Computers: What can't they do!
Computers: What can't they do?

Part 3: TMs (and Undecidability)

November 29, 2011

CS 42: Principles and Practice of Computer Science

The Story So Far...

- ✓ If we want to know what can/cannot be computed, it's enough to consider *decision problems*.
- ✓ Every decision problem corresponds to a *language*, a set of inputs (finite strings) where the answer is yes.
- ✓ DFAs (a.k.a. deterministic finite automata a.k.a. deterministic finite state machines) can model any deterministic mechanism with a finite number of configurations (finite memory).

A Jewel of Theoretical Computer Science



The following are equivalent:

- ✓ There is a DFA accepting the language L.
- ✓ [Rabin and Scott] There's an NFA accepting L.
- ✓ [Kleene] L can be described by a regular expression.

Nonregular Language Theorem

Given a language L, two strings x and y are *distinguishable* if there is some string z such that $xz \in L$ and $yz \notin L$, or vice-versa.

If strings are distinguishable, running them through a state machine for L must lead you to *different* states.

If there is an infinite set of pairwise-distinguishable strings, then none of them can lead you to the same state-machine state. Thus, there cannot by a finite state machine recognizing L.

A language is *regular* if it can be recognized by a finite state machine, and *nonregular* otherwise.

Conclusions

Any computer is a finite state machine.

Hence, there are questions it provably cannot answer. (E.g., "is the input a palindrome?")

But it sure seems like computers can answer these questions (until we run out of memory), so maybe this isn't so helpful in practice?

Perhaps we should consider more powerful models of computation.
(But not too powerful! Why?)

Turing Machines

A simple model of universal computation.

- ✓ Control/Program: a finite state machine
- ✓ Data: An *infinite* tape, with a read/write head.
  Initially blank except for the input string.
- ✓ Steps determined by current state + current symbol;
  Can overwrite current symbol, and move left or right.
- ✓ Can stop (and say yes/no) **or not**.

Why Turing Machines?

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. Turing.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes

Why Turing Machines?

Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, *i.e.* on a tape divided into squares. I shall also suppose that the number of symbols which may be printed is finite. If we were to allow an infinity of symbols, then there would be symbols differing to an arbitrarily small extent†. The effect of this restriction of the number of symbols is not very serious. It is always possible to use sequences of symbols in the place of single symbols. Thus an Arabic numeral such as

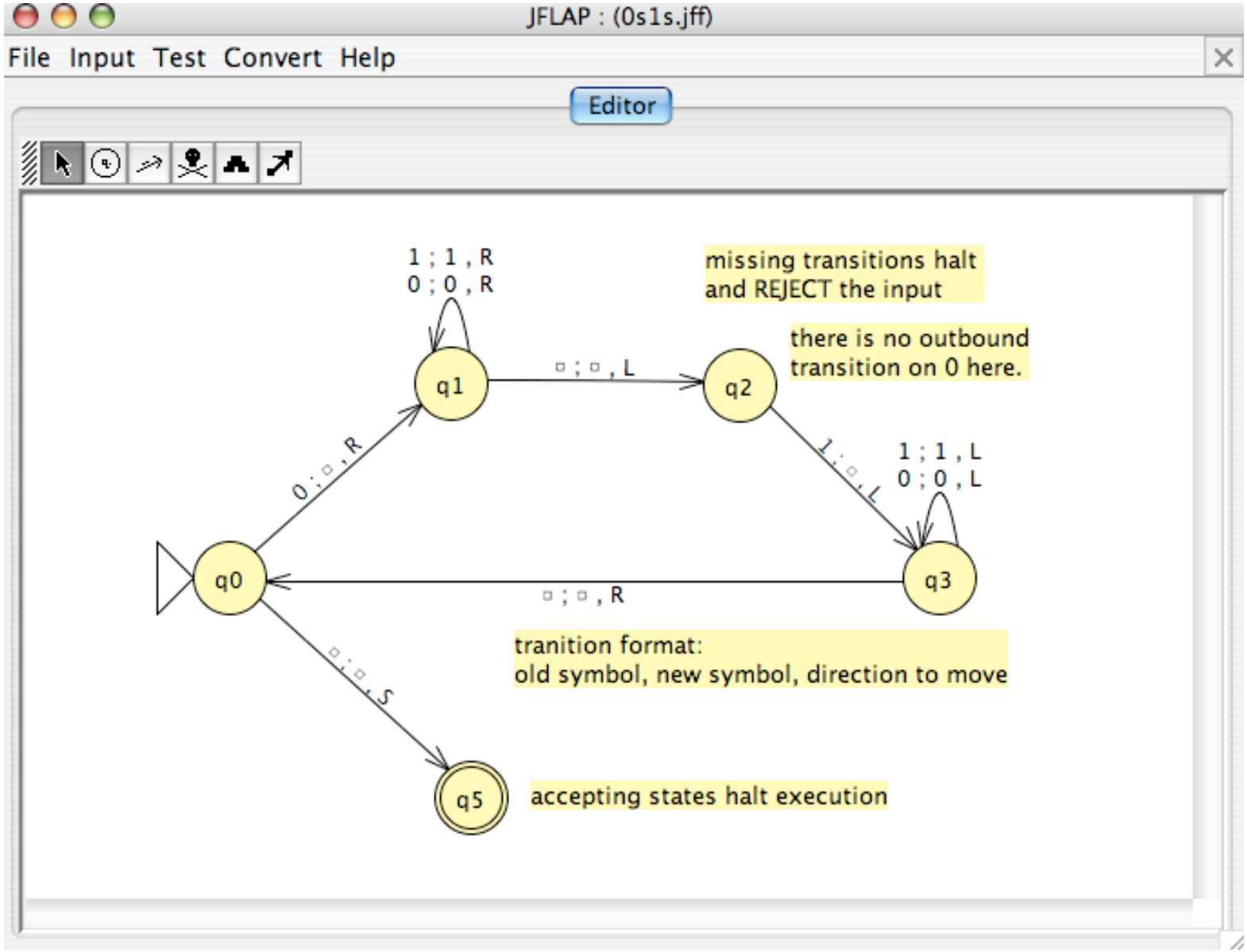
Why Turing Machines?

17 or 999999999999999 is normally treated as a single symbol. Similarly in any European language words are treated as single symbols (Chinese, however, attempts to have an enumerable infinity of symbols). The differences from our point of view between the single and compound symbols is that the compound symbols, if they are too lengthy, cannot be observed at one glance. This is in accordance with experience. We cannot tell at a glance whether 9999999999999999 and 999999999999999 are the same.

The behaviour of the computer at any moment is determined by the symbols which he is observing, and his "state of mind" at that moment. We may suppose that there is a bound B to the number of symbols or squares which the computer can observe at one moment. If he wishes to observe more, he must use successive observations. We will also suppose that the number of states of mind which need be taken into account is finite. The reasons for this are of the same character as those which restrict the number of symbols. If we admitted an infinity of states of mind, some of them will be "arbitrarily close" and will be confused. Again, the restriction is not one which seriously affects computation, since the use of more complicated states of mind can be avoided by writing more symbols on the tape.

Let us imagine the operations performed by the computer to be split up

Turing Machines in JFLAP



Universal Turing Machines

A *Universal Turing Machine* takes a description of any TM on its tape, and simulates that TM.

UTMs can be shown to exist by constructing them.

Think about what would be required.

- ✓ The tape has to hold the tape of the machine being simulated.
- ✓ The tape has to hold the state machine of the machine being simulated.
- ✓ The tape has to hold the current state of the machine being simulated.

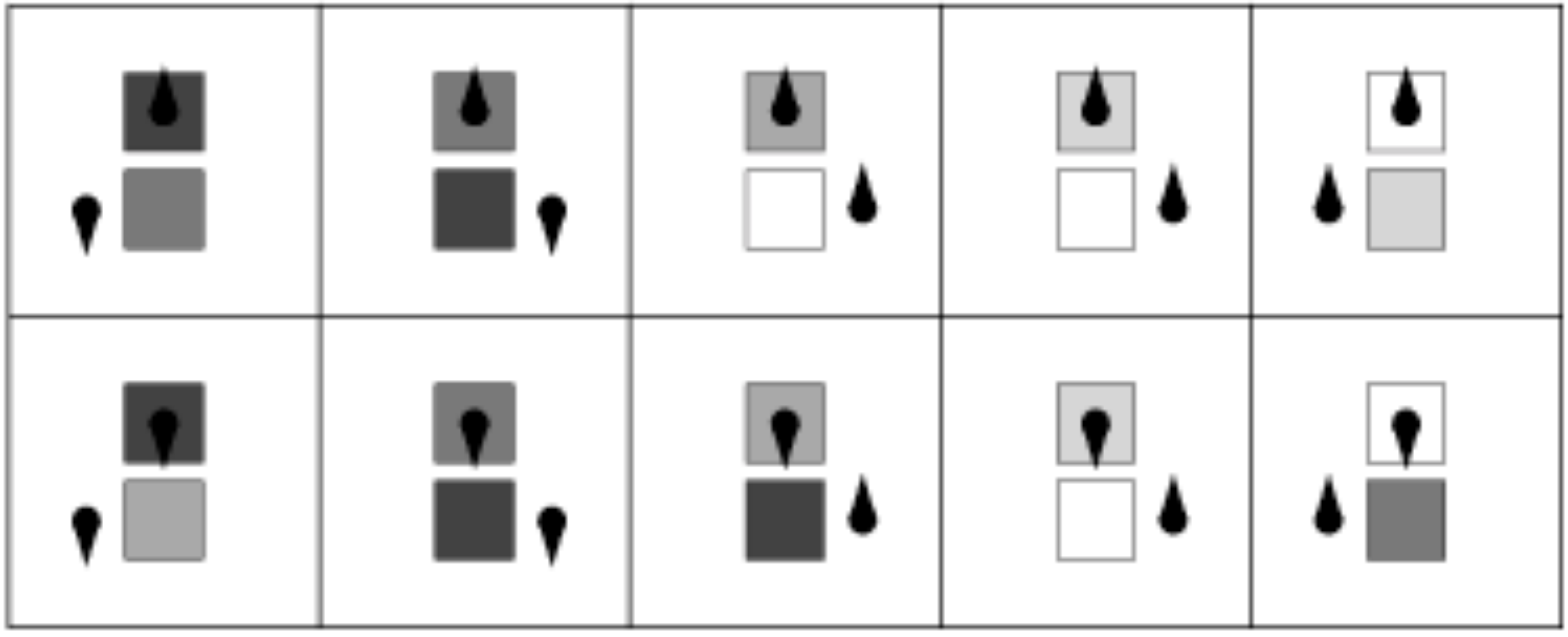
All this is possible, if somewhat laborious to construct.

Specific UTMs

- ✓ The first was constructed by Turing himself.
- ✓ Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine (with lots more states or symbols, respectively).
- ✓ Minsky (1960) gave a 7-state 6-symbol machine.
- ✓ Watanabe (1961) gave an 8-state 5-symbol machine.
- ✓ Minsky (1962) gave a 7-state 4-symbol machine.
- ✓ Rogozhin (1996) gave a 4-state 6-symbol machine
- ✓ Wolfram and Reed (2002) gave a 2-state 5-symbol machine.
- ✓ Smith and Wolfram (2007) gave a 2-state 3-symbol machine.
- ✓ No 2-state 2-symbol UTM exists.

A Specific UTM

2-state, 5 symbol UTM published by Wolfram in 2002



adapted from Wolfram, S. *A New Kind of Science*. Wolfram Media, p. 707, 2002.

TM Variations

The following yield no extra power:

- ✓ Adding the option to write or not on each step.
- ✓ Adding the option to stay-in-place rather than moving L/R.
- ✓ Adding an extra "Erase Entire Tape" ability.
- ✓ Multiple tapes with multiple (independent) read/write heads
- ✓ 2-D infinite tape
- ✓ Nondeterminism (guessing the correct next step)

Can Turing Machines Solve All Problems?

Church-Turing Thesis: any *plausible* model of computation is no more powerful than a Turing Machine.

- ✓ Proof: by lack of counterexample.

TM Reviews

"Two thumbs way up!"
"The Turing Machine is as good as it gets"

—Alonzo Church and Alan Turing

"A woeful Disappointment"
"Missing most of what I'd hope for"

—Georg Cantor

Set Cardinality

What is the size (cardinality) of the following sets? How do we know?

$$\{\ 42,\ 65,\ 1,\ 7\ \}$$
$$\updownarrow \quad \updownarrow \quad \updownarrow \quad \updownarrow$$
$$1 \quad 2 \quad 3 \quad 4$$

$$\{\, 800, 2, 76, 99, 101, 222 \,\}$$

Set Cardinality

What is the size (cardinality) of the following sets? Which is biggest?

$$\{\ 1,\ \ 2,\ \ 3,\ \ 4,\ \ 5\ \ \ldots\ \}$$

$$\{\ 2,\ \ 4,\ \ 6,\ \ 8,\ \ 10\ \ \ldots\ \}$$

$$\{\ 1,\ \ 4,\ \ 9,\ \ 16,\ \ 25\ \ \ldots\ \}$$

$$\{\ \ldots,\ -1,\ 0,\ \ 1,\ \ \ldots\ \ \}$$

Countably Infinite

A set is *countably infinite* if there is a one-to-one mapping between the members of the set and the natural numbers.

Show that the following sets are countably infinite:

- ✓ Powers of two: $\{1, 2, 4, 8, 16, 32, \ldots\}$
- ✓ All nonnegative rational numbers
- ✓ All rational numbers.
- ✓ All finite strings of 0's and 1's.
- ✓ All computer files.
- ✓ All Turing Machines

Uncountable Sets

A set is *countably infinite* if there is a one-to-one mapping between the members of the set and the natural numbers.

Show that the following sets are *not* countably infinite:

- ✓ The real numbers in the interval $[0, 1]$.
- ✓ Languages over the alphabet $\Sigma = \{0, 1\}$.

Some Problems Can't Be Solved with TMs

Given a finite Σ, how many strings are there?

Given a finite Σ, how many languages are there?

How many TMs are there?

QED

A Strategy[1] for CS Programming Assignments[2]

Or: "How to succeed in CS 70 without programming at all!"

1. For each programming assignment, note that the desired program's behavior can be rephrased as a function from the integers $\mathbb{N}$ to the integers $\mathbb{N}$ by encoding the input and output appropriately.

2. Point out that **with 100% probability**, a function chosen from the set of all functions from $\mathbb{N}$ to $\mathbb{N}$ is not computable by any program at all.

3. Proceed to your choice of (a) the mountains, or (b) the beach.

4. Consult the fine print for disclaimers.

---

[1] Please do not mention Prof. Stone when using the Strategy

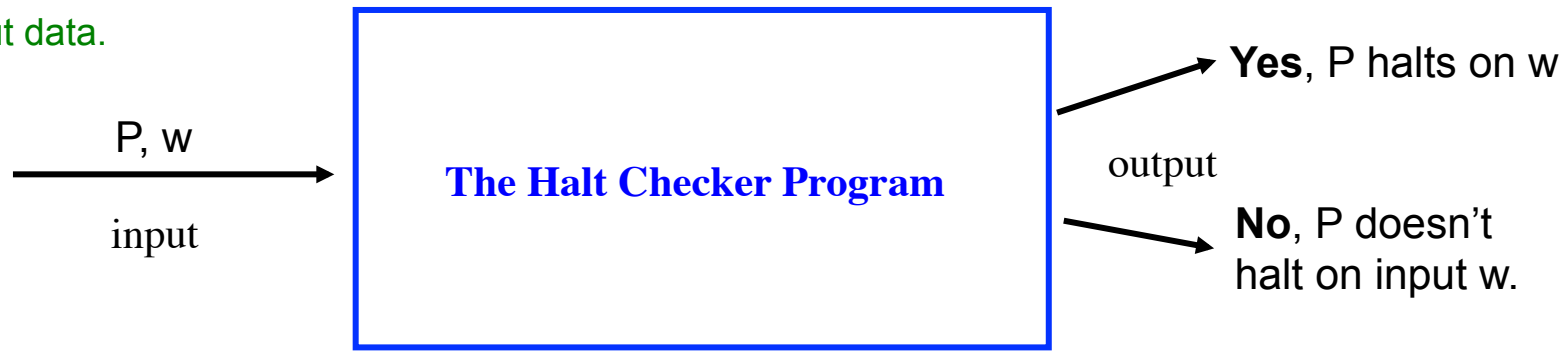
[2] This strategy not applicable in CS 42.

The TM Halting Problem

A specific undecidable problem. Is it obvious that it can't be solved?

P is an input program, i.e., a *turing machine, encoded suitably*.

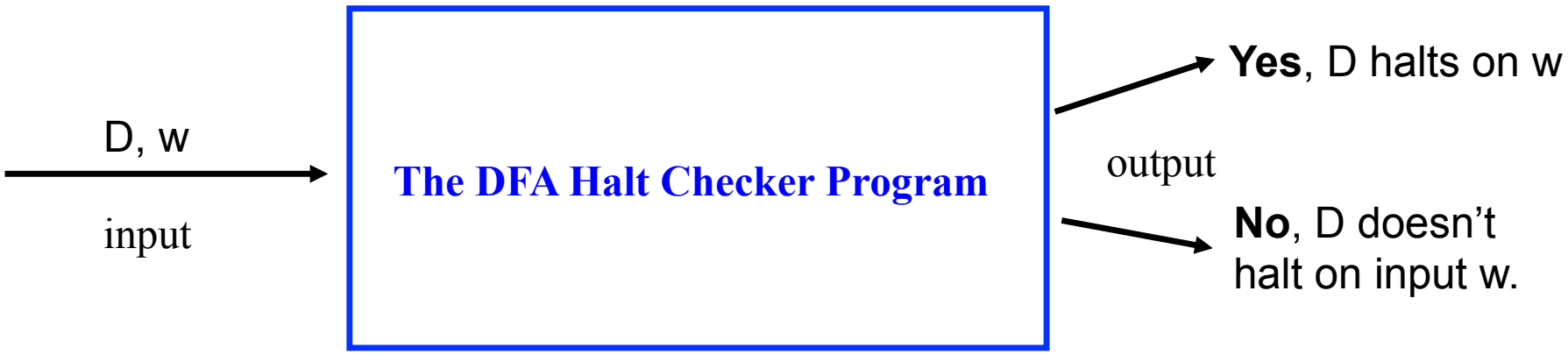
w is P's input data.



Note that this blue halt checker is ***not*** allowed to run forever.

The DFA Halting Problem

Could we solve the problem for DFAs?



Note that this blue halt checker is ***not*** allowed to run forever.

The TM Halting Problem, Revisited

P is an input program, i.e., a *turing machine, encoded suitably*.

w is P's input data.

Note that this blue halt checker is ***not*** allowed to run forever.

Weird things. Are these ok?

- ✓ The blue program takes two inputs, one of which is a program.
- ✓ If the blue program runs $P(w)$, it may never finish!