Turing Machines

Klaus Sutner

Carnegie Mellon University

Fall 2009

Outline

- Turing Machines
- The Busy Beaver Problem
- Wolfram Prize
- Church-Turing Thesis

Where Are We?

- We have a formal notion of computability based on register machines.
- One can use RMs to define decidability and semi-decidability.
- The Halting Problem is not decidable.
- The Halting Problem is semi-decidable since there is a universal register machine.
- We can think of the Halting Problem as an application of the jump operation.

Iterating the Jump

. . . creates monsters.

And Intuitive Computability?

This is all very nice, but how close are we to capturing the intuitive notion of computability?

Historically, the first notions of computability suggested around 1930 were

- Gödel-Herbrand general recursive functions, and
- Church's λ-calculus.

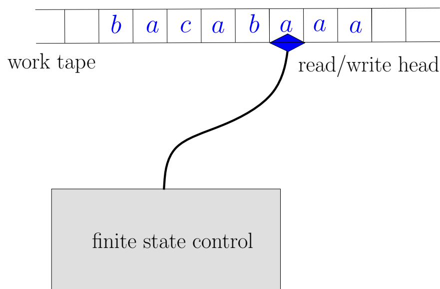
See the CDM website for details.

Gödel was not convinced that these proposals properly capture computability.

Turing's Brilliant Idea

To settle the debate about how to formalize computability, Turing suggested in the mid 1930's to model the only entities known at the time capable of performing computations: human computors.

A very careful study of the a human computes leads to a machine of the form



The Pieces

- A tape: a bi-infinite strip of paper, subdivided into cells. Each cell contains a single letter; all but finitely many contain just a blank. So we have a tape inscription.
- A read/write head that is positioned at a particular cell. That head can move left and right.
- A finite state control that directs the head: symbols are read and written, the head is moved around and the internal state of the FSC changes.

Formalization

- alphabet Σ: finite set of allowed symbols, special blank symbol $\underline{b} \in \Sigma$
- state set Q: finite set of possible mind configurations
- $\delta : Q \times \Sigma \to Q \times \Sigma \times \{-1, 0, 1\}$: transition function
- a special initial state $q_0 \in Q$
- a special halting state $q_H \in Q$.

Definition

A Turing machine is a structure $M = \langle\, Q, \Sigma, \delta, q_0, q_H \,\rangle$.

Example: Successor Function

Tape alphabet $\Sigma = \{\underline{b}, 1\}$

States $Q = \{0, 1, 2, 3\}$

Initial state $q_0 = 0$, final state $q_H = 3$.

The transition function δ is given be the following table:

p	σ	$\delta(p, \sigma)$		
0	$\underline{b}$	1	$\underline{b}$	$+1$
1	$\underline{b}$	2	1	0
1	1	1	1	1
2	$\underline{b}$	3	$\underline{b}$	0
2	1	2	1	-1

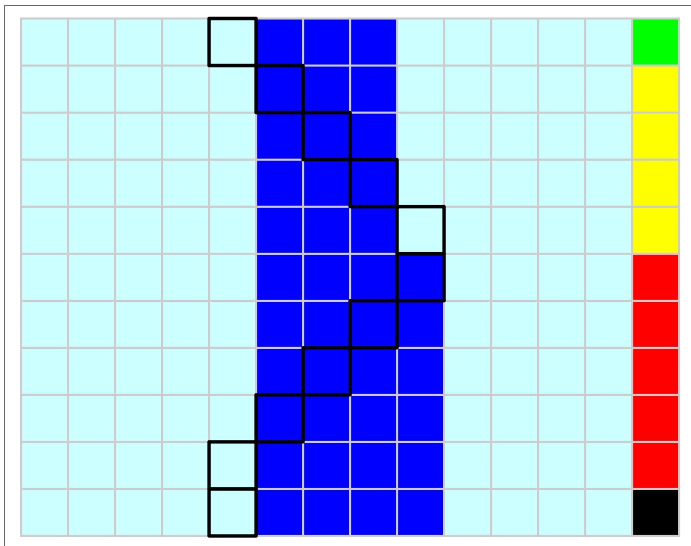
No other transitions are needed.

Simulation

It is not hard to write a program that simulates a Turing machine. Hence, do some computational experiments on these machines. The following visualization method is sometimes useful to understand behavior of small Turing machines. In the pictures:

- The tape is represented by a row of colored squares.
- Time flows from top to bottom (usually; sometimes from left to right).
- The position of the tape head is indicated by a black frame around a square.
- The rightmost column indicates the state of the machine.

Sample Run



Addition

We are using unary notation, n is represented by

$$n \mapsto \underbrace{111\ldots11}_{n+1}$$

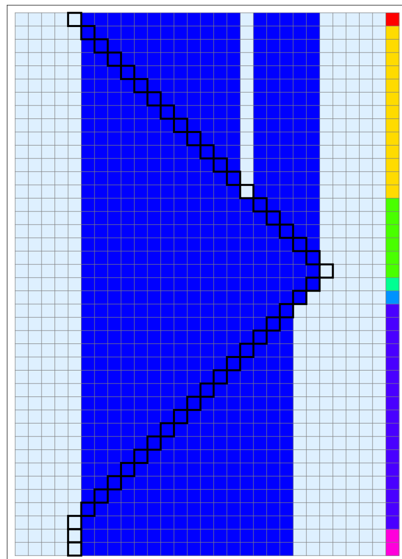
Hence we have to erase two 1's at the end:

$$\begin{array}{cc|ccc}
0 & \underline{b} & 1 & \underline{b} & 1 \\
1 & \underline{b} & 2 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 \\
2 & \underline{b} & 3 & \underline{b} & -1 \\
2 & 1 & 2 & 1 & 1 \\
3 & 1 & 4 & \underline{b} & -1 \\
4 & 1 & 5 & \underline{b} & -1 \\
5 & \underline{b} & 6 & \underline{b} & 0 \\
5 & 1 & 5 & 1 & -1
\end{array}$$

0 is the initial state, 6 is the final state

Sample Run

A Longer Run



Correctness

Note that for very simple tasks such as addition in unary one can read off a correctness proof from these pictures.

Exercise

What would a complete correctness proof for the Turing machine that performs unary addition look like? What is difficult about the proof?

Exercise

Construct a Turing machine that performs addition when the input is given in binary. What should be pictures look like in this case? How hard is a correctness proof?

Palindromes

Comments

In the pictures, time flows from left to right and the states are in the top row.

The first picture represents a unsuccessful computation.

The second picture represents a successful computation.

Exercise

Figure out how this machine works.

Formalizing Turing Computation

As with register machines we need to describe a snapshot during a computation; we need

- the current tape inscription
- the current head position
- the current state

For the following, assume for simplicity that $Q \cap \Sigma = \emptyset$.

Definition

A configuration or instantaneous description (ID) is a word $b\,p\,a$ where $a, b \in \Sigma^\star$ and $p \in Q$:

$$b_m b_{m-1} \ldots b_1\ p\ a_1 a_2 \ldots a_n$$

means that the read/write head is positioned at a_1 and the whole tape inscription is $b_m \ldots b_1 a_1 \ldots a_n$.

The Step Relation

We need to explain the one-step relation $upv \mathrel{\vdash^{1}_{M}} u'p'v'$.

Let $\delta(p, a_1) = (q, c, \Delta)$ Then the "next configuration" is defined by

$$\begin{array}{ll} b_m \ldots q\, b_1 c a_2 \ldots a_n & \Delta = -1 \\ b_m \ldots b_1\, q\, c a_2 \ldots a_n & \Delta = 0 \\ b_m \ldots b_1 c\, q\, a_2 \ldots a_n & \Delta = +1 \end{array}$$

Here we assume that b is non-empty, otherwise we can set $b_1 = \underline{b}$.

Many Steps

As always, we extend the "one-step" relation to multiple steps by iteration:

- $C \vdash^{1}_{M} C'$: one step
- $C \vdash^{t}_{M} C'$: exactly t steps
- $C \vdash_{M} C'$: any finite number of steps

Input and Output

Given any input $x_1, x_2, \ldots, x_n \in \Sigma$ the initial configuration is

$$C_x = q_0\, \underline{b}\, x_1 x_2 \ldots x_n$$

A final configuration is of the form

$$C_y^H = q_H\, \underline{b}\, y_1 y_2 \ldots y_n$$

For input we have chosen to place the head at the last blank to the left of the input symbol, there are lots of other possibilities (e.g, $q_0 x_1 \ldots x_n$).

Also, we require our machines to erase the tape (except for the output) and return the tape head to the initial position before halting; nothing important changes if we drop this condition.

Computations

If $C_x \vdash_M C_y^H$ then $y = y_1, y_2, \ldots, y_m$ is the output of the computation of machine M on input x.

M computes $f : \mathbb{N}^k \rightarrow_p \mathbb{N}$ if, for all $\vec{x} \in \mathbb{N}^k$,

- If f is defined on $\vec{x}$ then $C_x \vdash_M C_y^H$ and $f(\vec{x}) = y$.
- If f is undefined on $\vec{x}$ then the computation of M on x does not halt.

Here $\vec{x}$ is a k-tuple of natural numbers and $x \in \Sigma^\star$ is the corresponding tape-inscription.

Turing Computability and Robustness

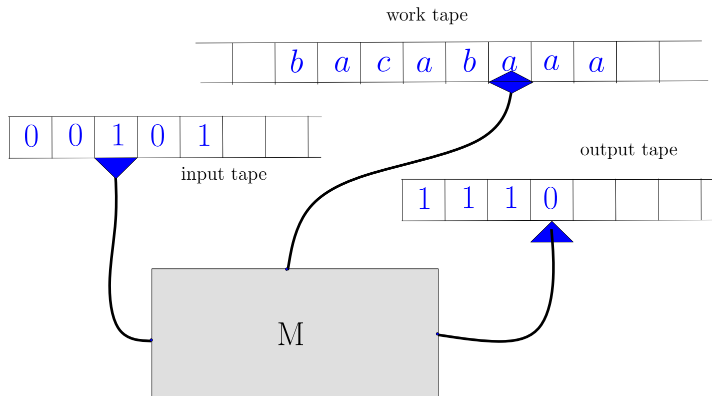
So we have a notion of a function being Turing-computable and a relation being Turing-decidable.

These notions do not change if we modify our definitions slightly:

- one-way infinite tapes
- multiple tapes
- multiple heads
- different input/output conventions

Note that without this kind of robustness our model would be essentially useless.

Input/Output Tapes



Input/Output Tapes

In this model, the

- input tape is read-only, one-way infinite
- output tape is write-only, one-way infinite
- work tape is read/write, two-way infinite

As we will, see this is just the right model for space complexity.

The Important Ideas

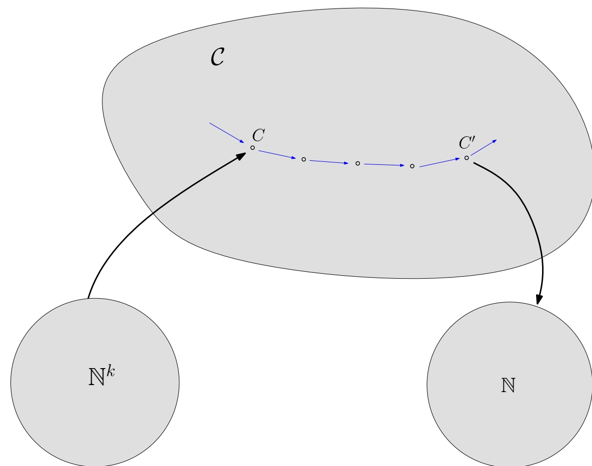
We can specify a model of computation by defining

- a space $\mathcal{C}$ of possible configurations (snapshots),
- a "next configuration" relation,
- an input and output convention.

The details vary, but it's always the same pattern.

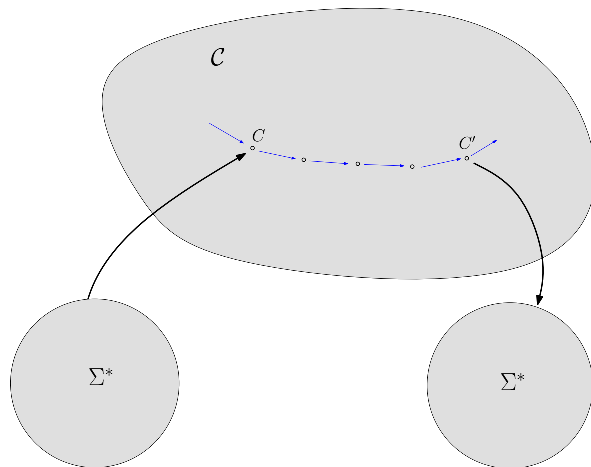
Details about input/output conventions become really important in low complexity classes; higher up they are all interchangeable.

Models of Computation



The number-theoretic scenario (input and output are natural numbers).

Models of Computation, II



The string scenario (input and output are words over some alphabet).

TMs versus RMs

It is tedious but straightforward to show that Turing machines can simulate register machines: store the register contents on the tape, hardwire the RMP in the finite state control of the the Turing machine.

For the opposite direction use sequence numbers to simulate the tape of the Turing machine, hardwire the actual Turing program. This is a bit easier than the URM construction.

Exercises

Exercise

Explain how a multi-tape TM can be simulated by a single-tape TM.

Exercise

Determine how long it takes to recognize palindromes on two tapes versus one tape.

Exercise

Determine the general slow-down caused by switching from two tapes to one.

Exercise

Show that it does not matter whether the tape head is required to return to the standard left position at the end of a computation.

- Turing Machines
- The Busy Beaver Problem
- Wolfram Prize
- Church-Turing Thesis

Busy Beaver Problem

Here is a famous problem due to Tibor Rado (1962). Consider only TMs on tape alphabet $\Sigma = \{0, 1\}$ (where 0 is the blank symbol) and n states.

Question: What is the largest number of 1's any such machine can write on an empty tape, and then halt?

We write $\beta(n)$ for this number and refer to $\beta : \mathbb{N} \to \mathbb{N}$ as the Busy Beaver function.

Halting is crucial, otherwise we could trivially write infinitely many 1's. We will not insist that the 1's are contiguous (that is a variant of the problem with much the same properties).

Variants

It is standard to ignore the halting state in the count, so n means "n ordinary states plus one halting state." There are several variants of the BBP in the literature:

- What is the largest number written in unary (one contiguous block of 1's) that can be written by a halting n-state machine?
- What is the largest number of moves a halting n-state machine can make (time complexity)?
- What is the largest number of tape cells a halting n-state machine can use (space complexity)?

These are all closely related, we will only discuss the "total number of 1's" version.

Exercise

Establish a hierarchy of corresponding Busy Beaver functions.

Busy Beaver $n = 1$

- OK, this is nearly trivial, but still . . .
- $\beta(1) = 1$ because if we were to write a second "1" we would never halt.

Busy Beaver $n = 2$

Amazingly, the answer is no longer obvious: $\beta(2) = 4$.

	0	1
p	(q,1,R)	(q,1,L)
q	(p,1,L)	halt

$$p0 \vdash 1q0 \vdash p11 \vdash q011 \vdash p0111 \vdash 1q111$$

Busy Beaver $n = 3$

How bad can it be?

n	1	2	3	4	5	6
$\beta(n)$	1	4	6	13	≥ 4098	$\geq 4.6 \times 10^{1439}$

Already for $n = 5$ we only have a lower bound, not the exact value.

The champion machine due to Marxen and Buntrock is a small miracle: there are 819 628 286 980 801 possible machines (this is a brute-force count, one can trim this number a bit).

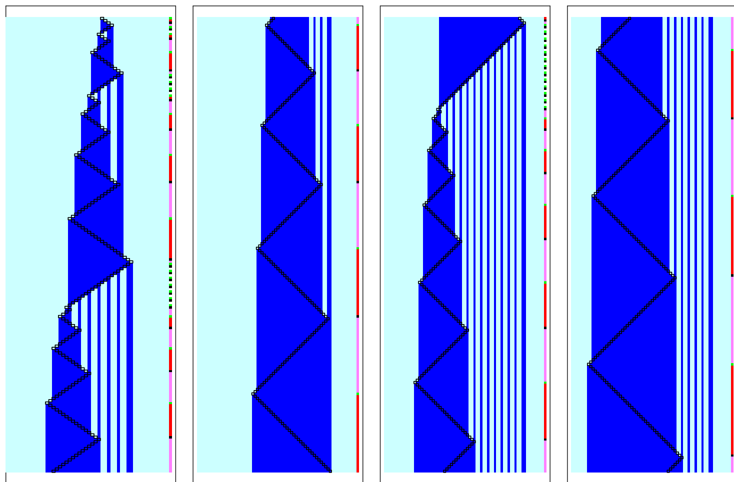
The Marxen-Buntrock Machine

Entry (q, b, X) in position (p, a) means: in state p, scanning an a, go into state q, write b and move X.

	0	1
1	(2,1,R)	(3,1,L)
2	(3,1,R)	(2,1,R)
3	(4,1,R)	(5,0,L)
4	(1,1,L)	(4,1,L)
5	halt	(1,0,L)

Currently this machine is the 5-state champion: no other halting 5-state machine is known that produces more ones.

Marxen-Buntrock 800 Steps



Misleading Pictures

Looking at a run of the Marxen-Buntrock machine for a few hundred or even a few thousand steps one invariably becomes convinced that the machine never halts: the machine zig-zags back and forth, sometimes building solid blocks of 1's, sometimes a striped pattern.

Whatever the details, the machine seems to be in a “loop” (not a an easy concept for Turing machines). Bear in mind: there are only 5 states, there is no obvious method to code an instruction such as “do some zig-zag move 1 million times, then stop”.

Still, this machine stops after 47 176 870 steps.

Why is this Hard?

There are several fundamental obstructions to computing $\beta(n)$.

- No one has a strong theory of the behavior of Turing machines (or any other model of computation).
- Hence, one has to use brute-force search, at least to some degree. But the number of Turing machines on n states grows wildly exponentially.
- Most important is the last problem: Even if we could somehow consider all machines on, say, 10 states, there is the problem that we don't know if a machine will ever halt – it might just keep running forever.
  The Halting Problem is directly responsible for BB being so hard.

Tip of an Iceberg

Define a (n, k)-Turing machine to be a TM that has n states and a tape alphabet of size k.

Clearly, there is a Busy Beaver problem for (n, k) TMs, the standard problem is just the special case $(n+1, 2)$. Very little is known about the general case.

In a similar spirit, one can ask for small values of n and k if there is a universal (n, k) machine. One would expect a trade-off between n and k. Some values where universal machines are known to exist are

$$(24, 2), (10, 3), (7, 4), (5, 5), (4, 6), (3, 10), (2, 18)$$

Busy Beaver Exercises

Exercise

Derive the transition table of the 3-state Busy Beaver machine from the last picture.

Exercise

Give an intuitive explanation of how this machine works.

Exercise

Prove that the last machine is indeed the champion: no other halting 3-state machine writes more than 6 ones.

Exercise (Hard)

Find the Busy Beaver champion for $n = 4$.

Exercise (Extremely Hard)

Organize a search for the Busy Beaver champion for $n = 5$.

- Turing Machines
- The Busy Beaver Problem
- Wolfram Prize
- Church-Turing Thesis

A Prize Question

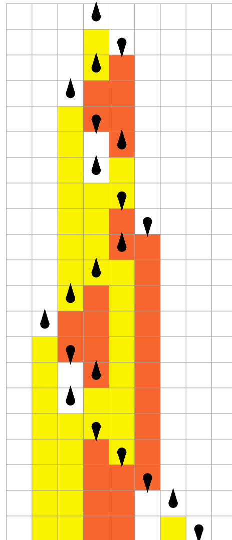
In May 2007, Stephen Wolfram posed the following challenge question:

Is the following (2,3)-Turing machine universal?

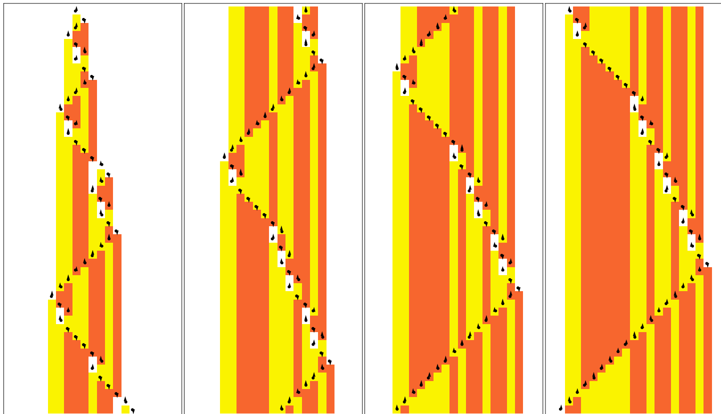
	0	1	2
p	(p,1,L)	(p,0,L)	(q,1,R)
q	(p,2,R)	(q,0,R)	(p,0,L)

Prize money: $25,000.

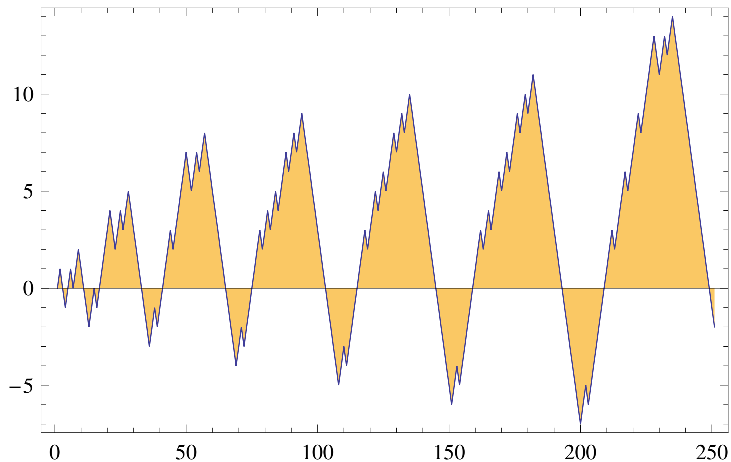
A Run



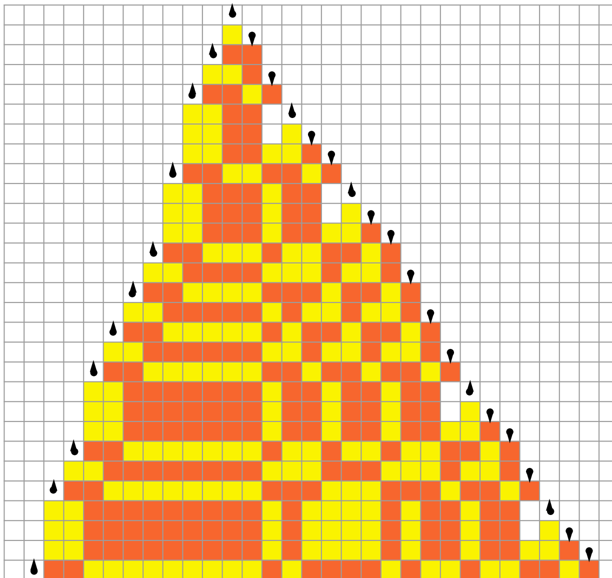
Another



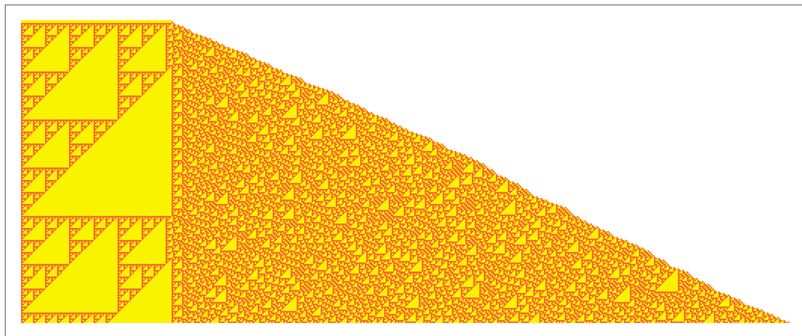
Head Movement



Compressed Computation



Compressed Computation with Different Initial Condition



The Big Difference

- We saw how to construct a universal register machine.
- Likewise we could construct a universal Turing machine.
- But the prize machine is not "designed" to do any particular computation, much less to be universal.
- The problem here is to show that this tiny little machine can simulate arbitrary computations – given the right initial configuration (presumably a rather complicated initial configuration).
- Alas, that's not so easy.

The Big Controversy

- In the Fall of 2007, Alex Smith, an undergraduate at Birmingham, submitted a “proof” that the machine is indeed universal.
- The proof is painfully informal, fails to define crucial notions and drifts into chaos in several places.
- A particularly annoying feature is that it uses infinite configurations: the tape inscription is not just a finite word surrounded by blanks.
- At this point, it is not clear what exactly Smith’s argument shows.

- Turing Machines
- The Busy Beaver Problem
- Wolfram Prize
- Church-Turing Thesis

Models of Computation

There are many other models of computation: general recursive functions, λ calculus, Turing machines, Post systems, Markov algorithms, register machines, random access machines, ...

Historically, Gödel-Herbrand general recursive functions and Church's λ-calculus were the first fully developed models.

Once their equivalence was known, Church unsuccessfully tried to persuade Gödel to accept the following proposition:

Claim

A function is computable in either model iff it is computable in the intuitive sense.

Turing's Contribution

But Gödel was convinced when Turing published his paper in 1936.

Interestingly, he always gave full credit to Turing for having settled the question of what computability really is – but none for Church and himself.

The ideas that all these models of computation reflect the true meaning of computation has become known as the Church-Turing thesis.

Summary

- Turing machines are another model of computation; important historically and the standard model in complexity theory.
- The Busy Beaver function is a good example of computational hardness that appears at very low levels of a hierarchy.
- There are several models of computation that are all equivalent in the sense that they produce the same notion of computability and decidability.
- The Church-Turing thesis states that various formal notions of computability precisely capture the intuitive notion of computability.