Introduction

Part I - Computability-Turing Machines 10 lectures

II - Lambda Calculus 5 lectures

Part III - Complexity - Introduction to Complexity 5 lectures

Coursework - I Turing Machines due 10 Nov
- II Lambda Calculus: details later

Alan Turing - biographical notes

1912 - Born in London, father in colonial service in India. Fostered in Sussex

1921 - 31 Boarding school, prep then Sherborne

He had a passion for science & experimentation

1931-4 - Kings College Cambridge, Mathematics
--> 39 then research at Cambridge & Princeton

1938 started work on the Enigma enciphering machine
39 formed the Bletchley Park Code and Cypher team - led to development of Colossus machines with hundreds of staff ..success by 1941, giving early warning of German moves but needing cover to keep deciphering secret.

1946 - at NPL and Univ of Manchester, competing with other projects for funds to develop ideas for new computers.

- a fine athlete, narrowly missing Olympic selection in 1948

Socially, he was openly gay at Cambridge, despite illegality; post-war, he holidayed in the Mediterranean..risked blackmail and attracted the attention of the security services.

At Manchester in 1952 he was robbed by a youth he had taken home, and on reporting the crime was charged with gross indecency.. convicted and given compulsory hormone treatment (it was a very different world)
By 1953..working on math. models of biological systems..
1954..died fron cyanide poisoning..suicide verdict disputed by family who said he was always careless during experiments..

See: www.turing.org.uk

Start with a paradox:

define: "the least number not definable by an English sentence with fewer than 100 letters"

a dedication: "this book is dedicated to all those people who haven't got a book dedicated to them"

define: "the set of all sets which are not members of themselves" (Russell's Paradox)

The problem? - self-reference within the definition

and in programming languages

-which permit self reference (can self-modify code)..

are there similar paradoxes?

A Programming Paradox

Given: a high level imperative programming language.

A program: a character string containing letters, numbers, punctuation.

Now **list all syntactically correct programs** in alphabetic order:

P_1, P_2, P_3,....

all programs appear in this list.

each program outputs a string of characters (which may be empty)

suppose this o/p string is binary (and if not, encode it in binary).

Now define the program P as follows:

Program P

```
repeat forever
generate the next program P_n
run P_n as far as the nth output bit
if P_n terminates or prompts for input before
  the nth bit has been output
   then output 1
   else if the n^th bit of P_n's output is 0
      then output 1
         else if the n^th bit of P_n's output is
  1
                     then output 0
 end if
 end repeat
```

Use **auxiliary program** to **generates all text strings** in alpha order & call an **interpreter to check syntax** - if no errors we have the next P

The Operation of P

at each iteration the interpreter runs P_n
it halts if:

P_n halts or
P_n prompts for input or tries to read a file or
P_n has produced n bits of output.

Transcribed formally into a programming language, P is a valid program so $P=P_i$ for some i.

Suppose $P=P_7$ ie. P is generated on the 7th iteration of the loop in P.

Continued..

continued..**The Operation of P**..

if P_7 halts or
P_7 prompts for input } **P outputs 1**

if P_7 outputs…bit 7 = 0 } **P outputs 1**

if P_7 outputs…bit 7 = 1 } **P outputs 0**

but $P=P_7$.. a contradiction

So $P \neq P_7$

continued..

continued..The Operation of P

In general $P \neq P_n$ for any n as the n^{th} output bit is different
ie. P is not in our list of programs

BUT the list was defined to contain **all** programs..a contradiction..

..we want our programming to exclude such paradoxes..

..where is the problem?

We see later, using a simple model of a computer, that this is a general problem, independent of the power of the computer or of the programming language we use

ie. **P cannot be "patched"** - we have an **unsolvable** problem.

Other **unsolvable problems**:

- Checking mechanically whether an arbitrary program will halt on a given input (the **Halting Problem**)

- Printing out all true statements of arithmetic and no false ones (**Goedel's Incompleteness Theorem**)

- Deciding whether a given sentence of first-order logic is valid or not (**Church's Theorem**)

All these problems are **unsolvable** by a computer as we will see later in the course…

Summary

Computability - **what problems are solvable?**

- **are there unsolvable problems?**

- **if so, how do we prove this?**

Part I (Turing Machines) and Part II (Lambda Calculus)

-**study different (but equivalent) approaches to this problem.**