | 933 | 934 | 935 | 936 | 937 |
| 938 | 939 | 940 | 941 | 942 | 943 | 944 | 945 | 946 |
| 947 | 948 | 949 | 950 | 951 | 952 | 953 | 954 | 955 |
| 956 | 957 | 958 | 959 | 960 | 961 | 962 | 963 | 964 |
| 965 | 966 | 967 | 968 | 969 | 970 | 971 | 972 | 973 |
| 974 | 975 | 976 | 977 | 978 | 979 | 980 | 981 | 982 |
| 983 | 984 | 985 | 986 | 987 | 988 | 989 | 990 | 991 |
| 992 | 993 | 994 | 995 | 996 | 997 | 998 | 999 | 1000 |

1944

1976

Konrad Zuse
Plankalkül

$$\underline{1} \quad A2 = (A9, A\Delta 1)$$

$$\underline{2} \quad P1 \quad \left| \begin{array}{l} R(V) \Rightarrow R \\ V \quad 0 \quad 0 \\ A \quad \Delta 1 \quad \Delta 1 \end{array} \right.$$

$$\underline{3} \quad V \quad 0 \quad 0$$

$$\underline{4} \quad A \quad \Delta 1 \quad \Delta 1$$

$$\underline{5} \quad \left| \begin{array}{l} \sqrt{|V|} + 5 \times V^3 \Rightarrow R \\ V \quad 0 \quad 0 \quad 0 \\ A \quad \Delta 1 \quad \Delta 1 \quad \Delta 1 \end{array} \right.$$

$$\underline{6} \quad V \quad 0 \quad 0 \quad 0$$

$$\underline{7} \quad A \quad \Delta 1 \quad \Delta 1 \quad \Delta 1$$

$$\underline{8} \quad P2 \quad \left| \begin{array}{l} R(V) \Rightarrow R \\ V \quad 0 \quad 0 \\ A \quad 11 \times \Delta 1 \quad 11 \times 2 \end{array} \right.$$

$$\underline{9} \quad V \quad 0 \quad 0$$

$$\underline{10} \quad A \quad 11 \times \Delta 1 \quad 11 \times 2$$

$$\underline{11} \quad \left[\begin{array}{l} W2(11) \left[\begin{array}{l} R1(V) \Rightarrow Z \\ V \quad 0 \quad 0 \quad 0 \\ K \quad i \\ A \quad \Delta 1 \quad \Delta 1 \end{array} \right. \right.$$

$$\underline{12} \quad V \quad 0 \quad 0 \quad 0$$

$$\underline{13} \quad K \quad i$$

$$\underline{14} \quad A \quad \Delta 1 \quad \Delta 1$$

$$\underline{15} \quad \left[\begin{array}{l} Z > 400 \Rightarrow (i, +\infty) \Rightarrow R \left[\begin{array}{l} (10-i) \\ 0 \end{array} \right. \\ V \quad 0 \end{array} \right.$$

$$\underline{16} \quad V \quad 0$$

$$\underline{17} \quad K$$

$$\underline{18} \quad A \quad \Delta 1 \quad 9 \quad 2 \quad 9$$

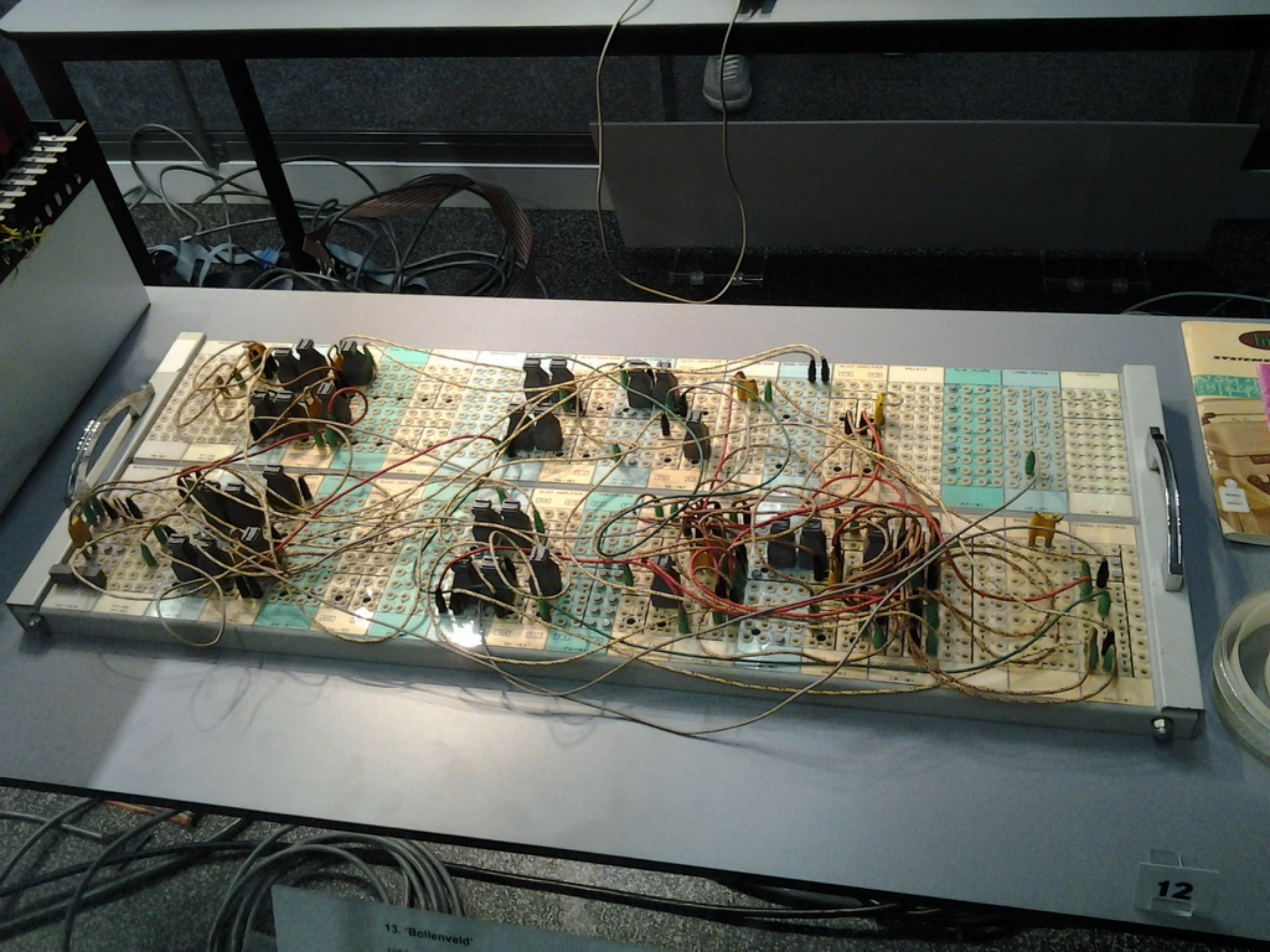
$$\underline{19} \quad \left[\begin{array}{l} Z > 400 \Rightarrow (i, Z) \Rightarrow R \left[\begin{array}{l} (10-i) \\ 0 \end{array} \right. \\ V \quad 0 \end{array} \right.$$

$$\underline{20} \quad V \quad 0$$

$$\underline{21} \quad K$$

$$\underline{22} \quad A \quad \Delta 1 \quad 9 \quad \Delta 1 \quad 2 \quad 9$$



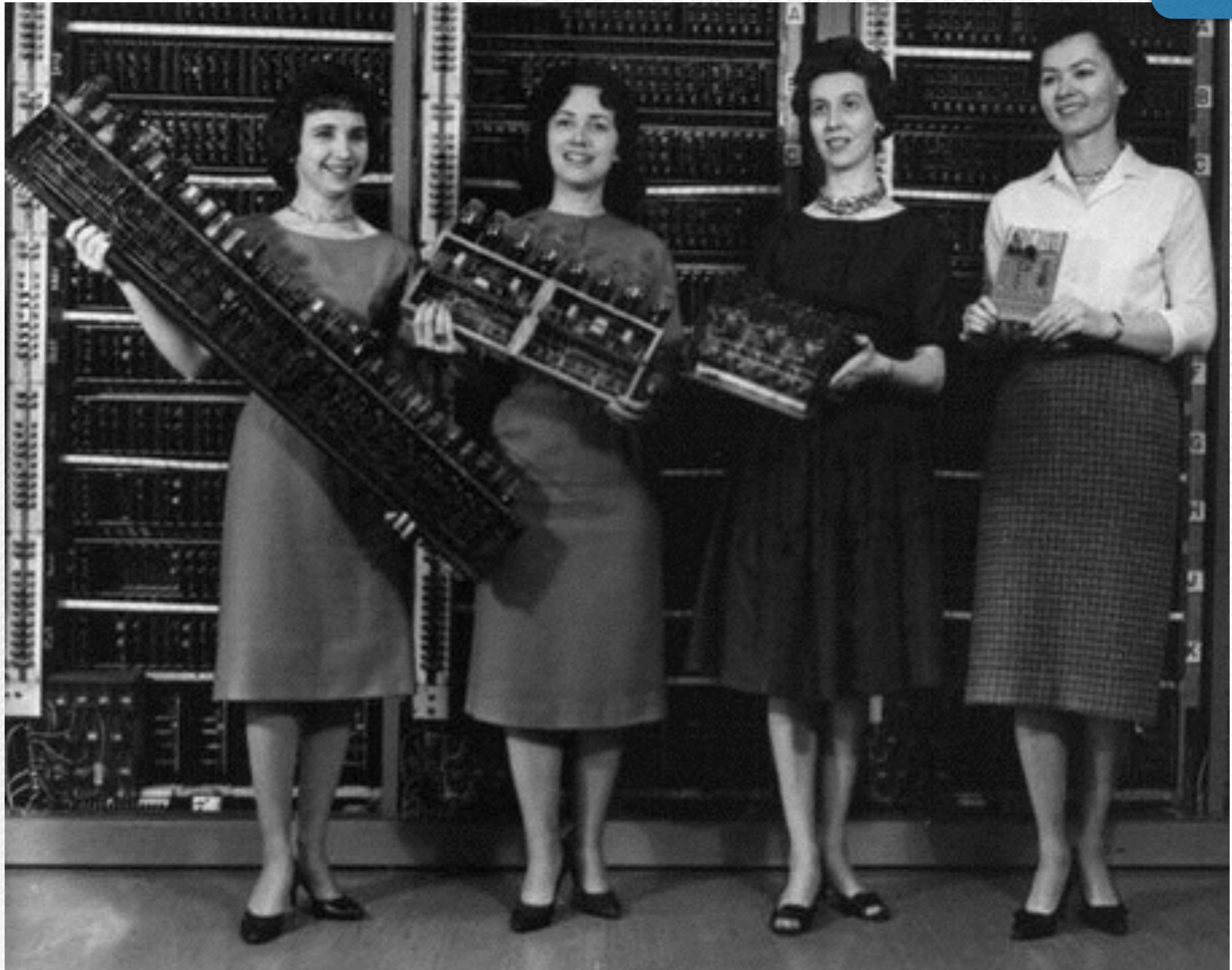


13. 'Bollenveld'

12

Adele Katz Goldstine (1920–1964)

1946



1945

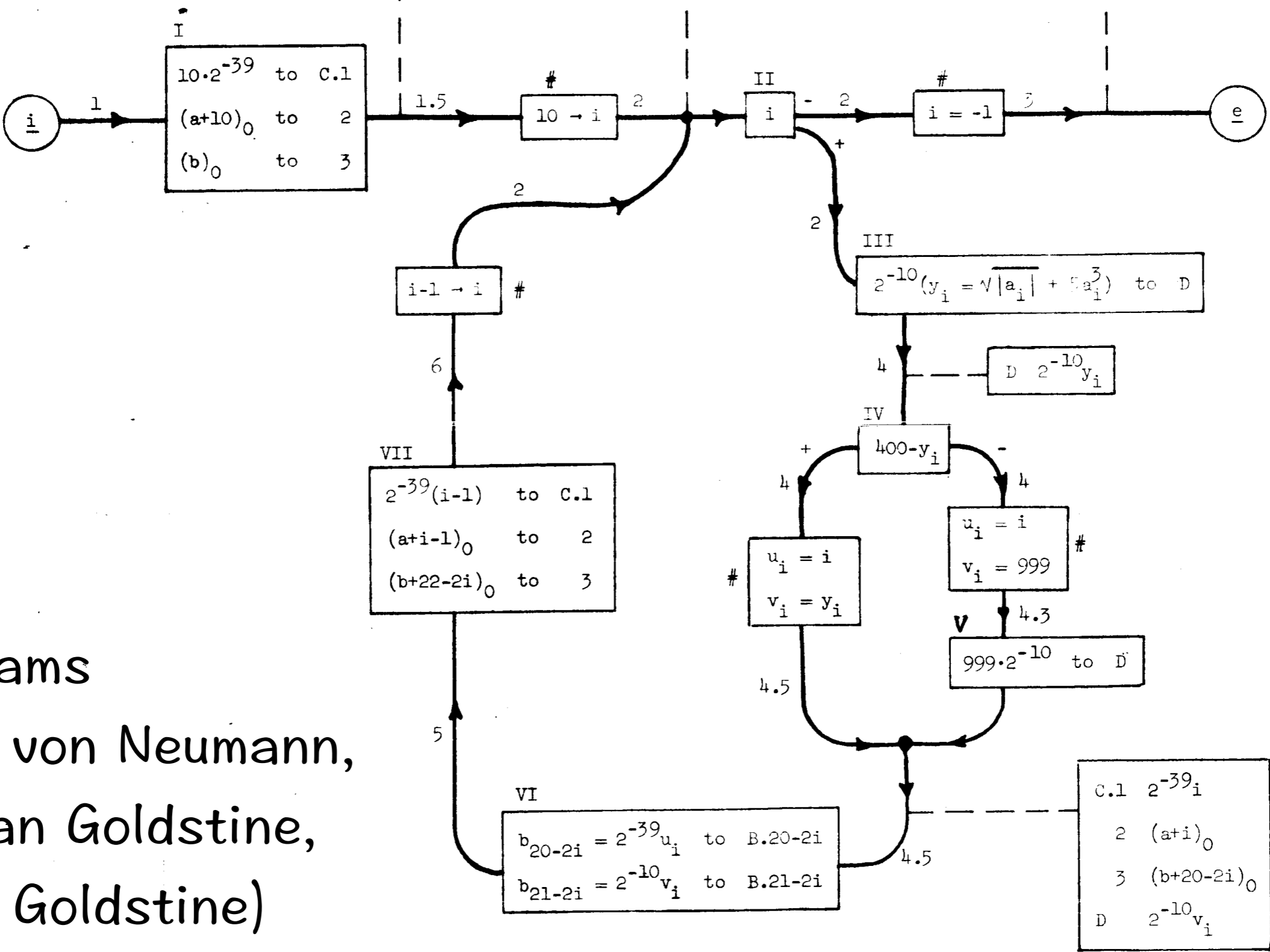


John von Neumann (1903–1957)

A.j $2^{-10}a_j$ ($j = 0, \dots, 10$)
 C.1 $10 \cdot 2^{-39}$
 2 $(a+10)_0$
 3 $(b)_0$

A.j $2^{-10}a_j$ ($j = 0, \dots, 10$)
 B.j b_j ($j = 0, \dots, 19-2i$)
 C.1 $2^{-39}i$
 2 $(a+i)_0$
 3 $(b+20-2i)_0$

B.j b_j ($j = 0, \dots, 21$)



Flow
 Diagrams
 (John von Neumann,
 Herman Goldstine,
 Adele Goldstine)

1950

"the technique of program composition can be mechanised"



Haskell Brooks Curry (1900–1982)

Equations

Coded representation

| | | |
|-----------|---|-------------------|
| <u>00</u> | $i = 10$ | 00 00 00 W0 03 Z2 |
| <u>01</u> | 0: $y = (\sqrt{\text{abs } t}) + 5 \text{ cube } t$ | T0 02 07 Z5 11 T0 |
| <u>02</u> | | 00 Y0 03 09 20 06 |
| <u>03</u> | $y \text{ 400 } \text{ if}_{\leq} \text{to } 1$ | 00 00 00 Y0 Z3 41 |
| <u>04</u> | $i \text{ print, 'TOO LARGE' print-and-return}$ | 00 00 Z4 59 W0 58 |
| <u>05</u> | 0 0 $\text{ if}_{=} \text{to } 2$ | 00 00 00 Z0 Z0 72 |
| <u>06</u> | 1: $i \text{ print, } y \text{ print-and-return}$ | 00 00 Y0 59 W0 58 |
| <u>07</u> | 2: T0 U0 shift | 00 00 00 T0 U0 99 |
| <u>08</u> | $i = i - 1$ | 00 W0 03 W0 01 Z1 |
| <u>09</u> | 0 i $\text{ if}_{\leq} \text{to } 0$ | 00 00 00 Z0 W0 40 |
| <u>10</u> | stop | 00 00 00 00 ZZ 08 |

```

1   Für i = 10(-1)0
2   ai ⇒ t
3   (Sqrt Abs t) + (5 x t x t x t) ⇒ y
4   Max(Sgn(y-400), 0) ⇒ h
5   Z 0i ⇒ b20-2i
6   (h x 999) + ((1-h) x y) ⇒ b21-2i
7   Ende Index i
8   Schluss

```

1954



Alexander Brudno (1918–2009)

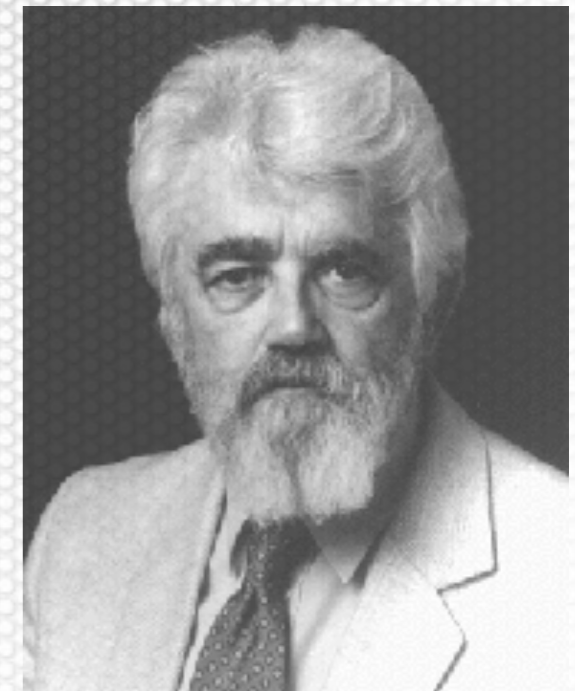
Перед каждой командой в кодированной форме Брудно писал еще ее адрес. Получилось так:

| | | | | A | |
|-------------------|------|-----|------|------|------|
| $0 + 0 = S$ | 1003 | 001 | 0 | 0 | 2570 |
| $1 \rightarrow n$ | 1004 | 056 | 0101 | 2571 | 1006 |
| $n + 1 = n$ | 1005 | 001 | 2571 | 0101 | 2571 |
| $n \times n = R1$ | 1006 | 005 | 2571 | 2571 | 0011 |
| $1 : R1 = R2$ | 1007 | 004 | 0101 | 0011 | 0012 |
| $S + R2 = S$ | 1010 | 001 | 2570 | 0012 | 2570 |
| $n < 10$ | 1011 | 036 | 2571 | 0112 | 1005 |
| Стоп | 1012 | 017 | 0 | 0 | 0 |
| 1 | 0101 | 101 | 4000 | 0 | 0 |
| 10 | 0112 | 104 | 5000 | 0 | 0 |

Теперь, чтобы запомнить, какую ячейку отвели данной букве, и чтобы ячейки не налезали друг на друга, понадобилась шпаргалка (роспись памяти).

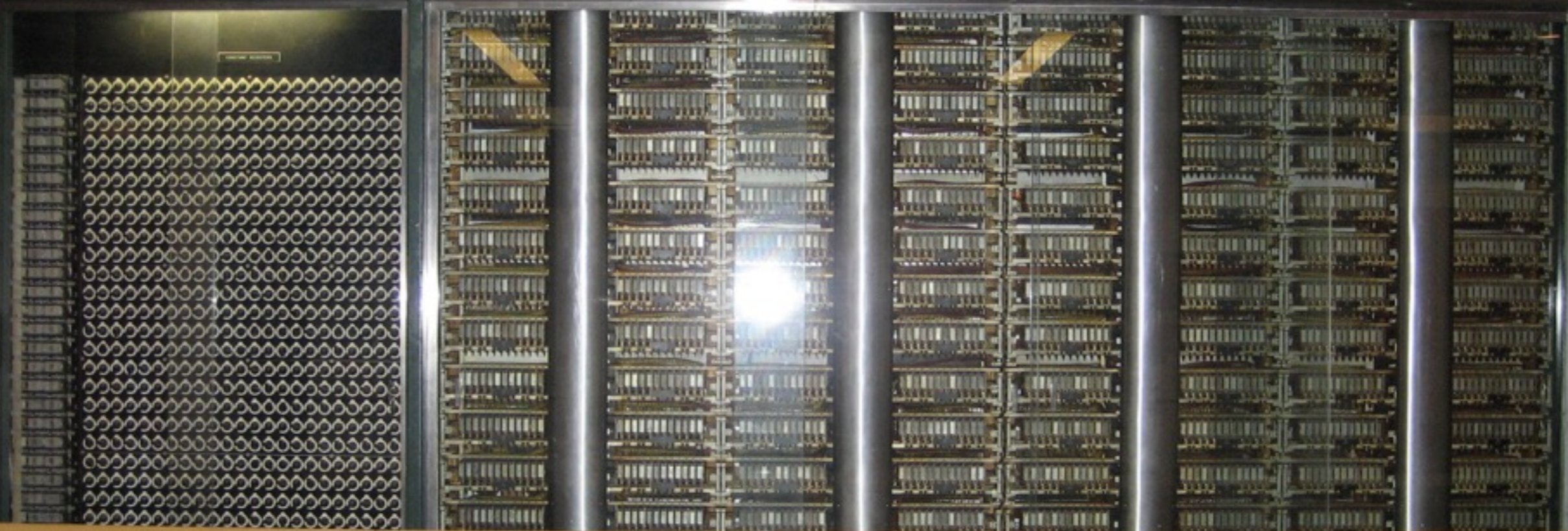
Она выглядела так:

| | | | | |
|---------------|-----------------|----------------|------|------|
| 1260 | 1261 | 1252 | 1263 | 1254 |
| <i>a</i> | <i>b</i> | <i>c</i> | 17,5 | |
| 1320 Δ | 1321 <i>R1</i> | 1322 <i>R2</i> | | |
| 1340 | Блок подготовки | | | |
| 1360 | | | | |
| 1400 | 1401 | 1402 | 1403 | 1404 |
| 1420 | 1421 | 1422 | 1423 | 1424 |



(from A. S. Kronrod, "Discussions about Programming", 1963)

AIKEN - IBM AUTOMATIC SEQUENCE CONTROLLED CALCULATOR



29

photo credit: <http://techie.com/amazing-grace/> / personal.psu.edu



Grace Murray Hopper (1906–1992)

- (1) READ-ITEM A(11) .
- (2) VARY I 10(-1)0 SENTENCE 3 THRU 10 .
- (3) J = I+1 .
- (4) $Y = \text{SQR } |A(J)| + 5 * A(J)^3$.
- (5) IF Y > 400, JUMP TO SENTENCE 8 .
- (6) PRINT-OUT I, Y .
- (7) JUMP TO SENTENCE 10 .
- (8) Z = 999 .
- (9) PRINT-OUT I, Z .
- (10) IGNORE .
- (11) STOP .

MATH-MATIC

- (1) COMPARE PART-NUMBER (A) TO PART-NUMBER (B) ; IF GREATER GO TO OPERATION 13 ; IF EQUAL GO TO OPERATION 4 ; OTHERWISE GO TO OPERATION 2 .
- (2) READ-ITEM B ; IF END OF DATA GO TO OPERATION 10 .

FLOW-MATIC

```
MOVE B"1" TO MY-BOOLEAN. *>Set to True to begin with
MOVE ZEROS TO MY-COUNT.
*> Primer read to see if we have data to read back
INVOKE DATAREADEROBJ "Read" RETURNING MY-BOOLEAN. *> True if data, False if none
PERFORM WITH TEST BEFORE UNTIL MY-BOOLEAN NOT EQUAL B"1"
    INVOKE DATAREADEROBJ "GetString" USING 0 RETURNING FIRSTNAME
    INVOKE DATAREADEROBJ "GetString" USING 1 RETURNING LASTNAME
    INVOKE DATAREADEROBJ "GetString" USING 2 RETURNING HOMEPHONE
    ADD 1 TO MY-COUNT
    DISPLAY "Record: ", MY-COUNT, " NAME: ", FIRSTNAME, " ", LASTNAME, " ", HOMEPHONE
    INVOKE DATAREADEROBJ "Read" RETURNING MY-BOOLEAN *> True if data, False if none
END-PERFORM.
```

COBOL

1996

“...it’s only a matter of time before all the existing COBOL programmers die of old age...”

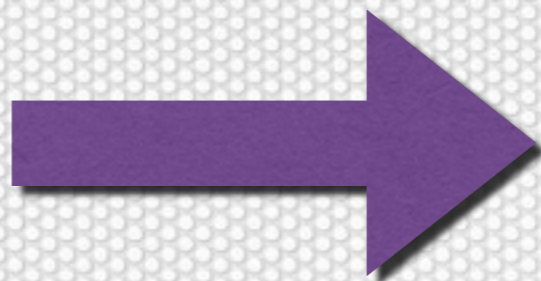


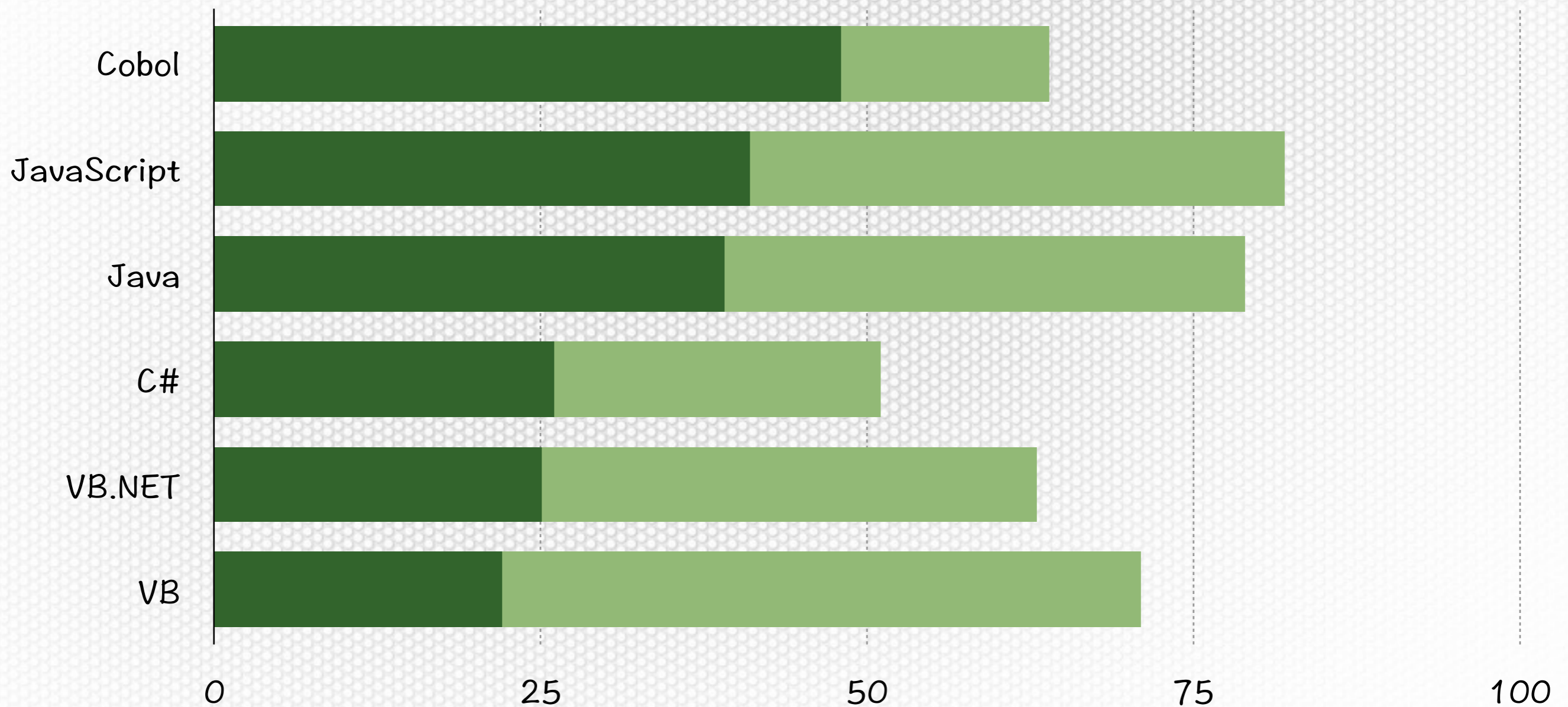
photo credit: Ed Yourdon, CC-BY-SA,

[https://commons.wikimedia.org/wiki/File:Ed_Yourdon_\(1970s\).jpg](https://commons.wikimedia.org/wiki/File:Ed_Yourdon_(1970s).jpg)

[https://commons.wikimedia.org/wiki/File:Yourdon,_Edward_\(2008\).jpg](https://commons.wikimedia.org/wiki/File:Yourdon,_Edward_(2008).jpg)

2012

To what extent do you use these languages?



1957

```
FTN 5.1+642      87/02/02. 10.44.20 PAGE 18
SUBROUTINE RCKELI 74/175 OPT=0,ROUND= A/ S/ M/-D,-DS
DO=-LONG/-OT,ARG=-COMMON/-FIXED,CS= USER
/-FIXED,OB= TB/ SB/ SL/ ER/-ID/ PHD/-ST,-AL,PL=5000
FTN5,I=CHRLGS,L,LO,DB,PN,PW=90,PS=80.
```

```
1  SUBROUTINE RCKELI(N,A,Y,X)
2  PARAMETER (IA=50)
3  REAL A(IA,IA), Y(IA), X(IA)
4  INTEGER N
5  *
6  *-----*
7  *
8  * SUBROUTINE ZUR RUECKWAERTSEELIMINATION, D.H. ZUR LOESUNG VON
9  * R*X=Y
10 *
11 * ERKLAERUNG DER FORMALEN PARAMETER:
12 *
13 * NAME      TYP      DIMENSION  ERKLAERUNG
14 *-----*
15 * N        INTEGER   0          DIMENSION DER MATRIX A (E)
16 * A        REAL      2          DREICKSZERLEGUNG DER MATRIX A
17 *          (UEBERSPEICHERT) (E)
18 * Y        REAL      1          VEKTOR, DER DIE LOESUNG VON
19 *          L*Y=B ENTHAELT (E)
20 * X        REAL      1          VEKTOR ZUR AUFNAHME DER LOESUNG VON
21 *          R*X=Y (A)
22 *
23 *
24 *
25 *
26 * E: EINGABEPARAMETER
27 * A: AUSGABEPARAMETER
28 *
29 *-----*
30 *
31 * KONSTANTEN ('PARAMETER' IN FORTRAN):
32 *
33 * IA: MAXIMALE DIMENSION DER MATRIX A
34 *
35 *-----*
36 * X(N)=Y(N)/A(N,N)
37 * DO 10 K=N-1,1,-1
38 *   S=0.
39 *   DO 20 J=K+1,N
40 *     S=S+A(K,J)*X(J)
41 * 10 X(K)=(Y(K)-S)/A(K,K)
42 * RETURN
43 * END
```

```
--VARIABLE MAP--(LO=A/R)
--NAME--ADDRESS --BLOCK--PROPERTIES--TYPE--SIZE--REFERENCES--
A          2    DUMMY-ARG          REAL      2500      1    3    36
           40    41
J          201B          INTEGER          39/C    40    40
K          176B          INTEGER          37/C    39/C    40
           41    41
           41    41
N          1    DUMMY-ARG          INTEGER          1    4    36
           36    36
           37/C  39/C
S          200B          REAL          38/S    40    40/S
           41
X          4    DUMMY-ARG          REAL        50      1    3    36/S
           40    41/S
Y          3    DUMMY-ARG          REAL        50      1    3    36
           41
```

```
--SYMBOLIC CONSTANTS--(LO=A/R)
--NAME--TYPE-----VALUE--REFERENCES--
IA    INTEGER          50      2/S  3    3    3    3
```

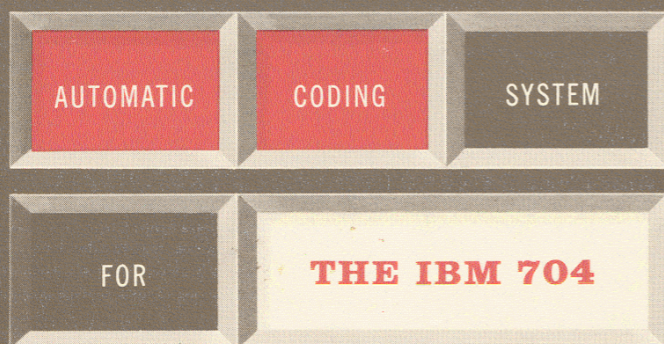


John Backus (1924–2007)

1957

PROGRAMMER'S REFERENCE MANUAL

Fortran



John Backus (1924–2007)


```

proc abs max = ([,] real a, ref real y, ref int i, k) real:
comment The absolute greatest element of the matrix a, of size  $r_a$  by  $2r_a$ 
is transferred to y, and the subscripts of this element to i and k; comment
begin
  real y := 0; i :=  $L_a$ ; k :=  $2L_a$ ;
  for p from  $L_a$  to  $r_a$  do
    for q from  $2L_a$  to  $2r_a$  do
      if abs a[p, q] > y then
        y := abs a[p, q];
        i := p; k := q
      fi
    od
  od;
  y
end # abs max #

```

1958

"a language so far ahead of its time that it was not only an improvement on its predecessors but also on nearly all its successors"

```

proc abs max = ([,]real a, ref real y, ref int i, k)real:
comment The absolute greatest element of the matrix a, of size  $\uparrow a$  by  $2\uparrow a$ 
is transferred to y, and the subscripts of this element to i and k; comment
begin
  real y := 0; i :=  $\uparrow a$ ; k :=  $2\uparrow a$ ;
  for p from  $\uparrow a$  to  $\uparrow a$  do
    for q from  $2\uparrow a$  to  $2\uparrow a$  do
      if abs a[p, q] > y then
        y := abs a[p, q];
        i := p; k := q
      fi
    od
  od;
  y
end # abs max #

```



Tony Hoare (b. 1934)

CC-BY-SA; code credit: https://en.wikipedia.org/wiki/ALGOL#ALGOL_68;

photo credit: Rama, Sir Charles Antony Richard Hoare, https://commons.wikimedia.org/wiki/File:Sir_Tony_Hoare_IMG_5125.jpg

BNF

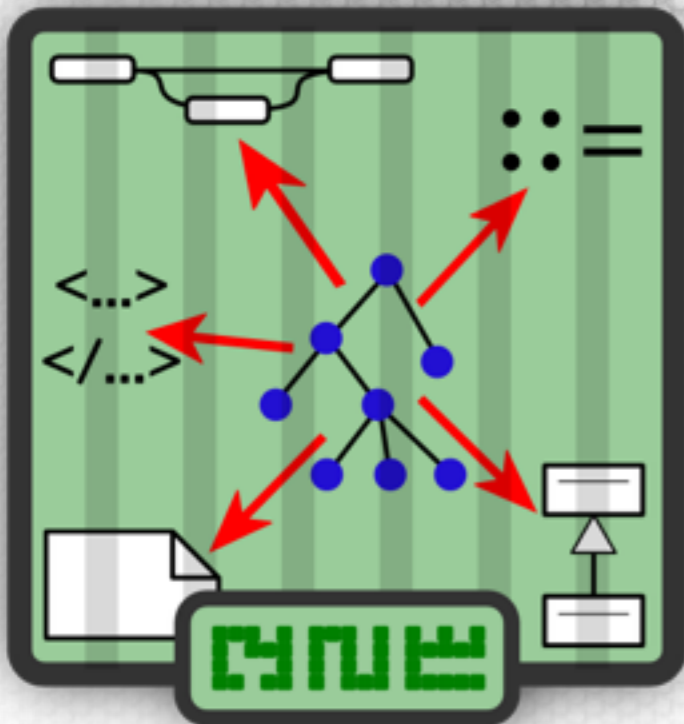
```
compilation ::=  
    compilation_unit*
```

```
compilation_unit ::=  
    visibility_restriction? "separate"? unit_body
```

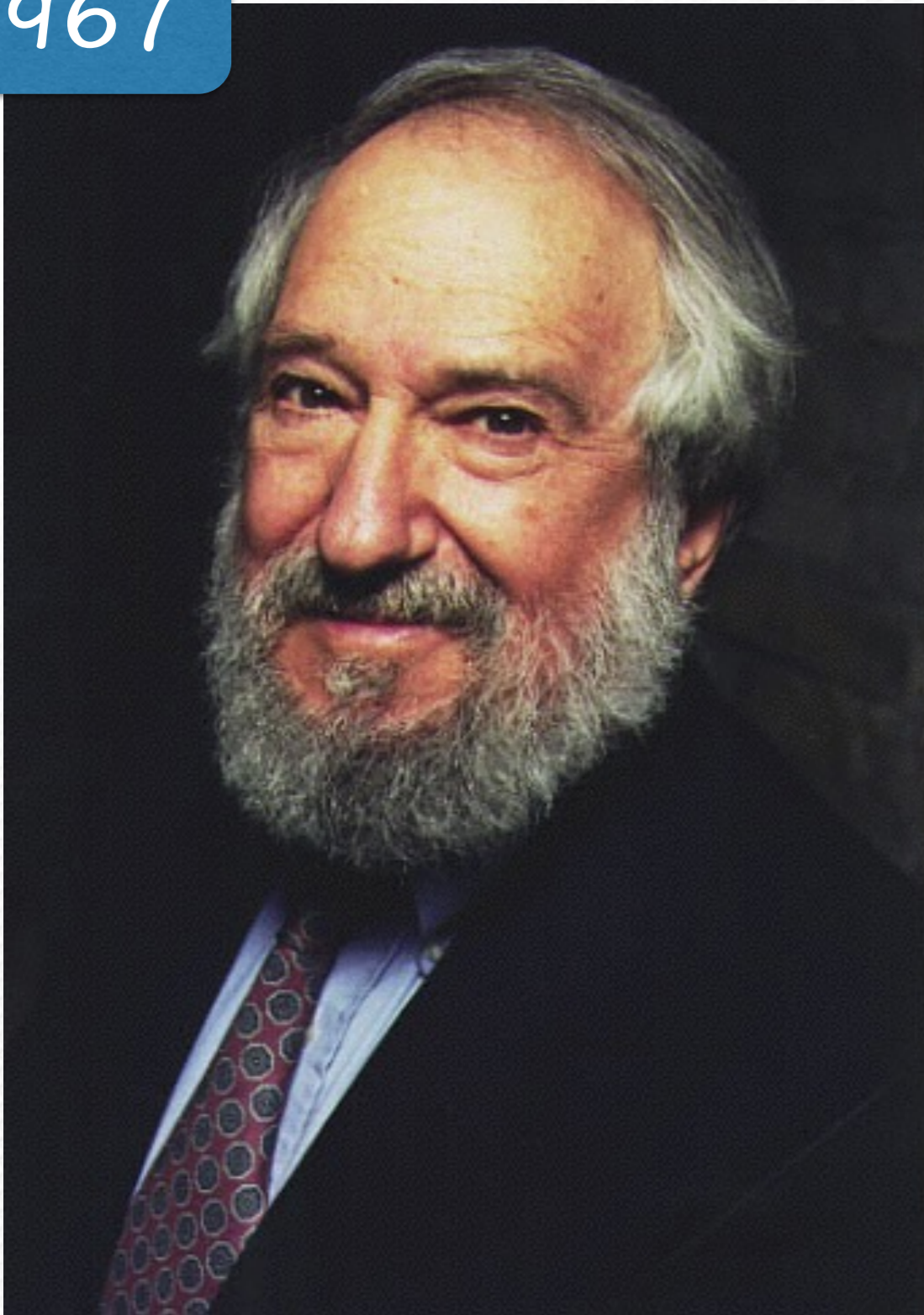
```
visibility_restriction ::=  
    "restricted" visibility_list?
```

```
visibility_list ::=  
    "(" <unit_name>:name ("," <unit_name>:name)* ")"
```

```
unit_body ::=  
    subprogram_body  
    module_specification  
    module_body
```



1967

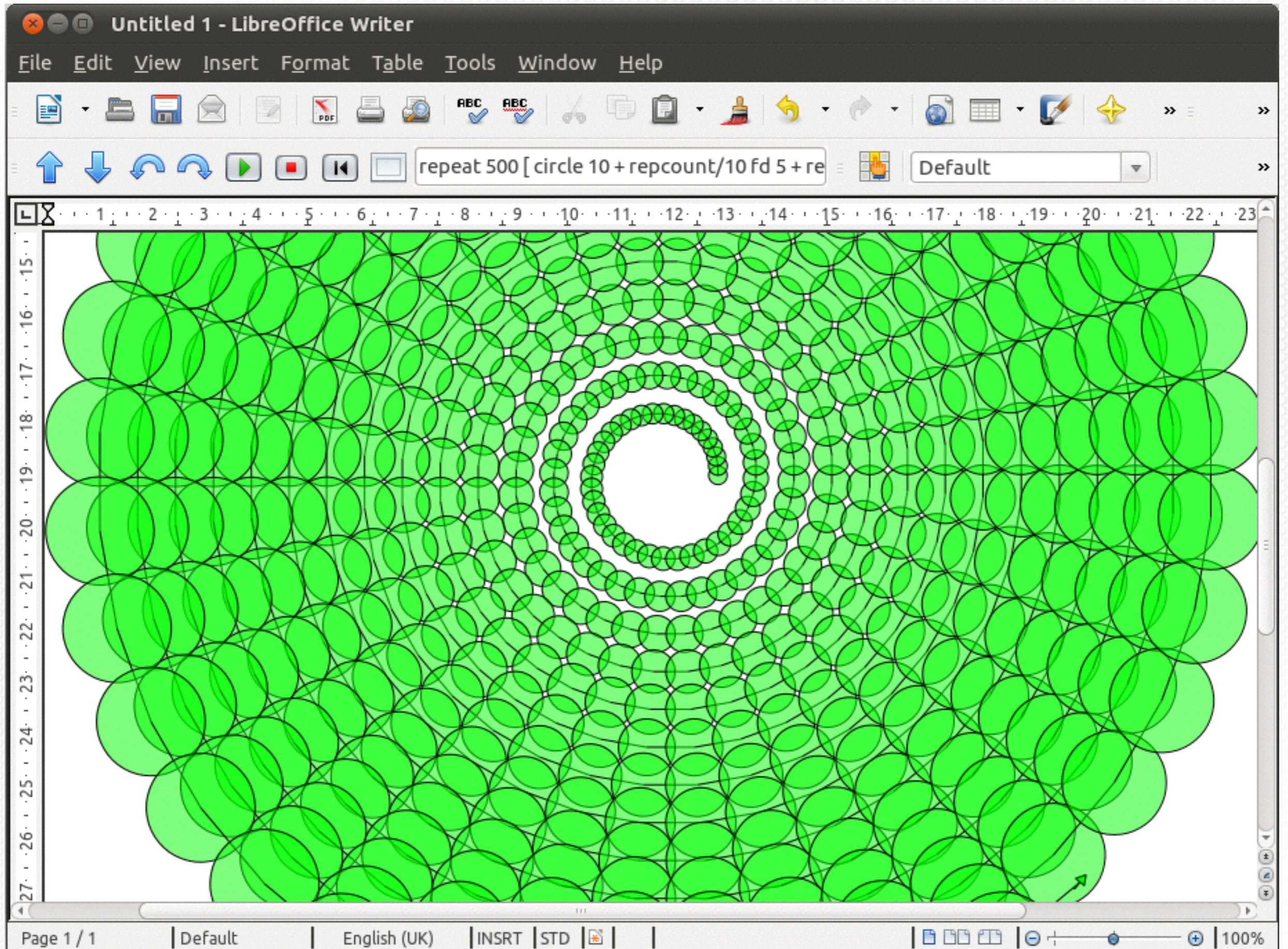


Seymour Papert
(b. 1928)

1974



Radia Perlman
(b. 1951)



2006

The image shows a screenshot of the Scratch web editor interface from 2006. The browser address bar displays `scratch.mit.edu/projects/10015059/#editor`. The page title is "Animate the Crab on Scratch" by Scratchteam. The main stage area shows a red crab sprite on a blue background, with its current position at `x: 138 y: -180`. The left sidebar contains the "Sprites" panel with a "Scratch Cat" sprite and the "Stage" panel with "3 backdrops". The central "Scripts" panel lists various block categories: Motion, Looks, Sound, Pen, Data, Events, Control, Sensing, Operators, and More Blocks. The right panel shows the script editor with three event-driven scripts:

- Script 1:** "when green flag clicked" followed by "go to x: 0 y: 0", "set rotation style don't rotate", and a "forever" loop containing "move pick random 1 to 3 steps", "turn 15 degrees", and "if on edge, bounce". A yellow callout box labeled "Move around randomly" points to the loop.
- Script 2:** "when green flag clicked" followed by "switch costume to starter crab" and a "forever" loop containing "next costume" and "wait 1 secs". A yellow callout box labeled "Switch between costumes" points to the loop.
- Script 3:** "when green flag clicked" followed by a "forever" loop containing "play sound human beatbox1 until done". A yellow callout box labeled "Play music" points to the loop.

The bottom of the script editor shows a search bar and a "x position" button.

Scratch, <http://scratch.mit.edu>

1964



IDEs

- 40% time spent editing code
- 50% efficiency boost with better tools
- We want:
 - editing support, code navigation
 - debugging, profiling
 - build integration, versioning
 - quality assurance, acceptance testing

Everything is a...

- ...machine?
- ...bit string?
- ...word?
- ...object?
- ...model?



Jean Bézivin

System

represented by



Metamodel

conforms to



Model



represented by



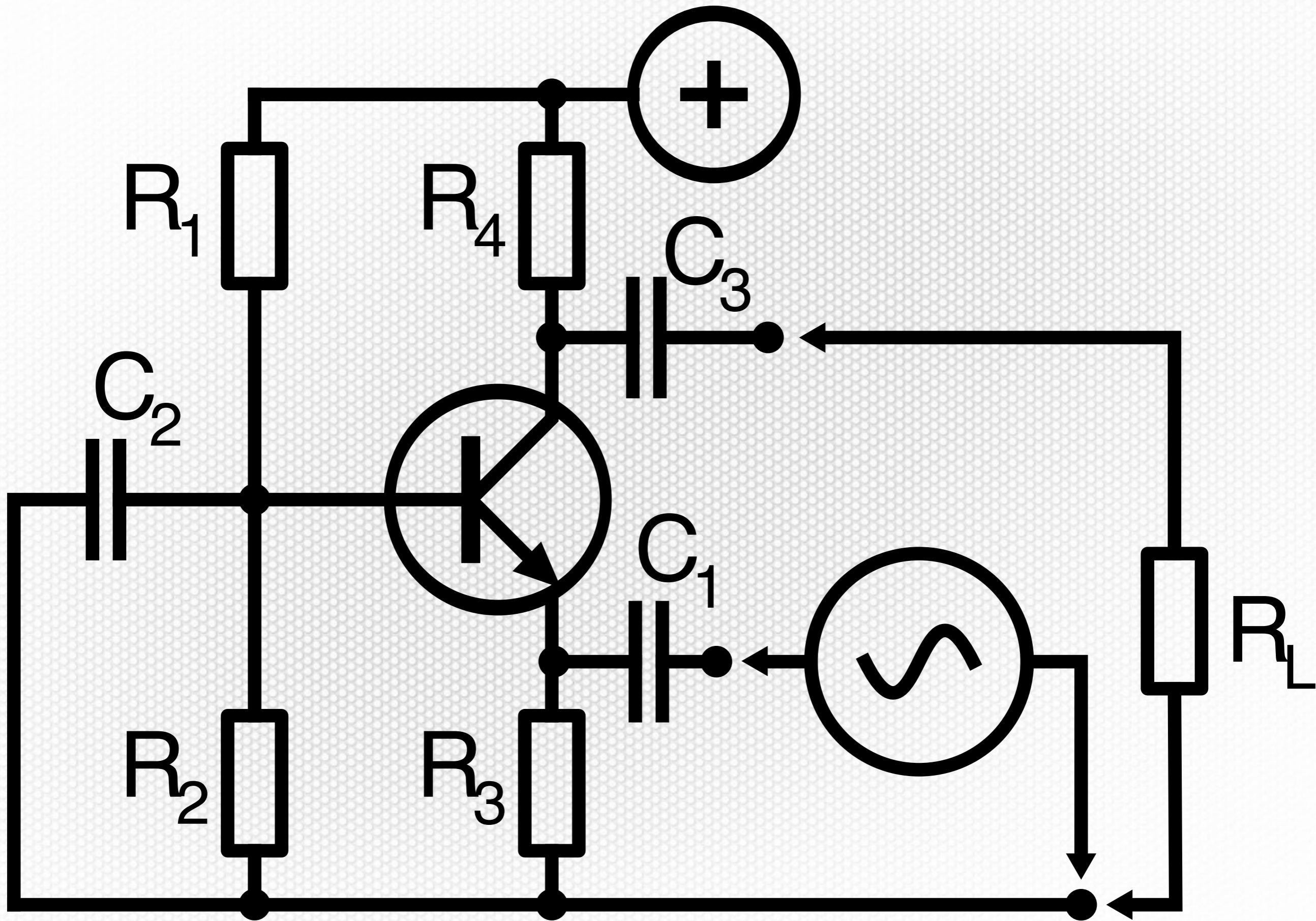
conforms to





VIA 9GAG.COM

Ceci n'est pas un chat



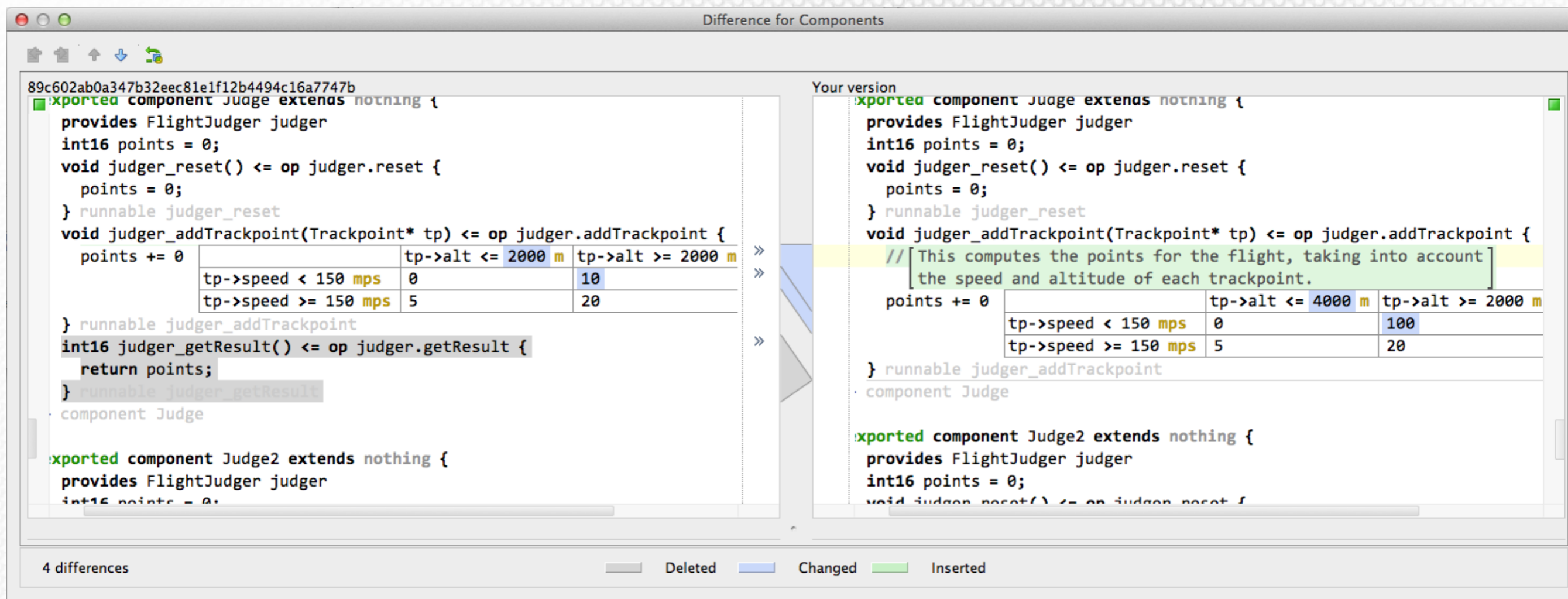
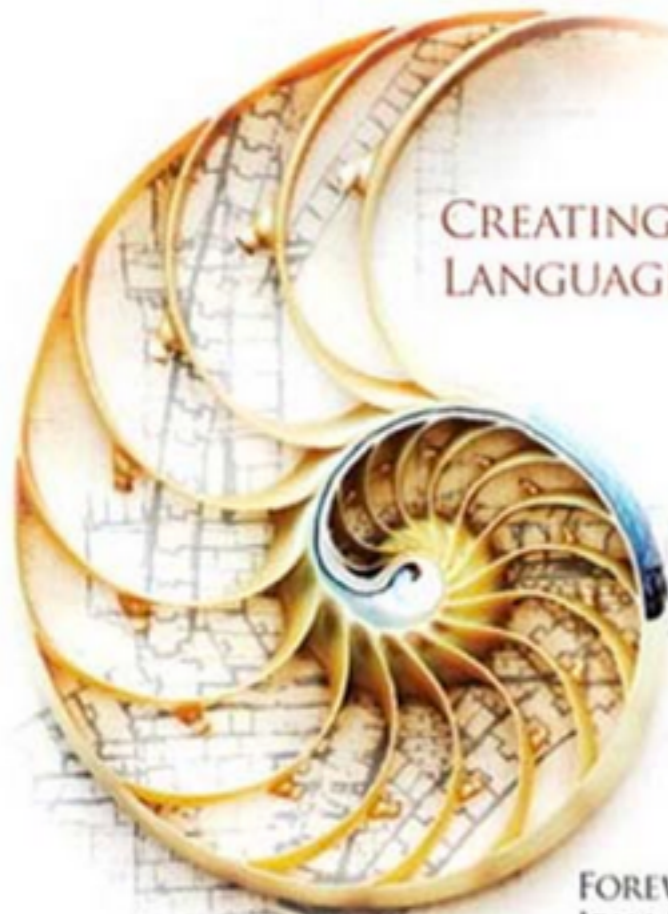


Figure 4.20 This dialog shows the diff between the local version and the latest version from git before a commit. The local version has changed two of the values in the decision table, has added a comment and has removed one runnable.

SOFTWARE LANGUAGE ENGINEERING



CREATING DOMAIN-SPECIFIC
LANGUAGES USING METAMODELS

ANNEKE KLEPPE

FOREWORD BY
JEAN-MARIE FAVRE

SOFTWARE LANGUAGE ARCHAEOLOGIST
AND SOFTWARE ANTHROPOLOGIST, IIG, ACONIT,
UNIVERSITY OF GRENOBLE, FRANCE



2009

2013

The Pragmatic
Programmers

Language Implementation Patterns

Create Your Own Domain-
Specific and General
Programming Languages

Terence Parr



DSL Engineering

*Designing, Implementing and Using
Domain-Specific Languages*

Markus Voelter

with Sebastian Benz, Christian Dietrich, Birgit Engelmann
Mats Helander, Lennart Kats, Eelco Visser, Guido Wachsmuth

dslbook.org

Software languages

- Not (just) textual
- Exist in many formats
- (Now) easy to develop
- Easy to map to one another
- Language-parametric methods

Grammars in a broad sense

- Exist in various forms and formats
 - metamodels, schemata, domain models, class dictionaries
- Grammar recovery, convergence, transformation, ...
- Grammars as contracts/commitments
 - for people and tools

Language workbenches

- Extensive IDE support beyond compilation

- Syntax highlighting & visualisation

- Interactive recommendations

- Beyond textual

- TDD and debugging

```
test bool testCode0() = unambiguous("");
test bool testCode1() = unambiguous("int x");
test bool testCode2() = unambiguous(";");
test bool testCode3() = unambiguous(":=");
test bool testCode3a() = unambiguous(":= ");
test bool testCode3b() = parses(":= ");
test bool testCode4() = unambiguous(": =");
test bool testCode5() = unambiguous("int FACES = 6;");
test bool testCode6() = unambiguous("1");
test bool testCode6a() = unambiguous("1 ");
test bool testCode7() = unambiguous("1 //");
test bool testCode7a() = parses("1 //");
test bool testCode8() = unambiguous("//");
test bool testCode9() = unambiguous("1 // comment");
test bool testCode9a() = parses("1 // comment");
```

Milestone summary

- Universal hardware + programs
- Automated code generation
- Programming with words
- Language documentation
- Domain-specific languages
- Merging/bridging disciplines: MDE, PLT, CC, TT, CT, NLP...

Questions?

