SWE 212 Analysis of Programming Languages Week 1 Summary

Summary of Paper: 'First Draft of a Report on EDVAC' by John von Neumann

Samyak Jhaveri

The paper starts with a briefing on the process a code would follow to be executed on the ideal EDVAC system, which, in a highly abstracted way, feels relevant today as well. Besides the strange terminology of the various components, many of the concepts presented in this paper can be mapped out to modern-day computing concepts. Logical and algebraic operations would be meticulously and exhaustively expressed as something the computer can understand. Assuming the machine is faultless, all its internal workings shall be abstracted from the user once it is fed the input - from the calculations to its memory and storage - only giving out the result of the operations. The paper goes further to mention that the device might have some errors and malfunctions and might even have its own way of recognizing and addressing such malfunctions automatically. This draft contained von Neumann's ideas on how a "stored program" computer should be built for faster operations. He divides it into six major subdivisions: a central arithmetic part (*CA*) for arithmetic operations, a central control part (*CC*) for carrying out the given instructions in the correct sequence in the computer's machinery, memory (*M*), input (*I*), output (*O*), and external memory. Von Neumann compares the working of the EDVAC system with the human brain's neurons such that the numerical or other information can be transferred in and out of the outside recording medium (*R*) such that it enters into (as input) and exits from (as output) the specific parts such as *C* or *M*. A large portion of the paper is dedicated to elaborating into minute details the workings of the the 5 subdivisions mentioned above and how they would function together.

Von Neumann offers many insights into the workings of an ideal computer down to the procedures to be used to compute operations like addition, subtraction, multiplication, division, square root, etc. He proposes many ideas such as table lookups, interpolation, logarithmic and trigonometric functions. Furthermore, the paper elaborates on the treatment of elements as vacuum tubes as they proved to be the fastest approach to computing at the time as compared to relays. Another radical idea, for its time, is the idea of the e-element, which is a hypothetical element of the computer that essentially functions like a vacuum tube with the appropriate RLC-circuit but can be expressed as an individual unit. This e-element, is expressed as the author's analogy to the human neuron. The paper discusses various arithmetic and logical operations using this e-element concept like adder, discriminator, multiplier and even a square-root finder circuit, along with other components that work with the e-element such as a terminal output equipment l_k.

Another interesting series of concepts presented in the paper are the cohesive working of the *CC* and *M* components. The author, for the first time in the paper, hints at the importance of listing a set of code instructions (here called *code words*) for the *CC* to receive, interpret and carry them out, or to properly stimulate those organs that carry them out. From this point in the paper, the connection between *CC* and *M* and the code instructions starts to look increasingly similar to the modern way of coding we use today with our programming languages. Here are the different types of code instructions for and between the *CC* and *M* that von Neumann proposes, along with my interpretation of their modern-day code analogy:

1. Orders for *CC* to instruct *CA* to carry out one of its ten specific operations - Unary, Binary, Ternary operators like +, -, *, /, ?, :, etc.

2. Orders for *CC* to cause the transfer of a standard number from one place to another - Assignment operation of the r-value of a constant number into the l-value of a variable located in the memory
3. Orders for *CC* to transfer is own connection with *M* to a different point in *M* with the purpose of getting its next order from there - Loosely similar to the concept of control flow that dictates the flow of execution of the text lines present in memory as per the rules and syntax of the programming language under consideration
4. Orders controlling the operation of the input and the output of the device - Simple I/O operations like taking input from the user, or printing data onto the terminal, or reading and writing data to and from a given file.

Moreover, the paper states the disjunctions and definitions. It suggests that the memory units in *M*, each one characterized by binary digits *1* or *0* are grouped together to form 32-unit *minor cycles*. These *minor cycles* have a direct significance in the code and fall into two categories: *Standard Numbers* and *Orders*. They are distinguished from each other by the value of their respective first digits i_0. These two *minor cycles* seem to be analogous to *constants* and *operators* respectively prevalent in modern-day programming languages. The remaining 31 bits of a *Standard Number* must express the number itself in reverse binary notation (i.e. with the lowest positional value first). On the other hand, the remaining 31 bits of an *Order* express the nature of the order(s). These orders could be either (α) orders for *CC* to instruct *CA* to carry out one or more of its ten operations, or (β) orders for *CC* to transfer a *Standard Number* from a definite *minor cycle* in *M* to *CA*, or even (δ) order for *CC* to transfer a *Standard Number* from *CA* to definite *minor cycle* in *M*.

A total eight (8) orders are established in the paper, and neither of them are very efficient in using the 31 available digits. Consequently, the idea of pooling 2 or more orders in one *minor cycle* is discussed which is justified by the objective to simplify the overall logical structure of the code. However, it is not very practical to pool several orders since it may not be logically possible and since it is also good practice to spare some space in the logical part of the *minor cycle* for changeability. Additionally, even though it may be logically possible to pool another order with the (α) order, only (β) and (γ) can be pooled with it. With many other considerations of the practicality of possible order pools, the paper devises a table enumerating standard order pools that specifies four (4) things for each possible minor cycle:

I. The *type* i.e. its relationship to the classification (α) - (η) and to the relevant pooling procedures
II. The *meaning*
III. The *short symbol* to be used in verbal or written discussion of the code, and also when setting up the device
IV. The *code symbol* i.e. the 32 binary digits which correspond to the 32 units of the minor cycle in questions.

What I find the most fascinating about this paper is how the concepts mentioned and foreseen by von Neumann are still relevant today, albeit in different forms or as different concepts ranging from the hardware design of a computer to the way instructions are compiled, interpreted and executed from a relatively higher level to the binary operation level. It shows how far the industry and research has come from this initial seed of ideas about the ideal electronic computer and is also a testament to power and longevity of simple ideas as they are recycled over generations.