Undecidabilty

Read K & S 5.1, 5.3, & 5.4.
Read Supplementary Materials: Recursively Enumerable Languages, Turing Machines, and Decidability.
Do Homeworks 21 & 22.

Church's Thesis
(Church-Turing Thesis)

An **algorithm** is a formal procedure that halts.

The Thesis: Anything that can be computed by any algorithm can be computed by a Turing machine.

Another way to state it: All "reasonable" formal models of computation are equivalent to the Turing machine.

This isn't a formal statement, so we can't prove it. But many different computational models have been proposed and they all turn out to be equivalent.

Examples:
- unrestricted grammars
- lambda calculus
- cellular automata
- DNA computing
- quantum computing (?)

The Unsolvability of the Halting Problem

Suppose we could implement the decision procedure

```
HALTS(M, x)
        M: string representing a Turing Machine
        x: string representing the input for M
        If M(x) halts then True
                        else False
```

Then we could define

```
TROUBLE(x)
        x: string
        If HALTS(x, x) then loop forever
                        else halt
```

So now what happens if we invoke TROUBLE("TROUBLE"), which invokes HALTS("TROUBLE", "TROUBLE")

If HALTS says that TROUBLE halts on itself then TROUBLE loops. IF HALTS says that TROUBLE loops, then TROUBLE halts. Either way, we reach a contradiction, so HALTS(M, x) cannot be made into a decision procedure.

Another View

The Problem View: The halting problem is undecidable.

The Language View: Let H =
{"M" "w" : TM M halts on input string w}
H is recursively enumerable but not recursive.

Why?

H is recursively enumerable because it can be semidecided by U, the Universal Turing Machine.

But *H cannot be recursive*. If it were, then it would be decided by some TM MH. But MH("M" "w") would have to be:
If M is not a syntactically valid TM, then False.
else HALTS("M" "w")

But we know cannot that HALTS cannot exist.

If H were Recursive

H = {"M" "w" : TM M halts on input string w}

Theorem: If H were also recursive, then every recursively enumerable language would be recursive.

Proof: Let L be any RE language. Since L is RE, there exists a TM M that semidecides it.

Suppose H is recursive and thus is decided by some TM O (oracle).

We can build a TM M' from M that decides L:
1. M' transforms its input tape from ◊❑w❑ to ◊❑"M""w"❑.
2. M' invokes O on its tape and returns whatever answer O returns.

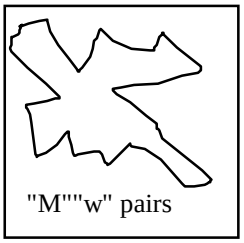
So, if H were recursive, all RE languages would be. **But it isn't.**

Undecidable Problems, Languages that Are Not Recursive, and Partial Functions

The Problem View: The halting problem is undecidable.

The Language View: Let H =
{"M" "w" : TM M halts on input string w}
H is recursively enumerable but not recursive.

The Functional View: Let f (w) = M(w)
f is a partial function on Σ*



Other Undecidable Problems About Turing Machines

- Given a Turing machine M, does M halt on the empty tape?
- Given a Turing machine M, is there any string on which M halts?
- Given a Turing machine M, does M halt on every input string?
- Given two Turing machines M_1 and M_2, do they halt on the same input strings?
- Given a Turing machine M, is the language that M semidecides regular? Is it context-free? Is it recursive?

Post Correspondence Problem

Consider two lists of strings over some alphabet Σ. The lists must be finite and of equal length.

$A = x_1, x_2, x_3, \ldots, x_n$
$B = y_1, y_2, y_3, \ldots, y_n$

Question: Does there exist some finite sequence of integers that can be viewed as indexes of A and B such that, when elements of A are selected as specified and concatenated together, we get the same string we get when elements of B are selected also as specified?

For example, if we assert that 1, 3, 4 is such a sequence, we're asserting that $x_1x_3x_4 = y_1y_3y_4$

Any problem of this form is an instance of the Post Correspondence Problem.

Is the Post Correspondence Problem decidable?

Post Correspondence Problem Examples

i	A	B
1	1	111
2	10111	10
3	10	0

i	A	B
1	10	101
2	011	11
3	101	011

Some Languages Aren't Even Recursively Enumerable

A pragmatically non RE language: L_1={ (i, j) : i, j are integers where the low order five digits of i are a street address number and j is the number of houses with that number on which it rained on November 13, 1946 }

An analytically non RE language: L_2={x : x = "M" of a Turing machine M and M("M") does *not* halt}

Why isn't L_2 RE? Suppose it were. Then there would be a TM M* that semidecides L_2. Is "M*" in L_2?

- If it is, then M*("M*") halts (by the definition of M* as a semideciding machine for L_2)
- But, by the definition of L_2, if "M*" ∈ L_2, then M*("M*") does not halt.

Contradiction. So L_2 is not RE.

Another Non RE Language

$\overline{H}$

Why not?

Reduction

Let $L_1, L_2 \subseteq \Sigma^*$ be languages. A **reduction** from L_1 to L_2 is a recursive function $\tau: \Sigma^* \rightarrow \Sigma^*$ such that

$x \in L_1$ iff $\tau(x) \in L_2$.

Example:

$L_1 = \{a, b : a,b \in N : b = a + 1\}$

$\Downarrow$ τ = Succ

$\Downarrow$ a, b becomes Succ(a), b

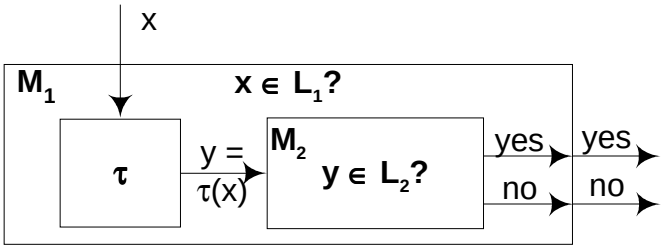
$L_2 = \{a, b : a,b \in N : a = b\}$

If there is a Turing machine M_2 to decide L_2, then I can build a Turing machine M_1 to decide L_1:

1. Take the input and apply Succ to the first number.
2. Invoke M_2 on the result.
3. Return whatever answer M_2 returns.

Reductions and Recursive Languages

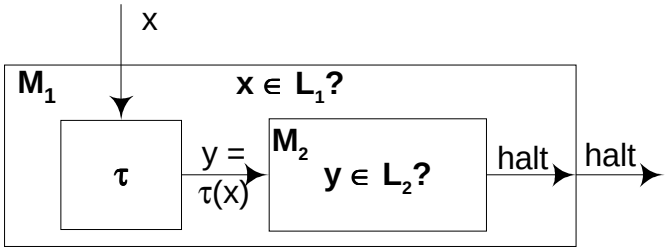
Theorem: If there is a reduction from L_1 to L_2 and L_2 is recursive, then L_1 is recursive.



Theorem: If there is a reduction from L_1 to L_2 and L_1 is not recursive, then L_2 is not recursive.

Reductions and RE Languages

Theorem: If there is a reduction from L_1 to L_2 and L_2 is RE, then L_1 is RE.



Theorem: If there is a reduction from L_1 to L_2 and L_1 is not RE, then L_2 is not RE.

Can it be Decided if M Halts on the Empty Tape?

This is equivalent to, "Is the language L_2 = {"M" : Turing machine M halts on the empty tape} recursive?"

L_1 = H = {s = "M" "w" : Turing machine M halts on input string w}

$\Downarrow$ τ

(?M_2) L_2 = {s = "M" : Turing machine M halts on the empty tape}

Let τ be the function that, from "M" and "w", constructs "M*", which operates as follows on an empty input tape:

1. Write w on the tape.
2. Operate as M would have.

If M_2 exists, then $M_1 = M_2(M_\tau(s))$ decides L_1.

A Formal Reduction Proof

Prove that L_2 = {⟨M⟩: Turing machine M halts on the empty tape} is not recursive.

Proof that L_2 is not recursive via a reduction from H = {⟨M, w⟩: Turing machine M halts on input string w}, a non-recursive language. Suppose that there exists a TM, M_2 that decides L_2. Construct a machine to decide H as $M_1(\langle M, w\rangle) = M_2(\tau(\langle M, w\rangle))$. The τ function creates from ⟨M⟩ and ⟨w⟩ a new machine M*. M* ignores its input and runs M on w, halting exactly when M halts on w.

- $\langle M, w\rangle \in H \Rightarrow$ M halts on w $\Rightarrow$ M* always halts $\Rightarrow \varepsilon \in L(M^*) \Rightarrow \langle M^*\rangle \in L_2 \Rightarrow M_2$ accepts $\Rightarrow M_1$ accepts.
- $\langle M, w\rangle \notin H \Rightarrow$ M does not halt on w $\Rightarrow \varepsilon \notin L(M^*) \Rightarrow \langle M^*\rangle \notin L_2 \Rightarrow M_2$ rejects $\Rightarrow M_1$ rejects.

Thus, if there is a machine M_2 that decides L_2, we could use it to build a machine that decides H. Contradiction. $\therefore L_2$ is not recursive.

Important Elements in a Reduction Proof

- A clear declaration of the reduction "from" and "to" languages and what you're trying to prove with the reduction.
- A description of how a machine is being constructed for the "from" language based on an assumed machine for the "to" language and a recursive τ function.
- A description of the τ function's inputs and outputs. If τ is doing anything nontrivial, it is a good idea to argue that it is recursive.
- Note that machine diagrams are not necessary or even sufficient in these proofs. Use them as thought devices, where needed.
- Run through the logic that demonstrates how the "from" language is being decided by your reduction. You must do both accepting and rejecting cases.
- Declare that the reduction proves that your "to" language is not recursive.

The Most Common Mistake: Doing the Reduction Backwards

The right way to use reduction to show that L_2 is not recursive:

1. Given that L_1 is not recursive,
2. Reduce L_1 to L_2, i.e. show how to solve L_1 (the known one) in terms of L_2 (the unknown one)

L_1 $\Downarrow$ L_2

Example: If there exists a machine M_2 that solves L_2, the problem of deciding whether a Turing machine halts on a blank tape, then we could do H (deciding whether M halts on w) as follows:

1. Create M* from M such that M*, given a blank tape, first writes w on its tape, then simulates the behavior of M.
2. Return M_2("M*").

Doing it wrong by reducing L_2 (the unknown one to L_1): If there exists a machine M_1 that solves H, then we could build a machine that solves L_2 as follows:

1. Return (M_1("M", "")).

Why Backwards Doesn't Work

Suppose that we have proved that the following problem L_1 is unsolvable: Determine the number of days that have elapsed since the beginning of the universe.

Now consider the following problem L_2: Determine the number of days that had elapsed between the beginning of the universe and the assassination of Abraham Lincoln.

Reduce L_1 to L_2:
$L_1 = L_2 + (\text{now} - 4/9/1865)$

L_1 ⇓ L_2

Reduce L_2 to L_1:
$L_2 = L_1 - (\text{now} - 4/9/1865)$

L_2 ⇓ L_1

Why Backwards Doesn't Work, Continued

L_1 = days since beginning of universe
L_2 = elapsed days between the beginning of the universe and the assassination of Abraham Lincoln.
L_3 = days between the assassination of Abraham Lincoln and now.

Considering L_2:
Reduce L_1 to L_2:
$L_1 = L_2 + (\text{now} - 4/9/1865)$

L_1 ⇓ L_2

Reduce L_2 to L_1:
$L_2 = L_1 - (\text{now} - 4/9/1865)$

L_2 ⇓ L_1

Considering L_3:
Reduce L_1 to L_3:
$L_1 = \text{oops}$

L_1 ⇓ L_3

Reduce L_3 to L_1:
$L_3 = L_1 - 365 - (\text{now} - 4/9/1866)$

L_3 ⇓ L_1

Is There Any String on Which M Halts?

$L_1 = H =$ {s = "M" "w" : Turing machine M halts on input string w}

⇓ τ

(?M_2) $L_2 =$ {s = "M" : there exists a string on which Turing machine M halts}

Let τ be the function that, from "M" and "w", constructs "M*", which operates as follows:

1. M* examines its input tape.
2. If it is equal to w, then it simulates M.
3. If not, it loops.

Clearly the only input on which M* has a chance of halting is w, which it does iff M would halt on w.

If M_2 exists, then $M_1 = M_2(M_\tau(s))$ decides L_1.

Does M Halt on All Inputs?

L_1 = {s = "M" : Turing machine M halts on the empty tape}

$\Downarrow \tau$

(?M_2) L_2 = {s = "M" : Turing machine M halts on all inputs}

Let τ be the function that, from "M", constructs "M*", which operates as follows:

1. Erase the input tape.
2. Simulate M.

Clearly M* either halts on all inputs or on none, since it ignores its input.

If M_2 exists, then $M_1 = M_2(M_\tau(s))$ decides L_1.

Rice's Theorem

Theorem: No nontrivial property of the recursively enumerable languages is decidable.

Alternate statement: Let P: $2^{\Sigma^*} \rightarrow$ {true, false} be a nontrivial property of the recursively enumerable languages. The language {"M": P(L(M)) = True} is not recursive.

By "nontrivial" we mean a property that is not simply true for all languages or false for all languages.

Examples:

- L contains only even length strings.
- L contains an odd number of strings.
- L contains all strings that start with "a".
- L is infinite.
- L is regular.

Note:
Rice's theorem applies to languages, not machines. So, for example, the following properties of machines are decidable:

- M contains an even number of states
- M has an odd number of symbols in its tape alphabet

Of course, we need a way to define a language. We'll use machines to do that, but the properties we'll deal with are properties of L(M), not of M itself.

Proof of Rice's Theorem

Proof: Let P be any nontrivial property of the RE languages.

L_1 = H = {s = "M" "w" : Turing machine M halts on input string w}

$\Downarrow \tau$

(?M_2) L_2 = {s = "M" : P(L(M)) = true}

Either P(∅) = true or P(∅) = false. Assume it is false (a matching proof exists if it is true). Since P is nontrivial, there is some language L_P such that P(L_P) is true. Let M_P be some Turing machine that semidecides L_P.

Let τ construct "M*", which operates as follows:

1. Copy its input y to another track for later.
2. Write w on its input tape and execute M on w.
3. If M halts, put y back on the tape and execute M_P.
4. If M_P halts on y, accept.

Claim: If M_2 exists, then $M_1 = M_2(M_\tau(s))$ decides L_1.

Why?

Two cases to consider:

- "M" "w" $\in$ H $\Rightarrow$ M halts on w $\Rightarrow$ M* will halt on all strings that are accepted by M_P $\Rightarrow$ L(M*) = L(M_P) = L_P $\Rightarrow$ P(L(M*)) = P(L_P) = true $\Rightarrow$ M_2 decides P, so M_2 accepts "M*" $\Rightarrow$ M_1 accepts.

- "M" "w" $\notin$ H $\Rightarrow$ M doesn't halt on w $\Rightarrow$ M* will halt on nothing $\Rightarrow$ L(M*) = $\varnothing$ $\Rightarrow$ P(L(M*)) = P($\varnothing$) = false $\Rightarrow$ M_2 decides P, so M_2 rejects "M*" $\Rightarrow$ M_1 rejects.

Using Rice's Theorem

Theorem: No nontrivial property of the recursively enumerable languages is decidable.

To use Rice's Theorem to show that a language L is not recursive we must:

- Specify a language property, P(L)

- Show that the domain of P is the set of recursively enumerable languages.

- Show that P is nontrivial:
  - P is true of at least one language
  - P is false of at least one language

Using Rice's Theorem: An Example

L = {s = "M" : there exists a string on which Turing machine M halts}.
= {s = "M" : L(M) $\neq \varnothing$ }

- Specify a language property, P(L):
  P(L) = True iff L $\neq \varnothing$

- Show that the domain of P is the set of recursively enumerable languages.
  The domain of P is the set of languages semidecided by some TM. This is exactly the set of RE languages.

- Show that P is nontrivial:
  P is true of at least one language: P({ε}) = True
  P is false of at least one language: P($\varnothing$) = False

Inappropriate Uses of Rice's Theorem

Example 1:
L = {s = "M" : M writes a 1 within three moves}.

- Specify a language property, P(L)
  P(M?) = True if M writes a 1 within three moves,
  False otherwise
- Show that the domain of P is the set of recursively enumerable languages.
  ??? The domain of P is the set of all TMs, not their languages

Example 2:
L = {s = "M1" "M2": L(M1) = L(M2)}.

- Specify a language property. P(L)
  P(M1?, M2?) = True if L(M1) = L(M2)
  False otherwise
- Show that the domain of P is the set of recursively enumerable languages.
  ??? The domain of P is RE $\times$ RE

Given a Turing Machine M, is L(M) Regular (or Context Free or Recursive)?

Is this problem decidable?

No, by Rice's Theorem, since being regular (or context free or recursive) is a nontrivial property of the recursively enumerable languages.

We can also show this directly (via the same technique we used to prove the more general claim contained in Rice's Theorem):

Given a Turing Machine M, is L(M) Regular (or Context Free or Recursive)?

$L_1 = H = \{s = \text{"M" "w"} : \text{Turing machine M halts on input string w}\}$

$\Downarrow \tau$

$(?M_2) \quad L_2 = \{s = \text{"M"} : L(M) \text{ is regular}\}$

Let τ be the function that, from "M" and "w", constructs "M*", whose own input is a string
t = "M_*" "w_*"
M*("M_*" "w_*") operates as follows:

1. Copy its input to another track for later.
2. Write w on its input tape and execute M on w.
3. If M halts, invoke U on "M_*" "w_*".
4. If U halts, halt and accept.

If M_2 exists, then $\neg M_2(M^*(s))$ decides L_1 (H).

Why?
If M does not halt on w, then M* accepts $\varnothing$ (which is regular).
If M does halt on w, then M* accepts H (which is not regular).

Undecidable Problems About Unrestricted Grammars

- Given a grammar G and a string w, is $w \in L(G)$?
- Given a grammar G, is $\varepsilon \in L(G)$?
- Given two grammars G_1 and G_2, is $L(G_1) = L(G_2)$?
- Given a grammar G, is $L(G) = \varnothing$?

Given a Grammar G and a String w, Is $w \in L(G)$?

$L_1 = H = \{s = \text{"M" "w"} : \text{Turing machine M halts on input string w}\}$

$\Downarrow \tau$

$(?M_2) \quad L_2 = \{s = \text{"G" "w"} : w \in L(G)\}$

Let τ be the construction that builds a grammar G for the language L that is semidecided by M. Thus
$w \in L(G)$ iff M(w) halts.

Then $\tau(\text{"M" "w"}) = \text{"G" "w"}$

If M_2 exists, then $M_1 = M_2(M_\tau(s))$ decides L_1.

Undecidable Problems About Context-Free Grammars

- Given a context-free grammar G, is $L(G) = \Sigma^*$?
- Given two context-free grammars G_1 and G_2, is $L(G_1) = L(G_2)$?
- Given two context-free grammars G_1 and G_2, is $L(G_1) \cap L(G_2) = \varnothing$?
- Is context-free grammar, G ambiguous?
- Given two pushdown automata M_1 and M_2, do they accept precisely the same language?
- Given a pushdown automaton M, find an equivalent pushdown automaton with as few states as possible.

Given Two Context-Free Grammars G_1 and G_2, Is $L(G_1) = L(G_2)$?

$L_1 =$ {s = "G" a CFG G and $L(G) = \Sigma^*$}

$\Downarrow \quad \tau$

($?M_2$) $L_2 =$ {s = "G_1" "G_2" : G_1 and G_2 are CFGs and $L(G_1) = L(G_2)$}

Let τ append the description of a context free grammar G_{Σ^*} that generates Σ^*.

Then, τ("G") = "G" "G_{Σ^*}"

If M_2 exists, then $M_1 = M_2(M_\tau(s))$ decides L_1.

Non-RE Languages

There are an uncountable number of non-RE languages, but only a countably infinite number of TM's (hence RE languages).
$\therefore$The class of non-RE languages is <u>much</u> bigger than that of RE languages!

Intuition: Non-RE languages usually involve either infinite search or knowing a TM will infinite loop to accept a string.

{⟨M⟩: M is a TM that does not halt on the empty tape}
{⟨M⟩: M is a TM and $L(M) = \Sigma^*$}
{⟨M⟩: M is a TM and there does not exist a string on which M halts}

Proving Languages are not RE

- Diagonalization
- Complement RE, not recursive
- Reduction from a non-RE language
- Rice's theorem for non-RE languages (not covered)

Diagonalization

L={⟨M⟩: M is a TM and M(⟨M⟩) does not halt} is not RE

Suppose L is RE. There is a TM M* that semidecides L. Is ⟨M*⟩ in L?
- If it is, then M*(⟨M*⟩) halts (by the definition of M* as a semideciding machine for L)
- But, by the definition of L, if ⟨M*⟩ $\in$ L, then M*(⟨M*⟩) does not halt.

Contradiction. So L is not RE.

(This is a very "bare-bones" diagonalization proof.)

Diagonalization can only be easily applied to a few non-RE languages.

Complement of an RE, but not Recursive Language

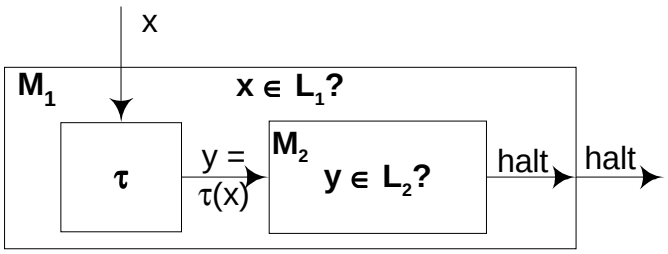
Example**:** $\overline{H} = \{\langle M, w\rangle: M \text{ does not accept } w\}$
Consider $H = \{\langle M, w\rangle: M \text{ is a TM that accepts } w\}$:

- H is RE—it is semidecided by U, the Universal Turing Machine.
- H is not recursive—it is equivalent to the halting problem, which is undecidable.

From the theorem, $\overline{H}$ is not RE.

Reductions and RE Languages

Theorem: If there is a reduction from L_1 to L_2 and L_2 is RE, then L_1 is RE.



Theorem: If there is a reduction from L_1 to L_2 and L_1 is not RE, then L_2 is not RE.

Reduction from a known non-RE Language

Using a reduction from a non-RE language:

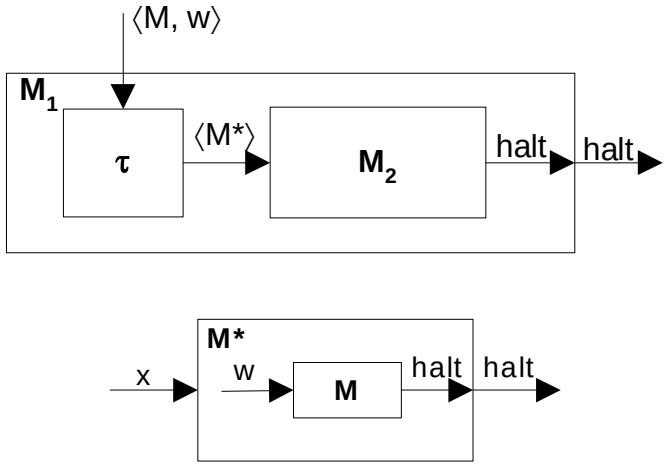
$L_1 = \overline{H} = \{\langle M, w\rangle: \text{Turing machine M does not halt on input string w}\}$

$\Downarrow \tau$

(?M_2) $L_2 = \{\langle M\rangle: \text{there does not exist a string on which Turing machine M halts}\}$

Let τ be the function that, from $\langle M\rangle$ and $\langle w\rangle$, constructs $\langle M^*\rangle$, which operates as follows:

1. Erase the input tape (M* ignores its input).
2. Write w on the tape
3. Run M on w.



$\langle M, w\rangle \in \overline{H} \Rightarrow$ M does not halt on w $\Rightarrow$ M* does not halt on any input $\Rightarrow$ M* halts on nothing $\Rightarrow M_2$ accepts (halts).
$\langle M, w\rangle \notin \overline{H} \Rightarrow$ M halts on w $\Rightarrow$ M* halts on everything $\Rightarrow M_2$ loops.

If M_2 exists, then $M_1(\langle M, w\rangle) = M_2(M_\tau(\langle M, w\rangle))$ and M_1 semidecides L_1. Contradiction. L_1 is not RE. $\therefore L_2$ is not RE.

Language
Summary

IN		OUT
Semidecidable Enumerable Unrestricted grammar	**Recursively Enumerable**	
Decision procedure Lexicicographically enumerable Complement is recursively enumer.	**Recursive**	Diagonalization Reduction
CF grammar PDA Closure	**Context Free**	Pumping Closure
Regular expression FSM Closure	**Regular**	Pumping Closure