Computation Series : John Balwit

An algorithm is a finite answer to an infinite number of questions.
Stephan Kleene

Learning Targets
Students completing this series will have a broad familiarity with the historical context and the key figures in the the development of modern computation. They will understand Turing's use of the Turing machine as an illustration of an algorithm. They will understand the Universal Turing machine--both as a template for the modern "stored program machine" and as a necessary step in Turing's diagonal proof. They will also be able to implement simple "programs" on the programmable Turing machine provided in this series. Students will also learn about the context and questions that surrounded the development of the computer. The concepts of Decidability and the classification of problems generally will be explored.

Essential Concepts
Systems described as the interaction of states.
Algorithms: The manipulation of states
Turing machine as example.
Turing machine as general purpose machine
Turing machine as tool
Turing machine as prop in proof
Halting Problem

Much of the complexity that we observe in Nature can be understood as the interaction of simple elements following a small set of rules. Rabbits have babies, nibble grass and run from foxes. Grass grows and gets nibbled by rabbits. Foxes reproduce, chase rabbits and evade hunters etc. Of course, in reality things are more complex but these KIND of interactions can and do give rise to the ever-shifting, organic dynamics that are typical of complex systems.

The rules that organisms follow are typically related to the situation they are in-- their state.

Our best bet at understanding real-world complexity lies in identifying the rules that individual organisms follow (the transition rules) and the various circumstances that they may find themselves in (their states).

The science of computation provides insight into this perspective on complex systems. It provides us with a language to talk about elements of a system, their relationships with other elements, and the limits of what is possible within such systems. This series of models introduces students to terms, tools, and ideas that are foundational to a computational approach to complexity.

Finite state machines

A fundamental motivation for science-- from antiquity to modern times--is to understand how the world works. That is, science seeks to understand how objects change and how they behave when they interact with other objects. The question is what causes what? If we understand these causes it is hoped that we will be able to predict and control future change. This is at the core of the scientific project and it has been remarkably successful for the past 400 years.

Since earliest times there have been those that maintain that every event that we witness or experience is caused by or related to other events that preceded it. It is obvious that *some* events are caused. Sometimes the causes of events are more obscure but revealed through investigation. The belief that everything we experience can ultimately be reduced to interactions between simple elements (atoms, quarks or energy) is called reductionism.

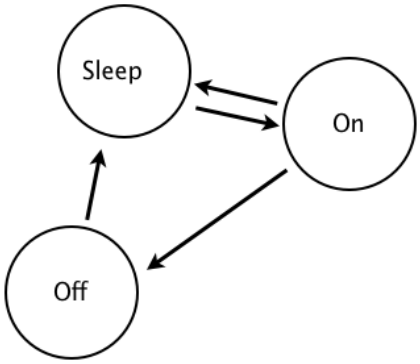
In 1800 Simon Pierre Laplace famously espoused the notion that all phenomena are ultimately to be explained in terms of the interactions of fundamental particles. He said that a sufficiently intelligent being, if given the positions and velocities of all the particles in the universe, could compute the universe's entire future and past. This notion is still at the core of normal science. Nobel laureate physicist Stephen Weinberg declares, "All the explanatory arrows point downward, from societies to people, to organs, to cells, to biochemistry, to chemistry, and ultimately to physics."

As we shall see in various exercises in the Complexity series, there is reason to question whether this reductionist position is appropriate as a general framework. For now, however, we will put these doubts aside and explore what we can learn by about the world using reductionist assumptions.

Reductionism is places great emphasis on understanding the current state of affairs since it is understanding of the current state that will allow us to predict future states. Consequently, the ability to simplify and model the world as a collection of relevant states is key to most sciences. In this series we introduce users to a formal ways to capture this notion of state and the regular change that happens as states interact.

Finite State Machines

A finite state machine (FSM) is perhaps the simplest model of states and the dynamics of change. FSMs can be represented graphically as nodes labeling states and lines showing transitions between states.



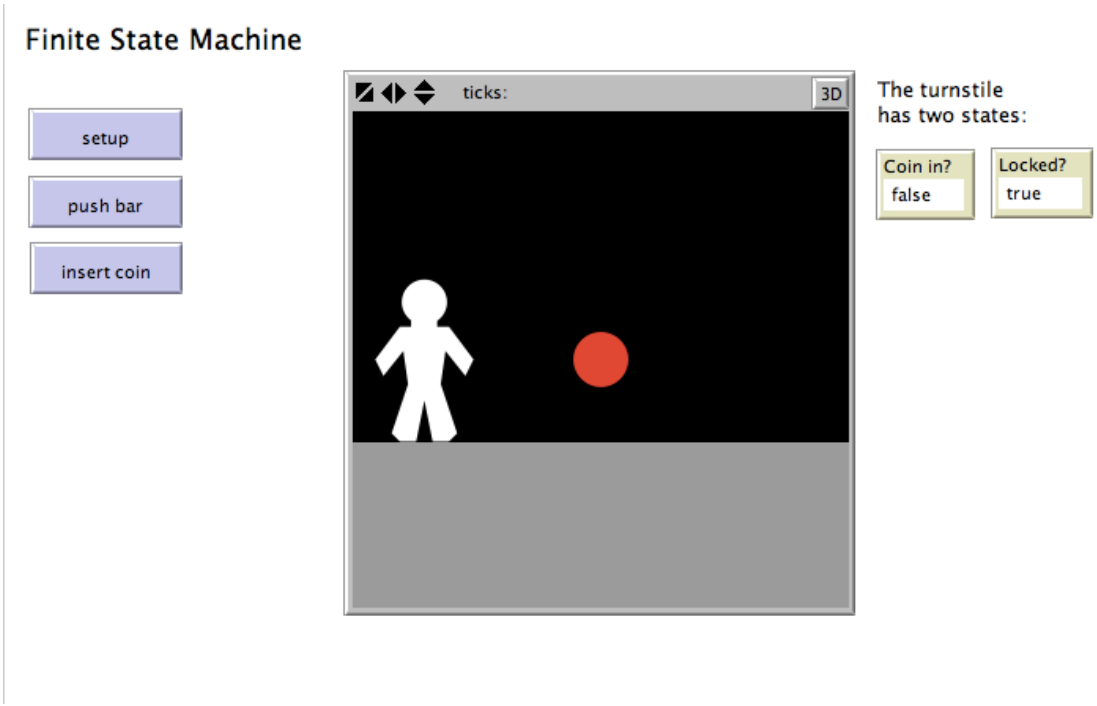
The beauty of FSMs is that they make explicit the assumption that states of a system are unambiguously found to be in a determined state and that the transition between states is also well understood and explicitly defined, These models are useful to engineers as they design real world mechanisms. They are also useful to us as an introduction to systems with well defined behavior.

The FSM NetLogo model in this series offers an example of the familiar turnstile seen at subway stations. The model turnstile has two states, locked and unlocked. Inserting a coin sets the state to unlocked and moving the turnstile bar sets the state to locked.

This is a rudimentary model. It is anticipated that students will elaborate upon the model as they add states and explore more realistic assumptions. As it turns out, even relatively simple mechanisms often have a variety of states and transitions.

An interesting observation made by Marvin Minsky is that FSM may have very complex behavior but as a consequence of their finite natures they will ultimately return to the initial, starting condition. Philosophically, this fact has been used by philosophers (Nietzsche) and even the cosmologist, Brian Greene, to argue that the entire history of the universe will eventually repeat (and repeat an infinite number of times.). The time scales for these repetitions are truly beyond comprehension. Minsky noted that even systems with a few hundred elements will require time spans the exceed the current age of the universe to explore all combinatory diversity of possible states and transitions.

The finite state machine introduced us to rule-bound systems of states. In the next section we will see these ideas applied to states and transitions that create that most exacting of all disciplines, mathematics.



Computation

Alan Turing is recognized as one of the architects of the modern computer. His contributions include the precise and formal description of the step-wise nature of computation itself: the algorithm.. By clarifying exactly what computing involves, he open the path for the construction of actual computers. He also explored the theoretical limits of computation.

In this section we are introduced to the problem that inspired Turing's investigation. Next, we look at the model of computation that he constructed to deal with the above problem -- an invention that is a direct ancestor of modern computers. Finally we turn to the proof that his invention made possible.

The Mathematics of Optimism

From the earliest examples of deductive math with Euclid through the enlightenment and up until the first couple decades of the 20th century, mathematics has moved in a sweep upward arc of optimism and increasing generality. Over the past 150 years mathematicians have attempted to find the essence of mathematics, that is, the foundation of simple rules from which everything else flows. By moving in the direction of increasing generality, they have tried to reach a very abstract and formal set of concepts that would allow them to construct mathematical knowledge which, because it is not bound to one interpretation, can support interpretations in multiple contexts. Alessandro Padoa, Mario Pieri, and Giuseppe Peano were pioneers in this movement.

Some mathematics, namely structuralist math (field theory, group theory,topology, vector spaces) goes even further and develops theories and axioms without reference to any application at all. The optimistic hope of setting all of mathematics on a sure foundation was implicit in German mathematician, David Hilbert's famous turn-of-the-century address to the International Congress of Mathematicians in Paris in 1900. In the talk he offered 23 unsolved problems --loose ends, really--that he anticipated would be cleared up in the decades to come. His influential address became a manifesto for what has come to be known as the Formalist school. According to this school the activity of mathematicians is simply the manipulation of symbols according to agreed upon formal rules.

David Hilbert (1912)

In 1920 Hilbert proposed explicitly a research project that we call today the Hilbert's program. Hilbert wanted mathematics to be formulated on a solid and

complete logical foundation. He believed that in principle this could be done, by showing that:

1. all of mathematics follows from a correctly chosen finite system of axioms; and
2. that some such axiom system is provably consistent through some effective method (definite procedural steps).

The Hilbert program represents a peak of optimism for mathematics that somewhat parallels the enthusiasm of Laplace with respect to the physical universe. Laplace felt that humankind was on the verge absolute, accurate prediction of the future through careful observation and comprehension of underlying laws and principles. Likewise, Hilbert was fervent in the belief the "We Must Know and We Will Know" -- a slogan that Hilbert invoked against his more skeptical colleagues[1],

The Formalist program enjoyed several early victories. Hilbert himself formalized Euclidean geometry and demonstrated the internal consistency of its axioms. Bertrand Russell and Alfred Whitehead published a huge three volume, that set out to place all of mathematics on a logical foundation--grounded in set theory, specifically Cantor's set theory.

The program began to falter after the 1931 publication of Austrian Kurt Gödel's incompleteness theorem. In a nutshell the theorem states that for any self-consistent recursive axiomatic system powerful enough to describe the arithmetic of the natural numbers, there are true propositions about the naturals that cannot be proved from the axioms. In order to create his proof, Gödel developed a technique which codes formal expressions as natural numbers (Gödel numbering).

Kurt Gödel

Godel's work raised serious doubts about the Formalist school's project. A few years later the star of our story, another mathematician, Alan Turing drove the final stake into the Formalist project with an idea that has had consequences far outside of the arena of mathematical debate.

Godel' proved that mathematics (or any formal system with a minimum of complexity) could not be both complete and consistent. It remained to be proved that all specific statements about numbers could be established to be true by some precise "definite method". The idea of a "definite method" is stepwise process that most of us associate with solving a math problem or developing a proof. This is also called and algorithm. It was to this problem that a young English mathematician devoted his attention in 1936.

In the terms that German-speaking Hilbert had presented the problem, it was known as the *Ensheidungsproblem* -- the decision problem. To answer the question of whether it could be proven that all problems where Decidable--that is, discoverable by an algorithm, Turing needed to first formalize the notion of what an algorithm was. He did this with a hypotheticalmachine that we call today, the Turing machine. Even though the machine was hypothetical, paper and pencil machine, it was articulated in concrete terms that made it relatively straightforward to implement physically with electrical switches and wiring.

Alan Turing

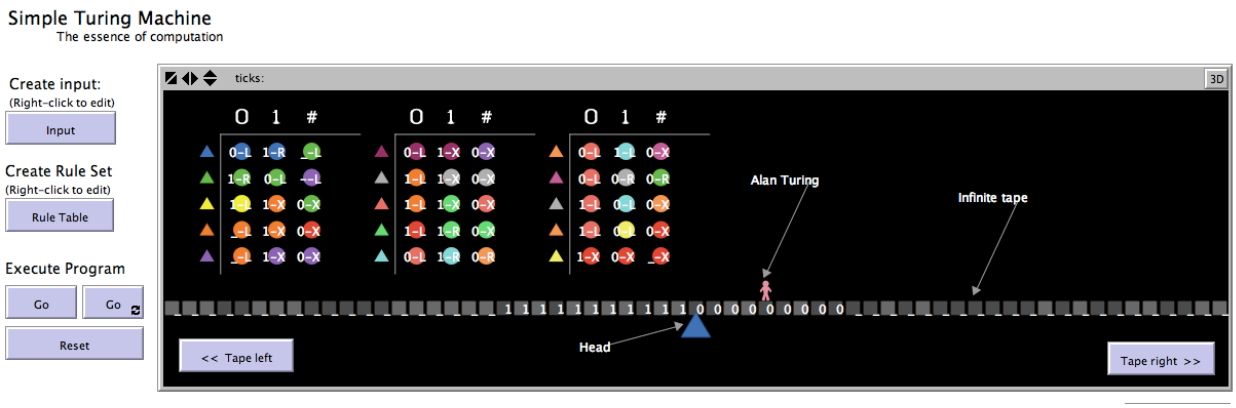
What is an algorithm?

In order to make the case that some assertions stand forever beyond the reach of mathematics (and other formal systems), Turing needed to carefully define the processes that are used to extend mathematics. He needed to define what he called a “effective method” of moving from the beginning (premises) of a problem to its conclusion. The simple addition of a column of numbers could be considered as an example of moving from a starting point with the individual numbers to and end point with the sum. The construction of a proof is a similar process. In each case, we focus on a subset of the initial terms, follow a set of rules, perhaps jotting down notes to keep our place, and finally arrive at our answer. This recipe, or algorithm, can be used in different context and with different numbers.

Each operation, addition, multiplication, division, has it own set of rules. Turing's first contribution was his ability to grasp the general shape of all of various operations and show that they could all be "mechanized" by a machine that he invented. "Mechanized" is in quotes because the machine was not a physical machine--it was an idealized thought experiment--but nevertheless, is evident that approximations to the ideal machine would certain be effective at running different algorithms. The machine that Turing described has come to be called the Turing machine and we will spend some time exploring what these simple machines can accomplish.

The Simple Turing Machine model illustrates a Turing machine and allows users to observe custom programs to solve simple computational tasks. The model offers a taxonomy of the parts of the system as well as an introduction to the conventions of the formalism used to describe the transitions.

Simple Turing Machine Exercises include creating or modifying programs (the rule table) to create Turing machines specific to particular tasks.



The Turing machine can express any ANY algorithm with the following parts and actions:

- an arbitrarily long paper tape divided into squares
- a read/write head that is in one of a finite number of states. The head can examine the tape directly over it.
- symbols on the tape drawn from a finite alphabet.
- an action table that that connects the any particular head state and read-symbol combination with a (possibly) new head state, new symbol, and a movement of the tape. (right or left).

The Turing machine draws its inspiration from the actions of an actual human solving a complicated math problem. The person (head) looks at a part of the problem and

consults the table to see what to do in this situation (state). The instructions provided by the table result in new information under the head--either a new number written or a new state of head.

Turing reasoned the a real human involved in a computation may, at any point, be interrupted and need to make notes of "what they were doing" so that they would be able to pick up where they left off. This possibility suggested to Turing that all "states" have a representation that could be captured in some kind of notation. Turing's notation of this fact involves five pieces of information. Each instruction is a quintuple of {the current state, the current symbol, the new state, new symbol, and tape action}. This can be stated compactly like this: {sym, state --> sym', state' dir }

This is not a complete atomization algorithmic thought (as Turing recognized) but things are broken down to a set of quite primitive operations.

A significant aspect of the Turing machine is the observation that this simple mechanism, given an appropriate action table, is equal to any task that can be described discrete steps. Even more surprising however, is the observation that certain action tables could create a machine with the ability to mimic the behavior of any dedicated Turing machine.

The **Universal Turing Machine** is an extension of the Simple Turing Machine model. It demonstrates Turing's fundamental theoretical insight that lead to the modern "stored program computer." The recognition that such a machine could exist was also an important step in developing the proof concerning the limits of mathematics.

As Turing wrote in his 1936 paper on On Computable Numbers, p. 128
"It is possible to invent a single machine which can be used to compute any computable sequence. If this machine U is supplied with the tape on the beginning of which is written the string of quintuples separated by semicolons of some computing machine M, then U will compute the same sequence as M."

Essentially Turing's idea was to include the rules that operate on data as a special kind of data that is feed into to machine. The rules for M can be encoded as data that is feed into this special Universal Turing machine along with the data for machine M. Machine U (the universal machine) uses M's rules to manipulate M's data and ultimately arrive at the same conclusion that M would have arrived at on its own.

This was a revolutionary idea at the time and represents one of the two pillars in computer science (the other is the idea of an algorithm which we have discussed above) Developing universal machines or interpreters as they are called today, opened the way for the general purpose computers that have become commonplace in modern life.

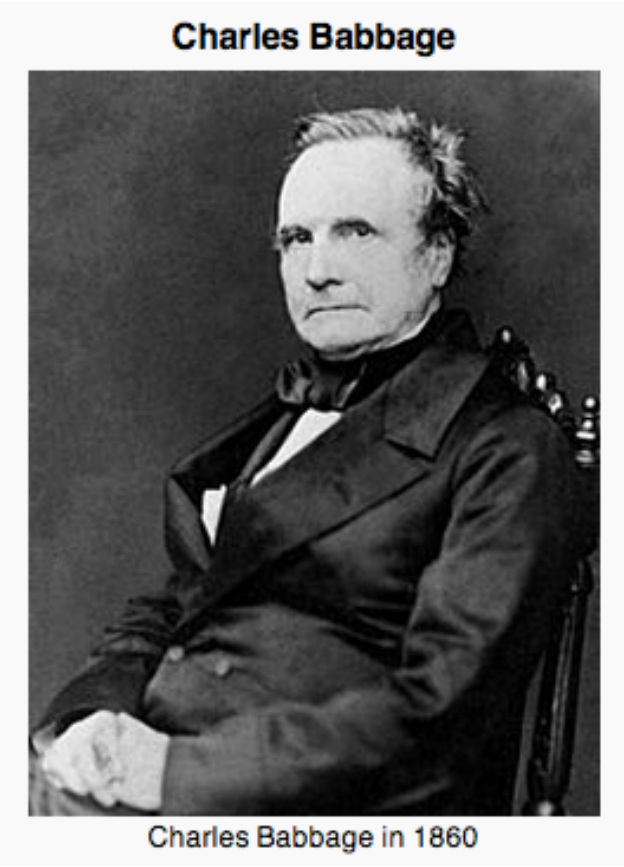
The notion that there could be a single machine (or, equivalently, a single process) that could handle any computation was a dream that reached back to the early 1800's. Charles Babbage was an English mathematician, philosopher, inventor and mechanical engineer who became obsessed with the thought that a single machine could calculate logarithms. He called this machine the Difference Engine. Although the machine was never completed during his lifetime due to funding problems, it was completed according to his design in 1991 and successfully performed calculations to 31 digits of precision. When progress on the construction of the Difference Engine began run into difficulties, Babbage began planning a truly universal machine, the Analytical Engine, which was intended to be programmed by punch cards. The machine as it was designed would have be capable of sequential control, branching and looping--the key features required for a machine to simulate the output of any arbitrary dedicated single-purpose machine.

[sidebar]
Could it be that the computer age was delayed by more than 100 years because of organ grinders?

Babbage never completed his work on the Analytic Engine in the 1800's. He felt that this was due in part to interference with his work from street musicians and, in particular, organ grinders (which he hated):

"It is difficult to estimate the misery inflicted upon thousands of persons, and the absolute pecuniary penalty imposed upon multitudes of intellectual workers by the loss of their time, destroyed by organ-grinders and other similar nuisances."

Babbage also disliked the then popular game of chasing a rolling hoop through the streets.

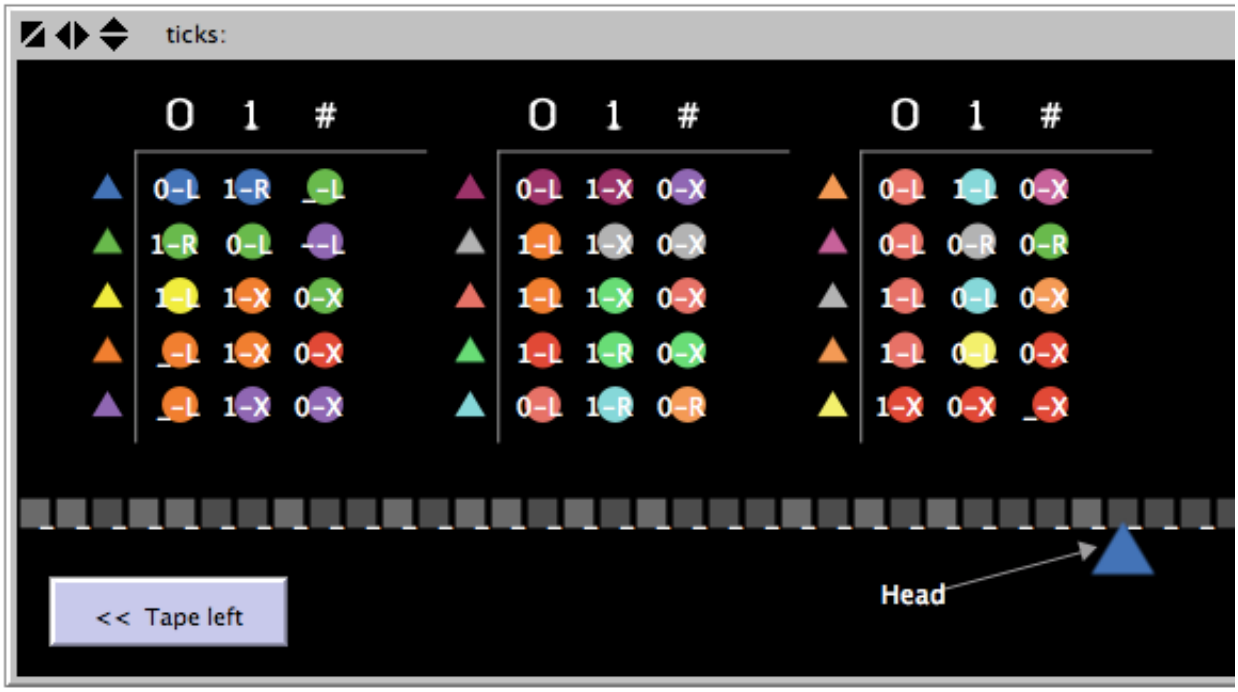


Charles Babbage in 1860

The Analytical Engine, too, was a hundred years ahead of its time. The Analytical Engine was never built and it is not clear that the intellectual legacy of Babbage ever contributed directly to the design of the universal computer that finally was design by Alan Turing in 1936. It is quite clear, however, that there were deep parallels in the underlying principles of Babbage's Analytical Engine and Turing's design. Babbage's Engine, had it been built, would have been capable of any computation that Turing's machine (or any modern day computer) can perform.

Exercises:
The NetLogo model reads in a string of 5-tuples and data from a csv file or from a hand-coded list . Users can create programs and input and "run" these on the U-machine. Users can also explore and optimize the built-in bootstrapping mechanisms.

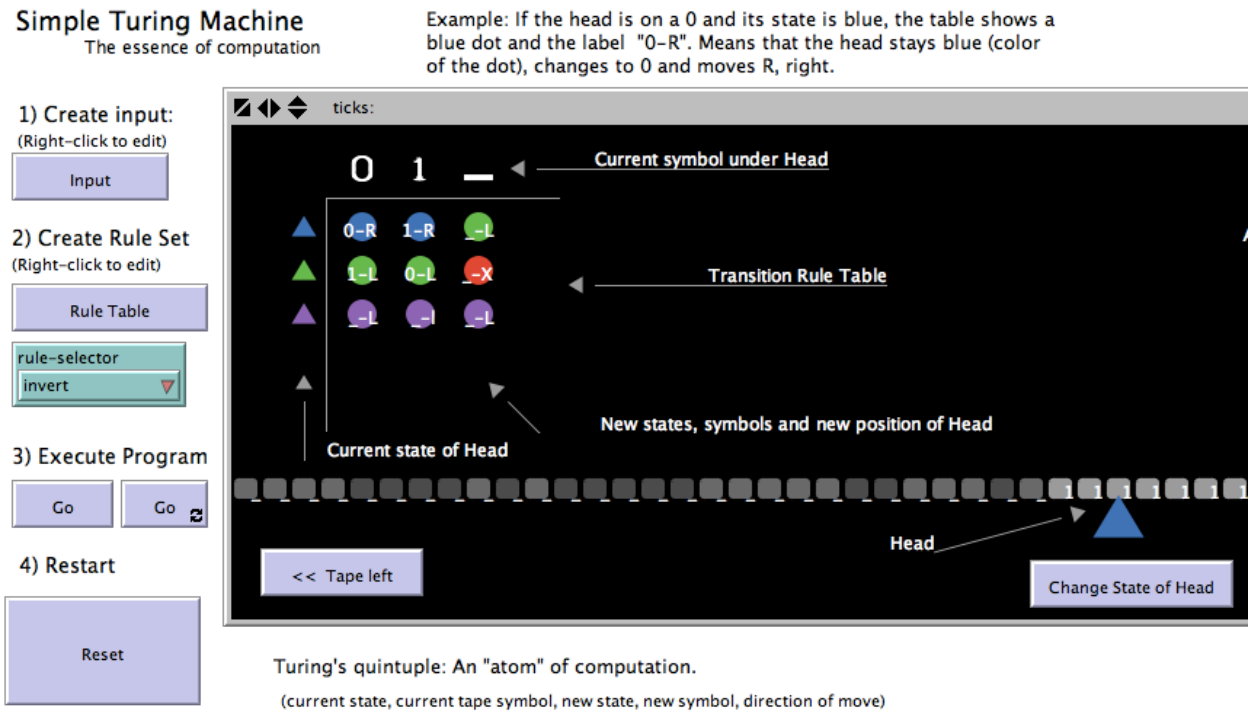
These model also offers users and opportunity to consider that each machine, along with its current state of progress through its input represents a unique number. This idea plays an important role in mathematical nature of computation.



The Programmable Turing Machine

This model offers users the ability to create commented rule tables in a external spreadsheet. The model has been optimized to make it as easy as possible to program so that users can focus on the logic and state transitions of the rule sets that they create. Feedback on which state is currently executing is provided by color coding and numbering states.

Programable Turing



A sample csv rule set is provided. The sample rule set adds two 3 digit binary numbers.

Current State	Current symbol	New State	New symbol	Dir	Comments
75	0	75	_	dir	Start state--READ
75	1	43	_	X	Start state--READ
75	_	15	_	X	Start state--READ
43	0	3	0	L	Move R series with 1
43	1	3	1	L	Move R series with 1
43	_	3	_	L	Move R series with 1
3	0	5	0	L	Move R series with 1
3	1	5	1	L	Move R series with 1
3	_	5	_	L	Move R series with 1
5	0	7	0	L	Move R series with 1
5	1	7	1	L	Move R series with 1
5	_	7	_	L	Move R series with 1
7	0	55	0	L	Move R series with 1
7	1	55	1	L	Move R series with 1
7	_	55	_	L	Move R series with 1
55	0	65	1	R	CARRY
55	1	85	0	R	CARRY
55	_	10	_	X	CARRY
65	0	21	0	R	Move left series
65	1	21	1	R	Move left series
65	_	21	_	R	Move left series
21	0	22	0	R	Move left series
21	1	22	1	R	Move left series
21	_	22	_	R	Move left series
22	0	23	0	R	Move left series
22	1	23	1	R	Move left series
22	_	23	_	R	Move left series
23	0	24	0	R	Move left series
23	1	24	1	R	Move left series
23	_	24	_	R	Move left series
24	0	75	0	X	Move left series

Compactly, for those who perfer the precision of formal descriptions:
In other terms :

- The Turing machine runs on a 2-way unbounded tape filled with zeros (no blank symbol).
- At each step, two conditions :

3. the machine's current "state" (instruction); and
4. the tape symbol the machine's head is scanning

define each of the following :

1. a unique symbol to write (the machine can overwrite a 0 or a previously written symbol) usually denoted from 0 to n;

2. a direction to move in (Left or Right but "none" is not allowed in this model except when the machine halts); and
3. a state to transition into (may be the same as the one it was in) usually numbered from 1 to m.

- A Turing machine is a 5-tuple of sets: M = (Q, Sigma, delta, q0, qH) where
- Q is a finite set of states including q0 and qH
- Sigma is a finite set of input symbols (there is no special 'blank' character, the background is a list of zeros)
- delta is a transition mapping Q-{qH} x Sigma to Q x Sigma x {1,-1,0}
- q0 is the initial state (1) in F
- qH is the halting state (0) in F
- F is a finite set of final states

The Halting Problem

The ultimate objective of Turing's mathematical work was the development of a proof that some statements about natural numbers were not "decidable". To build this proof he needed to formalize the idea of a definite method or an algorithm (which he did with the Turing machine). He also needed to establish that the rules that operate on numbers could themselves be represented by or encoded as numbers. With the Universal machine he introduced the possibility of one machine accepting input that can be interpreted as both rules and data. Finally, again, with the universal Turing machine, we consider the possibility of a machine recursively operating on its own rules--accepting itself as input.

The final step in Turing's proof was to illustrate the contradictions that arise in the event that one assumes that every statement is Decidable. Turing did this by linking the notion of decidability with the more concrete notion (in terms of his machine) of Halting. Halting was the property that well-behaved algorithms exhibited when they concluded with the results of some calculation. If the program failed to reach a definite conclusion is was said to "Loop".

Using the technique called *reductio ad absurdum,* demonstrating that certain assumptions led to impossible or absurd conclusions, Turing showed that it would be impossible to establish whether *every* machine (algorithm) would halt or loop.

He did this by first assuming that it was possible--that there DID exist a mechanism, H, which correctly sorted all statements (rules and data) into the proper category: "halts" or "loops". This mechanism would necessarily have rules itself and those rules could also be represented as data. Consider the case where H is asked to check if rules H running

data H will halt. Since we know that H always halts (with a “halts” or a “loop” statement, it is not hard to say that H(H,H) will halt.

Consider a slight modification to H (the new program will be called H’) and require that this new program base its output/behavior on the result of H. H’ is designed to take a single input which it duplicates and passes to H. If H says the input (used as both instructions and data--M running on M) halts, H’ loops. If, on the other hand, H says that M running on M loops, then H’ halts.

The absurdity arises when we run the code for H’ on H’. As described above the input, H’ is duplicated and passed to H. If H says that H’ running H’ halts then the real world H’ running H’ goes into a loop. If H says that H’ running H’ loops, then the opposite behavior of H’ running H’ is observed to actually occur.

One concludes from this that H (and H’) are impossible, contradictory machines.

It is easy to get lost in the notation and weird loops in the above description. I have offered a few other illustrations of this idea to try to make the paradox more accessible. This is an idea that is both fascinating and problematic as a consequence of its peculiar self-referential nature.

The Halting machine, H, describe above, might be thought of as a machine that looks briefly into the future to see if a process ends or loops. Consider a real device the took you could use to make decisions that showed you what another person would do 10 minutes into the future. You could base your own decisions on what you see in the crystal ball. Of course, everyone would be most curious to discover what THEY themselves will be doing 10 minutes hence. But therein lies the paradox, as soon as you set the device to display your own behavior in the future you may what to change your behavior based on what you see. This is the standard fare of sci fi movies written by screenwriters who do not fully appreciate Turing’s proof.

Turing, AI and Choice Machines

In his On Computable Numbers paper (1936) Turing makes a distinction between an "automatic machine"—its "motion ... completely determined by the configuration" and a "choice machine"...whose motion is only partially determined by the configuration ... When such a machine reaches one of these ambiguous configurations, it cannot go on until some arbitrary choice has been made by an external operator. This would be the case if we were using machines to deal with axiomatic systems. “

Turing was motivated in his work, not only by his interest in the foundation of mathematics, but also by his hope of finding “springs and levers” of human intelligence. The loss of a close personal friend at the age of 19 inspired Turing to consider the

essential nature of the personality or intelligence. He recognized that human intelligence follows rules but also that, at points, it seems to depart from a strictly deterministic pattern. Turing was hopeful of discovery the underlying mechanism that would support this kind of interaction. He felt that it would only be a matter of time before humans would successfully implement “human-like intelligence” in a mechanical (or electro-mechanical) device. He recognized that this would be a profoundly significant milestone for the human race. In order to identify when mechanical, artificial intelligence achieves equivalence with human intelligence, Turing proposed a “test”. The test, or Turing Test as it is now called, involved a series of blind interviews with an AIs and a human subjects. If the interviewer is unable to distinguish between the responses of the AI and the human subject, Turing reasoned, there can be no significant difference between the intelligences.

[sidebar in the opening scenes of the classic sci-fi movie BladeRunner. An actor resembling Alan Turing administers a Turing test to an AI who (perhaps) fails the test.

Holden: You're in a desert, walking along in the sand, when all of a sudden you look down...
Leon: What one?
Holden: What?
Leon: What desert?
Holden: It doesn't make any difference what desert, it's completely hypothetical.
Leon: But, how come I'd be there?
Holden: Maybe you're fed up. Maybe you want to be by yourself. Who knows? You look down and see a tortoise, Leon. It's crawling toward you...
Leon: Tortoise? What's that?
Holden: [*irritated by Leon's interruptions*] You know what a turtle is?
Leon: Of course!
Holden: Same thing.
Leon: I've never seen a turtle... But I understand what you mean.
Holden: You reach down and you flip the tortoise over on its back, Leon.
Leon: Do you make up these questions, Mr. Holden? Or do they write 'em down for you?
Holden: The tortoise lays on its back, its belly baking in the hot sun, beating its legs trying to turn itself over, but it can't. Not without your help. But you're not helping.
Leon: [*angry at the suggestion*] What do you mean, I'm not helping?
Holden: I mean: you're not helping! Why is that, Leon?
[*Leon has become visibly shaken*]
Holden: They're just questions, Leon. In answer to your query, they're written down for me. It's a test, designed to provoke an emotional response... Shall we continue?

Holden: Describe in single words only the good things that come into your mind about... your mother.
Leon: My mother?
Holden: Yeah.
Leon: Let me tell you about my mother.
[*Leon shoots Holden with a gun he had pulled out under the table*]

end sidebar]

On Computable Numbers, with an Application to the Entscheidungs problem (1936).

[sidebar: Formal Description of a Turing Machine

- A Turing machine is a 5-tuple of sets: M = (Q, Sigma, delta, q0, qH) where
- Q is a finite set of states including q0 and qH
- Sigma is a finite set of input symbols (there is no special 'blank' character, the background is a list of zeros)
- delta is a transition mapping Q-{qH} x Sigma to Q x Sigma x {1,-1,0}
- q0 is the initial state (1) in F
- qH is the halting state (0) in F
- F is a finite set of final states

]

1 the phrase is etched in Hilbert's gravestone. It was his enthusiastic response to a then popular group of academics that subscribed to the Ignoramus et ignorabimus "we do not know and will not know" school of thought. this group held that certain things were

forever beyond the reach of scientific knowledge “ "1. the ultimate nature of matter and force, 2. the origin of motion,... 5. the origin of simple sensations.” These were held to be “quite transcendent questions."

Parable:

There is a factory in this town that makes cookies. In that factory there is a machine that has an arrangement of hoppers and conveyors that feed ingredients into the machine. The internal workings of the machine are carefully designed to act on the ingredients just so--mixing, shaping, cooking and cooling--cookies roll out on a conveyor belt.

Next door to the cookie factory is a shop that makes cookie machines. Screws, springs, circuit boards and wires are fed in on conveyer belts and the internal workings of the machine arrange parts, solder chips and weld plastic housings. A cookie machine emerges.

There is a third shop in this town, perhaps the most marvelous shop of all. In this shop there is a single machine, the Acme Super U, with a single conveyor belt that stretches out of sight down a long dark hallway. On this belt is placed pages of specifications, blueprints or instructions and after that, a variety of parts. The belt rolls, the machine draw the instructions and parts into it it gaping maw. Lights flash and gears whir, the belt inches forward and the parts enter the machine one by one. The belt moves to and fro sometimes advancing,sometimes retreating, as it scans the instructions on one part of the belt, it rearranges parts on the other end. Parts are built into subassemblies and subassemblies become even larger units.

This machines progress is not straightforward. Those who watch may wonder if the process of scanning, manipulating parts and more scanning will ever end.

In this case, the process does end. A full blown cookie machine emerges, inches forward on the belt, and the machine halts.

A new set of specifications are placed on the belt followed by bowls of flour, sugar, egg whites and butter. the machine is set in motion once again. Before too long, the smell of fresh baked cookies fill the room. This marvelous machine is able, it would seem, to transform any effective set of specifications and set of parts into wonderful artifacts.

Whenever ACME has a customer for a new Super U, it builds one--with the Super U itself! The specifications for the Super U are long and the parts many but in a safe

somewhere there exists a dusty box holding the complete instructions and parts list for the Super U.

Placing the instructions and and a complete set of parts for the Super U on the belt of the original Super U results in a shiny, new Super U.

Over the years, upgrades to the machine revealed that it was not required that the actual parts be placed on the belt--just 10 digit numbers describing their location in inventory. Likewise, the assembly instructions could be represented by ten digit numbers representing an assembly action. This simplification allowed a single operator adept at generating 10 digit numbers, to produce both the assembly instructions and the parts lists for the create of a variety of machines. Of course, it was only a matter of time before the pranksters at Acme began feed the parts list in as instructions and the instructions in as parts list --just to see what the machine would build. The cleverest of all engineers would create a single list of 10 digit numbers, duplicate the list, and feed it in as both instructions and parts list.

Another prank undertaken by the night shift (when things drag a bit) involved encoded the instructions that build a Super U and then rearranging the inventory so that the list of Super U instruction could ALSO be used to look up the parts. In this way a single list could be used as instructions AND parts lists. The night shift called their Super U a Single LIst Self Duplicator. Needless to say, day shift was less impressed by this accomplishment than night shift had hoped.

There is only one problem with the Super U. If a single instruction out of the thousands that are needed to build a machine is missing or if a single part missing from the parts list , the Super U may not only fail to complete the machine but it may become mired in a long, meaningless cycles of fruitless looping--forever screwing and unscrewing the same part--painstakingly assembling a module and then just as painstakingly disassembling the same.

“Wouldn’t it be convenient”, thought the executives at Acme, “if we find a machine that would “preview” the instructions and parts lists and let the operator know if the operation that was about to be undertaken would result in complete machine or endless loopiness?” The engineers at Acme fell into two camps: those that thought that the loop detector was an impossible dream and others, who thought that the idea had merit.

No one was surprised when a competitor, Dubious Enterprises, began to advertise the Super Halting Teraphim . The Teraphim claimed to accept as input any set of instructions and corresponding list of parts and determine whether the combination would lead to endless looping or happy conclusion. In Dubious' technical language, whether the process would "Halt" or "Loop".

The Teraphim, an ugly black box with slot to accept 10-digit encoded input, displays its output with a curious disc that rotating disc. On the disc are the words "Halt" and "Loop". At the conclusion of any run, the Teraphim rotates the disc so that one of the words appears in a window and then the machine turns itself off. This last operation was advertised as a power-saving feature but also demonstrated the fact that the machine had reached a conclusion without itself entering the dreaded loop condition.

The clever night shift engineers at Acme were skeptical. They quickly threw together a simple device with an elaborate name, the Bivalent Limitrophic Logogram and Super Halting Teraphim Detector. The Detector, as they called it, would test Dubious's claim that they could determine whether ANY set of instructions and parts lists. To create the Detector, the engineers modified the encoded instructions that Dubious had provided with instructions from their Single LIst Self Duplicator. The Detector was designed to take any input, duplicate that input and pass the copied input to the Teraphim as BOTH instructions and parts list. In this way they were in effect asking the Teraphim "will this set of instructions Halt when given this (the same encoded numbers) parts lists?" Like the Teraphim that it was testing, the night shift's Detector, was not designed to build anything, it was only required to do one of two things:

a) if the internal Teraphim claimed that the input passed to it Halted, Detector would itself begin to whirr and enter an endless loop
b) if the internal Teraphim claimed that the input would Loop, the Detector would itself halt, after printing out the phrase "Voila!".

To test their creation night shift passed Detector a simple encoded set of instructions that they knew created a nice little treat: a set of encoded digits represented baking instructions and the same set of digits could be used to located flour, sugar butter and chocolate chips in inventory. They knew the program "Halted" in practice and when feed to Teraphim. Everything worked as expected when the input was feed to Detector. Detector duplicated the instruction set, feed both parts to its internal Teraphim, received the expected "Halt" result from Teraphim and, according to design, Detector began to whirr and entered an endless loop.

Now the rub!

But what happens when the Detector is asked to evaluate its own instruction? First the engineers rearranged the inventory numbers so that the same set of encoded numbers could refer both instructions and inventory. Then they feed this magic encoded number (which, for obvious reasons they called “Detector”) to their Detector. They call this situation “Detector running on input “Detector””.

What happens inside their little machine?Detector duplicates the number and asks Teraphim “Does Detector halt on input “Detector?” If Teraphim claims that Detector Halts on “Detector” the real Detector (running “Detector”) begins to loop. On the other hand, if the internal Teraphim claims that Detector Loops when given “Detector” as input, then the real Detector follows its instructions and halts after printing “Viola!”.

Teraphim doesn’t just fail to give the right answer, it gives exactly the wrong every time.

=----------=

[quotes:
I am thinking about something much more important than bombs. I am thinking about computers.-John von Neumann, 1946

There are two kinds of creation myths: those where life arises out of the mud, and those where life falls from the sky. In this creation myth, computers arose from the mud, and code fell from the sky.