The Halting Problem

or what computers can't do

Julian Bradfield
School of Informatics, University of Edinburgh

The Entscheidungsproblem

In the early 20th century, mathematicians were wondering whether maths could be automated. In 1928, at a congress, David Hilbert posed the *Entscheidungsproblem*, or *Decision Problem*.

The Entscheidungsproblem

In the early 20th century, mathematicians were wondering whether maths could be automated. In 1928, at a congress, David Hilbert posed the *Entscheidungsproblem*, or *Decision Problem*.

In modern terms (simplifying a bit), it says:

> *Write a computer program that can tell us whether any mathematical statement is true or false.*

Hilbert didn't know about computer programs – he talked about 'a procedure using a finite number of operations'.

The Entscheidungsproblem

In the early 20th century, mathematicians were wondering whether maths could be automated. In 1928, at a congress, David Hilbert posed the *Entscheidungsproblem*, or *Decision Problem*.

In modern terms (simplifying a bit), it says:

> *Write a computer program that can tell us whether any mathematical statement is true or false.*

Hilbert didn't know about computer programs – he talked about 'a procedure using a finite number of operations'. So this is the master problem of mathematics: solve it, and mathematicians never need to do proofs again!

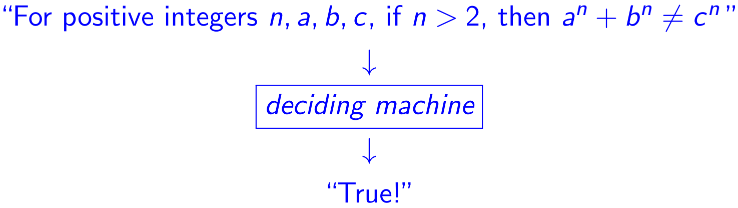
The Entscheidungsproblem

In the early 20th century, mathematicians were wondering whether maths could be automated. In 1928, at a congress, David Hilbert posed the *Entscheidungsproblem*, or *Decision Problem*.

In modern terms (simplifying a bit), it says:

> *Write a computer program that can tell us whether any mathematical statement is true or false.*

Hilbert didn't know about computer programs – he talked about 'a procedure using a finite number of operations'. So this is the master problem of mathematics: solve it, and mathematicians never need to do proofs again!



"For positive integers n, a, b, c, if $n > 2$, then $a^n + b^n \neq c^n$"

↓

deciding machine

↓

"True!"

The end of optimism

Hilbert thought the problem would be solved, just didn't know how long it would take. All part of a grand programme for mathematics.

The end of optimism

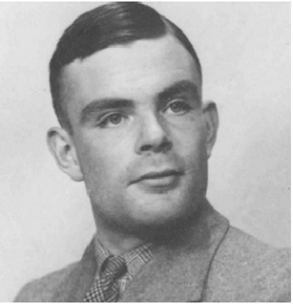
Hilbert thought the problem would be solved, just didn't know how long it would take. All part of a grand programme for mathematics.

In the early 1930s, three people shook the foundations of mathematics.

Kurt Gödel

Alonzo Church

Alan Turing



Alan Turing

Born in London in 1912, to parents who lived in India (his father was a civil servant).

Fairly miserable time at school (Sherborne), where he was considered to be wasting his time being interested in science.

Alan Turing

Born in London in 1912, to parents who lived in India (his father was a civil servant).

Fairly miserable time at school (Sherborne), where he was considered to be wasting his time being interested in science.

Went to King's College Cambridge in 1931 to read maths – a much more congenial environment, both for a mathematician, and for a homosexual, as he now knew he was.

First-class degree in 1934, Fellow in 1935.

Alan Turing

Born in London in 1912, to parents who lived in India (his father was a civil servant).

Fairly miserable time at school (Sherborne), where he was considered to be wasting his time being interested in science.

Went to King's College Cambridge in 1931 to read maths – a much more congenial environment, both for a mathematician, and for a homosexual, as he now knew he was.

First-class degree in 1934, Fellow in 1935.

By 1933, he had heard of Gödel's shattering result, that no logic could prove all theorems.

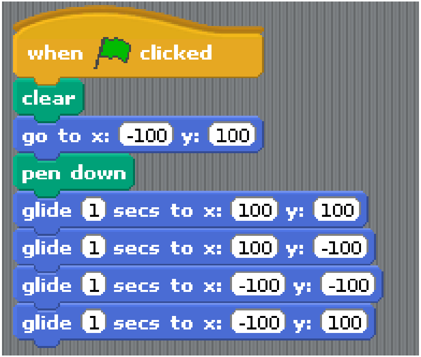
In 1935, he heard about the Entscheidungsproblem.

What is computing?

Turing had to make precise Hilbert's idea of 'procedure'.

He gave a theoretical pencil and paper method of calculation; any modern language will do just as well.

He argued that if we can compute something, we can compute it with one of these programs.



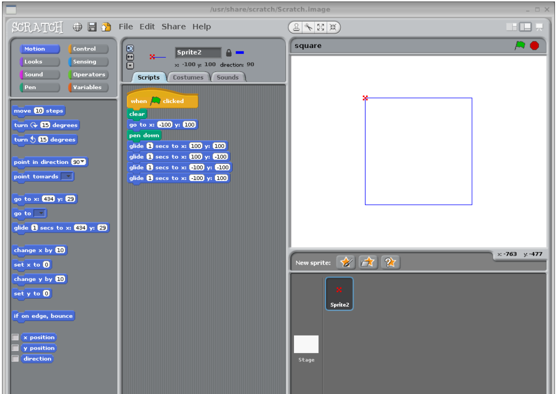
```
when ⚑ clicked
clear
go to x: -100 y: 100
pen down
glide 1 secs to x: 100 y: 100
glide 1 secs to x: 100 y: -100
glide 1 secs to x: -100 y: -100
glide 1 secs to x: -100 y: 100
```

Computing with computers

The second idea – based on Gödel's idea in logic – was that a procedure can calculate things about a procedure.

To us, this is natural: you write a program, and your program is compiled or interpreted by another program, and eventually machine code runs on a real physical computer.

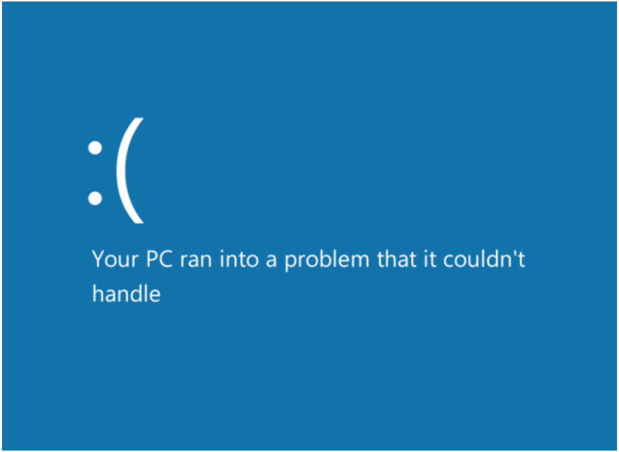
Then, it was not so natural.



```
when 🏁 clicked
clear
go to x: -100 y: 100
pen down
glide 1 secs to x: 100 y: 100
glide 1 secs to x: 100 y: -100
glide 1 secs to x: -100 y: -100
glide 1 secs to x: -100 y: 100
```

Impossible computations

The third idea – again, based on Gödel's technique in logic – was to find a statement about procedures that could not possibly be calculated by a procedure.

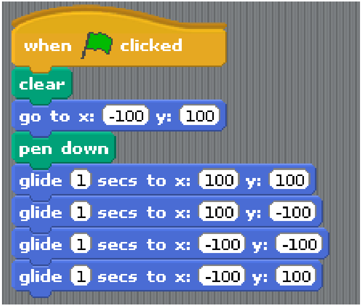


:(

Your PC ran into a problem that it couldn't handle

The Halting Problem

The statement is: given a procedure, does it eventually finish computing, or does it get stuck in a loop?



```
when ⚑ clicked
clear
go to x: -100 y: 100
pen down
glide 1 secs to x: 100 y: 100
glide 1 secs to x: 100 y: -100
glide 1 secs to x: -100 y: -100
glide 1 secs to x: -100 y: 100
```

↓

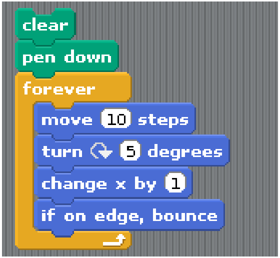
P, the halting decider

↓

“Good!”

The Halting Problem

The statement is: given a procedure, does it eventually finish computing, or does it get stuck in a loop?



```
clear
pen down
forever
    move 10 steps
    turn ↻ 5 degrees
    change x by 1
    if on edge, bounce
```

↓

P, the halting decider

↓

“Bad!”

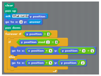
The Loop Snooper

Write a program that runs a procedure *on itself* and halts if it loops, and loops if it halts!

Q, the loop snooper

The Loop Snooper

Write a program that runs a procedure *on itself* and halts if it loops, and loops if it halts!



```
clear
pen up
ask What numbe y position
go to x: 0 y: answer
pen down
forever if y position > 1
  if y position mod 2 = 1
    go to x: x position + 5 y: y position * 3 + 1
  else
    go to x: x position + 5 y: y position / 2
```

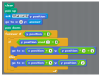
A

↓

Q, the loop snooper

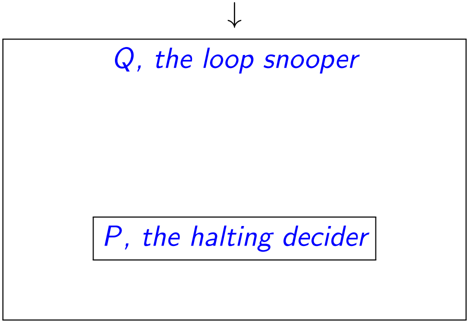
The Loop Snooper

Write a program that runs a procedure *on itself* and halts if it loops, and loops if it halts!



```
clear
pen up
ask What numbe y position
go to x: 0 y: answer
pen down
forever if y position > 1
    if y position mod 2 = 1
        go to x: x position + 5 y: y position * 3 + 1
    else
        go to x: x position + 5 y: y position / 2
```

A



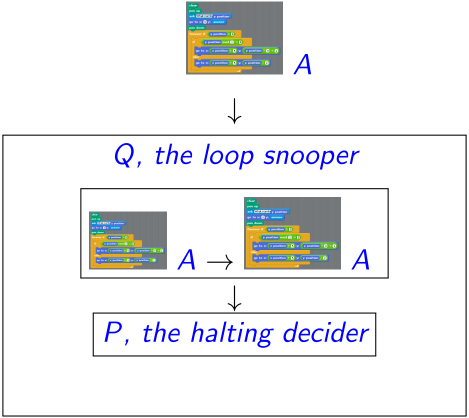
↓

Q, the loop snooper

P, the halting decider

The Loop Snooper

Write a program that runs a procedure *on itself* and halts if it loops, and loops if it halts!



```
clear
pen up
ask What numbe y position
go to x: 0 y: answer
pen down
forever if y position > 1
  if y position mod 2 = 1
    go to x: x position + 5 y: y position * 3 + 1
  else
    go to x: x position + 5 y: y position / 2
```

A

Q, the loop snooper

```
clear
pen up
ask What numbe y position
go to x: 0 y: answer
pen down
forever if y position > 1
  if y position mod 2 = 1
    go to x: x position + 5 y: y position * 3 + 1
  else
    go to x: x position + 5 y: y position / 2
```

A →

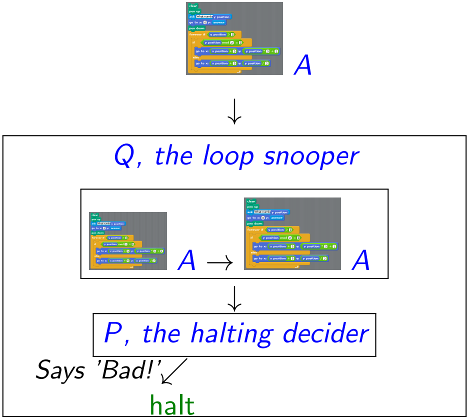
```
clear
pen up
ask What numbe y position
go to x: 0 y: answer
pen down
forever if y position > 1
  if y position mod 2 = 1
    go to x: x position + 5 y: y position * 3 + 1
  else
    go to x: x position + 5 y: y position / 2
```

A

P, the halting decider

The Loop Snooper

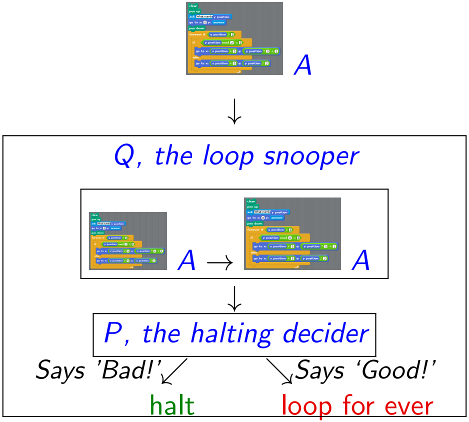
Write a program that runs a procedure *on itself* and halts if it loops, and loops if it halts!



```
clear
pen up
ask What numbe y position
go to x: 0 y: answer
pen down
forever if y position > 1
    if y position mod 2 = 1
        go to x: x position + 5 y: y position * 3 + 1
    else
        go to x: x position + 5 y: y position / 2
```

The Loop Snooper

Write a program that runs a procedure *on itself* and halts if it loops, and loops if it halts!



```
clear
pen up
ask What numbe y position
go to x: 0 y: answer
pen down
forever if y position > 1
    if y position mod 2 = 1
        go to x: x position + 5 y: y position * 3 + 1
    else
        go to x: x position + 5 y: y position / 2
```

A

↓

Q, the loop snooper

A → *A*

↓

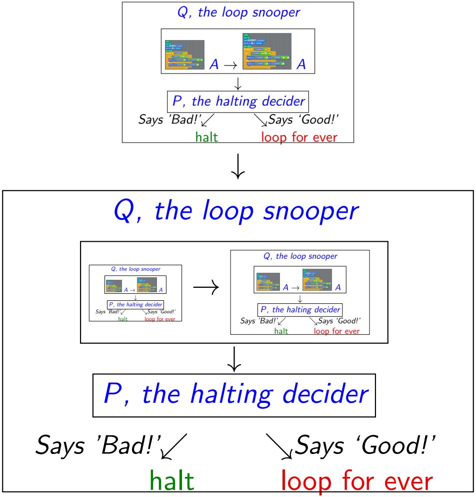
P, the halting decider

Says 'Bad!' ↙ halt

↘ *Says 'Good!'* loop for ever

The Loop Snooper Scoops the Halting Decider

Ask the Loop Snooper to analyse itself! What happens?



Q, the loop snooper

A → *A*

P, the halting decider

Says 'Bad!' ↙ halt

↘ *Says 'Good!'* loop for ever

↓

Q, the loop snooper

Q, the loop snooper — *A* → *A* — *P, the halting decider* — *Says 'Bad!'* halt — *Says 'Good!'* loop for ever

→

Q, the loop snooper — *A* → *A* — *P, the halting decider* — *Says 'Bad!'* halt — *Says 'Good!'* loop for ever

↓

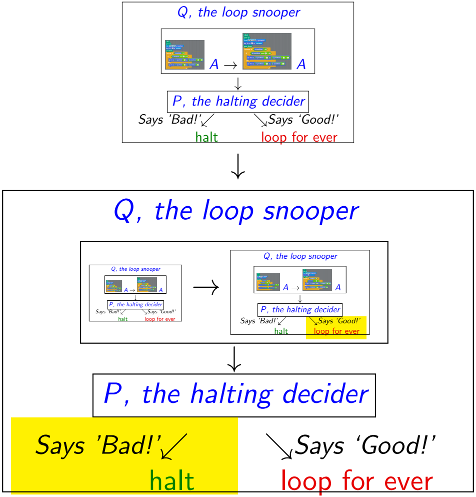
P, the halting decider

Says 'Bad!' ↙ halt

↘ *Says 'Good!'* loop for ever

The Loop Snooper Scoops the Halting Decider

If Q loops on Q, P says ‘Bad’ so Q halts on Q!



Q, the loop snooper

A → A

P, the halting decider

Says 'Bad!' ↙ ↘ *Says ‘Good!’*

halt

loop for ever

↓

Q, the loop snooper

Q, the loop snooper

A → A

P, the halting decider

Says 'Bad!' ↙ ↘ *Says ‘Good!’*

halt

loop for ever

→

Q, the loop snooper

A → A

P, the halting decider

Says 'Bad!' ↙ ↘ *Says ‘Good!’*

halt

loop for ever

↓

P, the halting decider

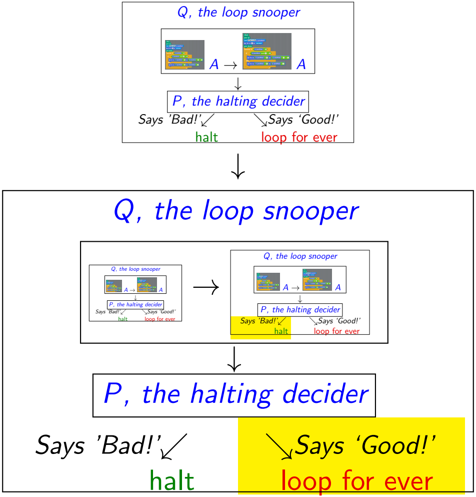
Says 'Bad!' ↙ ↘ *Says ‘Good!’*

halt

loop for ever

The Loop Snooper Scoops the Halting Decider

If Q halts on Q, P says ‘Good’ so Q loops on Q!



Q, the loop snooper

$A \rightarrow$ A

↓

P, the halting decider

Says ’Bad!’↙ ↘*Says ‘Good!’*

halt loop for ever

↓

Q, the loop snooper

Q, the loop snooper

$A \rightarrow$ A

P, the halting decider

Says ’Bad!’↙ ↘*Says ‘Good!’*

halt loop for ever

→

Q, the loop snooper

$A \rightarrow$ A

P, the halting decider

Says ’Bad!’↙ ↘*Says ‘Good!’*

halt loop for ever

↓

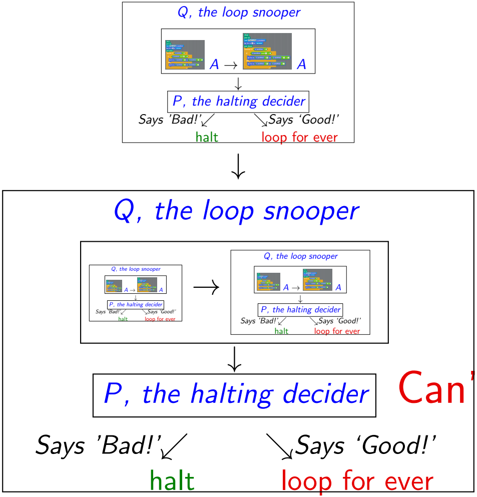
P, the halting decider

Says ’Bad!’↙ ↘*Says ‘Good!’*

halt loop for ever

The Loop Snooper Scoops the Halting Decider

Whatever P says Q does, Q does the opposite



Q, the loop snooper

$A \rightarrow$ A

↓

P, the halting decider

Says 'Bad!' ↙ ↘ *Says 'Good!'*

halt loop for ever

↓

Q, the loop snooper

Q, the loop snooper → *Q, the loop snooper*

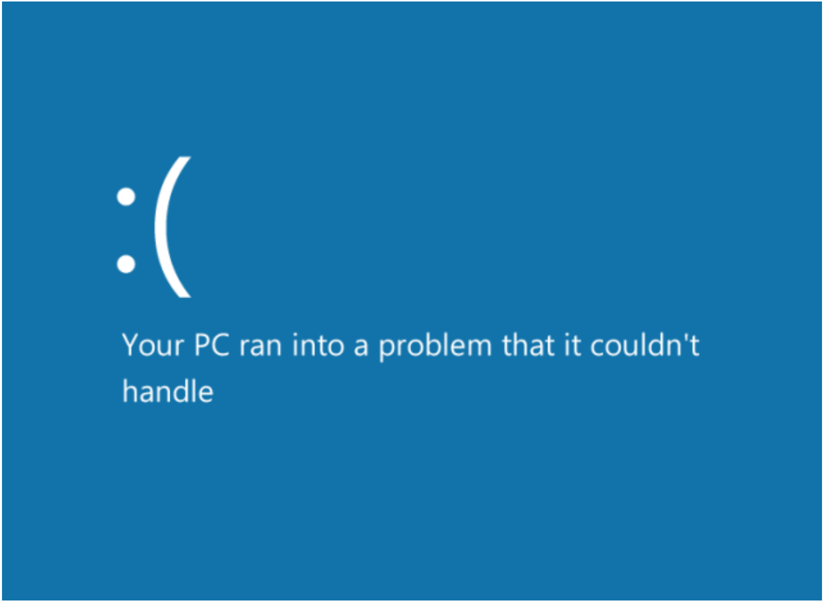
↓

P, the halting decider

Says 'Bad!' ↙ ↘ *Says 'Good!'*

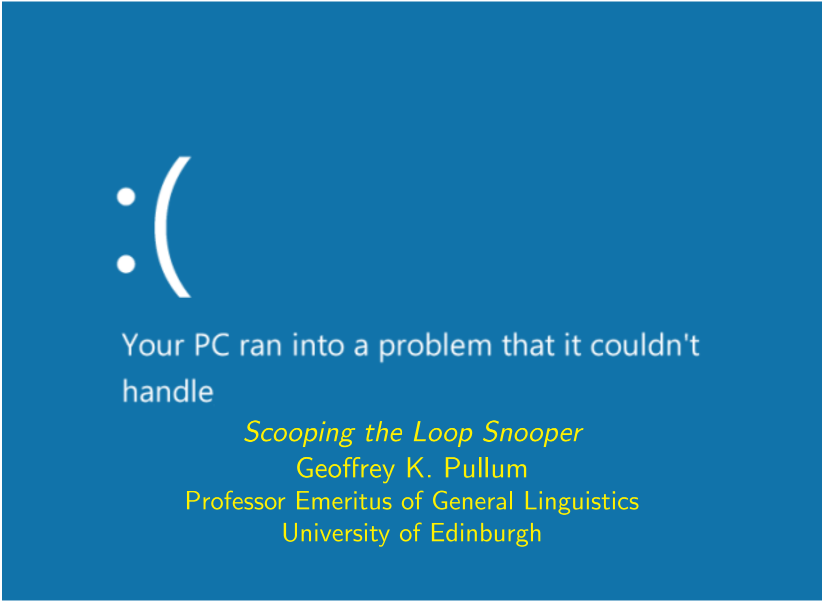
halt loop for ever

Can't exist!



:(

Your PC ran into a problem that it couldn't handle



:(

Your PC ran into a problem that it couldn't handle

Scooping the Loop Snooper

Geoffrey K. Pullum

Professor Emeritus of General Linguistics

University of Edinburgh

Turing and Theoretical Computer Science

Church had already broken the Entscheidungsproblem a few months previously. Turing learned this, and went to Princeton to study with him.

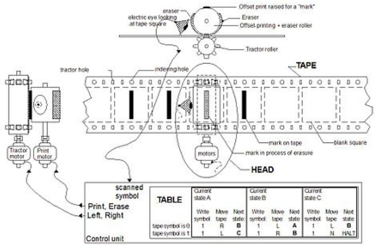
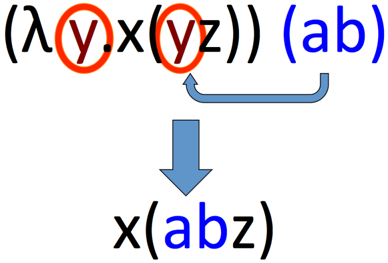


TABLE	Current state A			Current state B			Current state C		
	Write symbol:	Move tape:	Next state:	Write symbol:	Move tape:	Next state:	Write symbol:	Move tape:	Next state:
tape symbol is 0:	1	R	**B**	1	L	**A**	1	L	**B**
tape symbol is 1:	1	L	**C**	1	R	**B**	1	N	HALT

A fanciful mechanical Turing machine's TAPE and HEAD. The TABLE instructions might be on another "read only" tape, or perhaps on punch-cards. Usually a "finite state machine" is the model for the TABLE.



$$(\lambda y.x(yz))\ (ab)$$

$$x(abz)$$

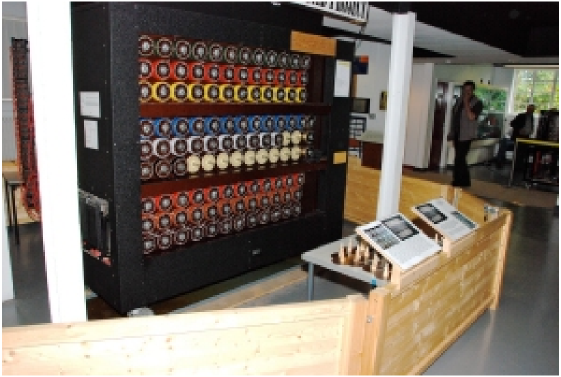
Turing's idea of computing is much more intuitive than Church's – but they were able to show they were the same. Now both are part of the foundations of computer science.

Turing continued working on computation and logic until war broke out.

Turing at Bletchley Park

Turing joined the code-breaking team at Bletchley Park.

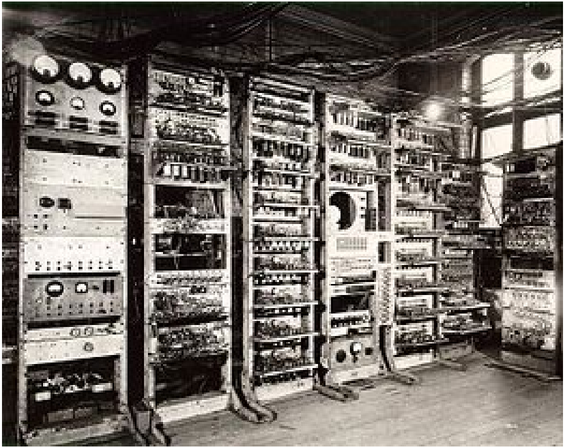
He was instrumental in many of their greatest achievements, including the breaking of Enigma.



Reconstruction of the Bombe – Bletchley Museum

Turing and Computers

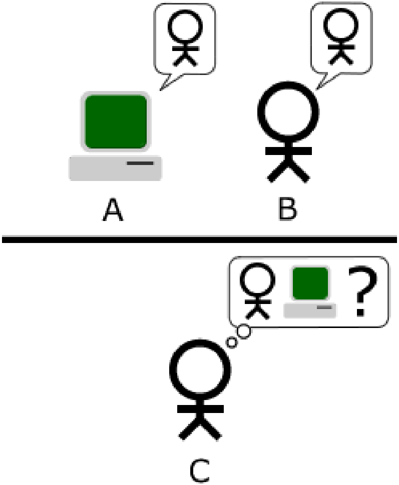
After the war, Turing worked in London and Manchester on the first real computers, as well as on mathematics.



The Manchester Mark 1

The Turing Test

He also thought about *Artificial Intelligence*, and proposed a test for truly intelligent computers:

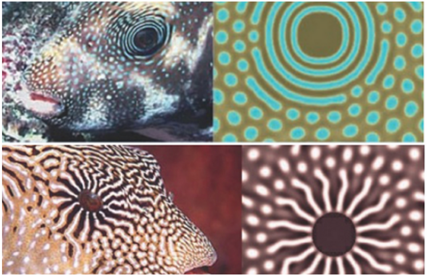


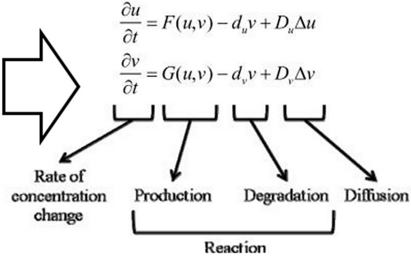
Turing the biologist

He also studied how patterns (e.g. spots and stripes on fish) arise in biology.

He made clever mathematical models of chemical reactions to do this.

Biologists still use his work.





$$\frac{\partial u}{\partial t} = F(u,v) - d_u v + D_u \Delta u$$

$$\frac{\partial v}{\partial t} = G(u,v) - d_v v + D_v \Delta v$$

Life and Death

Turing was homosexual, and in January 1952, after his house was burgled by an acquaintance of his boyfriend, he acknowledged this to the police.

He was charged with gross indecency, and pled guilty.

He was offered a choice between prison and an experimental hormone treatment intended ‘to reduce libido’. He took the treatment, for a year.

Another year later, on 8 June 1954, he was found dead of cyanide poisoning – he used cyanide for chemistry and electroplating experiments.

The inquest found suicide, and his biographers mostly agree. His relatives and closest friends thought it was accident.

912-1954
UBJ

Hilbert's Tenth Problem

Solve *Diophantine equations*: polynomial equations about integers.

E.g., find w, x, y, z such that $w^3 + x^3 = y^3 + z^3$.

(Many answers, e.g. $12, 1, 9, 10$.)

E.g., find a, b, c such that $a^3 + b^3 = c^3$.

(No solutions: the Fermat–Wiles Theorem, a.k.a. Fermat's Last Theorem.)

Hilbert's Tenth Problem

Solve *Diophantine equations*: polynomial equations about integers.

E.g., find w, x, y, z such that $w^3 + x^3 = y^3 + z^3$.

(Many answers, e.g. $12, 1, 9, 10$.)

E.g., find a, b, c such that $a^3 + b^3 = c^3$.

(No solutions: the Fermat–Wiles Theorem, a.k.a. Fermat's Last Theorem.)

It turns out that this is *undecidable* – no program can solve it.

Took 40 years to prove: final step by Yuri Matiyasevich in 1970, building on previous steps by Martin Davis, Hilary Putnam and Julia Robinson.