Models of Computation

What is Mathematical Computation?

At the turn of the 20th century, mathematicians believed that proof required skill and ingenuity. Logic had just been formalized, so the natural questions were

What is proof? What is a mathematical system?

In particular, mathematicians wanted to know if all mathematical proofs could be verified by an algorithm. This is very similar to the constructivist philosophy that any mathematical object must come with a means to generate it. The trouble was with infinities, like the set of real numbers or the set of points on a plane, neither of which can be constructed, or shown to actually exist.

The initial program was simply to show that the *arithmetic of natural numbers*, a very small portion of mathematical knowledge, could be placed on a firm, unambiguous basis.

In 1931, Kurt Godel's incompleteness theorem ended all hopes that mathematics generated certain knowledge or Truth.

Incompleteness theorem:

Any formal system as complex as *the arithmetic of positive integers* is either

incomplete:	there are some statements which cannot be proven to be either True or False
or	
inconsistent:	there are provable statements which are contradictory

Until very recently, inconsistent theories were felt to be intolerable, therefore the idea of incompleteness was necessarily accepted.

1936

In 1936 four independent mathematical models of the meaning of "computation" appeared. This was prior to the invention of silicon computers but inspired by large mechanical computers, the equivalent of today's one dollar calculators.

partial recursive functions Kurt Godel and Stephen Kleene

functions which can be defined through recursion on well-formed sequences

equational general recursive functions Jacques Herbrand

recursive functions combined with equality

lambda calculus	Alonzo Church

a very small functional system based on substitution and abstraction

Turing machines	Alan Turing

a mechanistic model of problem solving

Interestingly, each of these unrelated models defines *the same set of functions*, thus they are functionally identical.

Computer Science Models of Computation

Since the Turing machine model is mechanical, it appealed to computer scientists, who in the 1940s were dealing with large machines rather than mathematical systems. The Turing machine model is the most clumsy and difficult of the alternatives.

In 1963, a model of computation was introduced which was closely based on the structure of modern computers:

Universal register machines	Sheppardson and Sturgis

The *hardware* point of view is

An algorithm is a piece of machinery which realizes a desired computation.
The set of instructions for the algorithm is defined by the hardware architecture.

The *software* point of view is

An algorithm is a sequence of textual instructions.

Programming Language

In 1955, John McCarthy built the programming language LISP based on lambda calculus. LISP is the most elegant and abstract of all popular programming languages. However, the important point is that no matter what software language you use on what hardware architecture, and no matter how poorly an algorithm is implemented, all implementations are the same with regard to what can be expressed or computed.

A programming language is a formal language for specifying effective procedures.

Programming languages define a class of mathematical functions, those functions which can be stated in the language. At best a programming language specifies the set of recursive functions.

Church-Turing Thesis

All reasonable formulations of the intuitive notion of computability are equivalent.

Wolfram's Fundamental Theorem (2000)

All nontrivial interactions have effective computability as a minimal model.

Effective Procedures

All algorithms (and models of computation) have these properties:

1. finite set of instructions from a finite set of types of instruction
2. discrete stepwise process
3. deterministic (no random elements)
4. finite time and space for the process
5. each step involves a finite amount of data.

Algorithms are Not Functions

An *algorithm* computes a recursive *function*.

Algorithm	Function
a text	an idea
a string of tokens	a set of pairs
instructions	no instructions
input --> output	domain and range

Algorithms are effective procedures which solve a problem.

The types of problem that programming can solve are

1. deciding set membership
2. computing a recursive function

It is therefore desirable to specify problems for computers in one or both of the above ways.

Non-computational Mathematics

Here are some mathematical objects and operations which cannot be expressed as an algorithm or within an effective procedure.

real numbers	(absolute precision arithmetic)
transcendental numbers	(such as Pi and e)
infinity	(in any variety)
void	(non-symbolic grounds)
sets	(requires a parallel processor, one for each set member)
existence proof	(demonstration without producing a specific object)

Non-computational Functions

Here are some computational machine behaviors which cannot be expressed as recursive functions, and thus cannot be decided through computation.

Halting Problem

Does machine model M halt when given input string S?

Empty String Acceptance

Does machine model M accept the empty string?

Empty Language Acceptance

Does machine model M accept an empty language?

Regular Machine Recognition

Does a machine exist which can determine
if another machine accepts a regular language?

Rice's Theorem

No algorithm exists which accepts a machine description M
and determines if M accepts an effective procedure.

Self-terminating Machine

Does a machine halt when given a description of itself?