Rice's theorem for recursive index sets

- It is more complicated to find out whether $L_S$ is or is not recursively enumerable (rec. enum.).
- Rice has a theorem: $L_S$ is rec. enum. if and only if $S$ satisfies 3 conditions:

Rice's theorem - II

- 1) If $L$ is in $S$ and if $L'$ is any rec. enum. language such that $L \subseteq L'$, then $L'$ has property $S$.
- 2) If $L$ is an infinite language in $S$, then there exists at least one finite subset of $L$ that is in $S$. 
Rice's theorem - III

3) The set of all finite languages in \( \mathcal{S} \) is *enumerable*, i.e. a Turing machine can list all the finite languages in \( \mathcal{S} \).

Non rec. enum. properties - I

- The following properties of rec. enum. sets are NOT rec. enum. (the language \( L_{\mathcal{S}} \) is not rec. enum.)
  
  \( L = \emptyset \) (yes, we already knew \( L_e \) is not recursively enumerable)

  \( L \) is a *singleton*

  \( L = (0+1)^* \)

  \( L \) is a regular lang.

Non rec. enum. properties - II

- Also

  \( L \) is recursive (i.e. \( L_r \) is not rec. enum)

  \( L \) is a CFL

  \( L \) is not recursive (i.e. \( L_{nr} \) is not rec. enum)

  \( L - L_u \neq \emptyset \)

Rec. enum. properties

- The following properties of rec. enum. sets ARE rec. enum. (the language \( L_{\mathcal{S}} \) IS rec. enum.)

  \( L \neq \emptyset \) (i.e. \( L_{ne} \) is rec. enum)

  \( L \) contains at least 10 members

  \( w \) is in \( L \) for some fixed \( w \)

  \( L \cap L_u \neq \emptyset \)
**L = ∅** is not rec. enum. using Rice’s Theorem

- The property \( S \) is \( L = ∅ \)
- Consider \( L' = (0 + 1)^* \)
- We see that \( ∅ \) is in \( S \) (has the property)
- We see that \( ∅ \subseteq (0 + 1)^* \) and \( (0 + 1)^* \) is rec.enum.
- However, \( (0 + 1)^* \) is not in \( S \)
- Hence condition (1) is violated

---

**L ≠ ∅** is rec. enum. using Rice’s Theorem - I

- The property \( S \) is \( L ≠ ∅ \)
- Take any \( L \) that has the property
- Consider any rec. enum. \( L' \) such that \( L \subseteq L' \), clearly \( L' \) is not empty (it has property \( S \))
- Condition (1) is satisfied

---

**L ≠ ∅** is rec. enum. using Rice’s Theorem - II

- Consider an infinite language \( L ≠ ∅ \) (i.e. \( L \) has the property) and pick any element \( w \) in \( L \)
- Then \( \{w\} \) is a finite subset of \( L \) and obviously it has property \( S (\{w\} ≠ ∅) \)
- Condition (2) is satisfied

---

**L ≠ ∅** is not rec. enum. using Rice’s Theorem - III

- We can generate all finite non-empty sets (i.e. all finite sets with property \( S \)) using the same method we used to list all finite sets (just omit the empty set)
- Thus condition (3) is satisfied
  - you generate single strings on one tape and finite languages on another: \( 0\#1\#0\#1\#0\#0\#0\#0\#1\#0\#0\#... \)
  - each new string doubles the set of languages (+1)
The property $L$ is a singleton is not rec. enum. - I

- The property $\mathcal{S}$ is “$L$ has 1 element”
- Consider any such language \{w\}
- Consider $L' = (0 + 1)^*$
- We see that \{w\} $\subseteq (0 + 1)^*$ and $(0 + 1)^*$ is rec.enum.
- However, $(0 + 1)^*$ is not in $\mathcal{S}$
- Hence condition (1) is violated

The property $L$ contains at least 10 members is r.e.I

- The property $\mathcal{S}$ is $L$ has $\geq$ 10 elements
- Consider any rec. enum. $L'$ such that $L \subseteq L'$, clearly $L'$ has more elements that $L$, so certainly $\geq$ 10 elements
- Condition (1) is satisfied

The property $L$ contains at least 10 members is r.e.II

- Consider an infinite language $L$, it certainly has the property of $\geq$ 10 elements
- Pick any 10 elements \{w_1, w_2,\ldots, w_{10}\} of $L$ then and obviously \{w_1, w_2,\ldots, w_{10}\} has property $\mathcal{S}$
- Condition (2) is satisfied

The property $L$ contains at least 10 members is r.e.III

- Generate all finite languages on a tape as described earlier
- as every language is generated, count the elements in the language
- copy all languages with $\geq$ 10 elements on to the output tape
- Condition (3) is satisfied
\[ L = (0+1)^* \] is not rec. enum. using Rice's Theorem

- The property \( \mathcal{S} \) is \( L = (0+1)^* \)
- Take ANY finite \( L' \subseteq (0+1)^* \)
- Obviously \( L' \) is not in \( \mathcal{S} \)
- Hence condition (2) is violated

The property \( w \) is in \( L \) for some fixed \( w \) is r.e.I

- Fix \( w \)
- The property \( \mathcal{S} \) is \( L \) contains \( w \)
- Consider any rec. enum. \( L' \) such that \( L \subseteq L' \), clearly \( L' \) contains \( w \)
- Condition (1) is satisfied

The property \( w \) is in \( L \) for some fixed \( w \) is r.e.II

- Consider an infinite language \( L \) that contains \( w \)
- The set \( \{w\} \) is a finite subset of \( L \) and obviously it has the property \( \mathcal{S} \)
- Condition (2) is satisfied

The property \( w \) is in \( L \) for some fixed \( w \) is r.e.III

- Generate all finite languages as we have described but skip the string \( w \) from the tape that contains the single strings in canonical order
- As each finite set is generated, add \( w \) to it and output it
- Condition (3) is satisfied
Property $L$ is a regular lang. is not r.e.

- Consider $\{0^n1^n : n > 0\}$ and
- $\{01\}$ is regular and $\{0^n1^n : n > 0\}$ is not regular, but $\{01\} \subseteq \{0^n1^n : n > 0\}$
- Condition (1) is violated

Property $L \cap L_u \neq \emptyset$ is r.e. I

- Property $S$ is $L \cap L_u \neq \emptyset$
- Consider any rec. enum. $L'$ such that $L \subseteq L'$, clearly $L' \cap L_u \neq \emptyset$
- Condition (1) is satisfied

Property $L \cap L_u \neq \emptyset$ is r.e. II

- Consider an infinite language $L$ such that $L \cap L_u \neq \emptyset$
- Pick any $w$ in $L \cap L_u$; $\{w\}$ is a finite subset of $L$ and obviously it has the property $S (\{w\} \cap L_u \neq \emptyset)$
- Condition (2) is satisfied

Property $L \cap L_u \neq \emptyset$ is r.e. III

- Put $k = 1$ on tape 1
- Using the method described earlier generate the first non-empty language $L_1 (= \{0\})$ on tape 2
- Run a universal Turing machine $M$ on all the elements of $L_1$ (actually only 0) for $k$ steps (1 step)
- If $M$ accepts an element in $L_1$ within $k$ steps, copy $L_1$ to the output tape 3
Property $L \cap L_u \neq \emptyset$ is r.e. IV

- Increase $k$ on tape 1
- APPEND the $k$-th non-empty finite language $L_k$ to tape 2
- Run $M$ for $k$ steps on all the elements of all the languages on tape 2
- Whenever $M$ accepts a string in a language on tape 2, output that language to tape 3
- and remove the language from tape 2

Property $L \cap L_u \neq \emptyset$ is r.e. V

- go back and increment $k$
- eventually every finite language that intersects $L_u$ will be output on tape 3
- Condition (3) is satisfied
- The order in which languages appear on tape 3 depends on the order in which $M$ discovers elements of $L_u$ and has nothing to do with the canonical order in which their elements were generated

Property $L$ is a recursive lang. is not r.e.-I

- Consider $L_u$ and any element $w$ in $L_u$ then $\{w\} \subseteq L_u$ but $\{w\}$ is recursive and $L_u$ is not recursive (even though it is rec. enum.)
- Condition (1) is violated

Property $L$ is a CFL is not r.e.

- Condition (1) is violated by considering
  $\{0^n1^n : n > 0\} \subseteq \{0^n1^n2^m : n > m \geq 0\}$
- $\{0^n1^n : n > 0\}$ is a CFL
- $\{0^n1^n2^m : n > m \geq 0\}$ is recursive but not a CFL
Property $L$ is not recursive is not r.e.

- Take a $L_u$ and any finite subset $F$ of $L_u$
- $F$ is recursive (it is regular), i.e. it does not have the property but $L_u$ is infinite and does have the property
- Condition (2) fails

Property $L - L_u \neq \emptyset$ is not r.e. I

- The property can be rewritten $L \cap \overline{L_u} \neq \emptyset$
- We show that condition (3) must fail, otherwise $\overline{L_u}$ would be rec. enum.
- Suppose condition (3) holds

Property $L - L_u \neq \emptyset$ is not r.e. II

- Let a Turing machine generate all the finite languages $L$ satisfying $L \cap \overline{L_u} \neq \emptyset$
- Each time a language is generated that has only ONE element, output that element to another tape

Property $L - L_u \neq \emptyset$ is not r.e. III

- In this way we output all the elements of $\overline{L_u}$ to a tape (every element will eventually appear)
- This is impossible, see the next slide
Chapter 7 theorems - I

- It is shown in Chapter 7 that a language \( L \) is recursively enumerable IF AND ONLY IF there is a TM that outputs all elements of \( L \) on a tape in some order.

Chapter 7 theorems - II

- It is also shown in Chapter 7 that a language \( L \) is recursive IF AND ONLY IF there is a TM that outputs all elements of \( L \) on a tape in canonical order.

Closure Properties

- It is easy to show that
  - the intersection of 2 recursive languages is recursive and
  - the intersection of 2 recursively enumerable languages is recursively enumerable.
Proof (intersection of rec. enum.'s)

- Feed an input to the TM for the first language, if it accepts, feed the string to the TM of the second language
- Accept the string if both TM’s accept

Proof (intersection of recursives)

- Feed an input to the halting TM for the first language:
  - if it fails, reject the string
  - if it accepts, feed the string to the halting TM of the second language
  - if the second TM rejects, then reject
  - if both TM’s accept, accept the string

infinite unions (regular)

- The infinite union of regular languages may not be regular:
- union together *all* the singletons \{01\}, \{0^21^2\}, \{0^31^3\}, \{0^41^4\} ...
- each singleton is regular; the union is not: \{0^n1^n : n > 0\}

infinite unions (CFL)

- The infinite union of CFL’s may not be a CFL:
- union together *all* the singletons \{012\}, \{0^21^22^2\}, \{0^31^32^3\}, \{0^41^42^4\} ...
- each singleton is regular, hence a CFL; the union is not: \{0^n1^n2^n : n > 0\}
The infinite union of recursive languages may not be recursive:

- Repeat the same argument as before with \( L_d \) in place of \( L_u \)
- Each singleton is regular; hence recursive; the union is not: \( L_u \)
- Union together all the singletons \( \{w\} \), where \( w \) is in \( L_u \)

finite intersection of CFL's

- Even the intersection to 2 CFL's may not be a CFL
  \[
  \{0^n \mid n \geq 0, \text{any } m\} \cap \{0^m 1^2 \mid n \geq 0, \text{any } m\} = \{0^m \mid n \geq 0\}
  \]

- A grammar for the first language is

\[
S \rightarrow S_2 \mid A
\]

\[
A \rightarrow 0A1 \mid 01
\]

infinite intersections (regular)

- Just take the complements of the sets in the union of regular languages, \( \{01\}, \{0^21^2\}, \) etc.
- The intersection of the complements of the union:

\[
\begin{align*}
\{0^n 1^n \mid n > 0\} & \cup \{0^n 1^n 10^{(0+1)^r} \mid n > m \geq 0\} \\
\{0^n 1^n \mid m > n \geq 0\} & \cup \{0^n 1^n 10^{(0+1)^r} \mid m > n \geq 0\}
\end{align*}
\]

- Which is a CFL but not regular

infinite unions (recursive enum.)

- The infinite union of recursive languages may not be recursive:

\[
\{0^n 1^n \mid n > 0\}
\]
Regular language operations on CFL’s

- We leave the other intersections to think about
- We did not discuss regular expression operations on CFL’s
- The union of two CFL’s is a CFL: add a new start symbol $S_N$ and new productions $S_N \rightarrow S_1 \mid S_2$ where these are the start symbols for the 2 CFL’s

$L_1, L_2$ and $L^*$

- The concatenation of two CFL’s is a CFL: add a new start symbol and a production $S_N \rightarrow S_1S_2$
- The Kleene closure $L^*$ of a CFL $L$ is a CFL: add a new start symbol and productions $S_N \rightarrow SS_N \mid \varepsilon$

Intersection of a CFL and a regular lang. is a CFL

- Build a new PDA from the PDA of the CFL and a DFA of the regular language
- Accept if both reach a final state
- You only need one stack

Another result of TM’s

- Does a Turing machine ever reach a particular state?
- Suppose this were decidable
- Take a TM $<M>$, transform $<M>$ so that if $M$ accepts, it moves to a specific new state $p$ (otherwise it does not go to $p$)
reach the state if and only if accepts

- The new machine accepts its input if and only if it reaches state \( p \)
- We can apply the algorithm to the new machine to see if it reaches \( p \)
- This gives an algorithm to check if \( L(M) \) is not empty
- Impossible

another proof for \( M \) halting on \( w \)

- Suppose there were an algorithm \( A \) to decide if \( M \) halts on \( w \)
- Consider input \(<M,w>\)
- Use \( A \) to check if \( M \) halts on \( w \), reject if it does not
- If \( M \) halts on \( w \), run \( M \) on \( w \), accept \(<M,w>\) if \( M \) accepts \( w \), otherwise reject
- \( L_u \) is recursive!