Undecidability

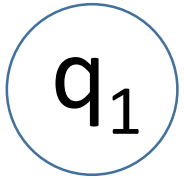
We are going to show that there are problems that can't be solved on any Turing Machine. We need to develop a standard way to represent TMs.

I. You should have learned in Discrete Math that the binary strings (strings of 0s and 1s) are countable. Here is a way to list them: 0, 1, 00, 10,01,11, 000, 100, 010, 110, 001, 101, 011, 111, ... There are many other ways. It doesn't matter which enumeration we use; just choose one and stick to it. Let B_i be the i^{th} binary string in this enumeration.

TMs are allowed to have more than one final state but we can always recode them to have just one.

We can rename the states of any TM q_1, q_2, ...q_n where q_1 is the start state and q_n is the only final state.

It will help to have two standard TMS. *Grumpy* has no final states, so it rejects everything:



Happy is the TM that immediately enters a final state, so it accepts everything:

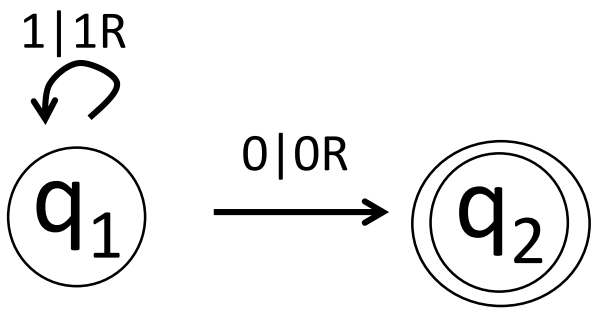
Here is a way to represent a TM as a binary string:

a) Represent state q_i with 0^i
b) Number the tape symbols $X_1..X_n$ and represent X_i with 0^i
c) Encode the directions L as 0^1 and R as 0^2.
d) Encode the transition $\delta(q_i,X_j)=(q_k,X_l,D_m)$ as $0^i10^j10^k10^l10^m$
e) Encode the complete transition function as $t_1 11 t_2 11 ... 11 t_n$ where the ti are the encodings of the individual transitions.
f) Encode the TM with final state t_n as $T1110^n$ where T is the encoding of the transition function.

Example: Grumpy is 111 (no transitions, no final state)

Example: Happy is 1110

Example:



The tape alphabet is {B,0,1} which we'll encode as {0, 00, 000}

Transitions

$\delta(q_1,1)=(q_1,1, R)$ is 01000101000100

$\delta(q_1,0)=(q_2,0, R)$ is 0100100100100

Altogether the TM is 0100010100010011010010010010011100

Since the set of all binary strings is countable, we can count the TMs: Suppose w_i is the i^{th} binary string. Let M_i be the TM represented by w_i if there is one, and Grumpy if there isn't. Every TM will be M_i for some i.

Since M_i is a TM we can ask if it accepts or rejects any string w. In particular we can ask if M_i accepts or rejects its own representation w_i. We know that Grumpy rejects its representation 111 and Happy accepts its representation 1110.

The *diagonal language* $\mathcal{L}_d$ = {w| w=w_i for some i and M_i does not accept w_i}. We know that 111, the representation for Grumpy, is in $\mathcal{L}_d$ as is every string that does not represent a TM.

Theorem: $\mathcal{L}_d$ is not recursively enumerable (i.e., not accepted by a TM)

Proof: Suppose TM M accepts $\mathcal{L}_d$. Since M is a TM it is M_i for some i. Now, is w_i in $\mathcal{L}_d$? If so then M_i accepts w_i (since M=M_i accepts all of $\mathcal{L}_d$.) But then by the definition of $\mathcal{L}_d$ w_i is not in $\mathcal{L}_d$. On the other hand, if w_i is not in $\mathcal{L}_d$ then M_i doesn't accept w_i, so w_i must be in $\mathcal{L}_d$. Any way we go there is a contradiction. So no TM can accept $\mathcal{L}_d$.

We are going to talk about the differences between recursive and recursively enumerable languages. Here are two easy results:

Theorem: If a language is recursive then its complement is also recursive.
Proof: If the language is recursive then it is accepted by a TM that always halts. "Complete" the TM by giving it a "dead" state and replacing every missing transition with a transition to the dead state. Now the TM always halts in either the final state or the dead state. Switching the final and dead states gives a TM for the complement of the language.

Theorem: If a language and its complement are both recursively enumerable then the language is recursive.

Proof: Suppose M1 accepts the language and M2 its complement. Use a 3-tape TM that has string w on tape 1 and simulates the tapes for M1 and M2 on the other two tapes. Simulate the running of M1 and M2 on w. One of them will eventually halt in an accept state. If M1 accepts w then the simulator halts and accepts w. If M2 accepts w then the simulator halts and rejects w. This is TM that always halts that accepts the language.

The *universal language* $\mathcal{L}_u$ is {m1111w | m is the encoding of a TM M and w is an input string and M accepts w}

Theorem: $\mathcal{L}_u$ is recursively enumerable.
Proof: We need a *universal simulator* -- a TM that takes as input the encoding for a TM and simulates it. We use a 3-tape TM. Tape 1 has the input m111w. Tape 2 simulates M's tape. This has two tracks -- one is the contents of M's tape the other has a pointer to the current square on M's tape. Tape 3 has M's current state.

(cont'd next slide)

At the start tape 1 has m1111w and the other tapes are blank. Copy w to tape 2 track 1 and set track 2 to point at its start. Write 0^1 onto tape 3 as the current state.

To take a step look at the current state (tape 3) and current tape symbol (tape 2). Look through m (tape 1) for a transition using these. If you find one update tapes 2 and 3. If you don't find one go to a REJECT state. If tape 3 ever gets a final state, halt and accept m1111w.

Theorem: $\mathcal{L}_u$ is not recursive.
Proof: We'll actually show that the complement of $\mathcal{L}_u$ is not recursively enumerable (as it would have to be if $\mathcal{L}_u$ was recursive).

Suppose there was a TM T that accepted the complement of $\mathcal{L}_u$.
Make a new TM T' so that T'(w) = T(w1111w). T' accepts w if w1111w is *not* in $\mathcal{L}_u$. This means T' accepts w if w is in $\mathcal{L}_d$. We know $\mathcal{L}_d$ is not recursively enumerable, so T' (and hence T) can't exist.

The structure of this argument is important. Suppose we know problem P_1 is not recursive (or RE). If we can show how to turn an instance of P_1 into a problem P_2 (*reducing* P_1 to P_2) then a decider (accepter) for P_2 would also decide (accept) P_1. This means P_2 also can't be recursive (RE).

So far we know $\mathcal{L}_d$ is not recursively enumerable and $\mathcal{L}_u$ is recursively enumerable but not recursive.

Here are some additional concrete languages:
$\mathcal{L}_e$ = {w | w is the encoding of a TM that accepts no strings}
$\mathcal{L}_{ne}$ = {w | w is the encoding of a TM that accepts at least one string}

Note that $\mathcal{L}_e$ is the complement of $\mathcal{L}_{ne}$.

111 (the encoding of Grumpy) is in $\mathcal{L}_e$.
Any binary string that doesn't represent a TM is in $\mathcal{L}_e$.

Theorem: $\mathcal{L}_{ne}$ is recursively enumerable
Proof: Make a non-deterministic TM that, given w writes 111 after w and then writes an arbitrary binary string s. This TM then runs the universal simulator on w111s. If w accepts any string there is a path for this non-deterministic TM to accept w.

Theorem: $\mathcal{L}_{ne}$ is not recursive.
Proof. Suppose TM T decides $\mathcal{L}_{ne}$. We will use T to build a decider for $\mathcal{L}_{u}$. Given a pair (M,w) build a new TM M':

For every x M' accepts x if M accepts w, and rejects x otherwise (i.e., M' ignores its input and simulates M on w.)
If M accepts w then M' accepts all strings. If M does not accept w then M' is in $\mathcal{L}_{e}$. A decider for $\mathcal{L}_{ne}$ will decide if M accepts w and so is a decider for $\mathcal{L}_{u}$, which can't exist.

Moral:

$\mathcal{L}_{ne}$ is recursively enumerable but not recursive.

$\mathcal{L}_{e}$ is not even recursively enumerable.

Rice's Theorem (H.G. Rice, 1951. This was his PhD dissertation):
Let $\mathcal{a}$ be any non-trivial property of recursively enumerable languages. Then $\mathcal{a}$ is undecidable.

What this means: We identify a *property* of languages (such as being nonempty) with the set of TMs that accept the languages with this property. . A *nontrivial* property is one that applies to some but not all languages.

Rice's Theorem, version 2: Let $\mathcal{a}$ be any set of TMs. Let

$\mathcal{a}^*$ = {M | M is a TM that accepts the same language as some TM in $\mathcal{a}$}

Then if $\mathcal{a}^*$ is neither empty nor the set of all TMs A* must be undecidable.

Proof of version 2: First assume $\mathcal{A}^*$ does not include the Grumpy TM G. Since $\mathcal{A}^*$ is not empty, let M* be any TM in $\mathcal{A}^*$. We will use a decider for $\mathcal{A}^*$ to build a decider for $\mathcal{L}_u$.

Given any (M,w) pair construct a new TM M' where

M'(x) = M*(x) is M accepts w
M'(x) = M(w) if M does not accept w

To do this M' first simulate M on w. If M ever accepts w, M' then similates M* on x.

Note that if M does not accept w then M' accepts nothing -- M' is equivalent to the grumpy TM, which we started out assuming is not in $\mathcal{a}^*$. So if M does not accept w then M' is not in $\mathcal{a}^*$. On the other hand, if M does accept w then M' accepts the same language as M*, so M' is in $\mathcal{a}^*$.

Altogether M' is in $\mathcal{a}^*$ if and only if M accepts w. A decider for $\mathcal{a}^*$ gives us a decider for $\mathcal{L}_u$. This can't be, so $\mathcal{a}^*$ must be undecidable.

We have assumed that $\mathcal{a}^*$ does not contain Grumpy. If it does consider the complement of $\mathcal{a}^*$. This is nontrivial and doesn't contain Grumpy. The argument above shows that the complement must be undecidable, so $\mathcal{a}^*$ must be undecidable.

The Post Correspondence Problem (Emil Post, prof at CCNY) Start with two sets of strings $A_1..A_n$ and $B_1..B_m$. Is there a set of indices i_1, i_2, ... i_k so that $A_{i1}A_{i2}..A_{ik} = B_{i1}B_{i2}..B_{ik}$.

For example:

A1 = 1	B1 = 111
A2 = 10111	B2 = 10
A3 = 10	B3 = 0

The solution is 2 1 1 3:

A2A1A1A3 = 101111110 = B2B1B1B3

We won't show it, but the Post Correspondence Problem is undecidable. A number of important "grouping" questions, such as whether a grammar is ambiguous, reduce to PCP.