Undecidability

CS254
Chris Pollett
Sep 18, 2006.

Outline

- Diagonalization
- Undecidability of the Halting Problem
- Facts about recursive and r.e. lanaguages

Universal Turing Machines

- It is natural to write Turing Machine programs as strings, say <M>.
- One might hope there is a decision procedure for the Halting problem:

H={ <M,w> | Here <M,w> is a coding as a string for pair M, w where M is a TM and w is a string and M halts on w}

- This language is recursive enumerable. Consider:

U=" On input <M,w>, where M is a TM and w is a string:

1. Simulate M on input w.
2. If M ever enters ia halt state, halt the same way; otherwise keep running"

- The above Turing Machine is called a **Universal Turing Machine (a UTM)** because it can be used to simulate any other Turing machine.
- However, as U on a given input does not necessarily halt, it is not a decision procedure for H.
- It turns out it is impossible to get a decision procedure for H.

Toward showing H is not recursive

- To see this we will use an idea known as diagonalization.
- Recall two sets have the same size if there is a 1-1, onto map (a *bijection*) between them.
- A set *A* is **countable** if there is a bijection between it either the natural numbers or an initial segment of the natural numbers.
- For example,
  - the set {a, b, c, d} is countable --- let f map a -> 0, b->1, c->2, d->3
  - the set {2, 4, 6, 8, …} is countable -- let f(k) --> (1/2)*k
  - the set of finite strings over an alphabet is countable. Map the empty string to 0, then map the strings of length 1 to the next group of natural numbers; then map the strings of length 2; etc.

Diagonalization

- Suppose f is a one-to-one function from a countable set A={a(0), a(1), a(2), …} to sequences of elements over some set B of size at least 2, such that the length of the sequence f(a(i)) is at least i.
- For example,

  f(a(0)) = (1, 0, 1)

  f(a(1)) = (0, 0, 0)

  f(a(2)) = (0, 1, 1)
- Let $f(a(i))_j$ denote the jth element of the sequence f(a(i)).
- The diagonal of this function is the function of f is the sequence $d(f)=(f(a(0))_0, f(a(1))_1, f(a(2))_2, \ldots)$.
- So in this case d(f) = (1, 0, 1).
- Call a sequence d'(f) a **complement** of the diagonal if $d'(f)_i$ is always different from $d(f)_i$.
- For example, for the f above a possible d'(f) is (0, 1, 0).
- The following theorem is an easy consequence of our definition.

Theorem (Diagonalization Theorem) If f satisfies the first bullet above then it does not map any element to a complement of its diagonal.

Example Use of the Diagonalization Theorem

Corollary. A countable set A is not the same size as its P(A).

Proof. Let f:A --> P(A) be a supposed bijection. Since A is countable, we have some function a(k) to list out its elements a(0), a(1), a(2), …An element {a(2), a(5), ..}∈P(A) can be view as an binary sequence (0, 0, 1, 0, 0, 1, …) where we have a 1 if a(i) is in P(A) and a 0 otherwise. So f satisfies the Diagonalization theorem. A complement of the diagonal for f will still be in P(A) but not mapped to by f.

- A set which is not countable is **uncountable**.
- Let **N** be the natural numbers. So P(**N**) is uncountable.

Non Recursively Enumerable Languages

Another corollary to the Diagonalization Theorem is the following:

Corollary. Some languages are not recursive enumerable.

Proof. The set of infinite sequences over {0,1} is uncountable, as we just indicated in the last proof there is a bijection between this set and P(N). On the other hand, each encoding <M> of a Turing Machine is a finite string over a finite alphabet and we argued earlier today that the set of finite strings over an alphabet is countable.

The Halting Problem is not Recursive

Theorem. The language H_{TM}= {<*M,w*> | *M* is a TM and *M* halts on *w*} is not recursive.

Proof. Suppose *H* is a decider for H_{TM}. Fix M_i and consider w's of the form <M_j> for some other TM, M_i. Then listing out encodings of TM's in lex order <M_0>, <M_1>,.. we can create an infinite binary sequence where we have a 1 in the *j*th slot if <M_j> causes M_i to halt and a 0 otherwise. If *H* is a decider H_{TM} then we can consider a variant on the complement of the diagonal of the map f:<M_i> |--> (H(<M_i,<M_0>), H(<M_i,<M_1>>),..). In particular, we can let D be the machine:

D="On input <*M*>, where *M* is a TM:

- Run H on input <*M*, <*M*>>
- If *H* says Yes, then run forever. If *H* says no, then say halt."

Now consider *D*(<*D*>). Machine D halts if and only if *H* on input <D, <D>> rejects. But *H* on input <D, <D>> rejects means that D did not halt on input <D>. This is contradictory. A similar argument can be made about if D does not halt <D>. Since assuming the existence of *H* leads to a contradiction, *H* must not exist. Q.E.D.

Another way to look at this is if you give an *H* which purports to be a decider for H_{TM} then we can give a specific input, <D, <D>>, which is calculated based on *H* on which *H* fails.

An Example of Undecidability

Proposition. The following language is not recursive:
L={<M> | M halts on all inputs}

Proof. Suppose D were a decider for L. Consider the machine M’ which when given an input x, checks if x=w, if it does the machine simulate M(w); otherwise, it halts. Then M’ halts on all inputs iff M halts on w. Further we can build <M’> from <M,w> using a Turing Machine . So given D we could decide H_{TM} by running the procedure:

“On input <M, w>:

(1) Build <M’>

(2) Run D on <M’> and accept if it does; reject otherwise”

Since we know there isn’t a decider for H_{TM} we therefore know D cannot exist.

More facts about Recursive Languages

Proposition. If L is recursive, the so is $\overline{L}$.

Proof. If M is a decider for L reverse its yes no states to get a decider for $\overline{L}$.

Proposition. If L and $\overline{L}$ are recursively enumerable, then L is recursive.

Proof. Let M and $\overline{M}$ be machines for L and $\overline{L}$. Have a tape that is used for a counter. On input x in stage t simulate M for t steps and then simulate $\overline{M}$ for t steps. If M halts with a yes then halt with a yes; if $\overline{M}$ halts with a yes; halt no. Otherwise, go on to stage t+1. At some point one of the two machines must halt.