Great Principles Project

Principles Summary

8/22/07

Computation

These principles define the nature of computational processes, both natural and artificial: what they can and cannot do, and how we cope with inherent and pervasive computational complexity.

Representations hold information.

A. A representation is a pattern of symbols that conveys information.

B. Every representation is embodied into physical phenomena, which we call the "carrier". We can encode the symbols of a representation as ink on paper, as magnetic patterns on a disk surface, as electromagnetic waveforms, as bumps and pits on a compact disk, or as 3D folded sequences of amino acids in DNA. The observer reads by observing carrier states. Thus representations are "strategic arrangements of stuff."

C. Meaning is discerned and acted on by the observers reading the pattern. Although this situates meaning in the perceptions of observers, many observers, all from the same background or community, will perceive the same meaning from a pattern of symbols.

D. The meaning (interpretation) of a representation depends on the means by which observers interact with the representation. Thus an observer of a linear string can parse it into a tree of embedded phrases; the tree is the string's meaning. A folded DNA strand is more interesting. The observers are RNA receptors, which match specific proteins neighboring in 3D space; the RNA cannot bind to the receptors of unfolded DNA because they are physically too far apart. (DNA sequencing reads unfolded DNA.) Therefore unfolded DNA does not embody all the information in folded DNA. Because the folded DNA is a minimum energy state of the molecular chain, unfolded DNA

can re-fold and the original information is once again recoverable. The physical embodiment of the representation affects what information can be derived from it.

E. The patterns of carrier configurations allowable as representations are described by rules. For example, we can tell if a string of symbols belongs to a computer program by checking whether the string can be parsed by the language's grammar. At present we have no language to describe DNA folding, so biologists work directly with DNA strands without trying to represent them as formal systems.

F. Representations are finite.

G. Representations are equivalent if they hold the same information. A folded DNA strand is not equivalent to an unfolded (linear) strand.

H. Because there are infinitely many ways to represent the same thing, the number of representations equivalent to any given one is infinite.

I. The representations traditionally associated with computing are linear -- strings in a language, tapes on a machine, sequences of 0 and 1. However most representations occurring in nature are nonlinear -- for example, folded DNA.

J. In some computations, electrical currents or fields, or mechanical positions, represent observable quantities in the world. These representations are finite and continuous (not discrete).

K. Symbols can be encoded with patterns of bits (0 and 1). Linear representations can be encoded as sequences of symbol patterns. Nonlinear representations can be encoded as symbol patterns with their coordinates in the representational medium. Therefore every representation is equivalent to a pattern of bits. This applies to continuous representations because any value of the representation function can be approximated by a binary number.

L. The size of a representation R is the number of symbols (or bits) encoding it. The size of the shortest representation equivalent to R is called the Kolmogorov complexity of R, written K(R). No representation equivalent to R can be shorter than K(R).

M. An algorithm is a representation of a method to accomplish a task or process. An algorithm transforms input data to output data. Input data are states of an input representation, which can be thought of as a set of symbols that represent values. Output data are likewise the states of an output representation. A program is an algorithm plus its data.

N. A complier translates a program to machine code, the low-level bit patterns that drive a machine. The machine code representation is

equivalent to the higher-level program. Because of the equivalence, an algorithm can be regarded as a logical machine.

O. The design of a program goes through various stages of precision, typically including requirements, specifications, source language, compiled code. The objective is that the successive representations are equivalent, that is, the compiled code means the same as the original requirements.

Computation is a sequence of representations.

A. In 1936, Alan Turing published a famous paper in which he defined computation, computing machines, and universal machines, and he showed that the halting problem for machines was not computable. He demonstrated that the century-old "eintsheidungsproblem", which posited an axiomatic universal logic system that would be compete and consistent, had no solution. The mathematician David Hilbert's dream of a method to tell "by inspection" whether a computation halted could not be realized. Turing showed that the very steps a mathematician might use to apply a "by inspection" method were fundamentally computational. Therefore the only general method of approaching the question of whether computations halt or produce useful results is to run them and see what happens. Turing thus showed that computation is unavoidable. This truly was the birth of computer science.

B. Turing defined algorithm and computation relative to his machine model (Turing machine). Since his time, the definitions have evolved so that they depend on representations rather than machines, and thus explain computation that occurs in nature. Computing machines are just one way of realizing computations. Computation is the principle, the computer is the tool.

C. An algorithm's total state is a record of the values of all its input, output, internal (private), and external variables. These variables are specified in the algorithm's data representation.

D. An operator causes a specific, precise change in total state. Most operators alter a confined, finite portion of a state. Some operators can alter the entire state.

E. Given an initial state, an algorithm specifies how operators from a finite set are applied to produce a final state: in what order and how many times. In an analog machine, the algorithm is the network of operators rerpresenting a differential equation and the state is the current flowing in the network.

F. A computing machine is a physical system or process for holding a representation and acting on it by an algorithm. Machine instructions are machine-level operators. Note that "machine" does not necessarily mean digital electronics. A DNA sequence is a representation, and the various enzymes that split, read, and recombine DNA strands are operators.

G. A computation is a sequence of states of a data representation caused by an algorithm. In an analog machine, a computation is the continuous function produced by the network of operators. Computations are also called processes.

H. Since states are themselves representations of values, there is no loss to say simply that a computation is a (possibly continuous) sequence of representations. The successive representations are controlled by the logic rules embodied in the operators.

I. If an algorithm's data representation has an infinite number of states, the algorithm can generate a potentially infinite number of computations. Thus an algorithm is highly compressed representation of a very large, potentially infinite, space of computations. This is why algorithms are difficult to understand and prove correct.

J. Turing introduced the universal machine, one that can reproduce the computations of any other machine. The rules of operation and data of the subject machine are encoded as standard-form representations; the universal machine interprets and updates the representations, applying the logic rules of the subject machine.

K. A consequence of the universal machine is that any computing machine can simulate any other. This includes what appeared to be different models of computation such as the Church lambda calculus, Post's production systems, and Kleene's recursive functions. The conclusion is summarized as the Church-Turing thesis, that any computational process can be simulated by a computing machine.

L. In the 1960s there was a strong concern for control structures: organizing programs so that their dynamic computations would mirror the textual structure. This made algorithms more understandable and reduced errors. A famous theorem said that all control structures could be expressed with just four basic forms: procedure call, sequence, alternatives, and iteration. These structures preserved a single-entry, single-exit property of code segments, enabling easy connection and nesting of segments, all without the need to know anything about the data on which the code segments operate. In this context the goto statement was considered harmful because it bypassed the single-entry-single-exit requirement.

M. The interrupt, now a fundamental structure in operating systems, was invented in the 1950s to give computations an orderly way to interact with their external environments. Signals from the environment triggered a procedure call on an operating system routine that received and acted properly on the incoming message. The interrupt is an elegant way to receive unpredictable data within a control-structure environment.

N. In recent decades, data-oriented computational forms have become more important than control-oriented forms. Large databases, natural language processing systems, interactive multiplayer games, and the World Wide Web emphasize computations that evolve according to the associations between data objects and dynamic interactions along agents (processes). DNA transcription, for example, evolves as reverse transcriptase enzyme reads elements of a DNA chain. Google searches depend on the hyperlink structure between documents and the frequencies of incoming links. Complex adaptive systems store information in their feedback loops, responding most strongly to associations between current input and prior states. These computations are all driven by interactions with the external environment of a computation and by the particular data that the computation receives at each interaction point.

O. In the future, new computational forms are likely to emerge from quantum computing, bioinformatics, semantic networks, economics, and more. The common factor is that all computational forms are sequences of representations.

Representations can be compressed, but not too much.

A. If a long representation L and a short representation S hold the same information, an algorithm for constructing S from L is called compression. If the process is reversible, the algorithm for recovering L from S is called decompression. If decompression fully and completely reconstructs L from S, the compression is lossless.

B. One possibility for lossless compression is to substitute shorter codes for symbols encoded in L. For example, 8-bit ASCII codes in L might be replaced by variable length codes of average length 6 to construct shorter representation S. S can be decompressed to L simply by restoring the original 8-bit codes.

C. The shortest possible lossless code has average code word length equal to the entropy of the symbols being encoded. (Entropy is the negative sum of the quantities p log p, where p is the probability of a symbol.)(Shannon's theorem.)

D. A second possibility for lossless compression is for S to be an algorithm which when executed generates L. This is not the same as the previous case, which did not count decompression algorithms as part of the compressed representation. Pseudo-random numbers are an example of algorithmic compression: L is the long sequence of random numbers and S is the compact algorithm for generating them. Although the sequence satisfies statistical tests for randomness, it cannot be truly random because S is so much shorter, and orderly, than L. Another example is the Mandelbrot set: an incredibly complex set of points generated by a very short algorithm.

E. An interesting case arises when there is no algorithm S that is significantly smaller than L: L is algorithmically incompressible. This is true randomness because there is no structure around which to compress. The algorithm S is basically nothing more than a statement to print out pre-recorded data. If L encodes messages by using a code with average codeword length equal to the message-source entropy, L will be algorithmically incompressible and the bits of L will pass statistical tests for randomness.

F. A third possibility for compression is value-preserving but lossy: the compression algorithm eliminates low-value symbols from L. This requires a valuation of symbols encoded into L and a value threshold to decide which symbols to preserve. In this case S may be considerably shorter than the entropy lower bound for L, but now L may not be accurately reconstructed from S.

G. An example of a value-preserving compression is the MP3 standard for music. By suppressing frequencies that the ear cannot hear, MP3 compresses music recordings by factors of 10 or more, far more than the typical factor of 2 for a lossless compression.

H. A value-preserving compression may suppress detail without reducing complexity. Complexity concerns interactions among components; detail the minutiae of components. For example, a model of a software system that displays only the module functions and their interactions will suppress many details, but will not reduce the complexity of the software system -- the modules still exist.

I. We often build models that both suppress details and ignore some interactions. These models are less complex than the original system. A queueing model of a system, for example, substitutes simple exponentials for the actual service time distributions and ignores initial transients. These assumptions enable a simple mathematical solution that can be quite useful for performance predictions.

Computations can be open or closed.

A. A computation is closed if its objective is to reach an end state in a finite time after being started in a beginning state.

B. Closed computations are associated with mathematical functions that map the beginning states to ending states.

C. A computation is open if its objective is to continue indefinitely. In certain states, the computation can exchange information with the environment. Incoming requests from the environment, called tasks, alter the state of the computation; responses to tasks are output to the environment.

D. Open computations are associated with interactive processes as well as non-terminating service processes such as web servers.

Computations have characteristic speeds of resolution.

A. Resolution time for a closed computation is the time to reach an end state, after starting at a beginning state.

B. Resolution time for an open computation means the time to respond to a task, after the task is submitted.

C. Resolution times for closed computations are expressed with order notations that express the way the resolution time depends on the size n of the input. Computations whose order is polynomial ($O(n^k)$ or better) are classified as “tractable”. Computations whose order is exponential ($O(2^n)$ or worse) are classified as “intractable”.

D. Some mathematical functions are noncomputable -- they cannot be computed at all. They have no resolution time. Examples include program halting, program equivalence, maximum possible running time, and finding Kolmogorov complexity.

E. Because they are not intended to stop, open computations use different measures of speed. The two most common are response time and throughput. Response time measures the time from when a user submits a task to a process and receives an answer. Throughput measures the number of tasks per unit time that the process completes.

Complexity measures the time or space essential to complete computations.

A. Complexity, a recurring theme throughout all of computing, measures the time or space required to complete an algorithm’s computations.

Assessing complexity is difficult because algorithms have potentially infinite spaces of possible computations, depending on their inputs. Larger input data sets usually require longer execution time and more memory space.

B. There are very simple algorithms that take enormous times to complete, and very complex algorithms that finish quickly: Algorithm size is not a good measure of complexity. (While the Kolmogorov complexity -- size of the shortest algorithm to compute a function -- might be a little better, it is not computable. Assuming it is computable leads to a contradiction similar to the paradox: "Define n as the smallest integer that needs at least fifteen words to define." Kolmogorov complexity is more useful to describe whether representations are incompressible and random.)

C. Therefore, the standard measure of algorithm complexity applies not to the algorithm, but to its computation space. Complexity expresses the relationship between execution time (or space) and the size of the input. The relation is expressed as "on the order of" -- denoted O(.) -- a function giving worst-case time (or space) for an algorithm to achieve its result. For example, a bubble sort algorithm takes time proportional to n(n-1)/2 to arrange a list of n items into ascending order, and it can do this in the same memory as its input. Therefore the time complexity of bubble sort is $O(n^2)$ and the space complexity is O(n).

D. A example of a simple algorithm with complex behavior is the request to print all n-digit numbers. This algorithm is very short. Its execution time is $O(10^n)$ because it enumerates all the numbers. The algorithm also needs space $O(10^n)$ to contain the answer. Because numbers raised to powers grow so rapidly, a small request can create impossible demands. For example, the time to generate all 30-digit numbers is $O(10^{30})$ steps, which on the fastest supercomputers (10^{12} operations per second) would take ten billion years. For all 10-digit numbers, which could be computed within a second, the output would require 2,000 reams of paper at 1,000 numbers per page.

E. We can lump together all algorithms of the same time (or space) complexity. For example, the linear algorithms (order O(n)), the quadratic algorithms (order $O(n^2)$), the polynomial algorithms (order $O(n^k)$, or the exponential algorithms (order $O(2^n)$). These sets of algorithms are called complexity classes.

F. Algorithms of complexity $O(2^n)$ or worse are considered "intractable" (print example). Algorithms of complexity $O(n^k)$ or better are considered "tractable" (sort example). This division of hard versus

easy is relative; even an $O(n^2)$ algorithm can demand more time than we can give when n is large enough.

G. Some intractable algorithms are slow because they have to enumerate many cases in search of a small set of optimal cases. These algorithms contain "choice-points" at which they can select one of several possible next configurations for testing. An algorithm has to remember its choices so that, when one does not work out, it can backtrack and try a different choice. The algorithm could avoid backtracking if it could successfully guess the best choice the first time it encounters each choice point. Such an algorithm is called nondeterministic. Numerous intractable deterministic algorithms have tractable nondeterministic counterparts. For these algorithms, an answer can be verified in polynomial time even if it takes exponential time to compute.

H. Tractability can be extended from individual algorithms to problems. A problem's complexity is the complexity of the best algorithm for solving it. P denotes the class of problems that are solvable with deterministic polynomial time algorithms, and NP the class of problems solvable with nondeterministic polynomial-time algorithms. It is unknown whether P = NP. At present, the best algorithms known for NP problems are exponential or worse. It is possible (but unlikely) that one day a polynomial time solution will be found for some problems in NP.

I. Heuristics are polynomial-time algorithms that employ simple rules of thumb to find approximate solutions to NP problems. While not guaranteed to find answers even close to optimal, heuristics are often good enough for practice. Experimental studies reveal which heuristics work well in practice.

J. Computation A is reducible to computation B if A can be solved by encoding it as an instance of B; the encoding process should take no worse than polynomial time.

K. There is a subset of NP called NP-complete (NPC). NPC problems are considered the hardest of the NP problems because every NP problem can be reduced to at least one of them. Moreover all the NPC problems are mutually reducible to one another. Over 3,000 problems belong to NPC, including most of the typical engineering, science, and commerce problems. Although no one knows of a fast (deterministic polynomial) algorithm for any member of NPC, if anyone ever finds one, it can be converted to a fast algorithm for every other member of NPC; that would prove definitively that P = NP. That no fast algorithm has been found for any NPC problem is taken as strong empirical evidence that $P \neq NP$.

Finite representations of real processes always contain errors.

A. A representation of a continuous real variable or process can never be exact because the finite number of represented points cannot cover the infinity of real points. Representation error is the difference between a real number and its nearest represented number. With careful planning, algorithms can be organized so that representation errors of outputs are limited. Mathematical software libraries are designed this way.

B. Floating point numbers are a basic way of representing numerical data in a machine. A floating point number consists of a mantissa M between 0 and 1 and an exponent E, meaning $M*10^E$. Suppose that E and M are stored as parts of 32-bit words. Then there are at most 2^{32} floating point numbers. Therefore the error between a real number and the nearest floating point number representing it has an error of as much as 2^{-32}. For N-bit words, every floating point has an error of as much as 2^{-N}. All floating point arithmetic therefore induces representation errors between the numbers in the machine and the numbers in the real process.

C. In a poorly organized long computation, representation errors can accumulate, culminating in a floating point result with a large error. For example, a sum of n floating point numbers can have an error as high as $n*2^{-32}$. Errors in small differences can also cause problems. For example, an algorithm that computes 1/(A-B) may give a divide-by-zero error if A and B are within 2^{-32} of each other.

D. Many real processes are represented with computational grids that cover the space of interest. For example, a weather calculation might compute pressure and wind at discrete points separated by 1 km; or an aircraft structure might calculate stress on the wings at discrete points separated by 10 mm. The algorithm itself computes values (weather data, stress) only at the grid points. There can be a small error between the value at any real point and the values at the nearest grid points. This type of representation error can usually be controlled by keeping the grid point separation sufficiently small.

