Introduction to Complexity

Week 5

The Quest for Intelligence

Alexandra Silva & Henk Barendregt

Although animals are not computers

for the simplest actions (like catching a fly) computations are needed

Nature has evolved a chemo-electrical computational model

Neural Net Synapse

These are not programmed using intention, but by trial and error

They run in parallel, and can be remarkably efficient

They need to be trained (by evolution via genes, taking a long time)

These from an approximation (an abstraction) of the NN in the brain

Netwerks are *taught* e.g. *supervised* or *trial-and-error* to fulfill a certain task

Equivalent to ordinary computability, but with different efficiency

Metaphor of the plastic tablecloth

This is a different computational model

Our complexity does not refer to this

A faster way is to compute using *rules* (mental programs, memes)

Many *numerical problems* can be answered by computing

"What is the area of a circle with radius $4\,m$?"

Answer: $4^2\pi\,m^2 = 50.2654824\,m^2$

Also many *qualitative problems* may be answered by computing

"Are points $A = (x_1, y_1)$, $B = (x_2, y_2)$ and $C = (x_3, y_3)$ in $\mathbb{R}^2$ collinear

i.e. do they lie on a straight line?"

Answer: if and only if $(x_1 - x_3)(y_2 - y_3) = (x_2 - x_3)(y_1 - y_3)$

Also for *daily life actions* or in the *performing arts* and *sports* one needs computations

Musicians can train themselves to be accurate to less than 100ms

This type of learning comes from mental programs using neural nets

Bartok: Sonata, Ivry Gitlis violin

Leibniz (1646-1716) conjectured:

Properly stated problems can be answered by computing (calculemus!)

He wanted to construct:

a universal language L for stating problems precisely

a machine M answering all these problems by computing

The first question Leibniz wanted to ask to such a machine is said to be

"Does God exist?"

Quite daring around 1700 to ask this question to a machine!

Science Fiction by Fredric Brown (1908-1972): *Answer* [1954]

Yes, now there is a God.

Restricted to mathematical problems, there is such a universal language L

Restricted to *numerical problems* there is such a machine M

A computer with a software package like Mathematica or Maple

However, Turing [1936]:

For *qualitative mathematical problems* M is impossible

E.g. "Are there twin primes (differing by 2 like $\{11, 13\}$) greater than googolplex $10^{10^{100}}$?"

is an example of such a qualitative problem

Turing [1936]: For *qualitative mathematical problems* M is impossible

Turing's negative answer about the qualitative problems

made possible the positive answer for the numerical problems

There is also a positive answer for a subclass of the qualitative problems

(symbolic computing, e.g. "Are points A, B, C collinear?")

So we have

numerical, symbolic, and qualitative mathematical problems

the latter involving quantifiers "for all" ($\forall$), "there exists" ($\exists$)

These have to do with mathematical infinity

Still we can answer some problems about infinity, by proving

Euclid: There are infinitely many primes: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, ...

Hilbert believed that eventually everything can be known (by proving or computing)

"Instead of the sad 'We don't know', my attitude is: 'We *have* to know, we *will* know'!"

Let $\mathbf{PA} \vdash \ldots$ denote provability in arithmetic as axiomatized by Peano

We suppose that $\mathbf{PA}$ is consistent. We are interested in the set

$$T_{\mathbf{PA}} = \{P \mid \mathbf{PA} \vdash P\}$$

One year after Hilbert's statement Gödel proved in [1931]

Theorem (*Incompleteness* of Arithmetic)

There is a statement G such that neither $G \in T_{\mathbf{PA}}$ nor $\neg G \in T_{\mathbf{PA}}$

Turing proved in [1937]

Theorem (*Undecidability* of Arithmetic)

There is no algorithm deciding whether for a $P \in L_{\mathbf{PA}}$ one has $P \in T_{\mathbf{PA}}$

Proposition. Undecidability implies incompleteness

Proof. Suppose $\mathbf{PA}$ is complete: always either $P \in T_{\mathbf{PA}}$ or $\neg P \in T_{\mathbf{PA}}$.
Then for P we can decide: just try to prove P or try to refute P
This is an enumerable process. By completeness we can wait until one succeeds. ■

How did Turing prove his negative result?

How could it have so much positive spin off?

Turing did the following

- ! Gave a well-motivated analysis of computability via *Turing Machines* (TM) an idealized class of machines (with infinite memory)
- !! Constructed a *universal Turing Machine* UTM that can simulate any TM via software (*programs*)
- ! Formulated the *halting problem* (HP) that cannot be decided by any TM argument like liar paradox
- ! Concluded qualitative mathematical problems cannot be decided by any TM as the HP is one of them

Therefore Leibniz's ideal cannot be fulfilled for mathematical problems
only be approximated, leave alone for philosophical problems

YouTube

`www.youtube.com/watch?v=E3keLeMwfHY`

Based on introspection: computing is done in *discrete* steps

A computational process goes from input to output (action)

Let M be a TM. It has

a finite set I of input

a finite set A of actions

The machine M transforms an input (i in I) into an action (a in A)

$$i \xrightarrow{M} a$$

As the same input may give rise at different times to different actions

Turing introduced a set S of *states* indicating how to react to an input

$$(i, s) \xrightarrow{M} (a, s')$$

The machine, given input and state, may choose action and (a new) state

M is given as a finite table of transitions $(i, s; a, s')$

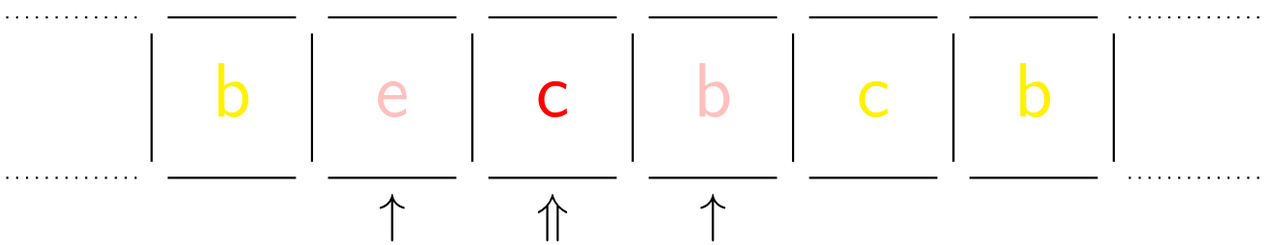
M and has a two-sided infinite memory tape

with cells containing nothing (a blank) or a symbol $i \in I$

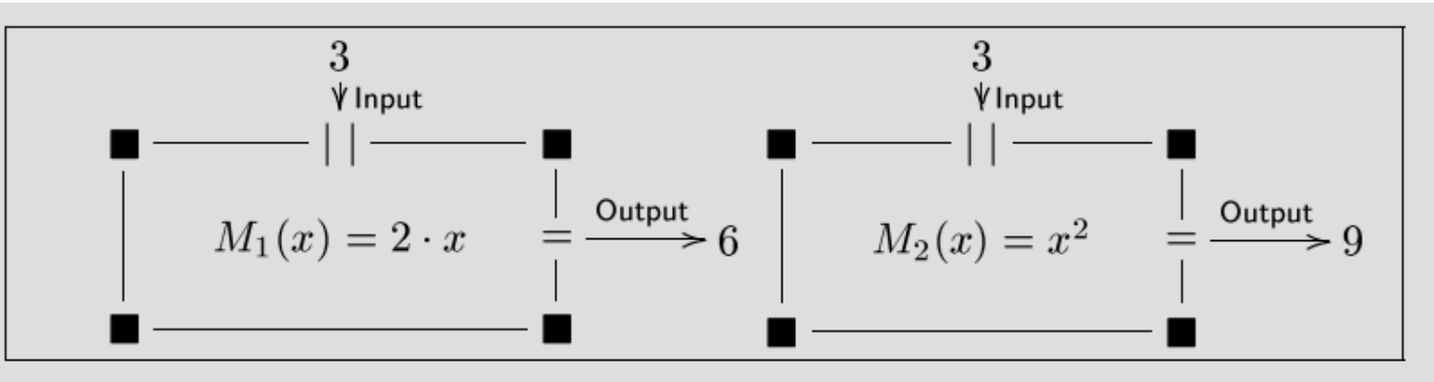
and also has a Read/Write device ('head') placed on one of the cells

An (*instantaneous*) *configuration* (at a given moment)

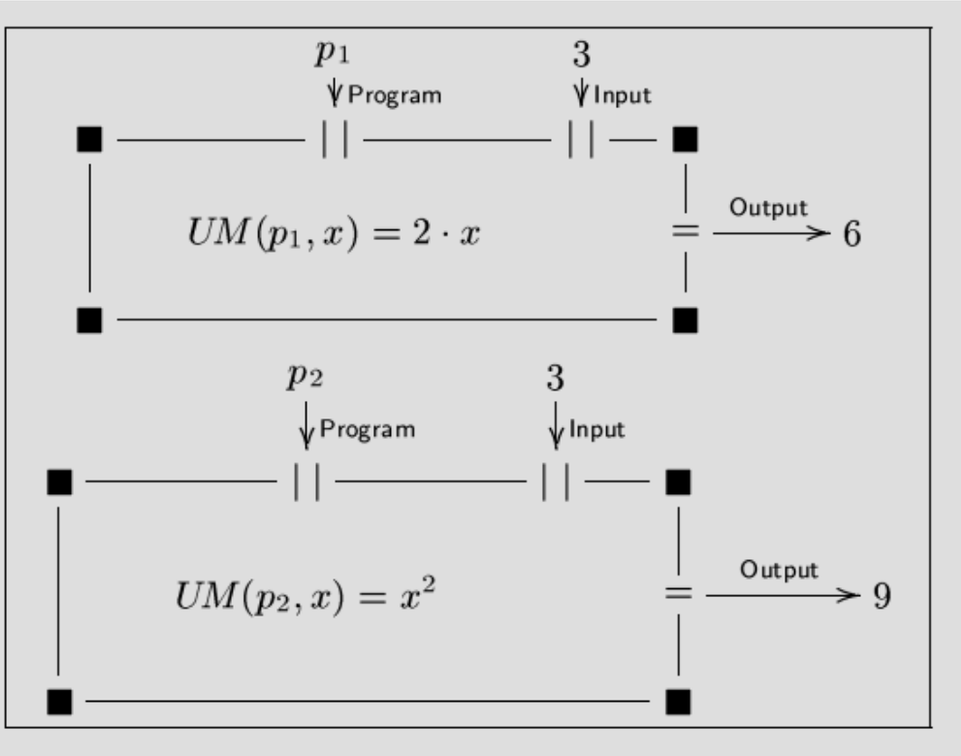
is the information on the tape & the position of the head



⇑		position of read/write head	Actions	
↑		potential next position of head	L	⇑ goes left
red	letter	present focus	R	⇑ goes right
pink	letters	possible next focus	W(c')	overwrite c by c'
yellow	letters	potential focus		



Two ad hoc machines M_1, M_2



UM simulating M_1, M_2

Write $M(n)\downarrow$ (resp. $(M(n)\uparrow)$ if machine M with input n does (doesn't) halt

Theorem. The problem $\mathrm{UM}(p,n)\downarrow$ is non-computable (*undecidable*)

Proof. Suppose $\mathrm{UM}(p,n)\downarrow$ is computable: then there is a machine H such that

$$\begin{aligned} \mathrm{UM}(p,n)\downarrow &\iff H(p,n)=1 \\ \mathrm{UM}(p,n)\uparrow &\iff H(p,n)=0 \end{aligned}$$

Modify H into H' making $H'(p,n)$ run forever if $H(p,n)=1$. Then

$$\begin{aligned} \mathrm{UM}(p,n)\downarrow &\iff H'(p,n)\uparrow \\ \mathrm{UM}(p,n)\uparrow &\iff H'(p,n)\downarrow \end{aligned}$$

Now define $D(n)=H'(n,n)$. It has a program p_D as UM is universal:

$$D(n)=\mathrm{UM}(p_D,n) \text{ for all } n$$

We get a contradiction

$$D(p_D)\downarrow \iff \mathrm{UM}(p_D,p_D)\downarrow \iff H'(p_D,p_D)\uparrow \iff D(p_D)\uparrow \blacksquare$$

$\mathrm{UM}(p,n)\downarrow \iff$ there exists a number k such that after k cycles $\mathrm{UM}(p,n)$ halts

The 'quantifiers'

exits ($\exists$)

for all ($\forall$)

ranging over the infinite set $\mathbb{N}$ usually cause undecidability

Theorem. Let $f : \mathbb{N} \to \mathbb{N}$ be computable. Define

$$\begin{aligned} H_f(n) &= \mathrm{UM}(n,n)+1 && \text{if } \mathrm{UM}(n,n) \text{ stops in } \leq f(n) \text{ steps;} \\ &= 0 && \text{otherwise.} \end{aligned}$$

Then H_f is computable. For each program p of H_f one has $T_p(p) > f(p)$

Proof. Let p be an algorithm for H_f. Then for all $n \in \mathbb{N}$

$$H_f(n) = \mathrm{UM}(p,n)$$

If $\mathrm{UM}(p,p)$ stops in $\leq f(p)$ steps, then $\mathrm{UM}(p,p) = H_f(p) = \mathrm{UM}(p,p)+1$

Therefore $H_f(p) = \mathrm{UM}(p,p) = 0$ takes more than $f(p)$ steps. ■

This function H_f is such that for every program p the cost of computing H_f via this program at p is more than $f(p)$. We can do better

Thm. Let f be a computable function. There exists a function $G = G_f$ such that for all algorithms p for G one has $T_p(n) > f(n)$ for all $n \geq p$

Notation. (i) Sequences of numbers like $a_0, a_1, \ldots, a_n$ can be coded as

$$\langle a_1, \ldots, a_n \rangle = 2^{a_0+1} \cdots p_n^{a_n+1}, \text{ where } p_n \text{ is the } n\text{-th prime}$$

and we can find computably the components

$$\langle a_1, \ldots, a_n \rangle_k = a_k \text{ for } 1 \leq k \leq n$$

$$\text{(ii)} \quad \begin{aligned} \mathrm{UM}(p, n, s) &= \mathrm{UM}(p, n), && \text{if it stops in} \leq s \text{ steps} \\ &= 0 && \text{otherwise} \end{aligned}$$

Proof. Define $G(n) = \langle \mathrm{UM}(0, n, f(n))_0 + 1, \ldots, \mathrm{UM}(n, n, f(n))_n + 1 \rangle$

Let p be a program of G: for all n one has $\mathrm{UM}(p, n) = G(n)$. Then for $n \geq p$

$\mathrm{UM}(p, n)_p = G(n)_p = \mathrm{UM}(p, n, f(n))_p + 1 = \mathrm{UM}(p, n)_p + 1$

if $T_p(n) \leq f(n)$, impossible; so $T_p(n) > f(n)$ ■

It is easy to improve this so that $T_G \in \Omega(f)$

We will consider only (classical) propositional logic

Suppose the following is given

(i) If it is autumn or it is misty, then the trees get leaves

(ii) If the trees get leaves, then it rains and there is no sun

(iii) There is sun

(iv) If it is not autumn, then it is hot

Conclude: *It is hot and it is not misty*

The truth is independent of the meaning of the words

The words are chosen a bit counter-intuitive

Proof. Let us first introduce abstract names (to avoid confusion)

$$\begin{array}{rrcl} (i) & A \vee M & \rightarrow & TL \\ (ii) & TL & \rightarrow & R \,\&\, \neg S \\ (iii) & & & S \\ (iv) & \neg A & \rightarrow & H \end{array}$$

We must show $H \,\&\, \neg M$

From

$$\begin{array}{rlcl} (i) & A \vee M & \to & TL \\ (ii) & TL & \to & R \,\&\, \neg S \\ (iii) & & & S \\ (iv) & \neg A & \to & H \end{array}$$

conclude

$$H \,\&\, \neg M$$

$$\dfrac{\dfrac{\dfrac{\dfrac{S}{\neg(R \,\&\, \neg S)}\,5}{\neg TL}\,(ii),1}{\neg(A \vee M)}\,(i),1}{\dfrac{\neg A}{H}\,(iv)}\,2,3a \qquad \dfrac{\dfrac{\dfrac{\dfrac{S}{\neg(R \,\&\, \neg S)}\,5}{\neg TL}\,(ii),1}{\neg(A \vee M)}\,(i),1}{\neg M}\,2,3b$$

$$\dfrac{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}{H \,\&\, \neg M}\,6$$

Steps used

$$\dfrac{A \to B}{\neg B \to \neg A}\,1 \qquad \dfrac{\neg(A \vee B)}{\neg A \,\&\, \neg B}\,2 \qquad \dfrac{A \,\&\, B}{A}\,3a \qquad \dfrac{A \,\&\, B}{B}\,3b \qquad \dfrac{A}{\neg\neg A}\,4$$

and since $(A \,\&\, \neg B) \to \neg B$, one has $\neg\neg B \to \neg(A \,\&\, \neg B)$, by 1, so

$$\dfrac{B}{\neg(A \,\&\, \neg B)}\,5$$

Also we have $\dfrac{A \quad B}{A \,\&\, B}\,6$

Propositions (abstract grammar)

$$\texttt{var} \quad ::= \quad \texttt{p} \mid \texttt{var}'$$

$$\texttt{prop} \quad ::= \quad \texttt{var} \mid \texttt{prop} \ \& \ \texttt{prop} \mid \texttt{prop} \vee \texttt{prop} \mid \texttt{prop} \to \texttt{prop} \mid \neg\texttt{prop}$$

The variables are $p, p', p'', \ldots$ but we write $x, y, z, \ldots$

For propositions we use $P, Q, R, \ldots$

A *valuation* is a map $\mathtt{v} : \mathtt{var} \to \{0, 1\}$

Such a v can be extended to $\overline{\mathtt{v}} : \ : \mathtt{prop} \to \{0, 1\}$

$$\begin{aligned}
\overline{\mathtt{v}}(x) &= \mathtt{v}(x) \\
\overline{\mathtt{v}}(\neg P) &= 1 - \overline{\mathtt{v}}(P) \\
\overline{\mathtt{v}}(P \ \& \ Q) &= \overline{\mathtt{v}}(P).\overline{\mathtt{v}}(Q) \\
\overline{\mathtt{v}}(P \vee Q) &= \max\{\overline{\mathtt{v}}(P), \overline{\mathtt{v}}(Q)\} \\
\overline{\mathtt{v}}(P \to Q) &= \max\{1 - \overline{\mathtt{v}}(P), \overline{\mathtt{v}}(Q)\}
\end{aligned}$$

A proposition P is *valid* iff $\overline{\mathtt{v}}(P) = 1$ for all $\mathtt{v}$

E.g. $p \vee \neg p$ is valid and $p \vee q$ is not valid

A proposition P is *satisfiable* if for some v one has $\overline{\mathrm{v}}(P) = 1$

E.g. $p \vee q$ is satisfiable, but $p \,\&\, \neg p$ not

Definition Let $\Gamma = \{P_1, \ldots, P_n\}$. We say Γ *entails (implies)* Q, notation

$$\Gamma \models Q$$

if for all v such that $\overline{\mathrm{v}}(P_1) = \ldots = \overline{\mathrm{v}}(P_n) = 1$ one has $\overline{\mathrm{v}}(Q) = 1$

The following is not difficult to show

Proposition. $\{P_1, \ldots, P_n\} \models Q \iff (P_1 \,\&\, \ldots \,\&\, P_n) \to Q$ is valid

So validity is an interesting notion

Proposition. $\neg P$ is valid iff P is not satisfiable

(ii) P is valid iff $\neg P$ is not satisfiable

So satisfiability is an interesting notion

Let us check that $\neg(P \to Q) \to P$ is valid

P	Q	$P \to Q$	$\neg(P \to Q)$	$\neg(P \to Q) \to P$
0	0	1	0	1
0	1	1	0	1
1	0	0	1	1
1	1	1	0	1

In particular $\neg(P \to Q) \models P$

Proposition. There are algorithms in $O(2^n)$ to test validity and satisfiability

Proof. Try all the 2^n many $\mathtt{v}$, where n is the number of atoms in a `prop`.

Open problem. (Worth one million dollar)

Can we do better?

It may be the case that satisfiability is polynimial:

we need only one $\mathtt{v}$ and checking that it works is polynomial

This is the famous P=NP problem

NP means *non-deterministic polynomial* (I'd prfer to call it 'Parallel Polynomial')