Decidability & Rice's Theorem

15-January-2019

Arjun Suresh
For GATE Overflow
Helped by Subarna and Sukanya

Decision Problem

A problem where the answer is yes or no.

Example: Is a given integer divisible by 2?

For numbers, 0, 2, 4, 6, 8, … answer is “yes”

For numbers 1, 3, 5, 7, … answer is “no”

We can get all the “yes” inputs and make a set - this will be the language of the problem. Here, L is the set of all even numbers.

Decision Problem

So, can we always decide a problem?
I.e., given an input whether we can always say “yes” or “no” ?

No, a typical example is HALTING problem.

Given an input Turing machine ‘M’ and an input word ‘w’ for it, we cannot always say “yes” or “no” for whether ‘M’ halts on ‘w’. (Sometimes we can say)

This is equivalent to me giving you an Executable program and its input and asking whether the program will finish running on the input.

- You can try running the program and say “yes” if it does finish. But what if the program keeps on running - can you be sure that it will never finish say even after 100 years?

What causes undecidability?

Our problem is to see if a string belongs to some language.

If we have a DFA or a PDA, we start with a given string and after some finite moves- we either reach a final ACCEPT state or REJECT state.

But what happens for a Turing machine?

It may loop forever and never reaches a FINAL state and this causes Undecidability

So, what makes Turing machine go to infinite loop?

- Its ability to move left and right of the tape and writing capability
- Since the tape is of infinite length this means Turing machine can produce infinite configurations
    - If the TM moves only in one direction on the tape, it cannot have infinite configurations. Because at each step, it can change the tape content - tape alphabet set is finite, change state - no. of states is finite, and move right. After the string is over, TM encounters BLANK symbol and then it can change the state but only “No. of states” times which is finite.
    - But if TM can move left also, then it can keep on changing the contents of the tape -- it works like changing the input string infinite times (no. of possible inputs are infinite). Tape contents can go on changing like below for the input string “aaa” and the input alphabet set {a, b}
      Aaa -> baa -> bbaa -> ….
    - Stack in a PDA is of infinite length, but it cannot cause infinite moves -- similar to one way moving TM

So, now you can imagine that “some inputs” can cause Turing machine to go to infinite loop and then we won’t get any output.

Otherwise output will be “Accept” or “Reject” as in the case of DFA or PDA.

You can consider a computer as a Turing machine and its memory simulating the input tape of the Turing machine.

Some Terminology

HALTING Turing machines

Language accepted by/decided by/recognized by

Recursive Set (Decidable)

Recursively Enumerable Set (Semi-decidable)

- Any problem that is not decidable is undecidable. Hence, all semi-decidable problems that are not decidable comes under undecidable class

Halting Turing Machine

If we take the subset of Turing machines that never goes to infinite loop for ANY input, we get HALTING Turing machines and its corresponding language is called RECURSIVE Set.

- RECURSIVE Set is the set of all Recursive languages - i.e., it is a set of set of strings.
- TM for Recursive set always halts - either accept or reject. So, the set of problems described by Recursive languages is called DECIDABLE.

Recursively Enumerable

Language of Turing machine (halting or otherwise) is recursively enumerable and the set of ALL recursively enumerable languages we call Recursively Enumerable Set. (denoted by RE from now own)

The language of any TM will be an element from RE

Language accepted by

We often say “Language accepted by a Finite Automata or PDA”. Because it means the Automata accepts all strings in the language and rejects everything else.

But for Turing machine this won’t hold as there is also a possibility of looping forever.

“Language recognized” is used to denote that a TM accepts all strings in the language. It may reject strings not in the language or loopforever

“Language decided” is used for halting turing machines -- so it will ACCEPT strings in L and REJECT strings not in L. (It basically decides a problem)

Property of a Set

Consider the set of Natural Numbers N = { 1, 2, 3, 4, …..
Assume x is an element of N for following statements:

Is x divisible by 2? - this is a property of the set and some elements satisfy it and some do not.

Is x/2 an element of N?

Is x+1 an element of N? - satisfied by ALL x

Is 1/(2x) an element of N? - satisfied by NO x

Non-Trivial Property

A property which is satisfied by all elements of a set or NO element of a set is called a TRIVIAL property.

Any other property is called non-trivial.

If P is a non-trivial property of a set S, then some elements of S will satisfy the property and some will violate the property. In First Order logic we can write this as:

$$\exists x, P(x) \land \exists y, \neg P(y)$$

Why non-trivial?

No one cares for trivial stuffs.

One real life example of a set and a property is as follows:

- Consider the set of all GATE 2019 applicants
- Whether an applicant has an application number - is a trivial question because everyone will be having it
- Whether an applicant location is Bangalore is not a trivial question
- For any non-trivial property P, we will have a Pyes instance and a Pno instance

Non-trivial property on RE

Now, we will see some property of RE set. Each element of this set is a recursively enumerable language. So, what can be some properties?

1. Is “0100” an element of L?
2. Are there more than 100 elements in L?
3. Is there a TM, M such that L is accepted by it?
4. Is there a TM, M such that L is not accepted by it?

Properties 3 and 4 are trivial as 3 is true for any recursively enumerable language and 4 is FALSE for any recursively enumerable language

The Useful theorem - Rice's Theorem Part 1

Since we have seen about non-trivial properties on RE, lets see Rice's theorem Part 1

Any non-trivial property of Recursively Enumerable set is **UNDECIDABLE**

i.e., The following problems are undecidable:

1. **If a given recursively enumerable language has a particular string**
2. **If a given recursively enumerable language is finite/regular/context-free**
3. **If number of strings in the given recursively enumerable language is 10 (or any other number)**

Rice’s Theorem Part 1 - Continued

How will be a recursively enumerable language given?

Since it is the language of a TM, it is usually given in the form of TM like below:

- L = {<M> | L(M) has 20 strings }
  Here <M> means an encoding of Turing machine M. Say if we are using binary <M> will be a sequence of 0’s and 1’s which when decoded will give every information about the Turing machine description like the number of states, transitions etc.

Rice's Theorem Part 1 - Examples

1. L = {<M> | L(M) has at least 10000 strings} - non-trivial - undecidable
2. L = {<M> | L(M) has at most 10000 strings} - non-trivial - undecidable
3. L = {<M> | (L(M) is finite} - non-trivial - undecidable
4. L = {<M> | L(M) = Σ* } - non-trivial - undecidable
5. L = {<M> | L(M) is infinite} - non-trivial - undecidable
6. L = {<M> | L(M) is recursive} - non-trivial - undecidable
7. L = {<M> | M is having even number of states} - property of TM, check in the description of <M> and we can say "yes" or "no" -decidable
8. L = {<M> | L(M) is a subset of Σ*} - trivial - decidable - any language is a subset of Σ*
9. L = {<M> | L(M) is a PROPER subset of Σ*} - non-trivial- undecidable
10. L = {<M> | L(M) is empty} - non-trivial - undecidable
11. L = {<M> | L(M) is non-empty} - non-trivial - undecidable

Rice's Theorem Part 1 - Examples Contd.

1. L = {<M> | L(M) has at least 10000 strings}
   L_{yes} = {Any infinite r.e. set or r.e. set having at least 10000 strings} e.g., $(0+1)^*$
   L_{no} = {Any language having less than 10000 strings} e.g. {} , {0} …

For other examples, you can see the L_{yes} and L_{no} at the end of the presentation.

Undecidable but is it semi-decidable?

- Undecidable - for some inputs we are not able to say “yes”/”no”
- Semi-decidable - for all inputs with answer “yes” we can say “yes”
- Decidable - for all inputs we can answer “yes” or “no”.
- So, some problems can be undecidable but still be semi-decidable
- All decidable problems are semi-decidable too.
- Halting problem is an undecidable problem which is semi-decidable

Why not use Intuition first?

Consider the following problem:

Given a graph G, does it have a cycle?

Is this problem decidable? - Yes of course. Just do a DFS and detect presence of backedge.

Likewise if you can give a procedure or algorithm any problem becomes decidable.

What procedure for semi-deciding?

Let's consider the halting problem only.

Given a TM M and a word ‘w’ we need to see if M halts on ‘w’.

Run M on ‘w’. If it halts on ‘w’ after some finite moves it will reach a halt state. Then we can output “yes”. If it continues running without halting, we won't output anything. But at least we semi-decided - gave output when the answer is “yes”.

Likewise if we can give a procedure which works for “yes” case, the problem becomes semi-decidable and its corresponding language becomes recursively enumerable.

Some examples:

All the below languages are undecidable as per Rice's theorem part 1. Here, we check for semi-decidability only

1. L = {<M> | L(M) contains "0"} - semi-decidable. Just simulate M on "0" and say "yes" if M accepts "0".
2. L = {<M> | M halts on 'w'} (Halting problem)- semi-decidable. Like above simulate M on 'w' and say "yes" if M halts. (This is not really a property of recursively enumerable language and hence Rice's theorem cannot be applied directly)
3. L = {<M> | M accepts 'w'} Same like above but M must halt in an ACCEPT state. (This is a property of recursively enumerable language and hence Rice's theorem can be applied directly)

Some more examples:

All the below languages are undecidable as per Rice's theorem part 1. Here, we check for semi-decidability only

1. L = {<M> | L(M) = {0}} - Our checking if M accepts "0" is not enough. We need to be sure M does not accept anything else - so no way to semi-decide -right?
2. L = {<M> | L(M) is regular} - Any way to semi-decide this?
3. L = {<M> | L(M) = {}] - We can say "no" is M accepts some string, but can we every say "yes" as we need to check infinite number of strings?
4. L = {<M> | L(M) contains at least 5 strings} - Give strings one by one to M in some order (lexicographic would do) and say "yes" if M accepts 5 strings. The thing is we cannot wait for M to finish running on some input as it may go to infinite loop. So, we will run each input for some steps, run the next input, go back to previous and run for some more steps and so on (dovetailing). Thus this is semi-decidable
   https://gateoverflow.in/1994/gate2014-2-35
5. L = {<M> | L(M) contains at most 0/no strings} - Same as problem 3
6. L = {<M> | L(M) contains at most 5 strings} - Similar to problem 3
7. L = {<M> | L(M) contains exactly 5 strings} - Similar to problem 3

Rice’s Theorem Part 2

In previous slides we saw some intuitive way to identify if a language is recursively enumerable (semi-decidable). Rice’s theorem part 2 can help here as a proof:

“Any non-monotonic property of recursively enumerable set is not even semi-decidable”

Compared to Part 1, we have

- Non-trivial becoming non-monotonic
- Undecidable becoming not even semi-decidable

Rice's Theorem Part 2

- Part 2 is stronger than Part 1 - i.e., non-monotonicity implies non-triviality and hence whenever we can apply Part 2 of theorem Part 1 can also be applied.

  Lets see what is meant by non-monotonicity.

  Non-trivial condition requires that we have Lyes and Lno. Non-monotonic condition requires an additional condition that

  Lyes SUBSET Lno (is it proper subset?)

  We can take ANY possible Lyes and Lno to ensure the above property. Exists and not Forall as in the below Formal condition:

$$\exists x, \exists y (L_{yes}(x) \wedge L_{no}(y) \mid x \subset y)$$

Rice’s Theorem Part 2 - Examples

1. L = {<M> | L(M) is finite} - non-monotonic, not Turing recognizable
2. L = {<M> | L(M) is infinite} - not non-monotonic, so Rice’s 2nd theorem not applicable - may or may not be recursively enumerable
3. L = {<M> | L(M) is regular} - non-monotonic, not Turing recognizable
4. L = {<M> | L(M) is not regular} - non-monotonic, not recognizable
5. L = {<M> | L(M) = {0} } - non- monotonic, not Turing recognizable
6. L = {<M> | L(M) has at least 100 strings}- not non-monotonic, so Rice’s 2nd theorem not applicable - may or may not be recursively enumerable
7. L = {<M> | L(M) has at most 100 strings} - non-monotonic, not Turing recognizable

Rice's Theorem Part 2 - Examples Contd.

L = {<M> | L(M) is finite}

$L_{yes} = \Phi$ (Any other finite language will also do)

$L_{no} = \Sigma^*$

$L_{yes} \subset L_{no}$ ($\Phi \subset \Sigma^*$) - non - monotonic

∴ L(M) is not Turing recognizable (not RE)

More examples at end of presentation

Rice’s Theorem Part 2 - Converse

If a property of RE set is non-trivial, it is undecidable

If a property of RE set is trivial, it is decidable (Trivial means it)

If a property of RE set is non-monotonic, it is not even semi-decidable

If a property of RE set is not non-monotonic, it may/may not be semi-decidable

- **Part 1 is 2 way**
- **Part 2 is 1 way**

Properties of TM and not its Language

- Whether a given Turing machine has 10 states? Similar to asking if a given C code has 10 lines of code - we can just see the TM description and say yes/no
- Whether a given Turing machine ever makes a LEFT move? - without making a LEFT move Turing machine cannot go into infinite configurations. So, this is decidable.
- Whether a given Turing machine enters a particular state? - State entry problem, undecidable even though it is not a property of RE set.
- Whether a given Turing machine ever outputs a particular symbol? - undecidable even though not a property of RE set

Some Trivial Properties

Just to be clear adding some trivial properties: (these are trivially decidable)

1. L = {<M> | L(M) is recursively enumerable}
2. L = {<M> | L(M) is countable}
3. L = {<M> | L(M) can be accepted by a Turing machine with even number of states}
4. L = {<M> | There exist a deterministic Turing machine which accepts L(M)}

Some NOT non-monotonic Properties

If a property of RE set is NOT non-monotonic, we cannot apply Rice's theorem Part 2 and must use either semi-deciding procedure to show that it is recursively enumerable or use reduction to show it is not recursively enumerable

1. L = {<M> | M accepts "0"} - We do have a semi-deciding procedure
2. L = {<M> | L(M) is infinite} - https://gateoverflow.in/78/l-m-is-infinite
3. L = {<M> | L(M) = Σ^*} - https://gateoverflow.in/79/l-m-%CF%83

Revising Reduction

1. If we reduce halting problem (known to be undecidable) to a problem ‘A’, A becomes undecidable
2. If we reduce complement of halting problem (known to be not even semi-decidable) to a problem ‘A’, A becomes not even semi decidable

End - Examples Continued...

Rice’s Theorem Part 1 - Examples Contd.

L = {<M> | L(M) has at most 10000 strings}

$L_{yes} = \Phi$

L_{no} = Any language more than 10000 strings

∴ **L(M)** is not Turing decidable(not recursive)

Rice’s Theorem Part 1 - Examples Contd.

L = {<M> | (L(M) is finite}

L_{yes} = Φ (or any other finite language)

L_{no} = Σ^*

∴ L(M) is not Turing decidable(not recursive)

Rice’s Theorem Part 1 - Examples Contd.

L = {<M> | (L(M) is infinite}

$L_{yes} = \Sigma^*$

$L_{no} = \Phi$ (or any other finite language)

∴ L(M) is not Turing decidable(not recursive)

Rice’s Theorem Part 1 - Examples Contd.

$L = \{\langle M \rangle \mid L(M) = \Sigma^* \}$

$L_{yes} = \Sigma^*$

$L_{no} = \Phi$ (or any other language)

$\therefore$ L(M) is not Turing decidable(not recursive)

Rice’s Theorem Part 1 - Examples Contd.

L = {<M> | L(M) is a subset of Σ*}

L_{yes} = Any language is a subset of Σ*

E.g : {Φ}, {01, 0101,...}, {10, 1010,...} etc.

We cannot get Lno. Trivial property, and hence no question of undecidability

Rice’s Theorem Part 1 - Examples Contd.

L = {<M> | L(M) is a PROPER subset of Σ*}

$L_{yes} = \Phi$

$L_{no} = \Sigma^*$

Rice's Theorem Part 1 - Examples Contd.

L = {<M> | L(M) is recursive}

L_{yes} = Any recursive language

L_{no} = Any non-recursive language

∴ **L(M)** is not Turing decidable

Rice’s Theorem Part 1 - Examples Contd.

L = {<M> | L(M) is empty}

$L_{yes} = \Phi$

$L_{no} = \{0\}$

∴ **L(M)** is not Turing decidable(not recursive)

Rice's Theorem Part 1 - Examples Contd.

L = {<M> | L(M) is non-empty}

L_{yes} = { Any language of Σ* other than empty language like {Φ}}

e.g: {0} , {01,0101,...} etc.

L_{no} = Φ

∴ **L(M)** is not Turing decidable(not recursive)

Rice’s Theorem Part 2 - Examples Contd.

Rice's Theorem Part 2 - Examples Contd.

L = {<M> | L(M) is regular}

L_{yes} = { ab }

L_{no} = { $a^n b^n$ |n≥1 }

$L_{yes} \subset L_{no}$ (ab $\subset$ $a^n b^n$,where n ≥ 1) - non - monotonic

∴ **L(M)** is not Turing recognizable (not RE)

Rice's Theorem Part 2 - Examples Contd.

L = {<M> | L(M) is not regular}

$L_{yes} = \{ a^n b^n \mid n \geq 0 \}$

$L_{no} = \Sigma^*$

$L_{yes} \subset L_{no}$ ($\{ a^n b^n \mid n \geq 0 \} \subset \Sigma^*$) - non - monotonic

∴ **L(M)** is not Turing recognizable (not RE)

Rice's Theorem Part 2 - Examples Contd.

L = {<M> | L(M) = {0} }

$L_{yes} = \{0\}$

$L_{no} = \Sigma^*$

$L_{yes} \subset L_{no}$ ($\{0\} \subset \Sigma^*$) - non - monotonic

∴ **L(M)** is not Turing recognizable (not RE)

Rice's Theorem Part 2 - Examples Contd.

L = {<M> | L(M) has at most 100 strings}

$L_{yes} = \Phi$

$L_{no} = \Sigma^*$

$L_{yes} \subset L_{no}$ ($\Phi \subset \Sigma^*$) - non - monotonic

∴ **L(M)** is not Turing recognizable (not RE)