16-0: **Enumeration Machines**

- An Enumeration Machine is a special kind of Turing Machine:
  - Takes no input
  - Produces strings as output
- Turn it on, and it start spitting out strings

16-1: **Enumeration Machines**

- Enumeration Machine M:
  - M has a a special (non-halting) "output state"
  - Whenever M enters "output state", contents of the tape are output
    - We will insist that the tape is of the form $\triangleright\underline{\sqcup}w$
  - M runs forever, outputting strings
  - $L[M] = \{w : w$ is eventually ouput by $M\}$

16-2: **Enumeration Machines**

- Enumeration machine for a^* (output state is q_{out})

	a	$\sqcup$
q_0		$(q_1, \rightarrow)$
q_1	$(q_1, \rightarrow)$	(q_2, a)
q_2	$(q_2, \leftarrow)$	$(q_{out}, \sqcup)$
q_{out}		$(q_1, \rightarrow)$

16-3: **Enumeration Machines**

- Enumeration machine for ba^*b

16-4: **Enumeration Machines**

- Enumeration machine for ba^*b

	a	b	$\sqcup$
q_0			$(q_1, \rightarrow)$
q_1			(q_2, b)
q_2		$(q_2, \rightarrow)$	(q_3, b)
q_3	$(q_3, \leftarrow)$	$(q_3, \leftarrow)$	$(q_{out}, \sqcup)$
q_4	$(q_4, \rightarrow)$	$(q_4, \rightarrow)$	(q_5, b)
q_5		$(q_6, \leftarrow)$	
q_6		(q_3, a)	
q_{out}			$(q_4, \rightarrow)$

16-5: **Enumeration & Recursive**

- All recursive languages can be enumerated
  - How?

16-6: **Enumeration & Recursive**

- All recursive languages can be enumerated
  - If a language is recursive, there exists some TM M that decides it
  - Generate each string in Σ^* in lexicographic order
  - Run each generated string through M.
    - If M says "yes", output the string

16-7: **Enumeration & r.e.**

- Can a recursively enumerable languages be enumerated?
  - The name does give a hint ...

16-8: **Enumeration & r.e.**

- Recursively enumerable languages can be enumerated!
  - The enumeration method for recursive languages doesn't work
  - Why?

16-9: **Enumeration & r.e.**

- Recursively enumerable languages can be enumerated!
  - We will use the same trick we used to show that a deterministic TM can be simulated by a non-deterministic TM
  - Try first (lexicographicly) string for 1 step
  - Try first and second strings for 2 steps each
  - Try first, second, and third strings for 3 steps each
  - . . . and so on

16-10: **Enumeration & r.e.**

- Recursively enumerable languages can be enumerated
- We have no idea in what order the strings in the language will be output.
  - Why?
- What if we had an enumeration machine that could output the strings of a language L in lexicographical order – what would we know about L?

16-11: **Enumeration & r.e.**

- If an enumeration machine outputs the strings of a language L in lexicographical order, then L is recursive.
- We can write a Turing Machine M that decides L
- To determine if w is in L:
  - Start running the enumeration machine
  - Eventually, either w will be output, or some string that appears after w lexicographically will be output.

16-12: **Enumeration & r.e.**

- If a language can be enumerated by an enumeration machine, it can be semi-decided by a (standard) Turing machine
  - Given an enumeration machine that generates L, how can we create a standard Turing machine that semi-decides L?

16-13: **Enumeration & r.e.**

- Given an enumeration machine that generates L, we can create a standard Turing machine that semi-decides L
  - Move tape head just beyond input
  - Start up enumeration machine
  - Each time a string is output, check to see if it matches input string. If so, halt and accept

16-14: **Recursive & r.e.**

- A Language L is recursive if and only if both L and $\overline{L}$ are recursively enumerable
  - L is recursive if there exists a Turing Machine M that decides L
  - L is recursively ennumerable if there exists a Turing Machine M that semi-decides L

16-15: **Recursive & r.e.**

- A Language L is recursive if and only if both L and $\overline{L}$ are recursively enumerable
  - "only if" (If L is recursive, then L and $\overline{L}$ are recursively enumerable)

16-16: **Recursive & r.e.**

- A Language L is recursive if and only if both L and $\overline{L}$ are recursively enumerable
  - "only if"
    - If L is recursive, then L is recursively enumerable
    - If L is recursive, then $\overline{L}$ is recursive (swap yes/no states), and hence $\overline{L}$ is recursively enumerable

16-17: **Recursive & r.e.**

- A Language L is recursive if and only if both L and $\overline{L}$ are recursively enumerable
  - "if" (If L and $\overline{L}$ are recursively enumerable, then L is recursive)

16-18: **Recursive & r.e.**

- A Language L is recursive if and only if both L and $\overline{L}$ are recursively enumerable
  - "if"
    - Run r.e. machines for L and $\overline{L}$ in parallel
    - Eventually one of them will halt

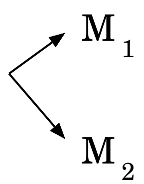
16-19: **Properties of r.e. Languages**

- Are the recursively enumerable languages closed under union?

- Given a Turing Machines M_1 and M_2, can we create a Turing Machine M such that $L[M] = L[M_1] \cup L[M_2]$?

16-20: **Properties of r.e. Languages**

- Are the recursively enumerable languages closed under union?
  - Given a Turing Machines M_1 and M_2, can we create a Turing Machine M such that $L[M] = L[M_1] \cup L[M_2]$?



16-21: **Properties of r.e. Languages**

- Are the recursively enumerable languages closed under intersection?
  - Given a Turing Machines M_1 and M_2, can we create a Turing Machine M such that $L[M] = L[M_1] \cap L[M_2]$?

16-22: **Properties of r.e. Languages**

- Given a Turing Machines M_1 and M_2, can we create a Turing Machine M such that $L[M] = L[M_1] \cap L[M_2]$?
  - First, make a backup copy of w
  - Run M_1 on w. If it halts and accepts ...
  - Restore w from the backup and run M_2 on w
  - Return result of running M_2 on w

16-23: **Properties of r.e. Languages**

- Are the recursively enumerable languages closed under complementation?
  - Given a Turing Machines M, can we create a Turing Machine M' such that $L[M'] = \overline{L[M]}$?

16-24: **Properties of r.e. Languages**

- Given a Turing Machines M, can we create a Turing Machine M' such that $L[M'] = \overline{L[M]}$?
- NO!
  - If L and $\overline{L}$ are r.e., then L is recursive.
  - There are some r.e. languages (the halting problem, for instance) that are not recursive

16-25: **Rice's Theorem**

- Determining if the language accepted by a Turing machine has any non-trivial property is undecidable
- "Non-Trivial" property means:
  - At least one recursively enumerable language has the property

  - Not all recursively enumerable languages have the property

- Example: Is the language accepted by a Turing Machine M regular?

16-26: **Rice's Theorem**

- Problem: Is the language defined by the Turing Machine M recursively enumerable?
  - Is this problem decidable?

16-27: **Rice's Theorem**

- Problem: Is the language defined by the Turing Machine M recursively enumerable?
  - Is this problem decidable? YES!
- All recursively enumerable languages are recursively enumerable.
- The question is "trivial"

16-28: **Rice's Theorem**

- Problem: Does the Turing Machine M accept the string w in k computational steps?
  - Is this problem decidable?

16-29: **Rice's Theorem**

- Problem: Does the Turing Machine M accept the string w in k computational steps?
  - Is this problem decidable? YES!
  - Problem is not language related – we're not asking a question about the language that is accepted, but about the language that is accepted *within a certain number of steps*

16-30: **Rice's Theorem – Proof**

- We will prove Rice's theorem by showing that, for any non-trivial property P, we can reduce the halting problem to the problem of determining if the language accepted by a Turing Machine has Property P.
- Given any Machine M, string w, and non-trivial property P, we will create a new machine M', such that either
  - $L[M']$ has property P if and only if M halts on w
  - $L[M']$ has property P if and only if M does not halt on w

16-31: **Rice's Theorem – Proof**

- Let P be some non-trivial property of a language.
- Two cases:
  - The empty language $\{\}$ has the property
  - The empty language $\{\}$ does not have the property

16-32: **Rice's Theorem – Proof**

- Properties that the empty language has:

  - Regular Languages
  - Languages that do not contain the string "aab"
  - Languages that are finite
- Properties that the empty language does not have:
  - Languages containing the string "aab"
  - Languages containing at least one string
  - Languages that are infinite

16-33: **Rice's Theorem – Proof**

- Let M be any Turing Machine, w be any input string, and P be any non-trivial property of a language, such that $\{\}$ has property P.
- Let L_{NP} be some recursively enumerable language that does *not* have the property P, and let M_{NP} be a Turing Machine such that $L[M_{NP}] = L_{NP}$
- We will create a machine M' such that M' has property P if and only if M does not halt on w.

16-34: **Rice's Theorem – Proof**

- M':
  - Save input
  - Erase input, simulate running M on w
  - Restore input
  - Simulates running M_{NP} on input

16-35: **Rice's Theorem – Proof**

- M':
  - Save input
  - Erase input, simulate running M on w
  - Restore input
  - Simulates running M_{NP} on input
- If M halts on w, $L[M'] = L_{NP}$, and $L[M']$ does not have property P
- If M does not halt on w, $L[M'] = \{\}$, and $L[M']$ does have property P

16-36: **Rice's Theorem – Proof**

- Let M be any Turing Machine, w be any input string, and P be any non-trivial property of a language, such that $\{\}$ *does not* have property P.
- Let L_{NP} be some recursively enumerable language that *does* have the property P, and let M_P be a Turing Machine such that $L[M_P] = L_P$
- We will create a machine M' such that M' has property P if and only if M does halt on w.

16-37: **Rice's Theorem – Proof**

- M':
  - Save input
  - Erase input, simulate running M on w
  - Restore input
  - Simulates running M_P on input

16-38: **Rice's Theorem – Proof**

- M':
  - Save input
  - Erase input, simulate running M on w
  - Restore input
  - Simulates running M_P on input
- If M halts on w, $L[M'] = L_P$, and $L[M']$ does have property P
- If M does not halt on w, $L[M'] = \{\}$, and $L[M']$ does not have property P

16-39: **Undecidability**

- How many undecidable languages are there?
  - A language is a set of strings
  - Set of all languages over Σ^* is the set of all subsets of Σ^*
  - Set of all languages over Σ^* is 2^{Σ^*}

16-40: **Undecidability**

- How many different langauges over Σ^* are there?
  - The set of all langauges over an alphabet Σ is 2^{Σ^*}
  - There is a bijection between strings and integers (lexigraphic ordering)
  - Thus, the number of different languages is $2^{\Sigma^*} = |2^{\mathbf{N}}|$
- How big is $2^{\mathbf{N}}$?

16-41: **Undecidability**

- $2^{\mathbf{N}}$ is uncountable
- Proof by contradiction (yet another diagonalization!)
  - Assume that there is a bijection between N and $2^{\mathbf{N}}$
  - Show that there must be an element of $2^{\mathbf{N}}$ that is not in the bijection – contradiction!

16-42: $2^{\mathbf{N}}$ **is Uncountable**

- Assume that there is a bijection between $\mathbf{N}$ and $2^{\mathbf{N}}$

0	$\{100, 8, 6\}$
1	$\{0, 1, 2, 3, 9, 11, 22\}$
2	$\{\}$
3	$\{2, 4, 6, 8, 10, 12, 14, 16, 18, 20, \ldots\}$
4	$\{2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, \ldots\}$
5	$\{3, 9, 11, 23, 54, 128\}$
...	...

- We will find an element of $2^{\mathbf{N}}$ that is not in the bijection

16-43: **$2^{\mathbf{N}}$ is Uncountable**

- Let S_n be the set mapped to by n in the bijection
- Consider the set $S = \{x : x \in \mathbf{N}, S \notin S_x\}$

0	$\{100, 8, 6\}$
1	$\{0, 1, 2, 3, 9, 11, 22\}$
2	$\{\}$
3	$\{2, 4, 6, 8, 10, 12, 14, 16, 18, 20, \ldots\}$
4	$\{2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, \ldots\}$
5	$\{3, 9, 11, 23, 54, 128\}$
...	...

$S = \{0, 2, 3, 5, \ldots\}$

16-44: **# of Turing Machines**

- How many Turing Machines are there?
  - Any Turing Machine can be represented by a finite string of 0's and 1's
  - There is a bijection between set of all Turing Machines and $\mathbf{N}$
  - Countable # of Turing Machines

16-45: **Undecidability**

- Each language represents a problem
- Each Turing Machine represents a solution to a problem
- There are a countable number of Turing Machines, and an uncountable number of languages
  - Vastly more undecidable problems than decidable problems!