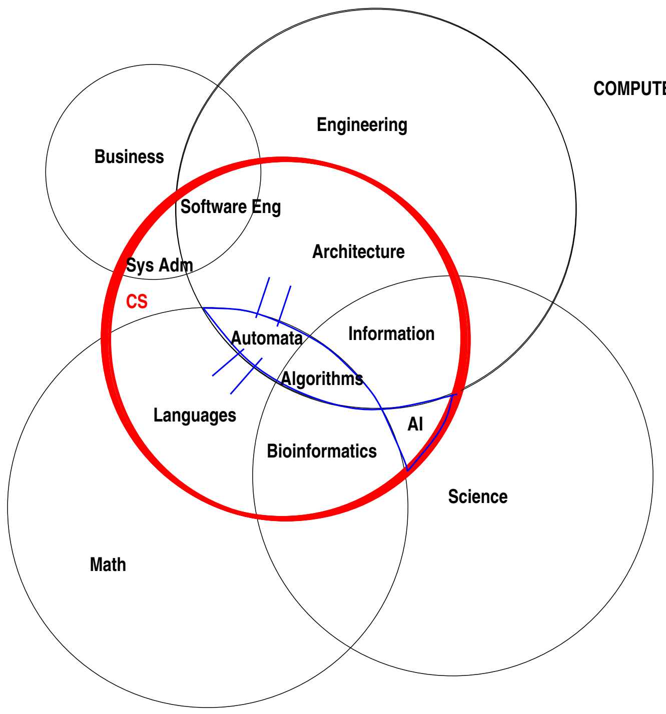
What is Computer Science?

We study:

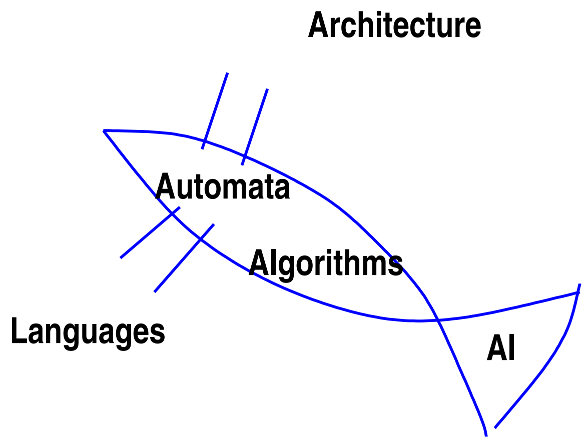
- *Math (algorithms, languages, automata)*
- *Engineering (design, information, machines)*
- *Science (physics, biology, cognitive sci.)*
- *Business (project management, system administration*

CS related fields



COMPUTER SCIENCE KNOWLEDGE SET

Turing Fish

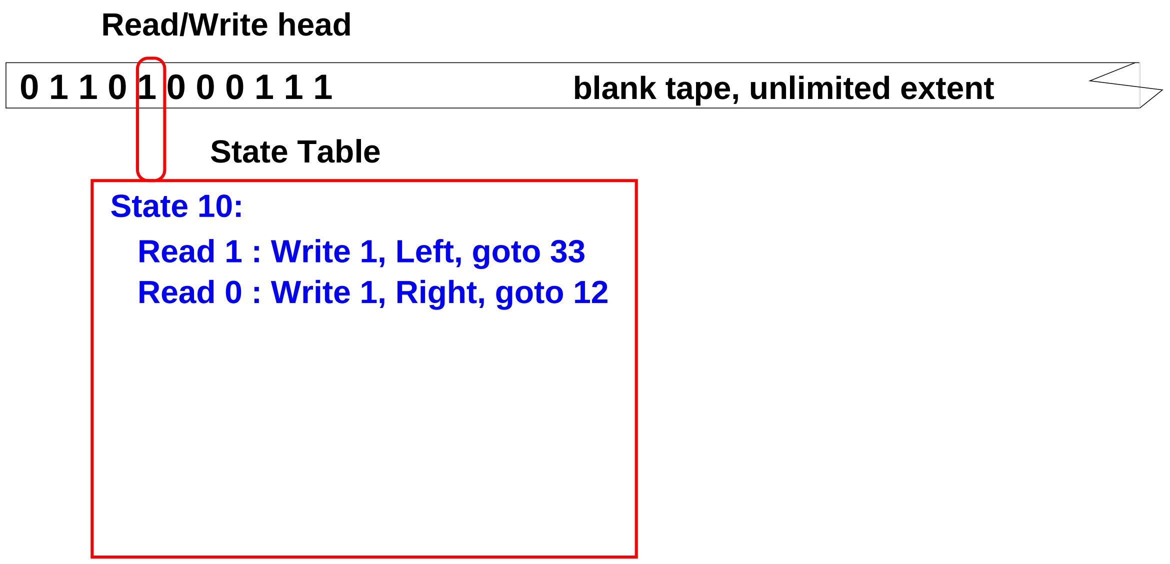


Is Mathematics Correct?

Hilbert's challenge - 23 unsolved problems in mathematics

- 2nd - can consistency of arithmetic be proved?
- 10th - is there an algorithm for Diophantine equations?
- *Principia Mathematica* - Russel and Whitehead. Attempt to derive all mathematical truths from axioms in a systematic, algorithmic way.
- Godel - proves that any system that includes arithmetic and logic must be incomplete or inconsistent - specifically, can not prove its own consistency. *Principia* is impossible.
- Godel does this by encoding arithmetic and logic as numbers, which allows the use of arithmetic to prove things about itself (How does this relate to Turing?)
- 10th problem (and *Principia*) - need to define what is an algorithm. Questions like - "What is a single step? How detailed do we need to be?"

The Turing Machine


Read/Write head
0 1 1 0 1 0 0 0 1 1 1
blank tape, unlimited extent
State Table
State 10:
Read 1 : Write 1, Left, goto 33
Read 0 : Write 1, Right, goto 12

Turing created the T.M. to define algorithm - Church-Turing Hypothesie. We can't prove this, but there are no counterexamples.

But how does this relate to Hilbert's 2nd problem?

Turing machine details

- Turing machines have finite states, finite alphabet (e.g 0,1)
- State table specifies the "hardware", tape is the memory, it is unlimited so we don't run out of space. Notice that we can't have random access, because addresses are some finite size and that limits the size of memory.
- Turing machines can read and write characters on the tape, move to different tape locations. Notic
- e that this is all that modern computers do - read and write 0 and 1 in memory.
- By reading and modifying tape, T.M. can do arithmetic and logic operations.

An example:

Alphabet is {0,1}; _ stands for blank tape

State	Read	Write	Move	New State
1	1	1	R	2
	0	0	R	4
2	1	1	R	3
	0	0	R	4
3	_	1	stop	TRUE
4	1	1	R	5
	0	0	R	5
5	_	0	stop	FALSE

This machine reads two characters on tape and computes "AND"

The Universal Turing Machine

- Turing Machine can be described in its own alphabet - we specify conventions for how to describe the table, separators for different sections of description, etc.
- We can define a Turing Machine that reads a T.M. description from its tape and executes that description - it is universal because we can run any other T.M.
- Note that the state table of a U.T.M. is fixed - it is like the hardware of a modern computer. The T.M. description is a program.
- Program and data can be on the same tape - like program and data are in memory of a modern computer.

Turing and languages

- During World War 2, Turing worked on encryption - his work earned him an O.B.E. This is his most recognized language related work.
- T.M. description in its own alphabet introduces machine language.
- Specific format of T.M. description depends on the state table of the U.T.M. that will run it - the machine language must match the machine hardware.
- T.M. can represent logic and arithmetic, therefore can carry out any mathematical operation.
- Therefore the programming language of T.M. can be used to formally describe mathematics (finally we see where Godel comes in!)

Turing and Godel

- Godel converted arithmetic and logic statements to numbers, so could use mathematical operations to express statements about mathematics.
- Godel showed that statements such as: “This statement is not provable” can be expressed - you have statements that are true only if they are not provable, and are false if they can be proved!
- Turing machines are capable of carrying out any mathematical operation - including logic and arithmetic. Therfore a result analogous to Godel’s theorem should be derivable with T.M.

Infinite loops

- If we can do logic and move to different memory (tape) locations, we can write a while loop on an arbitrary condition.
- Suppose we write "while(X) do Something" and X never becomes false?
- We can not limit loops to any arbitrary size - for example, limit loops to 1 trillion iterations, then we can not solve problems that require 1 trillion + 1.

The Halting Problem

- Given: A T.M. and its data. Can we devise an algorithm that tells us if the T.M. will enter an infinite loop?
- Since algorithms are Turing Machines, we want to write a particular T.M. - call it T-HALT - that takes any T.M. and its data as input.
- T-HALT(TM,data) answers YES if the given machine will halt on the data, NO if it will run forever.

The HALT Turing Machine.

- Given: A T.M. and its data. Can we devise an algorithm that tells us if the T.M. will enter an infinite loop?
- Since algorithms are Turing Machines, we want to write a particular T.M. - call it T-HALT - that takes any T.M. and its data as input and answers YES if the machine will halt, NO if it will run forever.
- Assume we can build T-HALT. We can use T-HALT to build a machine HALT that runs a T.M. on its own description as data. HALT always answers YES or NO.
- Any TM can be written on a tape, in some U.T.M. machine language. HALT just duplicates the description on the tape and runs T-HALT with the first description as machine and the second as data
- We can use HALT to build NOHALT - we take the part of HALT that answers YES and replace it with an infinite loop.
- NOHALT(T.M.) answers NO if T.M. does not halt, but goes into an infinite loop if T.M. will halt.
- What does NOHALT(NOHALT) do?

The limits of computability

- HALT always halts. HALT(NOHALT) must halt.
- Therefore NOHALT(NOHALT) does not halt.
- But if NOHALT acting on itself does not halt, then NOHALT(NOHALT) must answer NO and halt.
- CONTRADICTION - so our assumption that we can make T-HALT must be wrong, and there is no algorithm that solves the halting problem.
- The Halting Problem is the equivalent to Godel's incompleteness result.
- Just as Godel allows us to prove that some mathematical theorems are unprovable, the Halting Problem allows us to prove that some problems are uncomputable - because if we could solve them, we could solve the halting problem.
- Back to languages - we can prove that most interesting properties of languages can not be computed!