BBC: Chess champion loses to computer
(Tuesday, 5 December 2006, 22:10 GMT)

- Deep Fritz, a chess-playing computer, has beaten human counterpart world chess champion Vladimir Kramnik in a six-game battle in Bonn, Germany.
- Deep Fritz won by four points to two, after taking the last game in 47 moves in a match lasting almost five hours.

In 1948, Turing, working with his former undergraduate colleague, Champernowne, began writing a chess program for a computer that didn't yet exist. In 1952, lacking a computer powerful enough to execute the program, Turing played a game in which he simulated the computer, taking about half an hour per move. The game was recorded; the program lost to Turing's colleague Alick Glennie, although it is said that it won a game against Champernowne's wife.

Quote from a software magazine:

> "Put the right kind of software into a computer,
> and it will do whatever you want it to.
> There may be limits on what you can do with the machines themselves,
> but **there are no limits on what you can do with software**"

Really hard decision problems

Halting problem Is there an algorithm that, given **any** (Java, C++, etc.) program will eventually stop running on a given input **?**

Remember **Collatz**

```
c = n;
while (c != 1) {
    if (c % 2 == 0) { c = c / 2; }
    else { c = 3*c + 1; }
}
```

Diophantine equations Is there an algorithm that can decide whether any given Diophantine equation (a polynomial equation such as $x^n + y^n = z^n$) has a solution in integer numbers **?**

Equivalence of CFGs Is there an algorithm that can decide whether any two context-free grammars define the same language **?**

. . .

The Halting Problem

Decide, for any given Turing machine M and input word w, whether M will eventually halt on input w

Is there a general algorithmic method solving this problem?

If yes, then — according to the Church–Turing thesis — there should be a Turing machine solving it.

The Halting Problem is unsolvable

There is **no Turing machine** solving the halting problem:

there is no Turing machine H such that, given any input of the form $\langle M\rangle␣\langle w\rangle$,

$$\left[\begin{array}{ll} H \text{ halts and outputs } 1, & \text{if the Turing machine } M \\ & \quad \text{halts on input } w; \\ H \text{ halts and outputs } 0, & \text{if the Turing machine } M \\ & \quad \text{does not halt on input } w. \end{array}\right]$$

In the light of the Church–Turing thesis we can conclude:

there is no algorithmic method solving the halting problem

Understanding this ‘unsolvability’

Before presenting an argument for the unsolvability of the Halting Problem, let us try to understand what it really means.

For example: the Universal Turing machine U does simulate

every Turing machine M on every possible input w.

Isn’t it the case that U actually solves the Halting Problem?

NO!

On the one hand, if M halts on input w then U will halt on input $\langle M\rangle\llcorner\!\lrcorner\langle w\rangle$, and outputs the code of M’s output. Then it wouldn’t be too hard to extend U with instructions saying ‘erase the tape and write 1’. But this fancy new machine would still go on forever giving us no answer in case the input $\langle M\rangle\llcorner\!\lrcorner\langle w\rangle$ is such that M never halts on w.

So U is no help in **deciding** whether any M halts on any w or not.

Understanding this ‘unsolvability’ II

The unsolvability of the Halting Problem in no way implies that there may not be **some** circumstances when it is possible to decide whether **some** Turing machine M will halt on an input string w.

Take for example the Turing machine M_{eraser}

Or the \`copying machine'

Or the machine computing the function $f(n) = n + 1$

Surely in each case we can decide whether the machine in question halts on an input or not.

The point is that for different machines and inputs we may need
different methods for deciding whether they halt or not

And yes, there exist cases when we simply **don’t know** the answer.

Remember the program **Collatz**?

Understanding this ‘unsolvability’: summary

The unsolvability of the Halting Problem means:

Whatever general method we suggest to solve it, there will always be a Turing machine and an input which are out of the scope of our method:

- either our method does not give an answer,
- or the answer is incorrect.

In other words, **there is no** completely general method deciding correctly **ALL** cases.

How to show unsolvability?

So far we have discussed **what** the unsolvability of the Halting Problem means.

Let's now turn to discussing **how** to show this unsolvability.

How to show that there is no method . . . ?

- We try to find a method. Come up with a suggestion. It turns out to be wrong. Then try another one. It again turns out to be wrong. And so on. After how many attempts shall we conclude that there is no such???

- We will argue differently, using **proof by contradiction**

  or **reductio ad absurdum** :

  (1) Assume, hypothetically, that there is a Turing machine H solving the Halting Problem.

  (2) Then show that this assumption leads us to an absurd, impossible situation, that is, to a **contradiction**

Can a Turing machine run on 'itself'?

As a preparation for showing the unsolvability of the Halting Problem, let us examine the following question:

Can we give its own code $\langle M \rangle$ as an input to a Turing machine M?

Recall from that the code $\langle M \rangle$ of a Turing machine M is always a word using the 9 symbols $\boxed{\mathbf{a} \quad \mathbf{q} \quad \mathbf{0} \quad \mathbf{1} \quad (\quad) \quad , \quad \Rightarrow \quad \Leftarrow}$

– So yes, **if** the tape alphabet of M contains these symbols
then $\langle M \rangle$ **is a possible input for** M.

– And no, if the tape alphabet of M does not contain (some of) these symbols, then $\langle M \rangle$ is not a sensible input for M.

We will be interested in **'smart machines'**, that is, in those Turing machines M that **can** run on input $\langle M \rangle$
(for instance, the Universal Turing machine is such a 'smart' machine).

The Liar Problem

Before showing the unsolvability of the Halting Problem,

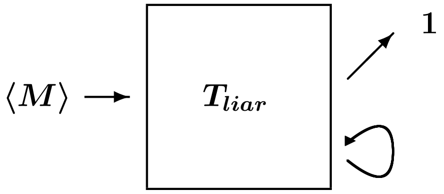
let us consider another problem:

The Liar Problem:

Is there a Turing machine T_{liar} which does the following:

given the code $\langle M \rangle$ of **any** 'smart' Turing machine M as input to T_{liar},

$$\left[\begin{array}{ll} T_{liar} \text{ halts and outputs } 1, & \text{if } M \text{ does not halt on } \langle M \rangle \\ T_{liar} \text{ loops,} & \text{if } M \text{ halts on } \langle M \rangle \end{array} \right] ?$$



if M does not halt on $\langle M \rangle$

in any other case T_{liar} gives no output, it runs forever

(Observe that if T_{liar} exists, it must be a 'smart' machine.)

The Liar Paradox

Is the following sentence true?

This statement is false

Is it false?

St Paul in "Epistle to Titus" speaking of Cretans:

> Even one of their own prophets has said
>
> "Cretans are always liars, evil brutes, lazy gluttons."
>
> This testimony is true.

This paradox is also known as the Epimenides paradox; see

http://en.wikipedia.org/wiki/Epimenides_paradox

See also Russell's paradox.

The Liar Problem is unsolvable

A 'proof by contradiction' argument:

Suppose there exists a machine T_{liar} solving the Liar Problem.

$$\boxed{\text{What would } T_{liar} \text{ do on input } \langle T_{liar} \rangle\text{?}}$$

There are two cases: either it halts or not. **But both cases are impossible!**

- **Case 1**: $\underline{T_{liar} \text{ halts on } \langle T_{liar} \rangle.}$
  Since T_{liar} is supposed to solve the Liar Problem,
  this would mean that T_{liar} should loop on $\langle T_{liar} \rangle$.

- **Case 2**: $\underline{T_{liar} \text{ does not halt on } \langle T_{liar} \rangle.}$
  Since T_{liar} is supposed to solve the Liar Problem,
  this would mean that T_{liar} should halt on $\langle T_{liar} \rangle$ (and output 1).

So our initial assumption was **wrong**:

there cannot exist a Turing machine T_{liar} solving the Liar Problem.

Understanding this proof: diagonalisation

To understand more why a Turing machine T_{liar} solving the Liar Problem **cannot exist**, we make a table.

We list all 'smart' Turing machines both down the rows and across the columns:

	M_1	M_2	M_3	M_4	$\dots$
M_1	1	0	1	0	$\dots$
M_2	0	0	1	0	$\dots$
M_3	1	1	0	1	$\dots$
M_4	0	0	1	1	$\dots$
$\vdots$					

The entries now tell whether the machine in a given row halts on the code of the machine in a given column as input or not:

- entry i, j is 1, if M_i halts on $\langle M_j \rangle$, and
- entry i, j is 0, if M_i does not halt on $\langle M_j \rangle$.

T_{liar} cannot exist: diagonalisation

Suppose there is a Turing machine T_{liar} solving the Liar Problem.

Then T_{liar} should show up somewhere in our list:

$$
\begin{array}{c|cccccc}
 & M_1 & M_2 & M_3 & \dots & T_{liar} & \dots \\
\hline
M_1 & \underline{1} & 0 & 1 & \dots & 1 & \dots \\
M_2 & 0 & \underline{0} & 1 & \dots & 0 & \dots \\
M_3 & 1 & 1 & \underline{0} & \dots & 0 & \dots \\
\vdots & & & & \vdots & & \\
T_{liar} & 0 & 1 & 1 & \dots & \boxed{?} & \dots \\
\vdots & & & & & &
\end{array}
$$

T_{liar} halts on the codes of those Turing machines that do not halt on their own codes, and loops otherwise.

So the row of T_{liar} contains the opposites of the diagonal entries.

The **contradiction** occurs at the point of $\boxed{?}$ where the entry must be the opposite of itself.

Unsolvability of the Halting Problem: further preparations I

- Recall the copying machine from:

  Given any input word w of a's and b's, this machine always halts and outputs $w\text{␣}w$ (and at the end the head is scanning the ␣ in the middle).

  Using the same ideas it is not difficult to write a Turing machine that does the same job, but now not only words of a's and b's but any word over the alphabet $\boxed{a \quad q \quad 0 \; 1 \; (\;\;) \; , \; \Rightarrow \quad \Leftarrow}$ can be given as input to this 'fancier' copying machine.

- Recall the encoding machine T_{code}:

  Given any word w over the alphabet $\boxed{a \quad q \quad 0 \; 1 \; (\;\;) \; , \; \Rightarrow \quad \Leftarrow}$ as input to T_{code}, this machine always halts and outputs the code $\langle w \rangle$ of w. In particular, if the input word is a code itself (like $\langle M \rangle$, for some Turing machine M), the T_{code} outputs the code of this code (like $\langle\langle M \rangle\rangle$).

Unsolvability of the Halting Problem: further preparations II

It is easy to write a Turing machine T_{swap} that does the following:

- The tape alphabet of T_{swap} consists of $\triangleright$, $\sqcup$, 0, 1.
- If the input is 0, then T_{swap} halts and outputs 1.
- If the input is 1, then T_{swap} does not halt (it loops).
- For any other input, we don't care what T_{swap} does.

s	0	h	1
s	1	s	1
s	$\sqcup$	s	$\sqcup$
s	$\triangleright$	s	$\rightarrow$

Unsolvability of the Halting Problem: the plan

We will show the unsolvability of the Halting Problem with the help of the following **proof by contradiction** argument:

- Suppose there exists a Turing machine $\boldsymbol{H}$ solving the Halting Problem.

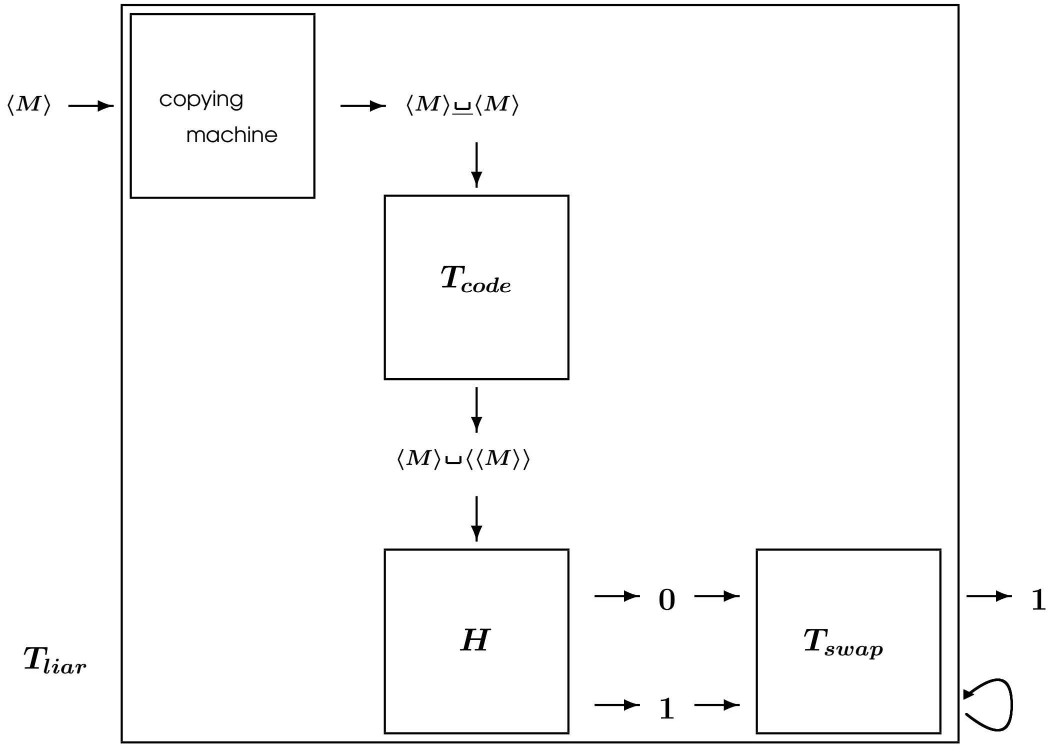
  (1) Then, with the help of this $\boldsymbol{H}$, the copying machine, and the machines $\boldsymbol{T_{code}}$ and $\boldsymbol{T_{swap}}$, we construct a Turing machine $\boldsymbol{T_{liar}}$ solving the Liar Problem (see next slide).

  (2) But, as we know, $\boldsymbol{T_{liar}}$ cannot exist.

- A **contradiction**. As we have just seen, all other 'building blocks' in our hypothetical $\boldsymbol{T_{liar}}$ do exist.

  So it is our original assumption that must have been **wrong**: a Turing machine $\boldsymbol{H}$ solving the Halting Problem cannot exist.

How to build T_{liar} if we have H



$\langle M \rangle$
copying machine
$\langle M \rangle \sqcup \langle M \rangle$
T_{code}
$\langle M \rangle \sqcup \langle \langle M \rangle \rangle$
H
0
1
T_{swap}
1
T_{liar}

This T_{liar} would solve the Liar Problem

Let us check that this $\boldsymbol{T_{liar}}$ would really solve the Liar Problem:

Given an input of the form $\langle M\rangle$ to this $\boldsymbol{T_{liar}}$, there are two cases:

- **Case 1.** $\underline{\boldsymbol{M} \text{ halts on } \langle \boldsymbol{M}\rangle}$:

  First, the copying machine runs on $\langle M\rangle$, and outputs $\langle M\rangle␣\langle M\rangle$. Second, the 'second part' of this output is given as input to $\boldsymbol{T_{code}}$, so $\boldsymbol{T_{code}}$ outputs $\langle M\rangle␣\langle\langle M\rangle\rangle$. Third, this output is given to $\boldsymbol{H}$ as input, so $\boldsymbol{H}$ outputs 1. Finally, $\mathbf{1}$ is given to $\boldsymbol{T_{swap}}$ as input, so $\boldsymbol{T_{swap}}$ $\underline{\text{loops}}$.

- **Case 2.** $\underline{\boldsymbol{M} \text{ doesn't halt on } \langle \boldsymbol{M}\rangle}$:

  First, the copying machine runs on $\langle M\rangle$, and outputs $\langle M\rangle␣\langle M\rangle$. Second, the 'second part' of this output is given as input to $\boldsymbol{T_{code}}$, so $\boldsymbol{T_{code}}$ outputs $\langle M\rangle␣\langle\langle M\rangle\rangle$. Third, this output is given to $\boldsymbol{H}$ as input, so $\boldsymbol{H}$ outputs 0. Finally, $\mathbf{0}$ is given to $\boldsymbol{T_{swap}}$ as input, so $\boldsymbol{T_{swap}}$ $\underline{\text{halts and outputs } 1}$.

Computers Ltd.?

Does the unsolvability of the Halting Problem **really** mean that the capabilities of computers are limited?

One might say **'the stupid Turing machines can't solve it, but look here, here is an amazingly powerful supercomputer, with an incredibly sophisticated programming language and a very very smart programmer equipped with modern state-of-the-art methodologies, they can surely solve this dumb problem'**

Maybe. But even if you do believe this (which means that you don't believe in the Church–Turing thesis), think about the following:

The amazing new stuff surely would be able to imitate all the 'tricks' we did with old fashioned Turing machines: copying and coding, diagonalisation, etc.

So the very same argument would show that the **Halting Problem of Amazing Supercomputers** is unsolvable by amazing supercomputers.

More unsolvable problems

- **Empty-Tape Halting Problem:**

  Given any Turing machine M, does M halt on the all-blank tape as input?

- **Some-Input Halting Problem:**

  Given any Turing machine M, is there any input string on which M halts?

- **All-Input Halting Problem:**

  Given any Turing machine M, does M halt on every input string?

- **Turing machine Equivalence Problem:**

  Given any two Turing machines M_1 and M_2, do they halt on the same input strings?

- **Tiling Problem:**

  Given any set T of tiles, decide whether an arbitrary large room can be tiled using only the available tile types