Limits of Computation

2 - Effective Procedures & Algorithmic Problems

Bernhard Reus

Last time

- we met our first non-computable (undecidable) problem: Hilbert's *Entscheidungs*problem.
- We motivated why we are interested in the limits of computability.
- We've seen a problem for which a brute-force solution is intractable (TSP).

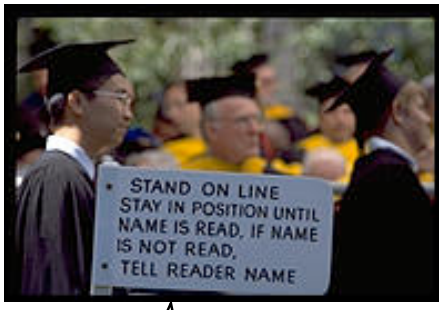
First Computability Question

Can every "problem" be solved, i.e. "computed" by some "program"?

need to sort out first the terms in "quotes"

Effective Procedures

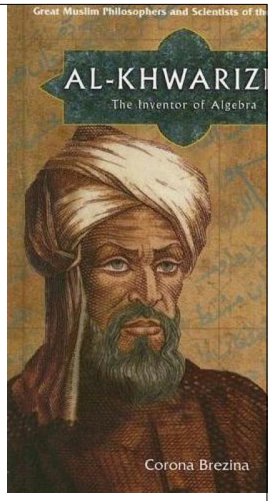
- in this lecture we fix the notion of
  - program ("effective procedure")
  - problem
  - computable problem



a procedure.

Algorithms

- algorithm: named (like algebra) after the ninth century Persian mathematician *Al-Khwarizmi.*
- Euclid's famous algorithm for "greatest common divisor" (gcd)
- algorithm is an "effective method"



Euclid ca 330-275 BC

Effective Procedures, Algorithms

- "*effective procedure*" or "*effective algorithm*" replaces our earlier "*program run on a computer*";
- *abstracting away* from concrete hardware; We can define it as we wish ...
- ... if we can justify the choice later.

Effective Procedures?

- What does **effective procedure** mean to you? Discuss!

effective
1 successful in producing a desired or intended result: *effective solutions to environmental problems*
(Oxford Dictionaries)

Effective Procedure

Alan Turing : 1936

The naming goes back to *Alan Turing*: "*A function is said to be effectively calculable if its values can be found by some purely mechanical process. . . . We may take this statement literally, understanding by a purely mechanical process one which could be carried out by a machine*"

Turing then defined a specific type of machine, now called "Turing machines", thus defining formally a notion of computation

Alan Turing : 1936

"Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, i.e., on a tape divided into squares. I shall also suppose that the number of symbols which may be printed is finite. If we were to allow an infinity of symbols, then there would be symbols differing to an arbitrarily small extent. The effect of this restriction on the number of symbols is not very serious. It is always possible to use sequences of symbols in the place of single symbols. ... The difference from our point of view between the single and compound symbols is that the compound symbols, if they are too lengthy, cannot be observed at a glance. .. We cannot tell at a glance whether 9999999999999999 and 999999999999999 are the same.

Effective Procedure

Copeland gives the following definition:

" 'Effective' and its synonym 'mechanical' … do not carry their everyday meaning. A method, or procedure, M, for achieving some desired result is called 'effective' or 'mechanical' just in case

1. M is set out in terms of a **finite number of exact instructions** (each instruction being expressed by means of a finite number of symbols);
2. M will, if carried out without error, always produce the desired result in a **finite number of steps;**
3. M can (in practice or in principle) be carried out by a human being unaided by any machinery save paper and pencil;
4. M demands **no insight or ingenuity** on the part of the human being carrying it out. "

What is not Effective?

- Can you give an example of a procedure that is not "effective"?

effective
1 successful in producing a desired or intended result: *effective solutions to environmental problems*
(Oxford Dictionaries)

Problem?

What's Our Problem?

- a general, **uniform** class of questions
  - *uniform* means there is some clearly defined "parameter" (input).
- each of which can be given a **definite and finite** answer
  - *answer* means there is some clearly defined solution.
- solving it means *providing a function* or *deciding membership in a set*

i.e. we need to be able to describe a finite solution to the problem

these are called algorithmic problems

Problem Examples

Given any tree *t*, what is its height?

defines function "height"

uniform

Given any number *n*, is it even?

decision problem (answer yes/no)

uniform

Problem Examples

uniform

Given any *graph G*, and *vertices s* and *t* what is the shortest path in *G* from *s* to *t*?

optimisation problem (minimal/maximal solution)

special kind of "function" problem

Computing Solutions to Problems

algorithmic problems

Definition Provided a certain choice of effective procedures P, a (function or decision) problem is called ***P-computable*** if, and only if, its solution can be computed (calculated) by carrying out a specific such effective procedure in P. A decision problem that is P-computable is also called ***P-decidable***.

- so programs (effective procedures) are solutions to computable/ decidable problems
- we drop the "*P*" in *P-computable* if the language is clear from the context.

Reminder on Logic

Formal notation

- we will need to be formal in this module, otherwise there is no point in talking about things like computability;
- we use some mathematical language (but this does not mean we do maths) for:
  - sets and relations
  - functions (partial and total), polynomials etc.
  - basic probability theory (towards the end)

will be introduced/ recapitulated in exercises and when needed

- and to denote logical arguments in proofs: *and, or, not, iff, forall, exists*

"if, and only if"

- it means: logical equivalence
- we will use this often and sometimes abbreviate it "iff"
- "A iff B" = "(A implies B) and (B implies A)"
- Consider examples:

  A = "Rain" B = "Wet Road"

  A ="divisible by 3" B = "sum of digits divisible by 3"

Sets

subset relation

forall

"in" (elementhood)

$$S_1 \subseteq S_2 \iff \forall x \in T.\ x \in S_1 \Rightarrow x \in S_2$$

$$S_1 = S_2 \iff \forall x \in T.\ x \in S_1 \Leftrightarrow x \in S_2$$

or (disjunction)

and (conjunction)

$$x \in S_1 \cup S_2 \iff x \in S_1 \vee x \in S_2$$

$$x \in S_1 \cap S_2 \iff x \in S_1 \wedge x \in S_2$$

$$x \in S_1 \setminus S_2 \iff x \in S_1 \wedge \neg(x \in S_2)$$

iff (if, and only if)

Set Notation

$\{e_1, e_2, \ldots, e_n\}$ — finite set with n elements

$\{ x \in S \mid P(x) \}$ — set of those elements in S that have property P

END

Next time:
`WHILE` programs in detail: Syntax and Semantics