Austin Herring

CS T680 – Evolution of Computing

Dr. Brian Stuart

7 March 2017

The Earliest Electronic Realizations of Turing Machines

Starting in the 1930s and continuing into the 1940s, an explosion of results in mathematics and engineering interacted and combined, eventually morphing into the field that is today recognized as computer science. One of the first major theoretical works in the field, helping form the foundation of the "theory of computation" and what can and cannot be done by a "computer," was done by Alan Turing, in his 1936 paper "On Computable Numbers." Very closely following this, two of the first actual physical devices that might reasonably be argued as electronic "computers" in the modern sense are the Colossus, developed by British codebreakers, largely Tommy Flowers, and the ENIAC, developed by John Mauchly and J. Presper Eckert. From there, many of the lessons learned from the ENIAC were then used to develop the first "stored-program computers" such as the Manchester SSEM, the EDSAC, and more, which truly began to lay the foundation for what are today referred to as computers. While all of these breakthroughs into digital, electronic computing share characteristics, with each of the machines displaying at least one aspect, and most of them many, of the theory presented in Turing's paper, the truest realization of the grandest possibilities laid out by Turing did not occur until an electronic, digital design of a stored program computer was implemented.

While the main point of Turing's paper is work towards determining which numbers are "computable" and, with that, an application to the *Entscheidungsproblem,* to achieve these results, he introduces a class of idealized machines, ultimately dubbed "Turing machines," that he describes as closely simulating the actions of a human computer. The machine has a finite number of "states of mind," referred to as "*m*-configurations"; an infinitely long "tape" made up of individual "symbols" from which to read information and on which to mark both scratch work and final results; and a "tape head" that scans exactly one symbol on the tape at a time. Depending on the machine's current *m*-configuration and scanned symbol, it can perform a "transition," consisting of writing a symbol, moving left or right, and going to another *m*-configuration. By supplying rules for each combination of scanned symbol and *m*-configuration, grouped together into a "transition relation," a specific computation can be carried out by the Turing machine, using arbitrary data on the tape as "input" and producing "output" on the final tape (Turing). Turing argues, convincingly, that these machines completely capture everything that an actual human computer can do in following an algorithm, taking input and converting it to output, thus offering one definition of a computer, in a machine sense.

However, in his paper, Turing also describes in detail the idea of a "universal Turing machine." The machine takes as input a description of another Turing machine and the input for that machine, and it uses the description of that machine to itself *simulate* that machine on the part of the tape given for its input. In the paper, Turing is able to provide a full set of *m*-configurations and transition rules, as described above, for such a machine (Turing). Such a machine is a "universal Turing machine" because it can be used to "run" any other Turing machine. This then means that the universal Turing machine, given the right description of another machine, can be used to compute anything that is computable by a Turing machine, which, as Turing argued, is anything that is algorithmically computable in general.

These two conceptions of machines give two different but related definitions of a "computer." Machines that do not take advantage of the aspects that make a Turing machine universal are something like “special-purpose” computers; they expect input in a particular format, and they will perform exactly one task to provide a particular output. An example of such a machine might be one that determines whether the length of a sequence of characters is a power of two. A physical machine automatically implementing this would, in a sense, be a “computer,” because it is automatically computing something, emulating a Turing machine in doing so, but it provides only one function. A more general sense of a computer, however, is one that, parameterized correctly, can compute *anything* that is computable, a “general-purpose” computer. This corresponds exactly to a universal Turing machine, with the parameterization corresponding to the description of another Turing machine on the input tape. Using a modern comparison, one might consider the difference between these two to be the difference between a program itself and an interpreter for a programming language; the program performs computations and is thus “computing” as a special-purpose “computer,” but the interpreter, given the right input, can compute anything computable in its language. Thus, in his paper, Turing presented the foundational conceptions of what can be considered a computer, in both a special-purpose and general-purpose sense.

Though there had previously been mechanical and electromechanical devices that could be said to have done so, one of the first digital, fully-electronic devices that can be said to realize a physical manifestation of a Turing machine is the Colossus, thus putting it somewhere early in the direct lineage to modern digital, electronic computers. The Colossus was developed by British codebreakers during World War II to break the encryption of what were dubbed “Tunny messages,” which were deduced to have been coming from “Tunny machines” operating a cryptographic Vernam cipher using rotating wheels to pseudo-randomly encrypt messages and remove linguistic properties from the ciphertext. After extensive analysis, it was determined that applying particular Boolean evaluations to the ciphertext could reveal statistical properties about the keys used to encrypt the plaintext, which could then be further analyzed to ultimately decrypt the messages. The Colossus is designed to quickly and automatically perform these extensive sets of evaluations on ciphertexts to reveal these statistical significances. Its main functionality is to emulate the turning wheels of the Tunny machines, run the desired Boolean operations on input, automatically count the instances that certain things occur, and, if a certain count threshold, called the "set total," is reached, print results, which, now knowing their likely statistical significance due to the count, can be manually analyzed (Flowers). The machine achieves these operations through the use of vacuum tubes to implement counters, Boolean operations, shift registers, and more, and the through the use of plugs and switches to "program" the machine and determine the Boolean operations to be applied and the wheel settings to be used (Flowers).

Clearly, the Colossus is, at the very least, a special-purpose digital computer and one that is, at that, configurable and programmable by its plugs and switches. After having been configured, given input ciphertexts, it automatically computes and prints the values that meet the requisite set totals. Though perhaps difficult and time-consuming, it would be possible to describe a formal theoretical Turing machine with *m*-configurations and transition rules that simulates the Colossus. However, the Colossus does not quite reach the level of a "general-purpose" computer, i.e., a *universal* Turing machine. Note that a centrally important aspect of the description of a Turing machine is the "decisions" made by its transition relation, i.e., "If in *m*-configuration *X* and reading symbol *a*, move direction *D*, write symbol *s*, and go to *m*-configuration *Y*; otherwise …." Thus, a universal Turing machine must have some way of simulating decisions about which *action* to take next, but the Colossus has no mechanism for doing this. Though decisions can be made at “run-time” about which *data* to send to where in the machine, decisions about which operations themselves should be performed cannot be determined from the internal state of the machine (Wells). The description that Howard Campaigne, a self-described cryptanalyst-programmer for the Colossus, gives for working with the machine is that he would “[tell] the machine to make certain calculations and counts, and after studying the results, [tell] it to do another job” (Flowers). If the Colossus were a fully universal machine, it would be able to perform the analysis and decide on the next job itself, but, in reality, as Campaigne describes, this would instead require reprogramming the switches and plugs to do so. Therefore, there is no way for the Colossus to implement any other Turing machine, restricting it to a “special-purpose” computer. Despite the high programmability and configurability, it is still more akin to a program and a “special-purpose” Turing machine, though perhaps one with many “command line options,” than an interpreter or a universal Turing machine.

The Electronic Numerical Integrator and Computer, or ENIAC, is also a digital, electronic computer developed during World War II, in the United States at the University of Pennsylvania, again making it one of the first physical realizations of a Turing machine electronically, placing it in the direct lineage of electronic and digital modern computers. The ENIAC’s main use is, as its name implies, for calculating the answers to arbitrary numerical problems, such as solving differential equations numerically, in contrast to the stricter use of the Colossus for a single cryptographic analysis. The machine is divided into arithmetic components, including accumulators, which do addition and subtraction, and combined divider/square rooters; memory components, including those same accumulators, which retain the values added to them, and function tables, which can be used to lookup constant values; the control components, namely, what is called the “master programmer”; and input and output devices, namely, IBM punch cards (Brainerd). These are all implemented, just as in the Colossus, through the use of vacuum tubes to act as discrete, digital signals.

The higher-level operation of the ENIAC occurs by sending “pulses” out on various “program trunks” to tell operations to start or to indicate operations have finished in such a way that operations are “chained” to produce the final result; in the words of Arthur Burks, “to set up a problem on the ENIAC, one establishes chains or sequences of local program controls by interconnecting the inputs and outputs …” of the different units. For example, an “initiating pulse” might be sent that tells the punch card reader to read data into an accumulator; when the punch card reader is done, it sends an output pulse to another accumulator, which reads its input pulse to learn it must add its value to the first accumulator; and when the addition is done, this sends another output pulse back to the input pulse of the card reader to “loop” the sequence. The machine is programmed to solve a particular numerical task by deciding which outputs of which units are to go to which inputs of which other units. In the machine, one final important source of pulses is the master programmer, which can conditionally produce output on its trunks. For example, if a particular counter in it has reached the value zero, the master programmer may send out a pulse on one line, and if it is non-zero, it will send out a pulse on a different. Again, in the words of Arthur Burks:

> [The master programmer] contains ten six-stage counters, each of which routes incoming program pulses. The positions of these counters may be controlled either by the number of pulses which have been supplied to the various output channels or by pulses received …. In this way, the schedule of sequences may be fixed in advance, or it may be made contingent on the results of the computation. (Burks)

What Burks is describing here is conditional choosing based on the internal state of the machine; the counters can be set based on input pulses to the master programmer, and the counters will cause incoming pulses to be routed to different outputs. This is at least one of the features, and the exact one described above, that prevent the Colossus from being a true universal Turing machine and, therefore, a true "general-purpose" computer.

Momentarily disregarding the conditional choosing, at the very least, the ENIAC, provided some wiring configuration, is clearly a physical manifestation of some Turing machine. Given that Turing argues that any Turing machine can be made to implement any algorithm, one could be made to implement any addition, subtraction, division, or square rooting algorithm, and each of these machines could be logically composed to simulate the sequence of stops followed by the ENIAC, though, again, it might require an extensive amount of *m*-configurations, transition relations, and work to do so. More interestingly, the ENIAC, to a reasonable degree, is a universal Turing machine. Informally, this is due to the choosing abilities of the master programmer. Treating the internal counters of the master programmer as part of the "tape," when it examines this part of the tape, the unit can choose what the next unit to activate is, putting it in a new "*m*-configuration" and looking at a new part of the "tape," and the results of the new unit can, in turn, trigger rewriting any other of the accumulators as part of the "tape" which can in turn retrigger the master programmer again which can decide the next "transition" to apply. Though informal, this description shows how, in general, the ENIAC can almost fully simulate, to within a factor of limited memory in the physical world, as opposed to the theoretical machine's infinite tape, a true, universal Turing machine, making it the first digital, electronic computer to do so, a true "general-purpose" machine in a direct precedent to what is today considered a computer.

However, there remains one notable difference between the ENIAC and a universal Turing machine. Namely, in a universal Turing machine, the description of the machine is provided in the same manner as the input for the machine, i.e., both are provided on the tape. In the ENIAC, however, while data is retrieved from punch cards for long-term storage and placed in the accumulators for active storage, the description of its Turing machine is provided by wiring, completely separated from the data input. Though this says nothing about the generality of the ENIAC, it does say something its particular method of modeling Turing machines, and actual, physical machines that even more closely resembled Turing machines would not arrive until the conception of “stored-program computers.”

Stored program computers essentially sprung out of, or are at least strongly characterized by, John von Neumann’s “First Draft of a Report on the EDVAC.” In the paper, von Neumann essentially lays out every required aspect to create a true, modern computer. In doing so, he specifies five main subdivisions of the system: one for “central arithmetic” to carry out such operations, one for “central control” of what the machine is doing, one for “memory” to store the intermediate results of operations, and one each for “input” and “output” to interact with the external world. He describes a variety of ways that, at the time, the central control component could be directed to itself tell the system what to do, such as through the use of punch cards, teletype tape, steep tape or wire, or, like with the ENIAC, a wiring into exchangeable plugboards. The concept of a “stored program” is one which the instructions affecting the central control component are stored into the memory component, just as the data itself is. This brings any machine fitting this description, and also able to simulate one, even closer to a “true” realization of a universal Turing machine; the specification of the Turing machine to be run is itself a data element in the memory, just as the Turing machine to be run is an input on the tape, meaning that such a stored program computer can modify its own behavior by modifying its own program, just as a true universal Turing machine would be able to do. Though the ENIAC could be considered a universal Turing machine, it did not have this particular property, and electronic, digital machines of this nature would not arise until after the ENIAC, with the first described being the EDVAC, the first operational being the Manchester Small-Scale Experimental Machine, and many others quickly following. Each of these machines is a full-fledged stored program computer, completely implementing all of the features outlined by von Neumann in his report on the EDVAC and, therefore, completely implementing universal Turing machines to within the realm of physical limits imposed by bounded memory.

With the implementation of these stored program computers in the late 1940s, a true realization had already been made, in fast, accurate, digital, electronic form, of the most powerful version of something, the universal Turing machine, which had only been a theoretical model in service of then-greater goals to Alan Turing barely ten years prior. Along the way, however, other machines had begun to hint at electronic versions of Turing's theoretical model. Colossus, as created by British codebreakers, is indeed a machine that automatically computes and can be configured to compute different operations, but these configurations are only parameterizations of the same cryptographic tasks, not capturing the full universality of Turing's theory. ENIAC, as created by John Mauchly, J. Presper Ecker, and others at the University of Pennsylvania, however, does capture the universality, within physical limits, in fulfillment of goals for solving numerical problems, with the ability to rewire the machine providing the capability to simulate any computation a Turing machine could make. On the other hand, that the ENIAC requires full rewiring to emulate different Turing machines means that it lacks the original spirit of Turing’s universal machine, would uniformly take in both its specification of the machine and that machine’s input on the same tape of data, and this is only finally realized with the implementation of stored program computers, which store instructions for the computer to run in the same way that the temporary values are stored, in memory, emulating the sharing of the tape in the model. Though there had been several false, near-false, or “true-but-with-caveats” starts along the way, with the implementation of the first generation of stored program computers, Alan Turing’s theoretical model of a machine that could compute anything that is computable, a “universal Turing machine,” had actually been physically realized with digital electronics, and, to this day, they act as the basis for what can be defined as a general-purpose computer.

Works Cited

Brainerd, John G., and T. Kite Sharpless. "The ENIAC." *Electrical Engineering* 67.2 (1948): 163-172.

Burks, Arthur Walter. "Electronic Computing Circuits of the ENIAC." *Proceedings of the IRE* 35.8 (1947): 756-767.

Flowers, Thomas H. "The Design of Colossus (foreword by Howard Campaigne)." *Annals of the History of Computing* 5.3 (1983): 239-252.

Turing, Alan Mathison. "On Computable Numbers, with an Application to the *Entscheidungsproblem*." *Proceedings of the London Mathematical Society* 2.1 (1937): 230-265.

Wells, B. "The PC-User's Guide to Colossus." *Colossus: The Secrets of Bletchley Park's Codebreaking Computers. Oxford University Press, Oxford* (2006): 116-140.