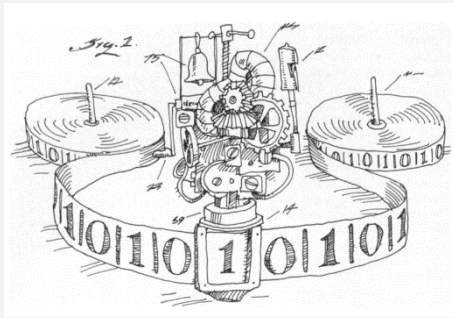
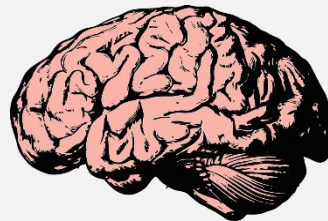
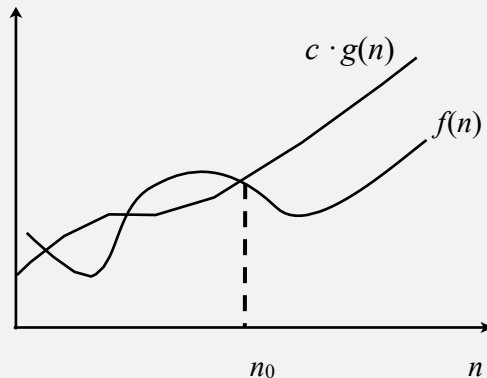


COMP 3602

Intro to Theory of Computation

Lecture 10



Logistics

- Test 1 day/time: Mon Feb 9
 - Written test on Turing machines, automata/regex, PDA/CFGs, pumping lemmas
 - Practice problems are up
 - One **letter-size double-sided handwritten** cheatsheet allowed
- First oral exam: as soon as we finish the next topic
- Presentation proposals?
- Drills are up, some due Feb 9th .
 - Do the sample drills first!
 - You can do the drills multiple times, but the grade is an average of all attempts
 - So if you open it and let it time out, it will add a 0 to your average!

Practice problems questions?

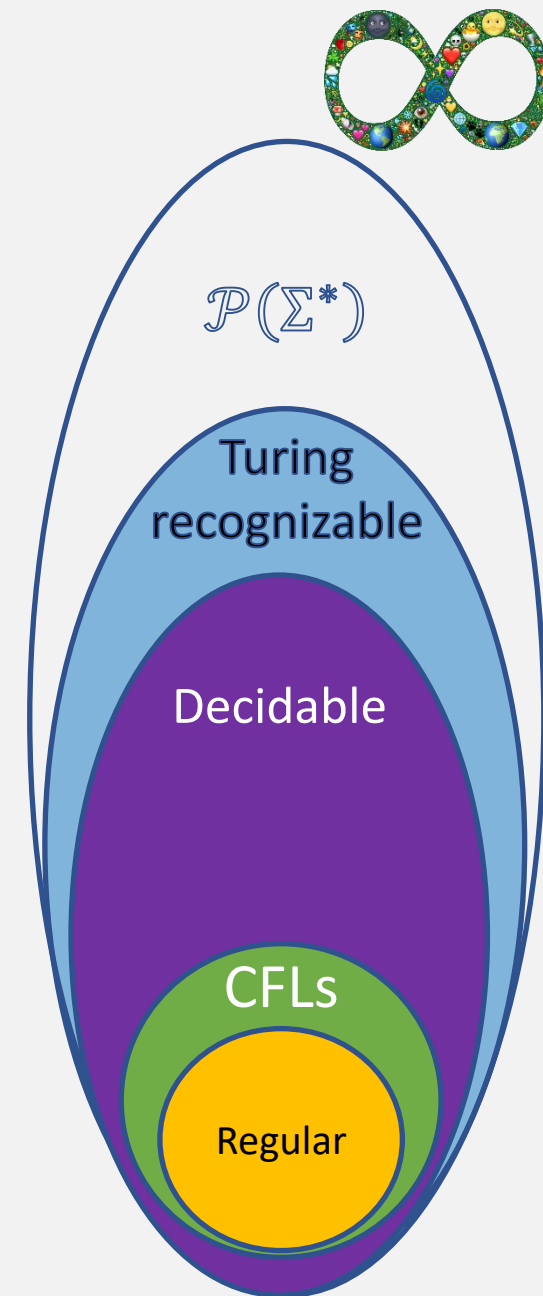
1. Give regular expressions and draw automata (DFAs or NFAs) for the following languages:
 - a. $\{w \in \{0,1\}^* \mid w \text{ consists only of the string } 010\}$
 - b. $\{w \in \{0,1\}^* \mid w \text{ contains a substring } 010\}$
 - c. $\{w \in \{0,1\}^* \mid w \text{ contains a substring } 010, \text{ starts with } 0 \text{ and ends with } 1\}$
 - d. $\{w \in \{0,1\}^* \mid w \text{ does not contain a substring } 010\}$
 - e. $\{w \in \{a,b,c\}^* \mid w \text{ contains exactly 2 "c" and at least one "a" }\}$
 - f. $\{w \in \{a,b,c\}^* \mid \text{length of } w \text{ is divisible by 5 and it ends at "a" }\}$
2. Give NFAs with the specified number of states for the following languages, then use the subset construction to get DFAs out of them.
 - a. $L = 1^*$, one state.
 - b. Strings with all 1s (if any) in the same substring (That is, strings without a substring 100^*1). Three states.
 - c. $0^*(1000^*)^*$ with three states (here, "1000*" means 100 followed by 0^*)
3. Give a pushdown automaton and a context-free grammar for the following languages (for extra practice, try writing grammars for languages above)
 - a. $\{w \in \{0,1\}^* \mid \text{the length of } w \text{ is odd}\}$
 - b. $\{w \in \{a,b,c\}^* \mid w \text{ contains more c's than a's}\}$
 - c. $\{w \in \{0,1\}^* \mid w \text{ is a palindrome or } w=0^n u 0^n \text{ for some } n, \text{ where } u \in \{0,1\}^* \}$
4. Show using the pumping lemmas that
 - a. $\{a^n b \{a,b,c,d\}^* c d^{2n}\}$ is not regular.
 - b. $\{w \in \{0,1\}^* \mid \text{contains a substring } 10 \text{ or } w=0^n 1^m \text{ where } n \neq m \}$ is not regular. Hint: use closure properties of regular languages.
 - c. $\{ww^r w \mid w = w_1 \dots w_n \in \{0,1\}^* \text{ where } w^r = w_n \dots w_1 \text{ is the reverse of } w \}$ is neither regular nor context-free.
 - d. $\{1^{n^2}\}$ is neither regular nor context-free.

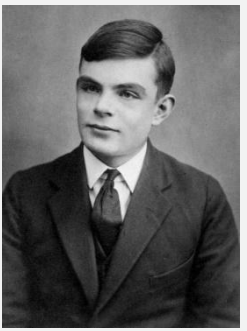
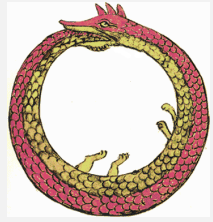
5. Describe Turing machines for the following languages. Use the definition of Turing machines with a single 2-way infinite tape, and movement directions $\{L,R\}$. Make sure to include both the informal outline of the algorithm, and the full description, that is, Σ , Γ , Q , q_0 , q_a , q_r and δ . Do not draw diagrams for Turing machines! Hint: first look at a few examples of strings that are in the language and strings that are not in the language, and don't forget the empty string.
 - $L_1 = \{w \in \{b,c,d\}^* \mid \text{non-empty strings of length at least 3 with the last three symbols all different.}\}$
 - $L_2 = \{w \in \{0,1\}^* \mid w=0^n1^n0^{2n} \text{ for } n \geq 0 \}$. That is, L_2 consists of all strings of length divisible by 4 (including length 0), where the first quarter of the string is all 0s, second quarter is all 1s, and the second half all 0s again.
6. Describe a Turing machine that decrements by 2 a natural number ≥ 2 written on its tape in binary. It should end in q_a pointing to the first symbol of the output if the tape contained a string starting with 1 encoding a natural number $x \geq 2$, and in q_r otherwise. Give both an informal algorithm and a full description of the TM.
7. Consider a TM where instead of "move left one cell" and "move right one cell" the only available moving operations are "jump left 11 steps", "jump right 5 steps" and "stay at the same place"; let's call this model 11L5RS Turing machines. Just as in the model of TM we defined in class, 11L5RS TMs have one two-way-infinite tape, the only difference is the movement operations.
 - Show that 11L5RS TM model is equivalent to the model we defined in class. Do both directions, with constructions (from M construct M'), and an outline of a proof why your construction works.
 - Show that if instead it were "jump left 10 steps" "jump right 5 steps", then this 10L5RS TM model would not be able to simulate the standard TM.
8. Show that single-tape Turing machines that only move right recognize exactly the regular languages. Hint: do both directions of the equivalence, and remember that the Turing machine is not guaranteed to stop.
9. Show that finite automata with two stacks can simulate Turing machines.

Do not draw diagrams for Turing machines!

Questions/comments?

- *Lemma:* Any Turing machine can be encoded as a binary string
 - We will use the notation $\langle M \rangle$ to mean a binary string encoding M
- *Theorem:* Set of all Turing machines is **countable**.
 - Proof: match encodings of TMs to binary numbers
- *Theorem:* Set of all languages is **uncountable**.
 - Proof: by diagonalization
- *Corollary:* There exists a language not recognizable by any TM
 - In fact, there are uncountably many such languages!
- *Definition:*
 - A language L is *undecidable* if no Turing machine decides it.
 - It is not Turing-recognizable if no Turing machine recognizes it.





The universal language

- Any Turing machine M can be encoded as a binary string $\langle M \rangle$
 - So it can be given as an input to another Turing machine.
- Pairs (tuples, finite sequences) of strings can also be encoded as strings.
 - Let $\langle M, w \rangle$ be an encoding of a pair (Turing machine M , a string w)

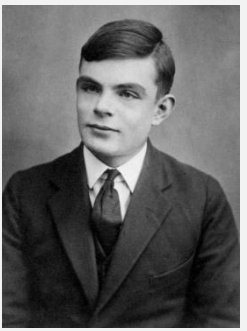
Definition: the universal language

$$A_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM and it accepts a string } w\}$$

That is, the universal language consists of all pairs $(TM\ M, string\ w)$ where M accepts w .

- If $x \neq \langle M, w \rangle$, then $x \notin A_{TM}$
- If $x = \langle M, w \rangle$ where M rejects w , or goes into infinite loop on w , then $x \notin A_{TM}$

The universal Turing machine

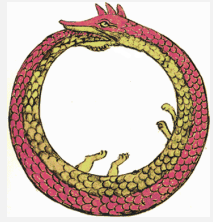


- *Definition:* the universal language
 $A_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM and it accepts a string } w\}$

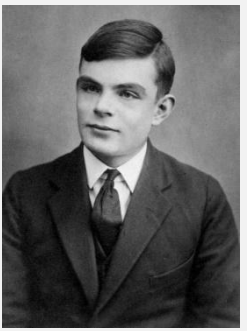
Claim: A_{TM} is Turing-recognizable

Proof: Let $M_{A_{TM}}$ work as follows:

$M_{A_{TM}} =$ " On input x
If $x \neq \langle M, w \rangle$, **reject**.
Simulate M on input w
If M ever enters accept state, **accept**.
If M ever enters reject state, **reject**. "



Undecidability of the universal language



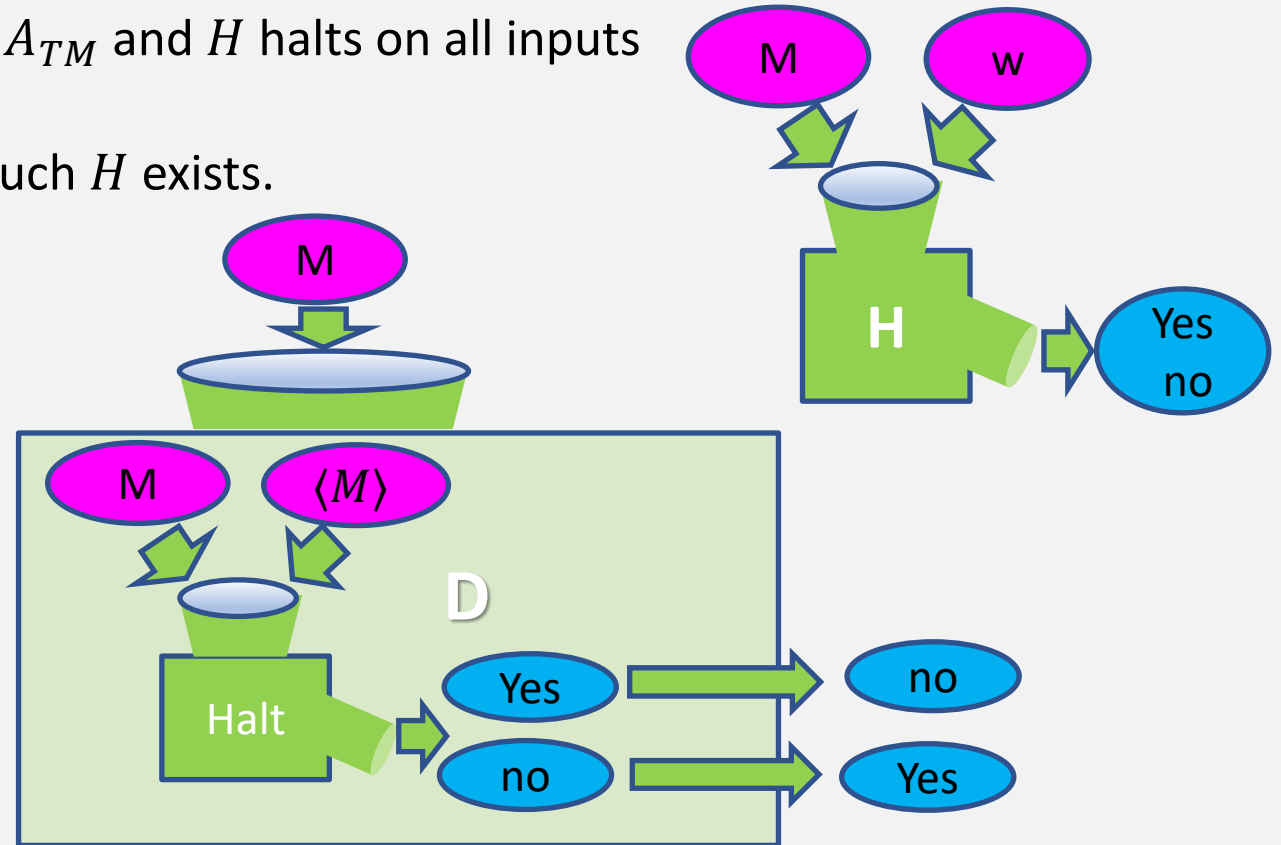
Theorem: A_{TM} is undecidable.

- That is, there is no TM H such that $\mathcal{L}(H) = A_{TM}$ and H halts on all inputs
 - Proof: by contradiction.
 - Suppose for the sake of contradiction that such H exists.
- Construct a Turing machine D as follows

D = “ On input x
 If $x \neq \langle M \rangle$, *reject*.
 Run H on input $\langle M, \langle M \rangle \rangle$
 If H accepts, *reject*.
 If H rejects, *accept*. ”

Now, what would D do on input $\langle D \rangle$?

- If D *accepts* $\langle D \rangle$, then D should *reject* $\langle D \rangle$
 - If D *rejects* $\langle D \rangle$, then D should *accept* $\langle D \rangle$
- Contradiction! So a decider H for A_{TM} cannot exist. Thus, A_{TM} is undecidable.

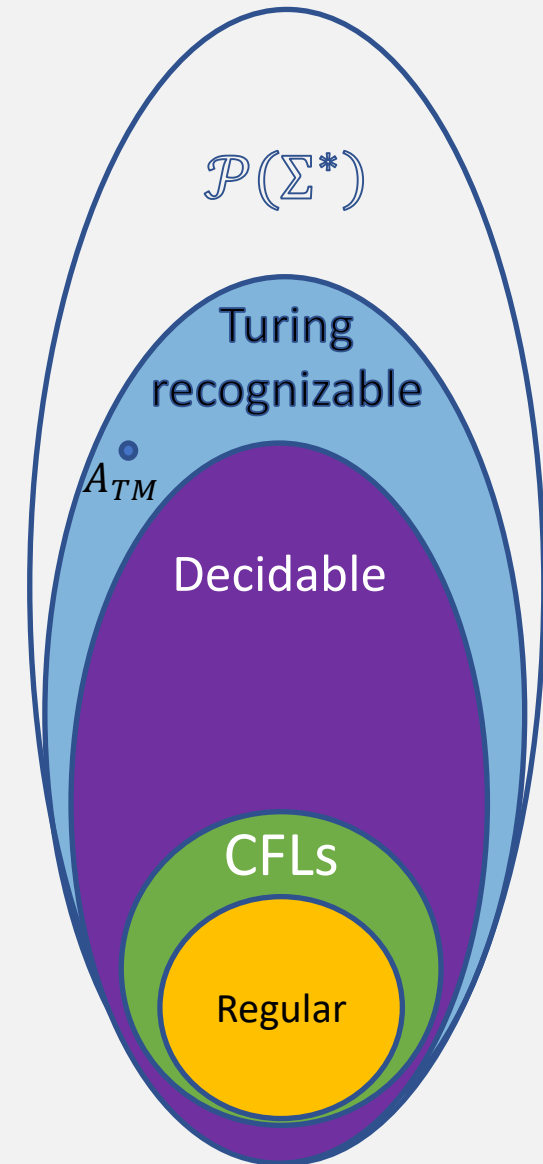


Decidable vs. recognizable

- *Corollary:* The set of all decidable languages is a strict subset of all Turing-recognizable languages.

Proof:

- Every decidable language is recognizable by definition.
- There is a recognizable language, A_{TM} , which is not decidable.



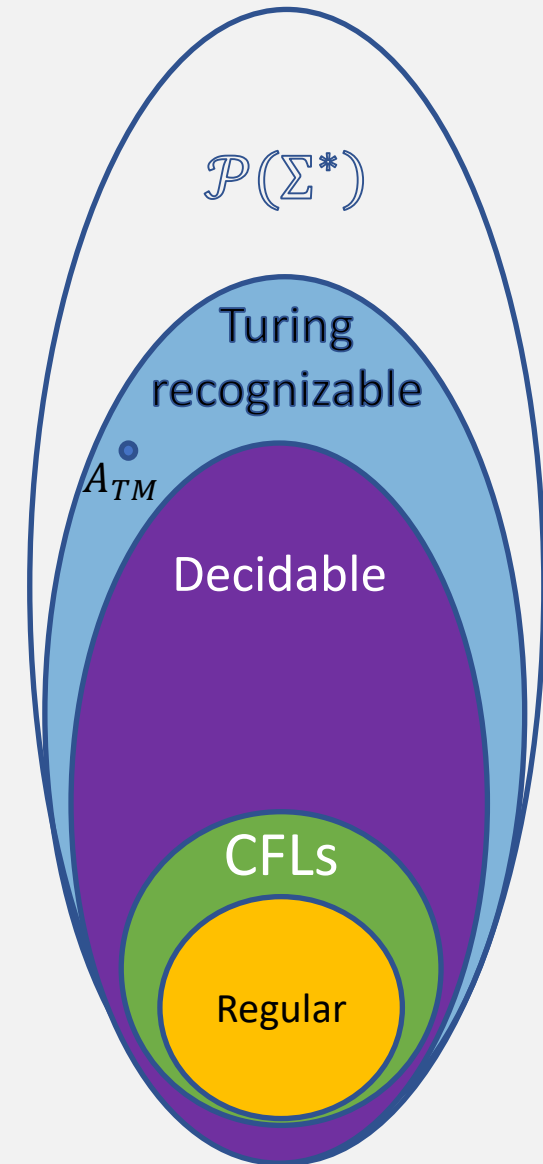
Decidable vs. recognizable

- *Corollary:* The set of all decidable languages is a strict subset of all Turing-recognizable languages.

Proof:

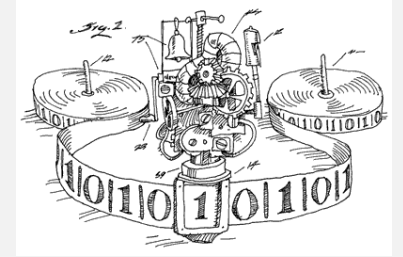
- Every decidable language is recognizable by definition.
- There is a recognizable language, A_{TM} , which is not decidable.

Recall: $A_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM and it accepts the string } w\}$



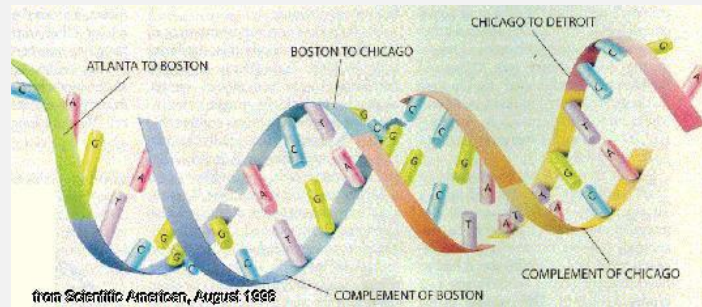
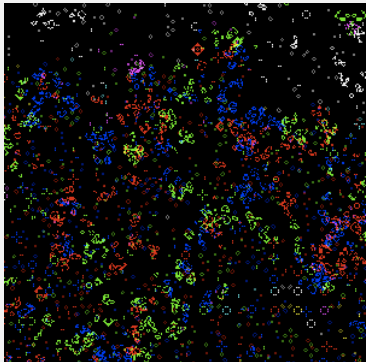


Undecidability



Are all hard languages “the same”, or some are harder than others?

***How would we know if a given L is hard
And how would we prove it?***



Closure properties

- Turing-recognizable (r.e.) languages are closed under
 - Regular operations: union, concatenation, Kleene star
 - Intersection.
 - We will show that they are not closed under complementation!
- Decidable languages:
 - Closed under regular operations, intersection, complementation...

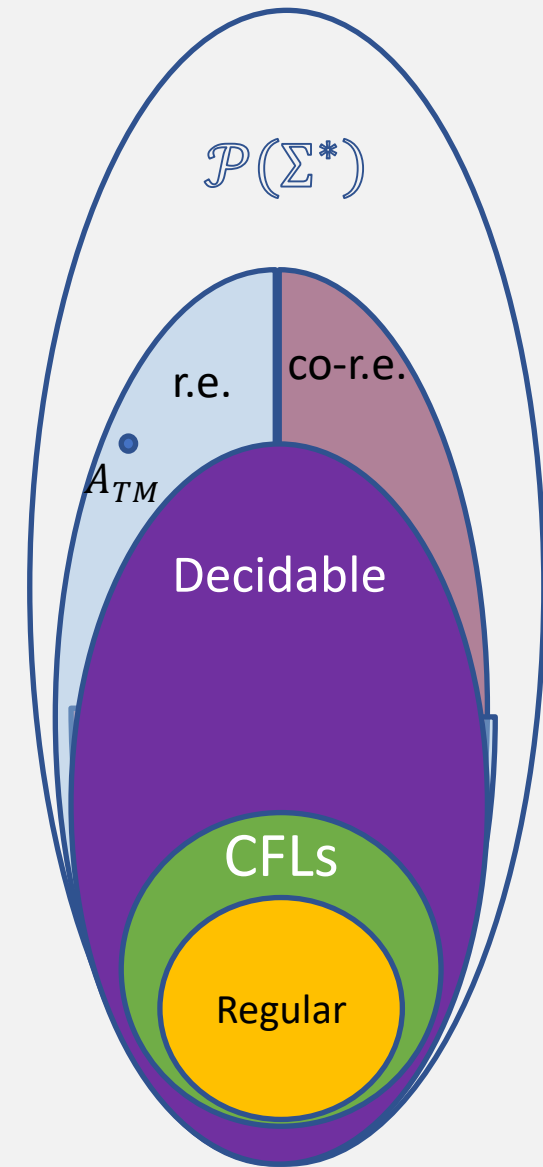
Co-Turing-recognizable

- *Definition:* A language is *co-Turing recognizable* (*co-semi-decidable*, *co-r.e.*) if its complement $\bar{L}$ is Turing recognizable.

– Example:

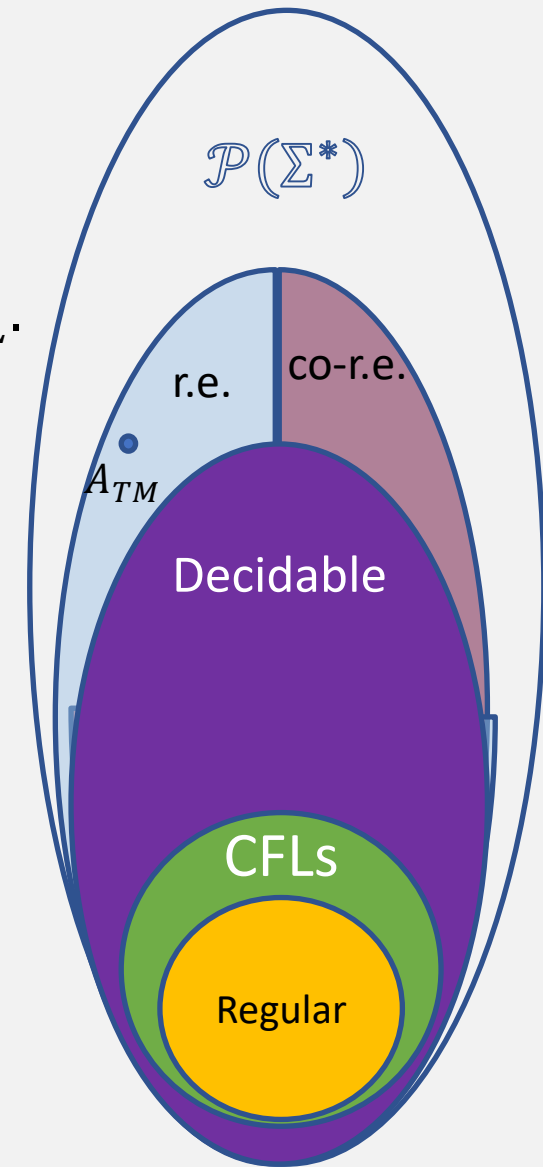
$$\overline{A_{TM}} = \{x \mid x \neq \langle M, w \rangle\} \cup \{\langle M, w \rangle \mid M \text{ rejects } w \text{ or loops on } w\}$$

- *Theorem:* A language L is decidable iff it is both Turing-recognizable and co-Turing-recognizable.



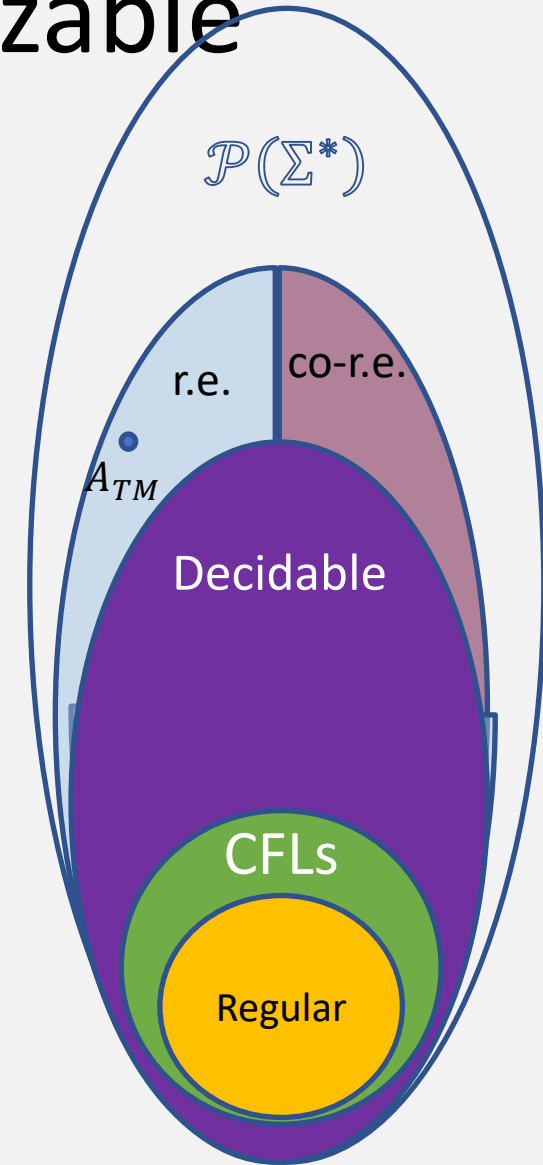
Theorem: A language L is decidable iff it is both Turing-recognizable and co-Turing-recognizable.

- Proof :
- $\Leftarrow$ If a language is decidable, then it is both r.e. and co-r.e.,
 - Let H_L decide L . Then H_L recognizes L , so L is r.e.
 - Make $H_{\bar{L}}$ that recognizes $\bar{L}$ by switching accept and reject states in H_L .
 - Since H_L never loops, neither does $H_{\bar{L}}$,
 - $H_{\bar{L}}$ accepts iff H_L rejects, so it recognizes $\bar{L}$
- $\Rightarrow$ if a language is both r.e. and co-r.e., then it is decidable.
 - Since L is Turing recognizable, there is M_L recognizing L
 - Since L is co-r.e., there is $M_{\bar{L}}$ recognizing $\bar{L}$
 - Let decider H_L for L be as follows:
 $H_L =$ “ On input x
 Run M_L and $M_{\bar{L}}$ in parallel on x
 If $M_{\bar{L}}$ accepts, **reject**.
 If M_L accepts, **accept**. “



A language that is not Turing-recognizable

- $\overline{A_{TM}} = \{x | x \neq \langle M, w \rangle\} \cup \{\langle M, w \rangle | M \text{ rejects } w \text{ or loops on } w\}$
- *Corollary:* $\overline{A_{TM}}$ is not Turing-recognizable.
 - Proof: otherwise A_{TM} would be decidable.
- *Corollary:* A complement of an undecidable language is always undecidable.
 - Decidable languages are closed under complementation.
- *Corollary:* An undecidable language L can be co-Turing-recognizable or Turing-recognizable (or neither), but not both.

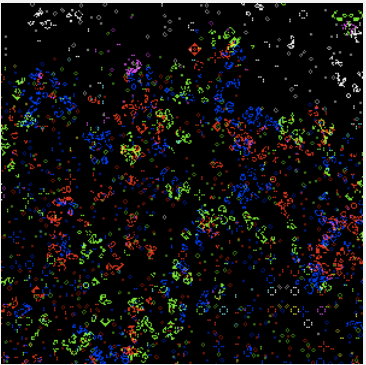
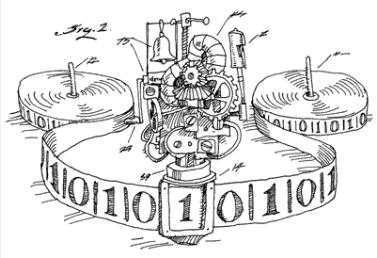


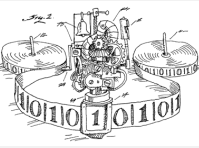


How to prove that languages are hard?

Reductions

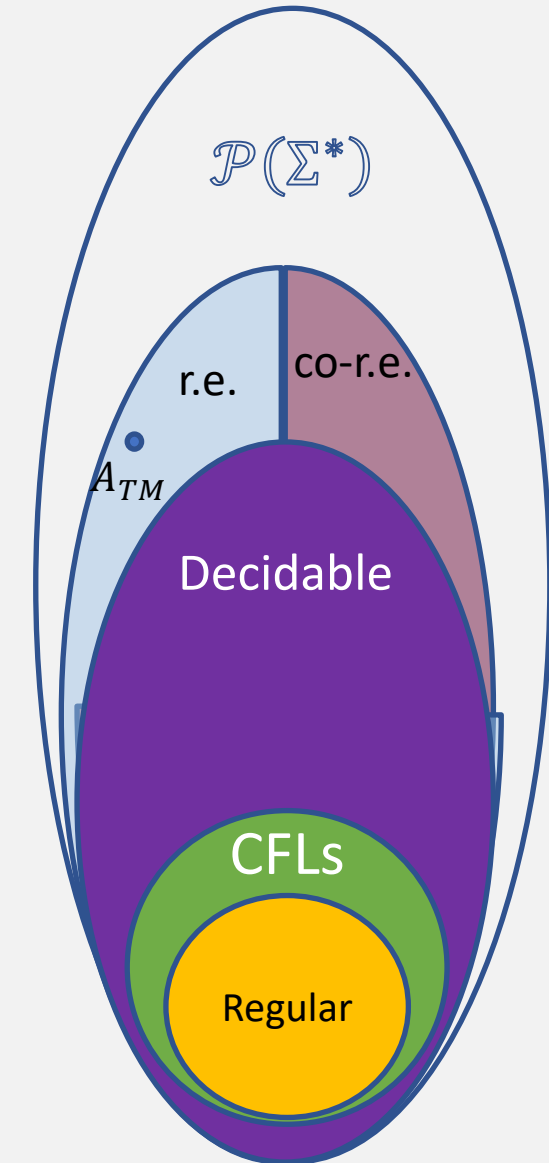
The most important
concept in this course.

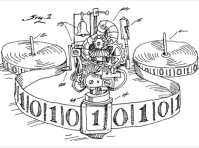




Halting problem

- $HALT_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM and it stops (accepts or rejects) on the string } w\}$
- *Claim:* $HALT_{TM}$ is Turing-recognizable, but not decidable.
 - That is, $HALT_{TM}$ is inside “r.e.”, but outside “decidable”.
 - To show that $HALT_{TM}$ is **Turing-recognizable**, build a Turing machine $M_{HALT_{TM}}$ recognizing it.
 - Easiness direction
 - To show that $HALT_{TM}$ is **undecidable**, “disguise” A_{TM} as $HALT_{TM}$
 - Hardness direction
 - Idea: Suppose it is possible to decide $HALT_{TM}$. Disguise A_{TM} as $HALT_{TM}$. An alleged decider for $HALT_{TM}$ will solve instances which are disguised A_{TM} instances. And so will decide A_{TM} -- but we just proved it is not possible to decide A_{TM} ! Contradiction.



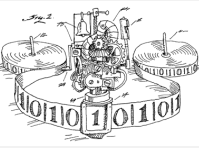


Halting problem

- $HALT_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM and it stops on the string } w\}$
- *Claim:* $HALT_{TM}$ is Turing-recognizable, but not decidable.
 - To show that $HALT_{TM}$ is Turing recognizable, build a Turing machine $M_{HALT_{TM}}$

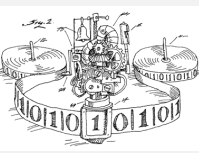
$M_{HALT_{TM}}$ = “ On input x
If $x \neq \langle M, w \rangle$, *reject*.
Simulate M on input w
If M ever enters accept state, *accept*.
If M ever enters reject state, *accept*. “

M_{ATM} = “ On input x
If $x \neq \langle M, w \rangle$, *reject*.
Simulate M on input w
If M ever enters accept state, *accept*.
If M ever enters reject state, *reject*. “



- $HALT_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM and it stops on the string } w\}$
- *Claim:* $HALT_{TM}$ is Turing-recognizable, but not decidable.
 - To show that $HALT_{TM}$ is **undecidable**, “disguise” A_{TM} as $HALT_{TM}$
 - Convert any instance $\langle M, w \rangle$ to an instance $\langle M', w' \rangle$ such that M accepts w if and only if M' halts on w'
 - That is, for each $\langle M, w \rangle$ we create a new instance $\langle M', w' \rangle$ such that
$$\langle M, w \rangle \in A_{TM} \Leftrightarrow \langle M', w' \rangle \in HALT_{TM}$$
 - Note: if M accepts w then it halts, but if M rejects w then it also halts
 - Cannot make $\langle M', w' \rangle$ the same as $\langle M, w \rangle$, the “only if” direction would not work.
 - We want to ensure that M' only halts on w' when M accepts w .
 - Idea: make M' from M by changing the code of M so whenever M goes to q_{reject} , M' would go into an infinite loop.
 - We can leave w unchanged this time.





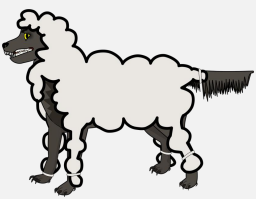
- $HALT_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM and it stops on the string } w\}$
- *Claim:* $HALT_{TM}$ is Turing-recognizable, but not decidable.
 - To show that $HALT_{TM}$ is **undecidable**, “disguise” A_{TM} as $HALT_{TM}$
 - Convert any instance $\langle M, w \rangle$ to an instance $\langle M', w' \rangle$ such that M accepts w iff M' halts on w'
 - Idea: make M' from M by changing the code of M so whenever M goes to q_{reject} , M' would go into an infinite loop.
 - We can leave w unchanged this time.
 - Define $\langle M', w' \rangle$ to be $\langle M', w \rangle$, where M' has
 - same Σ, Γ as M ,
 - All states of M , plus an extra state q_{loop}
 - All transitions of M , except change every q_{reject} to q_{loop}
 - transitions $(q_{loop}, s) \rightarrow (q_{loop}, s, R)$ for all $s \in \Sigma$.
 - Now, M' halts on w iff it accepts w , which happens iff M accepts w .
 - So $\langle M', w' \rangle \in HALT_{TM}$ iff $\langle M, w \rangle \in A_{TM}$





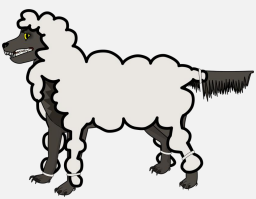
Computable functions

- *Definition:* A function $f: \Sigma^* \rightarrow \Sigma^*$ is **computable** if there exists a Turing machine M which halts on every input, and when M halts on a w , its tape contains only $f(w)$ and its head points to the first symbol of $f(w)$.
 - That is, it halts in a configuration of the form $q_a f(w)$, ignoring blanks.
 - The definition does not specify whether it has to be in q_a , but WLOG it is.
- Examples:
 - in a practice problem you showed that the function $f: \{0,1\}^* \rightarrow \{0,1\}^*$ defined as $f(x) = x - 2$ is computable
 - All arithmetic operations are computable.
 - A function $f: \{0,1\}^* \rightarrow \{0,1\}^*$ which takes $\langle M \rangle$ and outputs $\langle M' \rangle$ with accept/reject states switched is computable
 - For $x \neq \langle M \rangle$, define $f(x) = x$.
 - A function $f: \{0,1\}^* \rightarrow \{0,1\}^*$ which takes $\langle M, w \rangle$ and outputs 1 if M accepts w , 0 if M rejects w and blank if M loops on w is **not** computable.



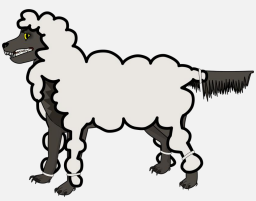
Mapping reduction

- *Definition:* Let L and L' be languages over Σ .
 - Then L is **mapping-reducible** to L' , written $L \leq_m L'$ iff there exists a computable function $f: \Sigma^* \rightarrow \Sigma^*$ such that $\forall x \in \Sigma^*$
$$x \in L \iff f(x) \in L'$$
The function f is called the **reduction** from L to L' .



Mapping reduction

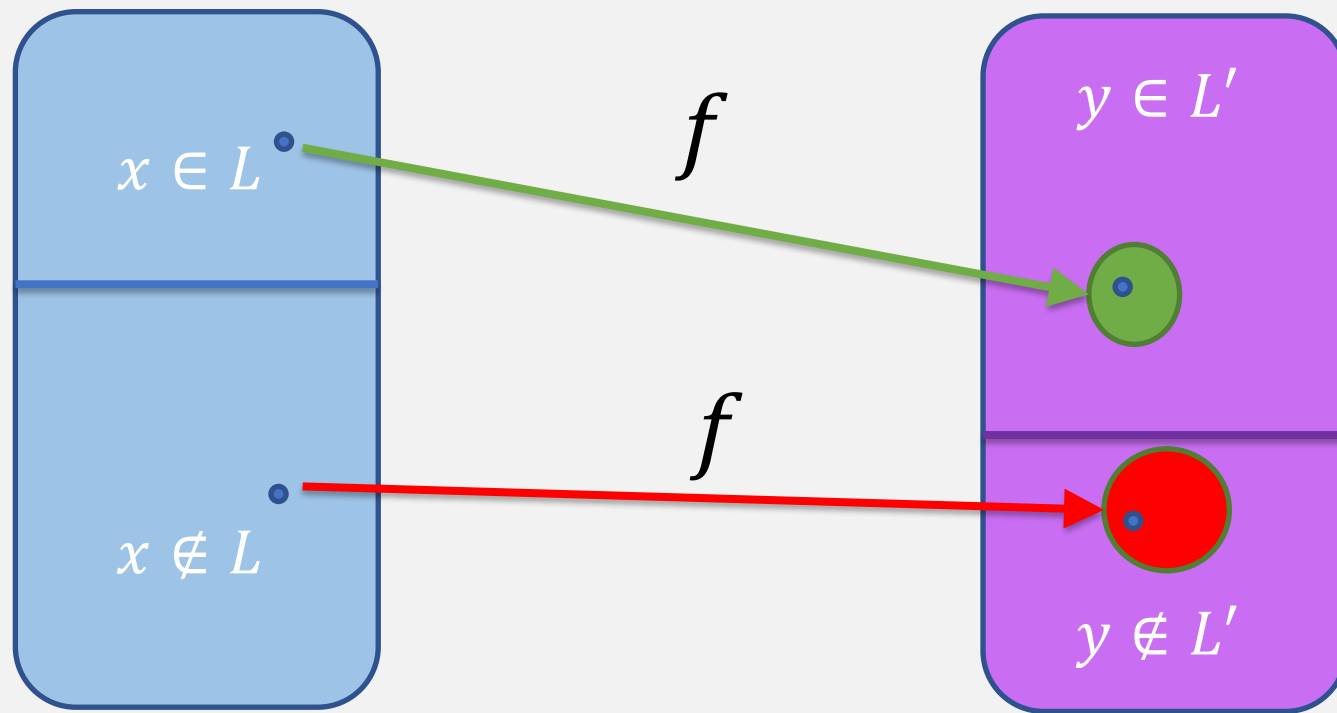
- *Definition:* Let L and L' be languages over Σ .
 - Then L is **mapping-reducible** to L' , written $L \leq_m L'$ iff there exists a computable function $f: \Sigma^* \rightarrow \Sigma^*$ such that $\forall x \in \Sigma^*$
$$x \in L \iff f(x) \in L'$$
The function f is called the **reduction** from L to L' .
- Such f “disguises” instances of L as instances of L'
- Our proof that $HALT_{TM}$ is undecidable reduced A_{TM} to $HALT_{TM}$



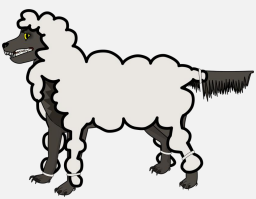
Mapping reduction

- Examples of reductions
 - Reduce checking if a graph is not bipartite to existence of an odd cycle
- Note: here we are just considering **mapping** reductions
 - **Not** where you run an algorithm for problem B, potentially multiple times and/or with extra computation on its output, to solve problem A
 - We might look at the latter kind (called Turing reductions) later in the course.

Mapping reduction: $L \leq_m L'$



- If f is a reduction from L to L' , then it also a reduction $\bar{L} \leq_m \bar{L}'$
 - So if $L \leq_m L'$, then $\bar{L} \leq_m \bar{L}'$
- Also, if $L_1 \leq_m L_2$ and $L_2 \leq_m L_3$, then $L_1 \leq_m L_3$



Reducibility for proving hardness

- The “ $\leq$ ” symbol indicates comparative hardness:
 - When $L \leq_m L'$, that means that L' is at least as hard as L
- Suppose that $L \leq_m L'$.
 - If L is **undecidable**, then L' is
 - If L is **not Turing-recognizable**, then L' is

Reduction is the main technique for proving hardness
For easiness, just give an algorithm for the original problem.

- Can prove that L is easy by proving that L' is easy and that $L \leq_m L'$.
 - This is done a lot in algorithm design

