What is the halting problem?

C. S. Calude

University of Auckland, NZ

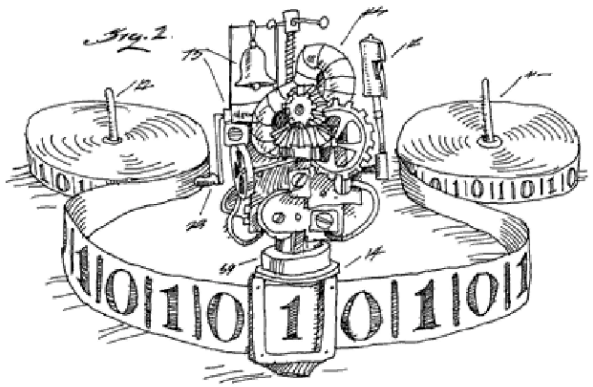
2006

Outline

Outline

The halting problem 1

The halting problem for Turing machines

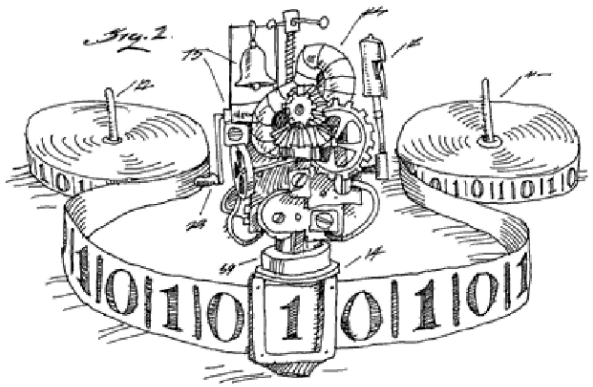


is the problem to decide whether an arbitrary Turing machine T (coded by a string, $code(T)$) eventually halts on an arbitrary input x.

Does there exist a Turing machine T_{halt} which given $code(T)$ and x, eventually stops and produces 1 if $T(x)$ stops and 0 if $T(x)$ does not stop?

The halting problem 1

The halting problem for Turing machines

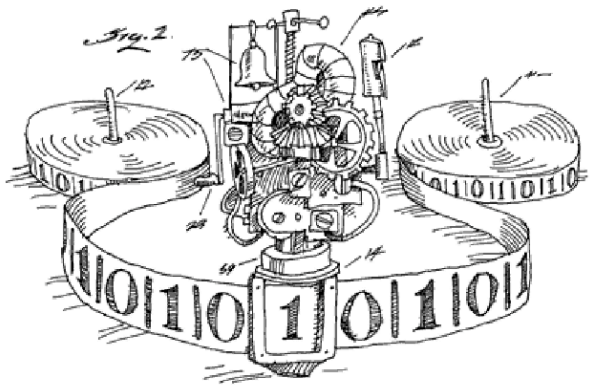


is the problem to decide whether an arbitrary Turing machine T (coded by a string, $code(T)$) eventually halts on an arbitrary input x.

Does there exist a Turing machine T_{halt} which given $code(T)$ and x, eventually stops and produces 1 if $T(x)$ stops and 0 if $T(x)$ does not stop?

The halting problem 1

The halting problem for Turing machines



is the problem to decide whether an arbitrary Turing machine T (coded by a string, $code(T)$) eventually halts on an arbitrary input x.

Does there exist a Turing machine T_{halt} which given $code(T)$ and x, eventually stops and produces 1 if $T(x)$ stops and 0 if $T(x)$ does not stop?

The halting problem 2

Turing's famous result states that this problem cannot be solved by any Turing machine, i.e. there is no such T_{halt}.

Some consequences:

- The negative solution of Hilbert's tenth problem: there is no Turing machine deciding whether an arbitrary Diophantine equation has or has not a solution in integers.
- Gödel's incompleteness theorem.
- Pour-El & Richards theorem (roughly, any closed unbounded operator on any Hilbert space takes some computable input to some uncomputable output).
- If one could solve it, then many mathematical problems . . . would be automatically solved.

The halting problem 2

Turing's famous result states that this problem cannot be solved by any Turing machine, i.e. there is no such T_{halt}.

Some consequences:

- The negative solution of Hilbert's tenth problem: there is no Turing machine deciding whether an arbitrary Diophantine equation has or has not a solution in integers.
- Gödel's incompleteness theorem.
- Pour-El & Richards theorem (roughly, any closed unbounded operator on any Hilbert space takes some computable input to some uncomputable output).
- If one could solve it, then many mathematical problems ... would be automatically solved.

The halting problem 2

Turing's famous result states that this problem cannot be solved by any Turing machine, i.e. there is no such T_{halt}.

Some consequences:

- The negative solution of Hilbert's tenth problem: there is no Turing machine deciding whether an arbitrary Diophantine equation has or has not a solution in integers.
- Gödel's incompleteness theorem.
- Pour-El & Richards theorem (roughly, any closed unbounded operator on any Hilbert space takes some computable input to some uncomputable output).
- If one could solve it, then many mathematical problems . . . would be automatically solved.

The halting problem 2

Turing's famous result states that this problem cannot be solved by any Turing machine, i.e. there is no such T_{halt}.

Some consequences:

- The negative solution of Hilbert's tenth problem: there is no Turing machine deciding whether an arbitrary Diophantine equation has or has not a solution in integers.
- Gödel's incompleteness theorem.
- Pour-El & Richards theorem (roughly, any closed unbounded operator on any Hilbert space takes some computable input to some uncomputable output).
- If one could solve it, then many mathematical problems . . . would be automatically solved.

The halting problem 2

Turing's famous result states that this problem cannot be solved by any Turing machine, i.e. there is no such T_{halt}.

Some consequences:

- The negative solution of Hilbert's tenth problem: there is no Turing machine deciding whether an arbitrary Diophantine equation has or has not a solution in integers.
- Gödel's incompleteness theorem.
- Pour-El & Richards theorem (roughly, any closed unbounded operator on any Hilbert space takes some computable input to some uncomputable output).
- If one could solve it, then many mathematical problems . . . would be automatically solved.

An informal proof 1

Instead of Turing machine we will deal with the informal notion of “program”. We assume that programs incorporate inputs—which are coded as natural numbers. So, a program may run forever or may just eventually stop, in which case it prints a natural number.

An informal proof 2

Assume that there exists a **halting program** deciding whether an arbitrary program will ever halt.

Construct the following program:

1. read a natural N;
2. generate all programs up to N bits in size;
3. use the **halting program** to check for each generated program whether it halts;
4. simulate the running of the above generated programs, and
5. output a number different from each output produced by the above programs.

An informal proof 2

Assume that there exists a **halting program** deciding whether an arbitrary program will ever halt.

Construct the following program:

1. read a natural N;
2. generate all programs up to N bits in size;
3. use the **halting program** to check for each generated program whether it halts;
4. simulate the running of the above generated programs, and
5. output a number different from each output produced by the above programs.

An informal proof 2

Assume that there exists a **halting program** deciding whether an arbitrary program will ever halt.

Construct the following program:

1. `read a natural N;`
2. `generate all programs up to N bits in size;`
3. `use the` **halting program** `to check for each generated program whether it halts;`
4. `simulate the running of the above generated programs, and`
5. `output a number different from each output produced by the above programs.`

An informal proof 3

The above program halts for every natural N. How long is it?

Answer: $\log N + O(1)$ bits.

Reason: to know N we need $\log_2 N$ bits; the rest of the program is a constant.

For large N, our program will belong to the set of programs having less than N bits (because $\log N + O(1) < N$). Accordingly, the program will be generated by itself at some stage of the computation.

In this case we have got a contradiction since our program will output a natural number two times bigger than the output produced by itself!

An informal proof 3

The above program halts for every natural N. How long is it?
Answer: $\log N + O(1)$ bits.
Reason: to know N we need $\log_2 N$ bits; the rest of the program is a constant.

For large N, our program will belong to the set of programs having less than N bits (because $\log N + O(1) < N$). Accordingly, the program will be generated by itself at some stage of the computation.

In this case we have got a contradiction since our program will output a natural number two times bigger than the output produced by itself!

An informal proof 3

The above program halts for every natural N. How long is it?
Answer: $\log N + O(1)$ bits.
Reason: to know N we need $\log_2 N$ bits; the rest of the program is a constant.

For large N, our program will belong to the set of programs having less than N bits (because $\log N + O(1) < N$). Accordingly, the program will be generated by itself at some stage of the computation.

In this case we have got a contradiction since our program will output a natural number two times bigger than the output produced by itself!

An informal proof 3

The above program halts for every natural N. How long is it?
Answer: $\log N + O(1)$ bits.
Reason: to know N we need $\log_2 N$ bits; the rest of the program is a constant.

For large N, our program will belong to the set of programs having less than N bits (because $\log N + O(1) < N$). Accordingly, the program will be generated by itself at some stage of the computation.

In this case we have got a contradiction since our program will output a natural number two times bigger than the output produced by itself!

An informal proof 3

The above program halts for every natural N. How long is it?
Answer: $\log N + O(1)$ bits.
Reason: to know N we need $\log_2 N$ bits; the rest of the program is a constant.

For large N, our program will belong to the set of programs having less than N bits (because $\log N + O(1) < N$). Accordingly, the program will be generated by itself at some stage of the computation.

In this case we have got a contradiction since our program will output a natural number two times bigger than the output produced by itself!

Outline

Fermat Last Theorem

It is perhaps surprising that many problems in mathematics can be reformulated in terms of the halting/non-halting status of appropriately constructed Turing machines.

Fermat's Last Theorem, stating that there is no integers $x, y, z, n > 3$ such that $x^n + y^n = z^n$, is an example.

We can construct a Turing machine T_{Fermat} which enumerates systematically all possible integers (for example, written in binary) $x, y, z, n > 3$, checks whether $x^n + y^n = z^n$, and stops if for some values x, y, z, n the relation is true; otherwise, T generates a new 4-tuple x, y, z, n and repeats the above procedure.

Fermat's Last Theorem is equivalent with the statement "T_{Fermat} never halts".

Fermat Last Theorem

It is perhaps surprising that many problems in mathematics can be reformulated in terms of the halting/non-halting status of appropriately constructed Turing machines.

Fermat's Last Theorem, stating that there is no integers $x, y, z, n > 3$ such that $x^n + y^n = z^n$, is an example.

We can construct a Turing machine T_{Fermat} which enumerates systematically all possible integers (for example, written in binary) $x, y, z, n > 3$, checks whether $x^n + y^n = z^n$, and stops if for some values x, y, z, n the relation is true; otherwise, T generates a new 4-tuple x, y, z, n and repeats the above procedure.

Fermat's Last Theorem is equivalent with the statement "T_{Fermat} never halts".

Fermat Last Theorem

It is perhaps surprising that many problems in mathematics can be reformulated in terms of the halting/non-halting status of appropriately constructed Turing machines.

Fermat's Last Theorem, stating that there is no integers $x, y, z, n > 3$ such that $x^n + y^n = z^n$, is an example.

We can construct a Turing machine T_{Fermat} which enumerates systematically all possible integers (for example, written in binary) $x, y, z, n > 3$, checks whether $x^n + y^n = z^n$, and stops if for some values x, y, z, n the relation is true; otherwise, T generates a new 4-tuple x, y, z, n and repeats the above procedure.

Fermat's Last Theorem is equivalent with the statement "T_{Fermat} never halts".

Fermat Last Theorem

It is perhaps surprising that many problems in mathematics can be reformulated in terms of the halting/non-halting status of appropriately constructed Turing machines.

Fermat's Last Theorem, stating that there is no integers $x, y, z, n > 3$ such that $x^n + y^n = z^n$, is an example.

We can construct a Turing machine T_{Fermat} which enumerates systematically all possible integers (for example, written in binary) $x, y, z, n > 3$, checks whether $x^n + y^n = z^n$, and stops if for some values x, y, z, n the relation is true; otherwise, T generates a new 4-tuple x, y, z, n and repeats the above procedure.

Fermat's Last Theorem is equivalent with the statement "T_{Fermat} never halts".

A universal prefix-free machine 1

A register machine has a finite number of registers, each of which may contain an arbitrarily large non-negative binary integer. The register machine *U* (labelled) instructions are:

L: EQ R1 R2 R3
L: SET R1 R2
L: ADD R1 R2
L: READ R1
L: HALT

A universal prefix-free machine 2

A register machine program consists of a finite list of labelled instructions from the above list, with the restriction that the HALT instruction appears only once, as the last instruction of the list.

The input data (a binary string) follows immediately after the HALT instruction.

A program not reading the whole data or attempting to read past the last data-bit results in a runtime error. Some programs have no input data.

Theorem

The register machine U is prefix-free and universal.

A universal prefix-free machine 2

A register machine program consists of a finite list of labelled instructions from the above list, with the restriction that the HALT instruction appears only once, as the last instruction of the list.

The input data (a binary string) follows immediately after the HALT instruction.

A program not reading the whole data or attempting to read past the last data-bit results in a runtime error. Some programs have no input data.

Theorem

The register machine U is prefix-free and universal.

A universal prefix-free machine 2

A register machine program consists of a finite list of labelled instructions from the above list, with the restriction that the HALT instruction appears only once, as the last instruction of the list.

The input data (a binary string) follows immediately after the HALT instruction.

A program not reading the whole data or attempting to read past the last data-bit results in a runtime error. Some programs have no input data.

Theorem

The register machine U is prefix-free and universal.

A universal prefix-free machine 2

A register machine program consists of a finite list of labelled instructions from the above list, with the restriction that the HALT instruction appears only once, as the last instruction of the list.

The input data (a binary string) follows immediately after the HALT instruction.

A program not reading the whole data or attempting to read past the last data-bit results in a runtime error. Some programs have no input data.

Theorem

The register machine U is prefix-free and universal.

Goldbach's Conjecture, Riemann Hypothesis, Collatz's Conjecture

The program T_{Goldbach}, which never stops if Goldbach's Conjecture is true, has 135 instructions totalling 3,484 bits.

The program T_{Riemann}, which never stops if the Riemann Hypothesis is true, consists of 290 instructions totalling 7,780 bits.

There is a *non-constructive* way to prove that there exists a program T_{Collatz} which never stops iff the Collatz' Conjecture is true.

Goldbach's Conjecture, Riemann Hypothesis, Collatz's Conjecture

The program T_{Goldbach}, which never stops if Goldbach's Conjecture is true, has 135 instructions totalling 3,484 bits.

The program T_{Riemann}, which never stops if the Riemann Hypothesis is true, consists of 290 instructions totalling 7,780 bits.

There is a *non-constructive* way to prove that there exists a program T_{Collatz} which never stops iff the Collatz' Conjecture is true.

Goldbach's Conjecture, Riemann Hypothesis, Collatz's Conjecture

The program T_{Goldbach}, which never stops if Goldbach's Conjecture is true, has 135 instructions totalling 3,484 bits.

The program T_{Riemann}, which never stops if the Riemann Hypothesis is true, consists of 290 instructions totalling 7,780 bits.

There is a *non-constructive* way to prove that there exists a program T_{Collatz} which never stops iff the Collatz' Conjecture is true.

What's the point?

Given that the halting problem is undecidable . . .

- First, we have a measure of the difficulty of problems which can be used for finitely refutable conjectures.
- Secondly, can we "attack" the halting problem, maybe in a different way?
  - quantum solutions
  - relativistic solutions
  - probabilistic approach

What's the point?

Given that the halting problem is undecidable . . .

- First, we have a measure of the difficulty of problems which can be used for finitely refutable conjectures.
- Secondly, can we "attack" the halting problem, maybe in a different way?
  - quantum solutions
  - relativistic solutions
  - probabilistic approach

What's the point?

Given that the halting problem is undecidable . . .

- First, we have a measure of the difficulty of problems which can be used for finitely refutable conjectures.
- Secondly, can we "attack" the halting problem, maybe in a different way?
  - quantum solutions
  - relativistic solutions
  - probabilistic approach

What's the point?

Given that the halting problem is undecidable . . .

- First, we have a measure of the difficulty of problems which can be used for finitely refutable conjectures.
- Secondly, can we "attack" the halting problem, maybe in a different way?
  - quantum solutions
  - relativistic solutions
  - probabilistic approach

What's the point?

Given that the halting problem is undecidable . . .

- First, we have a measure of the difficulty of problems which can be used for finitely refutable conjectures.
- Secondly, can we "attack" the halting problem, maybe in a different way?
  - quantum solutions
  - relativistic solutions
  - probabilistic approach

What's the point?

Given that the halting problem is undecidable . . .

- First, we have a measure of the difficulty of problems which can be used for finitely refutable conjectures.
- Secondly, can we "attack" the halting problem, maybe in a different way?
  - quantum solutions
  - relativistic solutions
  - probabilistic approach

Outline

The Omega number 1

Recall: The register machine U is prefix-free and universal.

Assume that programs are uniformly distributed. Chaitin's Omega, the "halting probability of U", is:

$$\Omega_U = \sum_{p \in \mathrm{dom}(U)} 2^{-|p|}.$$

Theorem

The number Ω_U is algorithmically random, i.e. if $\Omega_U = 0.\omega_1\omega_2\cdots$, then $H_U(\omega_1\omega_2\cdots\omega_n) \geq n -$ constant.

Theorem

With the first n bits of Ω_U we can solve the halting problem for every program of length less than or equal to n.

The Omega number 1

Recall: The register machine U is prefix-free and universal. Assume that programs are uniformly distributed. Chaitin's Omega, the "halting probability of U", is:

$$\Omega_U = \sum_{p \in \mathrm{dom}(U)} 2^{-|p|}.$$

Theorem

The number Ω_U is algorithmically random, i.e. if $\Omega_U = 0.\omega_1\omega_2\cdots$, then $H_U(\omega_1\omega_2\cdots\omega_n) \geq n - \text{constant}$.

Theorem

With the first n bits of Ω_U we can solve the halting problem for every program of length less than or equal to n.

The Omega number 1

Recall: The register machine U is prefix-free and universal. Assume that programs are uniformly distributed. Chaitin's Omega, the "halting probability of U", is:

$$\Omega_U = \sum_{p \in \mathrm{dom}(U)} 2^{-|p|}.$$

Theorem

The number Ω_U is algorithmically random, i.e. if $\Omega_U = 0.\omega_1\omega_2\cdots$, then $H_U(\omega_1\omega_2\cdots\omega_n) \geq n - \text{constant}$.

Theorem

With the first n bits of Ω_U we can solve the halting problem for every program of length less than or equal to n.

The Omega number 1

Recall: The register machine U is prefix-free and universal. Assume that programs are uniformly distributed. Chaitin's Omega, the "halting probability of U", is:

$$\Omega_U = \sum_{p \in \mathrm{dom}(U)} 2^{-|p|}.$$

Theorem

The number Ω_U is algorithmically random, i.e. if $\Omega_U = 0.\omega_1\omega_2\cdots$, then $H_U(\omega_1\omega_2\cdots\omega_n) \geq n - \text{constant}$.

Theorem

With the first n bits of Ω_U we can solve the halting problem for every program of length less than or equal to n.

The Omega number 2

Solving the halting problem for all programs up to 80 bits one can get:

Theorem [Calude, Dinneen, 2006]

The first 43 bits of Ω_U are:

$$0001000000010000101001110111000100000101110$$

Still, a long way till 3,484 bits for Goldbach's Conjecture!

Even worse,

Theorem

One can compute an integer N such that no Turing machine can compute any digit of Omega ω_n with $n > N$.

The Omega number 2

Solving the halting problem for all programs up to 80 bits one can get:

Theorem [Calude, Dinneen, 2006]

The first 43 bits of Ω_U are:

$$0001000000010000101001110111000100000101110$$

Still, a long way till 3,484 bits for Goldbach's Conjecture!

Even worse,

Theorem

One can compute an integer N such that no Turing machine can compute any digit of Omega ω_n with $n > N$.

The Omega number 2

Solving the halting problem for all programs up to 80 bits one can get:

Theorem [Calude, Dinneen, 2006]

The first 43 bits of Ω_U are:

$$0001000000010000101001110111000100000101110$$

Still, a long way till 3,484 bits for Goldbach's Conjecture!

Even worse,

Theorem

One can compute an integer N such that no Turing machine can compute any digit of Omega ω_n with $n > N$.

Can a program stop at an algorithmically random time? 1

Let $\text{bin}(i)$ be the ith non-empty binary string.
A time t is algorithmically random if

$$H(\text{bin}(t)) \geq |\text{bin}(t)| - \log(|\text{bin}(t)|).$$

Theorem [Chaitin, 1987]

There is a constant c such that if $U(\text{bin}(i))$ halts exactly in time t, then $H(\text{bin}(t)) \leq |\text{bin}(i)| + c$.

Theorem [Calude, Stay, 2006]

Assume that an N-bit program $U(p)$ has not stopped by time $2^{2N+2c+1}$, where $N \geq 2$ and c comes from the above theorem. Then, $U(p)$ cannot stop exactly at any algorithmically random time $t \geq 2^{2N+2c+1}$.

Can a program stop at an algorithmically random time? 1

Let $\text{bin}(i)$ be the ith non-empty binary string.
A time t is algorithmically random if

$$H(\text{bin}(t)) \geq |\text{bin}(t)| - \log(|\text{bin}(t)|).$$

Theorem [Chaitin, 1987]

There is a constant c such that if $U(\text{bin}(i))$ halts exactly in time t, then $H(\text{bin}(t)) \leq |\text{bin}(i)| + c$.

Theorem [Calude, Stay, 2006]

Assume that an N-bit program $U(p)$ has not stopped by time $2^{2N+2c+1}$, where $N \geq 2$ and c comes from the above theorem. Then, $U(p)$ cannot stop exactly at any algorithmically random time $t \geq 2^{2N+2c+1}$.

Can a program stop at an algorithmically random time? 1

Let $\text{bin}(i)$ be the ith non-empty binary string.
A time t is algorithmically random if

$$H(\text{bin}(t)) \geq |\text{bin}(t)| - \log(|\text{bin}(t)|).$$

Theorem [Chaitin, 1987]

There is a constant c such that if $U(\text{bin}(i))$ halts exactly in time t, then $H(\text{bin}(t)) \leq |\text{bin}(i)| + c$.

Theorem [Calude, Stay, 2006]

Assume that an N-bit program $U(p)$ has not stopped by time $2^{2N+2c+1}$, where $N \geq 2$ and c comes from the above theorem. Then, $U(p)$ cannot stop exactly at any algorithmically random time $t \geq 2^{2N+2c+1}$.

Can a program stop at an algorithmically random time? 2

A time t is called "exponential stopping time" if there is a program p which stops on U exactly at $t_p > 2^{2|p|+2c+1}$.

Theorem [Calude, Stay, 2006]

The set of exponential stopping times has effectively zero density.

Can a program stop at an algorithmically random time? 2

A time t is called "exponential stopping time" if there is a program p which stops on U exactly at $t_p > 2^{2|p|+2c+1}$.

Theorem [Calude, Stay, 2006]

The set of exponential stopping times has effectively zero density.

Halting according to a computable time distribution 1

Postulate *an a priori computable probability distribution on all possible runtimes*. Consequently, the probability space is the product of the space of programs and the time space.

More precisely, the probability space is

$$Space_{\{\rho_i\}} = B^* \times \{1, 2, \ldots\},$$

where N-bit programs are assumed to be uniformly distributed, and the observer time flows according to the computable probability distribution $\{\rho_i\}$.

Halting according to a computable time distribution 1

Postulate *an a priori computable probability distribution on all possible runtimes*. Consequently, the probability space is the product of the space of programs and the time space.

More precisely, the probability space is

$$Space_{\{\rho_i\}} = B^* \times \{1, 2, \ldots\},$$

where N-bit programs are assumed to be uniformly distributed, and the observer time flows according to the computable probability distribution $\{\rho_i\}$.

Halting according to a computable time distribution 3

Theorem [Calude, Stay, 2006]

Consider that the observer time flows from 1 to ∞ according to a probability distribution ρ_i which effectively converges to 1. Then, the probability that an N-bit program which hasn't stopped on U by time t_k (which can be effectively computed) will eventually halt effectively converges to zero.

Halting according to a computable time distribution 3

Here is an example.
For every program bin(i) let $t_{\text{bin}(i)}$ be the exact time $U(\text{bin}(i)$ stops, if $U(\text{bin}(i)$ halts, or $t_{\text{bin}(i)} = \infty$, otherwise.

Define the computable number:

$$0 < \Upsilon_U = \sum_{i \geq 1} 2^{-i} / t_{\text{bin}(i)} < 1.$$

Then, define the computable probability distribution

$$\rho_i = \frac{2^{-i}}{t_{\text{bin}(i)} \cdot \Upsilon_U}.$$

Halting according to a computable time distribution 3

Here is an example.
For every program $\text{bin}(i)$ let $t_{\text{bin}(i)}$ be the exact time $U(\text{bin}(i)$ stops, if $U(\text{bin}(i)$ halts, or $t_{\text{bin}(i)} = \infty$, otherwise.

Define the computable number:

$$0 < \Upsilon_U = \sum_{i \geq 1} 2^{-i} / t_{\text{bin}(i)} < 1.$$

Then, define the computable probability distribution

$$\rho_i = \frac{2^{-i}}{t_{\text{bin}(i)} \cdot \Upsilon_U}.$$

Halting according to a computable time distribution 3

Here is an example.
For every program bin(i) let $t_{\text{bin}(i)}$ be the exact time $U(\text{bin}(i)$ stops, if $U(\text{bin}(i)$ halts, or $t_{\text{bin}(i)} = \infty$, otherwise.

Define the computable number:

$$0 < \Upsilon_U = \sum_{i \geq 1} 2^{-i} / t_{\text{bin}(i)} < 1.$$

Then, define the computable probability distribution

$$\rho_i = \frac{2^{-i}}{t_{\text{bin}(i)} \cdot \Upsilon_U}.$$

Halting according to a computable time distribution 4

Theorem [Calude, Stay, 2006]

Assume that $U(p)$ has not stopped by time $T > k - \lfloor \log \Upsilon_U \rfloor$. Then, the probability (according to the probability space $Space_{\{\rho_i\}}$) that $U(p)$ eventually halts is smaller than 2^{-k}.

Theorem [Calude, Stay, 2006]

Assume that U and $Space_{\{\rho(i)\}}$ have been fixed. For every integer $k > 0$, the set of halting programs for U can be written as a disjoint union of a computable set and a set of probability effectively smaller than 2^{-k}.

Halting according to a computable time distribution 4

Theorem [Calude, Stay, 2006]

Assume that $U(p)$ has not stopped by time $T > k - \lfloor \log \Upsilon_U \rfloor$. Then, the probability (according to the probability space $Space_{\{\rho_i\}}$) that $U(p)$ eventually halts is smaller than 2^{-k}.

Theorem [Calude, Stay, 2006]

Assume that U and $Space_{\{\rho(i)\}}$ have been fixed. For every integer $k > 0$, the set of halting programs for U can be written as a disjoint union of a computable set and a set of probability effectively smaller than 2^{-k}.

Grand open problem

Open problem

Can the above analysis be used to develop a probabilistic solution for the halting problem?

Outline

Books

- C. S. Calude. *Information and Randomness – An Algorithmic Perspective*, Springer-Verlag 2002.
- G. J. Chaitin. *Meta Math!*, Pantheon, 2005.
- J. Gruska. *Foundations of Computing*, Thomson International Computer Press, Boston, 1997.

Papers

- C. S. Calude, Elena Calude, M. J. Dinneen. A new measure of the difficulty of problems, *Journal for Multiple-Valued Logic and Soft Computing* 10 (2006), 1–21.
- C. S. Calude, M. J. Dinneen. Approximations of Omega numbers, in preparation.
- C. S. Calude, M. J. Dinneen and C.-K. Shu. Computing a glimpse of randomness, *Experimental Mathematics* 11, 2 (2002), 369–378.
- C. S. Calude, B. Pavlov. Coins, quantum measurements, and Turing's barrier, *Quantum Information Processing* 1, 1–2 (2002), 107–127.
- C. S. Calude, M. A. Stay. De-Quantising non-halting programs, *CDMTCS Research Report* 284 (2006), 17 pp.