

# Nature of the Information Sciences

Robert L. Constable

Office of Computing and Information Science

Task Force on Cornell Research Initiatives -- 1997 Background Material

---

This document offers a definition of the Information Sciences and illustrates their impact, especially on universities. For want of popular accounts to serve as reference points for this discussion, it will describe the Information Sciences by comparison to the more traditional sciences. Many points are best made by examples from computer science which is at the core of this constellation of disciplines.

The document is a basis for discussing the opportunities for Cornell University to play a leading role in these sciences and for considering their impact on other research opportunities in the next century.

---

## 1. GENERAL

### 1.1 Analogies and Definitions

The concept of an *information system* helps organize the subject of the Information Sciences just as the notion of a *living system* helps organize the Biological Sciences.<sup>1</sup> An information system can receive data, send data, classify it, store it and process it. The data *represents* information through interpretation.

This is an abstract concept with diverse instances. A digital computer is one example. It can process streams of bits, some representing numbers, some representing words, etc. A living organism is another example, one that also stores information in its genes. Digital libraries are information systems, as is the Web and the so-called "global computer," markets in the economy are as well, and other examples of information systems will emerge as we go on.

Some Information Sciences like linguistics study particular systems, e.g. humans using natural languages. Essentially computer science is concerned with intrinsic abstract structures underlying *all* information processing systems. By analogy with biology, computer science concerns the study of what is common to all living systems -- earthly and alien.<sup>2 3</sup>

### 1.2 Scientific Questions

One of the most basic questions about information systems seems philosophical: *Are there limits to what they can do?* This is like the question "what can we know?" It turns out that computer science has discovered limits to what information systems can *compute*, see Theory below.

A related question is: *How fast can various kinds of information system process various kinds of data?* Are there problems that one kind of system can solve that another kind cannot? Can data that seems random to one system seem orderly to a faster one, to one that uses different representations? Again

---

<sup>1</sup> The Information Sciences include at least computer science, parts of computer engineering, software engineering, information theory, information retrieval, information technology, artificial intelligence, linguistics, cognitive psychology, cognitive science, library science, and computational economics.

<sup>2</sup> Computer science is perhaps the most identifiable and institutionalized pure information science today, a science that is called *informatics* in parts of Europe because of its core role in the Information Sciences.

<sup>3</sup> Computer systems are built from discrete algorithmic processes (mechanizable or *computable* processes) and process information that can be represented *digitally* (but a significant topic is discrete approximation of continuous data and processes), but it may not always be so, computers could be organic.

---

These colleagues contributed text to the document and provided a great deal of feedback and advice generally: Joseph Halpern, S. Keshav, Fred Schneider, Eva Tardos, Charles Van Loan, Steve Vavasis, and Richard Zippel. We also had input from Paul Chew, Dan Huttenlocher and Greg Morrisett.

computer science has been able to formulate many of these as precise scientific questions, and to answer some.

Another basic question is: *How do we organize data for various kinds of processing?* How do we store it for the long term, for the short term? How does processing speed influence storage and retrieval processes? The subject of information retrieval has gathered a great deal of quantitative data on these questions.

*How does a system react to failure of components?* How can we make a system robust against failures, what are the limits? How does it react to information overload? These questions interest biologists as well. They know how organisms respond to information overload, from cells to brains. Some of their data can be explained in terms of what computer scientists have learned about operating systems. Indeed, when organisms are studied in terms of their essential functions, it turns out that nearly half of them process information. Some of the hypotheses about the organization of organs, like the hypothesis that two-way feedback channels reduce error [Miller, 1978], are instances of design principles in computer systems.

*How does data represent information?* Can we transform all data representing the same information into the same canonical form? Can we tell when data represents the same thing? Does the representation of data limit what can be done with it? How does the representation influence processing speed? How can systems access data when it is given incompletely? Many of these questions can be made precise by semantics or information theory, see Semantics below.

*How does communication bandwidth and speed affect the organization of information within a system?* How does communication speed between components affect the organization of data and the trade off between re-computing information versus looking it up? How can systems that are operating concurrently coordinate their processing? How much faster can cooperating processes solve a problem compared to a sequential one? A great deal of data has been collected on these questions by hardware and software engineers, and theories have been formulated to explain it.

Some questions seem especially challenging and subtle: *What does it mean for an information system to be intelligent, to learn, and to evolve?* What is the simplest system that can learn? Can learning technology improve the productivity of industrial information systems? Although these are hard questions, progress is being made on them in a subject called *learning theory*.

From such questions as these, we can see that overall we are concerned with five functions of information systems.

- representing information as data
- storing data
- classifying and linking representations
- processing
- communicating

### 1.3 Theoretical Aspects

In general *information constraints* govern how information systems are organized, what they can perform and how they evolve. Those constraints are the laws which define the abstract space in which the systems “live”. Computer science is concerned with discovering the *universal laws* that apply to all information systems.

This generality of concern has led to mathematical theories of computation and information that address many of the scientific questions. Some results have the character of *laws*, like this one: “once information systems become sufficiently complex, there is no algorithmic procedure for telling whether two of them are functionally equivalent.” Such results tell us what can be computed in principle and what not, other results tell us what can be computed in practice and what not. Theory has led to fundamental concepts such as the *universal machine*, one that can be programmed to compute what *any* other system can. This result has had profound consequences for technology. Mathematical theory has also led to a *theory of representation* to explain how data corresponds to information and when static properties of systems determine their dynamic ones.

### 1.4 Experimental Aspects

Investigation in the Information Sciences also proceeds by experiment, especially in psychology, linguistics, information retrieval, and cognitive science. In computer science the term “experiment” also

covers the demonstration of what can be realized in a computer system as a proof of what is possible in general. Indeed this is the primary meaning of the term “experiment” in computer science today.

These demonstrations are sometimes the superior expression of the idea behind a system, “a demonstration is worth a million words.” For complex information systems demonstrations may be the only way to understand what is possible or the only way to discover the critical constraints governing behavior.

### 1.5 Shaped by Technology and Creating Technology

Just as astronomy, physics and biology are influenced by technology -- telescopes, accelerators and electron microscopes, so too the Information Sciences are shaped by computing, storage, visualization and communications technology -- from PET scanners that examine brain function, to multi-processor 500MIPS PC's, that simulate brain function! In the case of computer science, the advance has been especially rapid and broad. Technology has allowed demonstrations of ever more complex information systems, and experience with them suggests new properties that might be explained by laws. For example, all surviving complex systems, living and artificial, seem to be constructed in a modular way, decomposable into subsystems and sub-subsystems and so on all the way down to the smallest units. Also the more components a system has the more echelons it has, and the more complex a system, the more information flows between components than to the outside.

Like physics, computer science has created technologies. The transistor, laser and fiber optic cables have come from physics while universal machines (realized in hardware), digital signatures, electronic commerce, and prototype distributed digital libraries have come from computer science. The technology is sometimes so spectacular that people forget the science from which it came. It is one thing for government agencies to forget, quite another for universities.

### 1.6 Outline

To continue the discussion, let us first consider the nature of the mathematical theories in more detail, then examine the experimental method in computer science and finally look at the impact of the Information Sciences and some of the features that make them an enabling science.

## 2. THEORY

Every aspect of an information system can be explored mathematically. There is theory related to data storage, to information representation, to processing, to communications and to learning. The subject is too broad and too deep to summarize as a whole. Instead I will illustrate with two central categories of work, results dealing with processing power and those dealing with representing information as data. Presently the two theories are connected because what counts as a representation is only data that can be transformed to explicit *digital* form. The theory of what can be computed and how much it costs we call Computational Complexity, and the theory of representations we call Semantics.

### 2.1 Computability and Complexity

Some of the earliest theoretical results in computer science are from names that are now legend, like Alan Turing, Alonzo Church and Stephen Kleene. Turing introduced the universal machine (in the class of Turing machines) and a remarkable series of results revealed that this model is equivalent to every other model of sequential digital computing that has been proposed, including Church's first “high-level” programming language. Not only were these fundamental models discovered, but Turing and Church showed that there are classes of problems that no computation based information system can solve, such as the problem of whether programs fail to halt on their inputs, the so called *halting problem*. These results led to a classification of problems into *solvable* and *unsolvable*.

The subject of computational complexity which we discuss next arose from a desire to rate the “power” of an information processing system and to classify the solvable problems by their *intrinsic computational cost*. How much work can a particular system perform? To classify systems in this way requires a *quantitative* theory pioneered by Hartmanis and Stearns. The basis for measurement that they chose is to relate the expenditure of resource required to solve a task to the size of the task, say the number of bits in the data.

Cost can be measured in the time to execute a program or the amount of memory used by a program or the number of independent processes needed or any combination of these measures. The cost can be measured asymptotically over the whole range of inputs, as in saying that the process of parsing a sentence

is quadratic in  $n$ , the number of symbols of the string being parsed. It can be the “average cost” over the range. We also speak of the complexity of a problem in terms of the cost of the optimal algorithm to solve it if there is an optimal one. (Two classic textbooks in this area are *The Design and Analysis of Computer Algorithms*, by Aho, Hopcroft and Ullman [1974] and Kozen’s more recent *The Design and Analysis of Algorithms* [1991].)

**2.1.1 Traveling Salesman Problem as an Example.** Consider the so-called traveling salesman problem as an example: a salesman starts at home and must visit several cities before returning home, and he wants to determine the order in which to visit all of them so that he spends as little time traveling as possible. One way to compute the shortest tour is to check all possible orderings, and then to choose the best one. However, even for just 30 cities, there are more than 1,000,000,000 different ways to order the cities. And so this simple-minded approach is not feasible for the fastest computers today. Furthermore, as the computers become a 1000 times faster, with this approach we can only increase the number of cities by less than 10; so a 50 city problem is not even plausible for any computer of the future.

In complexity theory, we try to answer the question, “does any method to solve this problem need this much time, or could there be faster methods?” One of the major achievements of complexity theory has been the notion of NP-completeness, a characterization of the complexity of literally thousands of significant computational problems arising in virtually every discipline of science. We know that all NP-complete problems are equivalent in that if we can find an efficient algorithm (technically called a polynomial algorithm) for just one NP-complete problem, then this algorithm can serve as a master algorithm to solve each NP-complete problem efficiently. It is widely believed that no such algorithm exists, but to prove this remains the fundamental open problem of complexity theory, Cook’s famous  $P = NP?$  problem.

Knowing that a computational problem, such as the traveling salesman problem, is hard to solve might not seem of such practical importance, but it also guides us in what approaches should be considered instead. The salesman would, of course, settle for a route that is nearly the best, and so one further direction for NP-complete problems is to study approximation algorithms. That is, to answer the question, “Can we find solutions that are guaranteed to be nearly the best?”

The study of NP complete problems has revealed that diverse computational problems, such as discovering whether a graph has a closed route passing through each node exactly once or deciding whether a logical formula involving “and,” “or” and “not” is true under any assignment to its atomic parts or finding an optimal classroom schedule or solving a quadratic equation over the integers all have tightly related computational costs. Either all of them can be solved by a deterministic polynomial time algorithm (P-algorithm) or none can be. All of them can be solved by a nondeterministic polynomial time algorithm (NP-algorithm); and this leads to Cook’s famous  $P=NP$  problem. The class P of polynomial time algorithms are considered *feasible* and the problems that can be solved by these are called *tractable*. So asking whether P is equal to NP is asking whether all problems that are nondeterministically tractable are actually tractable.

**2.1.2 Cryptography.** Until this point, we have had the perspective that a computational problem being hard to solve is a *bad* thing. However, cryptography is one setting in which it is a *good* thing. One of major achievements of computer science is the notion of public key cryptography. The idea is that while the encrypted messages contain the information, unauthorized parties find it computationally too hard to deduce it. Advances in cryptography have led to methods for which one can formally show that decoding any particular message would require a general algorithm to factor numbers efficiently, and this is another computational problem that has resisted centuries of research. Similarly, in electronic commerce, we need ways to produce digital signatures, but here again this can be accomplished by relying on the intrinsic hardness of a computational problem, that of forging your digital signature.

## 2.2 Semantics

Useful data means something: a binary string, 100110, represents a natural number in hardware. A number in the field of a data base could represent a credit rating, and a triple of numbers along with a token might designate an airplane in Air Traffic Control software. To an organism, a protein might represent an intruder or food. To a natural language processing system, a string of bits might represent a concept providing the

key information needed to parse a sentence correctly, or it might be nonsense. Semantics is concerned with how data represents information.<sup>4,5</sup>

**2.2.1 Example of Numbers and Words.** To illustrate semantics, compare the subject of arithmetic to that of natural language processing. Computers can be said to perform standard arithmetic because of the way numbers are represented by bits and the way hardware is built. In this case the correspondence is especially simple because computers use a recognizable number system, say binary. They also implement as primitive the standard operations like addition and subtraction on small numbers (finite precision) in a way that can easily be extended to all numbers (“big numbers” or infinite precision). To say this precisely is to apply Semantics.<sup>6</sup>

Compare this to how computers represent words in a text editor like Word. First there is a table relating eight bit sequences to letters using a code (bytes to letters by ascii). The operation of looking up a word in a dictionary to find out its possible parts of speech is not built into the hardware. It must be programmed and the dictionary must also be built in software. The information about parts of speech will be “understood” only to the extent that the parsing program can use it. Thus knowledge of the representation is given by an algorithm -- meaning is in terms of rules of use.

**2.2.2 Semantics and Simulation.** Computational physics, chemistry and materials science use simulations to help understand nature. These scientists write programs and produce large systems. Semantics is the subject that connects the results of these programs to the mathematical theories underlying them. It attempts to guarantee that the numerals streaming off a printer to predict the location of a particle reflect the mathematics of mechanics used to code the program generating the numbers. It is what gives “meaning” to the results of computer programs.

This is a field where the computer scientist, the computational scientist and the experimental scientists work hand-in-hand. Each needs the others to produce meaningful results. The domain specialist knows which simplifying assumptions make sense in building a model, the computational scientist knows how to create a model of the phenomena being studied, and the computer scientist knows how to find the right algorithms, data structures and computing systems to produce results and knows how to guarantee that the algorithms are meaningful. This relationship is similar to the one between mathematicians and theoretical physicists.

**2.2.3 Semantics, Specification and Checking.** In general to know what a computer system is doing, we want to guarantee that it has certain properties. Semantics provides a language for stating these properties. It also provides the basis for deductive systems to prove the properties -- called *programming logics*.

In a remarkable example of *automation*, computers are used to manipulate formulas and proofs of these programming logics so that it is possible to check that critical systems behave as specified. One class of automated system are called *model checkers*, and they are used in industry to check chips for logical flaws - - by symbolically exploring all possible states that a chip can enter with respect to the critical properties of the system.

These checkers reveal the vast amounts of information that a system can process if it is represented in a way that takes account of structure. They can explore essentially 10 to the 120<sup>th</sup> power number of states,  $10^{120}$ , if there is enough structure. This kind of deductively constrained search might be critical to complex problem solving in humans. So these commercially useful model checkers are also testing methods of knowledge representation important to cognitive science.

Model checkers are also combined with *theorem provers* to check the reliability of hardware and software systems. The provers require large databases of formalized mathematics, and the creation of this formal material appears to be opening a new branch of mathematics, called *formalized mathematics*.

---

<sup>4</sup> Meaning depends on the context provided by the system and its environment. A key idea from Turing is that programs can be data -- its meaning is the program's execution. Another key idea is that some data is used to control the system, like the headers on messages telling how to route them. This is *metadata*, and there is great power in computing with it to control a system.

<sup>5</sup> Meaning is defined by the relationships between data in information processing systems and other kinds of data in the environment. The classification of data into *types* appears to be an adequate basis for meaning in the traditional sense, and is central to the organization of programming languages as well as natural languages.

<sup>6</sup> It is considerably more complicated to know that computers represent the notion of a *real number* in some sensible way. Indeed the exact story is not definitively known which accounts for some bizarre behavior from machines. This behavior can be dangerous when a computer program controlling an airplane does not “understand” real arithmetic in the same way as the engineers who built the plane or the Air Traffic Control system.

### 3. EXPERIMENTAL SYSTEMS

There is something very basic about observing an information system in action. One can read all the details of a design, hear descriptions of what the systems will do, but a deeper understanding comes from using it. Many people have experienced this phenomenon with the Web. Being told about it is one thing, using it is quite another.

The course of computer science is driven by demonstrations (in academia the saying is not only “publish or perish” but “demo or die”). The internet and Web are two familiar examples, but every programming language started as a demonstration. Computer scientists think of languages as on-going experiments in communicating algorithms. They are demonstrations of what concepts for communicating with machines work well together.

#### 3.1 The Internet and the Web

As remarked above, most of the work in the experimental aspects of computer science concern building demonstration systems. One reason for this is that we are capable of imagining and building computing structures that are far beyond our analytical capabilities today. So, we build things because we can’t predict how they will behave in one or another interesting dimension but we can measure that behavior. Then, by making measurements, we develop an intuition and refine theories to inform further efforts.

The origins of the Internet are an example. Originally an ARPA-funded research effort, the question under investigation was how to build computing systems where processors communicated with each other over telephone lines, which are low bandwidth/high latency devices (relative to the characteristics of processor to memory and processor to disk communication). Although we couldn’t predict the behavior of such a system -- it was enormously complex by the standard of the day -- we could build one. Would a useable system result or had some important dimension for performance been ignored? Would reliability problems be more significant than expected? Once the system was built, it was heavily instrumented and used. Data accumulated, and it directed further directions in networking. As load increased -- as it now has -- for example, we knew exactly what issues to concentrate on.

Another reason for building demonstration systems is because computer scientists deal in mental tools that are so general-purpose, that we cannot predict all the ways in which they will be used. But by seeing these uses, we learn where to extend and improve our work. Why is it necessary to build these tools? The answer derives from the observation that in explaining an object, one conveys a mental model of it. Users of general-purpose tools will find new uses for those tools by virtue of having different mental models. These alternative mental models are best developed by people actually using those tools. A demonstration system serves that purpose. The WWW is an example. It is simple to explain the notion of a web browser; but it is only through first-hand experience that one can see all the possibilities it provides.

#### 3.2 Programming Languages

The history of programming languages shows an evolution from *machine languages* to *high-level languages*. Machine languages provide commands that a machine can directly execute, such as shifting 8 bits left by 4 bits. High-level languages like C and Java provide commands to execute familiar mathematical operations such as adding two numbers. They also provide declarative text for defining new constants or new data types. These commands and declarations must be *translated* into machine language before a computer can execute them. This translation process is called *compilation*.

The difficulty of compiling a language increases as it becomes higher-level. One of the first high-level languages was *Fortran*; it is close to machine language and relatively easy to compile -- undergraduates are taught how to do it. But modern high-level languages such as Ada, High-Performance Fortran, ML and Java are considerably more difficult to translate. They offer features such as concurrent processes and computations on data types that are quite complex. But compiler technology is steadily improving based on results from Semantics and on forty years of tool building for this activity; all of which has enabled computer scientist to implement very high-level languages and increase programmer productivity significantly (some would argue that software advances have matched those in hardware until very recently).

The design, implementation and evaluation of high-level languages is an experiment in communication. These experiments along with mathematical theory are used to explore the design space for expressing algorithms and algorithmic theories (see Algorithmic Thinking). Computer scientists aspire to using

fragments of natural language as a programming language. The efforts to accomplish this are shedding light on the structure of natural language itself.

## 4. An Enabling Science

### 4.1 Impact

The Information Sciences are enabling disciplines whose role in the next century will greatly expand. The record of impact of computer science is remarkable even by the standards of 20<sup>th</sup> century science. It has touched all the sciences as we see in the rise of Computational Sciences -- computational physics, computational chemistry, and computational biology -- here we see a natural partnership enriching both sides. Perhaps even more profoundly, it has changed or will change nearly every academic discipline, as it has touched nearly every organized human activity -- art, advertising, banking, commerce, communications, dating, education, entertainment, health care, government, law, libraries, manufacturing, marketing, music, publication, travel, warfare, writing and so on. Here we want to see why these Information Sciences constitute an enabling discipline.

### 4.2 Behind the Impact

4.2.1 *Validating Imagination.* The simple answer to why the Information Sciences are proving to be so essential to all academic disciplines is that all disciplines are concerned with the organization and dissemination of information. So when capabilities demonstrated by new information systems are useful, they create a demand, sometimes strong enough to build industries like the computer hardware industry and the software industry.<sup>7</sup> The widespread use of powerful computer systems then creates further demand and stimulates *imagination, dreams and vision* in ever wider communities. Validating these ideas requires computer science, and realizing the dreams requires the technologies that this science has spawned.<sup>8</sup>

4.2.2 *Building Cyberspace.* What has happened is that the universities, government and industry have taken ideas from computer science and they have created a new world, a “cyberspace” in which we all work. There is the internet, based on the old arpa net, a computer science research project. On the net sits the World Wide Web, programmed in Java, a programming language incorporating advanced type systems created by Semantics. The Web itself is a new information resource made useful by the search engines built in the discipline of information retrieval, a thriving Information Science. On the Web people use digital signatures for security, a concept inconceivable without Complexity Theory.

The questions and opportunities provided by this cyber world are now sufficiently real to people who have experienced it that they see new ways to do their work and live their lives. This world then stimulates rich new visions that drive people to learn the results of computer science, to deploy the technology of computer engineering, and in more and more cases to learn Information Science to enable a new program of research.<sup>9</sup>

### 4.3 How Computer Science is Enabling

To understand in more detail how the information sciences are enabling new kinds of research in many fields, let us look at a few salient examples.

4.3.1 *Algorithmic Thinking.* One of the key ideas to arise from computer science is that of embodying theories as algorithms. This idea has spread to other fields as well. For example, in an area like natural language processing, theories for how humans process language are embodied as algorithms. A theory that cannot be so embodied is rightly viewed with some suspicion. The extent to which the algorithm is able to

<sup>7</sup> The computer and communications industry generated 500 billion dollars worth of business according to the HPCC study [cite]. *Business Week* reports profits in the software industry world wide at 300 billion dollars, 200 billion of which is in the United States. It is no surprise then that Microsoft has a greater market capitalization than the Big Three automobile manufacturers combined (155B to a combined 109B), figures as of July 1997, research data available from www.biz.yahoo.com.

<sup>8</sup> This is quite like the situation with physics. People dream of all manner of physical devices or conjecture intriguing ideas about the origins of the universe -- from perpetual motion and nuclear fusion to space ships, nuclear fission and big bangs. To validate this imagination and separate the reality from the nonsense requires detailed knowledge of the laws of physics. Likewise to sort out the reality of a distributed digital library from the nonsense of an “intelligent computer system” to decide whether any Fortran program will halt requires detailed knowledge of computer science.

<sup>9</sup> The accumulated knowledge in computer science alone defines a challenging undergraduate major whose graduate are in very high demand with an average of nine job offers. *Business Week* reports there are over 190K job openings for these graduates, and a shortage of programmers is expected to persist for at least a decade.

mimic human behavior is cited as evidence for the correctness of the approach. Tools from complexity theory are also applied. If a suggested approach to doing language processing leads to NP-complete problems, this is taken to be strong evidence that this is *not* how people approach the problem. Of course, this approach is equally applicable to other information processing systems as well. To take just one other example, from biology this time, thinking of the genome as an algorithm has suggested new approaches to understanding its functionality.

4.3.2 *Information Processing in Biological Systems.* Fifty years ago, before the discovery of the molecular structure of DNA, few would have suspected that the storage, transmission, and duplication of information lay at the heart of all biological processes. One of the major achievements of our age is the realization that the nature of the living world is determined in large part by the information content of the underlying DNA. Thus, information science, which deals with abstracted information, governs not only computational machines, but also the very processes of life.

The complete genome for many organisms will be digitally available in a few years. Then, a major research task, with potentially breathtaking consequences, will be to connect genes to function -- a subject called *functional genomics*. Success in this venture will allow things such as genetic medicine, the controlled modification of living things and potentially the elimination of many diseases! Unsurprisingly, this task has a large information sciences component. Indeed, one aspect of the problem looks quite similar to a well-studied subject in computer science, how binary machine code produces functionality. It might be possible to model a simple organism as a set of concurrent finite state processes of the kind computer science knows how to analyze. The issue of genetic function then becomes a matter of mapping DNA "program code" to higher level abstractions. More prosaically, the problem of finding a particular section of the genome can be solved by information technology developed for data bases and for information retrieval.

4.3.3 *Scientific Computing and Advances in Computational Science.* Scientific computing is concerned with the discrete solution of problems that are posed in the language of continuous mathematics. Integrals, derivatives, and nonlinearity give way to finite sums, divided differences, and the local linear model. Although the analysis of these approximations is very much in the domain of applied mathematics, their effectiveness in practice requires the full force of computer science more than ever before.

For example, it is one thing to solve a partial differential equation over a cube and quite another when the domain has irregular boundaries. The latter requires a mix of techniques from computational geometry and advanced data structures.

A multidimensional fast Fourier transform can be mathematically described on a 3-by-5 card, but performance on an advanced computer depends on just how the data flows between the registers and the various memory units during program execution. Smart compilers and related code development tools are able to optimize in this direction because computer scientists have invented a mathematics of data motion enabling others to build systems that address the key issues.

Numerical optimization sits on top of a few basic ideas from calculus: gradients, Hessians, Newton's method etc. Large scale optimization when thousands of variables are involved can require a clever data structure to represent sparse interconnections and/or a graceful interface with a complicated database.

The point we are making with these examples is that by giving us tools for reasoning about the codification of data and how it is moved, computer science can greatly widen the class of solvable numerical problems in science and engineering.

4.3.4 *High-Precision Simulations.* One of the central goals of the physical sciences is the modeling and simulation of physical phenomena. At the turn of the century, simulations were done via hand computations and this was reflected in structure and nature of the physical models developed (e.g., small, relatively simple systems of equations). Today, computers have qualitatively changed our simulation capabilities, and in turn have changed the types of models that are developed to describe physical phenomena. Vastly larger systems of equations can now be used, with far greater accuracy. Models that are amenable to parallel automatic computation, such as lattice gas models for fluid dynamics, are being used in place of classical techniques. These techniques allow us to produce high accuracy simulations of fluid dynamics, and guide space craft to Mars.

As we see it, computer models of reality are the representations of the natural world in the digital one. As we saw in the semantics section, one way to understand this interpretation is to see Nature itself as an

information system.<sup>10</sup> The Information Sciences may be destined to become more important in framing physical theory as well as in testing it through simulations in the broad area now called Computational Sciences. Clearly computer science is enabling this new branch of science, and other Information Sciences may play a role as well.

## 5. CONCLUSION -- THE DIGITAL FUTURE OF RESEARCH AND SCHOLARSHIP IN THE UNIVERSITY

The role of computing in the biological sciences is highly visible because of its connection to medicine and drug company research; also industrial resources have propelled the field rapidly revealing just how deeply dependent it is on the Information Sciences. These developments are a harbinger of changes in all fields of research and scholarship. The convergence of media to digital format is creating vast stores of information that can be linked, indexed, organized and processed in ways that challenge our imaginations.

Understanding how to exploit and manage this information requires not only the results of computer science, but its methodology as well, that is, placing *demonstration systems* in the hands of users and enabling them to create new content.

For instance, we know that it is possible in principle to create a massive *distributed digital library* accessible via the Web in which historians can view digitized original manuscripts from half way around the world and compare pages side by side from different primary sources with multiple translations linked by hypertext. We know it is possible to build expert systems that will alert a scholar to pertinent information as soon as it appears on the Web for example. We know that these *tools for thought* will greatly amplify creative expression.

Although we don't know exactly how such systems will function, we can be certain that they will change how scholars and scientists work. We can also be certain that many scholars and scientists will contribute to Information Science and many will contribute in partnership with specialists as is happening now with computer scientists. Our graduate and undergraduate students will help build these systems. It is also certain that Cornell University must be first rate in the Information Sciences if it is to remain first rate in other areas as well.

## REFERENCES

- AHO, A., HOPCROFT, J., AND ULLMAN, J. 1974. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA.
- BIRMAN, K. P. 1997. *Building Secure and Reliable Network Applications*. Manning Publishing Company and Prentice Hall.
- CONSTABLE, R. L., ALLEN, S. F., BROMLEY, H., CLEAVELAND, W., CREMER, J., HARPER, R., HOWE, D. J., KNOBLOCK, T., MENDLER, N., PANANGADEN, P., SASAKI, J. T., AND SMITH, S. F. 1986. *Implementing Mathematics with the Nuprl Development System*. Prentice-Hall, NJ.
- FAGIN, R., HALPERN, J. Y., MOSES, Y., AND VARDI, M. Y. 1995. *Reasoning About Knowledge*. Massachusetts Institute of Technology.
- GOLUB, G. H. AND VAN LOAN, C. F. 1997. *Matrix Computations* (3<sup>rd</sup> ed.). Johns Hopkins University Press, Baltimore, MD.
- HARTMANIS, J. AND LIN, H. Eds. 1992. *Computing the Future, A Broader Agenda for Computer Science and Engineering* (1992). Committee to Assess the Scope and Direction of Computer Science and Technology; Computer Science and Telecommunications Board; Commission on Physical Sciences, Mathematics, and Applications; National Research Council: National Academy Press.
- KESHAV, S. 1997. *An Engineering Approach to Computer Networking*. Addison-Wesley.
- KOZEN, D. 1991. *The Design and analysis of algorithms*. Springer-Verlag, New York.
- MILLER, J. G. 1978. *Living Systems*. McGraw-Hill, New York.
- MITCHELL, J. C. 1996. *Foundations of Programming Languages*. MIT Press.
- SCHNEIDER, F. B. 1997. *On Concurrent Programming*. Springer-Verlag, New York.
- VAN LOAN, C. F. 1992. *Computational Frameworks for the Fast Fourier Transform*. SIAM Publications, Philadelphia, PA.
- WHITEHEAD, A. N. 1925. *Science and the Modern World*. Macmillan, New York.

<sup>10</sup> Indeed this is a view of science that Alfred North Whitehead developed in *Science and the Modern World* [1925]: "... nature is a structure of evolving processes ... the realities of nature are the events in nature."