The Halting Problem for Turing machines

This lecture defines the Halting Problem for Turing machines and proves that it is unsolvable by a Turing machine, and therefore unsolvable by any algorithm (using the Church-Turing thesis.)

Statement of the Halting Problem

For motivation, we first consider the Turing machine with the instructions:

```
((q1, b, q1, 1, R)
 (q1, 0, q42, 0, L)
 (q1, 1, q1, 1, R))
```

If we start this machine on any finite string of 1's, it moves right forever writing 1's as it goes – it does not halt. If instead we start this machine on a string with 0's and 1's, it halts when it reads the first 0. For example:

```
-------------------------------------------------------------
  |     |  1  |  1  |  0  |  1  |     |     |     |     |
-------------------------------------------------------------
           ^
           q1
-------------------------------------------------------------
  |     |  1  |  1  |  0  |  1  |     |     |     |     |
-------------------------------------------------------------
                 ^
                 q1
-------------------------------------------------------------
  |     |  1  |  1  |  0  |  1  |     |     |     |     |
-------------------------------------------------------------
                       ^
                       q1
-------------------------------------------------------------
  |     |  1  |  1  |  0  |  1  |     |     |     |     |
-------------------------------------------------------------
                 ^
                 q42
```

It would be convenient to have an algorithm to determine whether a Turing machine will halt or run forever for a given input tape. This leads to the following question:

```
The Halting Problem:

Given a Turing machine T and string t, does T eventually halt
when started in state q1 with its head on the leftmost
symbol of t and the rest of the tape blank?
```

To formalize this as a computational problem to be solved by a Turing machine, we have to specify the representation of inputs and outputs. For concreteness, we will assume that the Turing machine `T` has tape symbols blank, 0, and 1. The input `t` will just be a string of 0's and 1's.

We represent T as a string of 0's and 1's by translating each character of its representation into a 4-bit code as follows.

```
b = 0000
0 = 0001
1 = 0010
( = 0011
) = 0100
, = 0101
2 = 0110
3 = 0111
4 = 1000
5 = 1001
6 = 1010
7 = 1011
8 = 1100
9 = 1101
L = 1110
R = 1111
```

The translation of the instructions listed for the Turing machine above would begin (omitting the redundant `q`'s from the states):

```
(    (    1    ,    b    ,    1    ,    1    ,    R    )    (    ...
0011 0011 0010 0101 0000 0101 0010 0101 0010 0101 1111 0100 0011 ...
```

For visual assistance, I have put blanks between the 4-bit codes, but you should imagine the result as a string `y` of 144 0's and 1's.

For a Turing machine `H` to solve this problem would mean that if we start `P` on the leftmost symbol of a string `x` of 0's and 1's, `P` eventually halts with just one nonblank symbol on the tape and its head on that symbol. The output symbol should be 1 if in the representation above, the initial portion of `x` represents a Turing machine `T` and the rest of `x` is the string `t`, and `T` eventually halts when started on `t`. Otherwise, the output symbol should be 0. (This also covers the case when the string `x` does not have the correct syntax to represent a Turing machine `T` and input string `t`.)

Recall that `y` is the string that represents the Turing machine in the example above. As examples of the values that should be computed we have the following. The first one represents the example Turing machine with input `111` and the second one the example Turing machine with input `1101`.

```
y111  =>  0
y1101 =>  1
```

We show that the Halting Problem is algorithmically unsolvable. That is, there is no program that always halts and correctly answers 1 or 0 for every possible pair of inputs `T` and `t`. Specifically, we show that no Turing machine can solve this problem, and apply the Church-Turing thesis to conclude that it is algorithmically unsolvable. We also use the term undecidable for algorithmically unsolvable yes/no questions like this one.

The Proof

How do we show that the Halting Problem cannot be solved by a Turing machine? We use a proof by contradiction: we assume to the contrary that there is a Turing machine `H` to solve the Halting Problem, and show that this leads to a contradiction.

We assume `H` exists and has the following property:

```
For any inputs T and t,
H halts and outputs 1 if T halts on input t,
and H halts and outputs 0 if T doesn't halt on t.
```

Note that `H` is required to halt in both cases.

We construct a new Turing machine `Q` consisting of the instructions of `H` with two new instructions added. For concreteness, suppose that state `q86` is the halt state for machine `H` when it outputs a 1. That is, in this case, `H` halts because there is no instruction `(q86, 1, ...)`. Suppose also that state `q152` does not appear in the instructions of `H`. Then, to construct `Q`, we add to the instructions of `H` the following two instructions:

```
(q86, 1, q152, 1, R)
(q152, b, q152, b, R)
```

The effect of these two instructions is to cause `Q` to run forever in those cases in which `H` would have halted with output 1. In those cases in which `H` would have halted with output 0, `Q` still halts with output 0. Thus, the behavior of `Q` is as follows:

```
For inputs on which H halts with output 1, Q doesn't halt.
For inputs on which H halts with output 0, Q halts with output 0.
```

Now we create a new Turing machine, `C` that takes an input string `x` and makes a copy of the string `x` immediately to the right of the string `x`, moves to the leftmost symbol of the string `x`, and halts. One example of input and output of the machine `C` is:

```
-------------------------------------------------------------
  |     |  1  |  1  |  0  |  1  |     |     |     |     |
-------------------------------------------------------------
           ^
           1
-------------------------------------------------------------
  |     |  1  |  1  |  0  |  1  |  1  |  1  |  0  |  1  |
-------------------------------------------------------------
           ^
```

The machine `C` is very similar to the copying machine described in an earlier lecture. Finally, we construct the machine `S` that consists of running `C` followed by running `Q`. To do this, we renumber the states of `Q` so that they don't conflict with the state numbers in the machine `C`, and then we take `S` to be the instructions of `C`, the instructions of `Q` with the states renumbered, and new instructions for the halt state of `C` that transfer to the renumbered version of state `q1` in `Q`. Thus, the machine `S` takes its input `x`, makes a copy of `x` to the right of `x` and runs `Q` on the resulting string, that is, on `xx`.

Then `S` is a Turing machine, represented by the list of its instructions. Translate those instructions into a string of 0's and 1's as above, and call the resulting string `s`. What happens when we run `S` on input `s`, that is, run the machine `S` on the representation of its own instructions? It makes a copy of its input and calls `Q` on the result, that is, it calls `Q` on the input `ss`. Thus:

```
The result of running S on input s is the same
as the result of running Q on input ss.
```

And what is the result of running `Q` on input `ss`?

```
Q fails to halt on input ss if H halts with output 1 on this input.
Q halts with output 0 on input ss if H halts with output 0 on this input.
```

So, what is the result of running `H` on input `ss`? Because `ss` correctly represents the Turing machine `S` and input `s`, we have that

```
H halts with output 1 if Turing machine S halts on input s.
H halts with output 0 if Turing machine S doesn't halt on input s.
```

Recall that this is just what it means for `H` to solve the halting problem.

Thus, working backwards, if Turing machine `S` halts on input `s` then `H` halts with output 1 on input `ss`, and `Q` fails to halt on input `ss`, so `S` fails to halt on input `s`. In other words, if `S` halts on input `s`, then `S` fails to halt on input `s`. (This is not a contradiction, yet, because it might just be that `S` fails to halt on input `s`.)

However, if we assume `S` does not halt on input `s`, then, again working backwards, `H` halts with output 0 on input `ss`, and therefore `Q` halts with output 0 on input `ss`. Thus, `S` with input `s` halts. This completes the contradiction, because we've shown that `S` halts on input `s` if and only if it doesn't halt. Thus, the Turing machine `H` cannot exist. This completes the proof that the Halting Problem is algorithmically unsolvable.

Lest you think that this is just nonsense incurred by the folly of running a program on its own instructions, consider the benefits of having a compiler compile its own code, or a C syntax-checker written in C run on its own code as a test case.