Issues in Multimedia Authoring Lecture 10: Limitations of Computers

Keith Douglas

Summary

- How do computers operate?
- What's a program? Input and output
- Mathematical proofs
- Reductio ad absurdum
- Non-computing example
- Halting problem and why it would be nice to solve
- The proof of the (recursive) unsolvability of the HP

How do computers operate?

- Computers apply a program to an input and produce an output
- Interactive programs most common these days; this makes no difference

What's a Computer Program? (I)

- For our purposes a computer program is just a list of precise, simple instructions
- Can be organized into *functions*, which can be looked at as "machines" that when given inputs churn out an output
- A function is just a way to reuse certain instructions repeatedly and give it a name (in principle they are dispensable)

What's a Computer Program? (II)

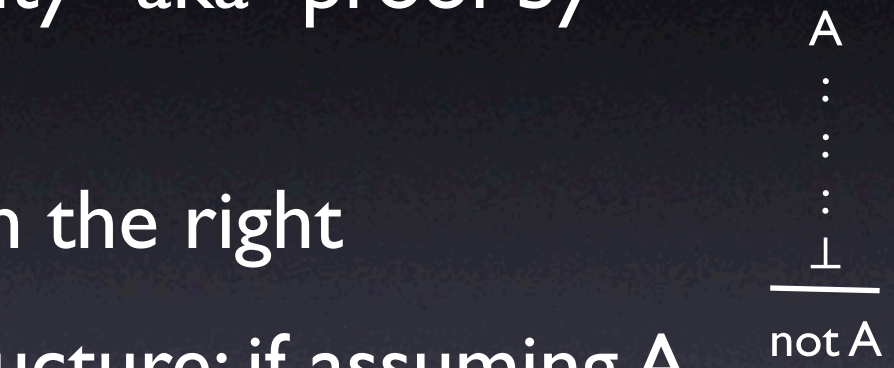
- We shall assume that given the same inputs the same output is produced, if there is one
- Functions need not always produce an output - think of a program to search through an infinite set looking for a value with a specific property
- Outputs and inputs are anything you might "get out" or "put into" a program
- We assume that any output of a function can be fed in as input in some fashion to another

Mathematical proofs

- Don’t have anything to do with the world directly
- Instead tell us rigorous consequences of our starting points; a generalization of logic

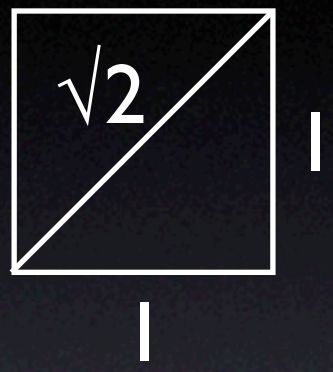
Reductio ad absurdum

- Technique of mathematical proof meaning "reduction to absurdity" aka "proof by contradiction"
- Has form depicted on the right
- Has the following structure: if assuming A leads to absurdity (contradiction), then A is false
- Assumes (as we do most of the time that if A is not true, it is false, and that statements which imply false statements are false)



RAA example

- **To prove**: the square root of 2 is not expressible as a fraction p/q.
- Assume that p/q is in lowest terms.
- Then $p = \sqrt{2}q$; hence $p^2 = 2q$.
- Hence p is even; so p^2 is even.
- So q is odd. Let $p = 2r$. Then $(2r)^2 = 2q^2$.
- Therefore $4r^2 = 2q^2$. Conclude $2r^2 = q^2$.
- So q is even. But we already concluded q is odd. Contradiction.
- So square root of 2 is not expressible as a fraction. QED.



The Halting Problem (I)

- Historically important example of a (recursively) unsolvable problem
- Its unsolvability first proved by A. Turing in 1936
- Prior to computers as we know them
- Turing's paper concerned the operations of an ideal clerk
- Since then it has been taken to apply to any computer as well; this extension rigorously formulated and proved for wide class of computers by R. Gandy and W. Sieg

The Halting Problem (II)

- It asks: can we write a computer program that will, when fed a programs input tell us yes or no whether or not that program will halt on a given input or, instead, loop forever?
- Nice in itself - would be nice if we could tell whether a program is looping or will stop (think of the Firefox message if you have seen it)
- If we could solve this problem we could also solve dead code problem - useful for programmers

The Halting Problem (III)

- **To prove:** The halting problem is (recursively) unsolvable.
- Assume (for reductio) that there is such a function. Call it Halt. Halt takes two inputs, a function (f) and an input (x) for the program.
- So intuitively, what Halt is supposed to do is look at f and x and see whether or not f would stop at some point when run on input x and output true if it does, false if it loops forever.

The Halting Problem (IV)

- Note that Halt has to be general. If all we want is one that will tell us whether it will loop, for example and not worry if it correctly gives us an answer for non-looping, just use a function that always outputs false.
- Note also that we cannot "guess" false and the "change" to true if f(x) eventually stops because that could take any amount of time

The Halting Problem (V)

- If Halt is possible, then surely the following "sneaky" function is possible. Write a function Diag which, given an input x it runs Halt(Diag, x) and loops forever if the result is true. [Diag calls Halt which itself passed Diag as a input. Remember that Halt takes a function and an input.] Otherwise, it returns 0.
- Now consider what happens if we try running Halt (Diag, 0).

The Halting Problem (VI)

- Halt (Diag, 0) either outputs true or false, per our assumption.
- Suppose first it is true. Then Diag(0) halts.
- But Diag(0) halts only if Halt(diag, 0) is false, per the definition of diag. But we already assumed that Halt(Diag, 0) is true. Contradiction.

The Halting Problem (VII)

- So try the other case; assume Halt(Diag, 0) is false.
- Then Diag (0) doesn't halt.
- But Diag (0) doesn't halt only if Halt (Diag, 0) outputs true. Again we have a contradiction.
- So either way we have a contradiction.
- So there is no such program Halt.

Slight Aside 1: "Recursive"?

- This is a technical term - it refers to a certain class of mathematical functions; loosely speaking ones equivalent to computer programs as we understand them
- Since we are interested in limitations of our computers, we ignore the possibility that there is another way, not a program, to check for halting
- That there is such a way is extremely unlikely; most computer scientists believe there isn't. We won't get into why in this course

Slight Aside 2: Diag?

- "Why is the weird function called Diag?"
- If you ever take courses in logic or computability theory you will learn that the proof we just saw is an example of a "diagonal argument".
- So "Diag" is short for "diagonal"; the name is typical.

TTQ

- The (fictional) character Crab in the book *Gödel, Escher, Bach: an Eternal Golden Braid* can apparently do something like Halt by looking at programs (or statements of integer arithmetic, which amount to the same thing for our purposes) as musical scores, playing them, and then listening to the tune that results. Based on what you have seen, how plausible is Crab's ability?