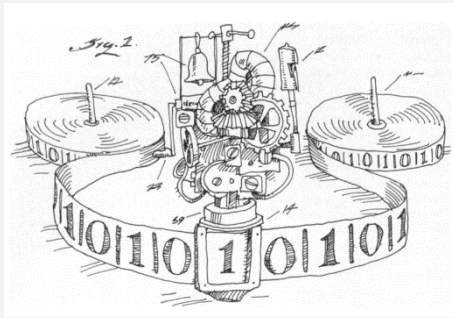
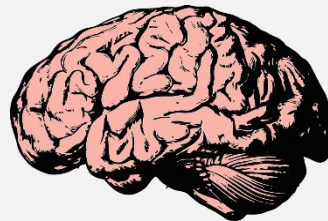
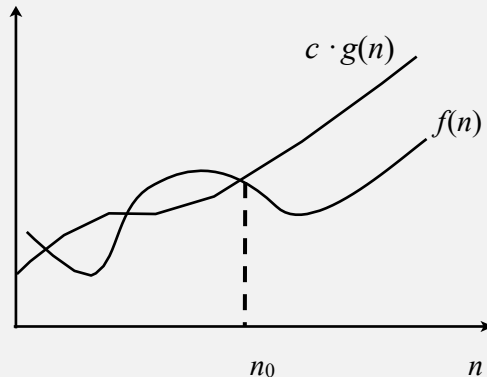


COMP 3602

Intro to Theory of Computation

Lecture 14



Logistics

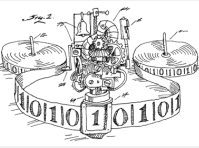
- First test graded: average 60%
- First oral exam: as soon as we finish the next topic
 - Mon Mar 2nd : schedule TBA, likely 11-12:30, 2-3:30 and 4-5:30.
 - I will assign the time slots according to the survey
 - Practice problems posted
 - Extra office hours?
- Presentation proposals? To say which topic you picked. Mar 4th?
 - Submit into the assignment folder (just the name and preferred date/place)
 - About 30 (50?) minutes intro to a topic in complexity (or computability)
 - In the reading group on in class?
 - Quantum computing, proof complexity, fine-grained complexity, circuit complexity, communication complexity, pseudorandomness, etc...

1. Prove that the following language is decidable by giving an algorithm to decide it.
L = all NFAs over the alphabet $\{0,1\}$ which do not accept any string with even number of 0s
Hint: you can use the fact that testing whether a DFA/NFA/PDA accepts nothing is decidable.
2. Prove that the following sets are countable:
 - a. A set of all integer points in 3D space (that is, all triples of integers).
 - b. A set of all Turing machines over the alphabet $\{a,b,c\}$ which accept the empty string.
3. Prove by diagonalization that the set of all languages over the alphabet $\{a,b,c\}$ which contain the empty string is uncountable. What can you conclude from this together with the previous question?

4. For each of the following languages, do the following
- Write the language description using quantifiers
 - Write the description of the complement of the language
 - Determine if the language is decidable, undecidable and Turing-recognizable, undecidable and co-Turing-recognizable, or neither Turing-recognizable nor co-Turing-recognizable, and prove that by constructing Turing machines for easiness proofs, and giving reductions for hardness proofs.
- a. $L_1 = \{ \langle M \rangle \mid M \text{ accepts an even-length string} \}$
- b. $L_2 = \{ \langle M \rangle \mid \text{the language of } M \text{ contains only strings starting with } 1 \}$
- c. $L_3 = \{ \langle M, w \rangle \mid M \text{ accepts } w \text{ without moving to a blank cell} \}$
- d. $L_4 = \{ \langle M \rangle \mid M \text{ loops on input "0" or does not accept any string encoding a prime number in binary} \}$
- e. $L_5 = \{ \langle M \rangle \mid M \text{ accepts all strings ending in "11"} \}$
- f. $L_6 = \{ \langle M \rangle \mid M \text{ rejects all strings of length at most } 3 \}$
- g. $L_7 = \{ \langle M_1, M_2 \rangle \mid \text{exactly one of } M_1, M_2 \text{ accepts "00"} \}$
- h. $L_8 = \{ \langle M \rangle \mid M \text{ accepts infinitely many strings containing a substring "101"} \}$

5. A *useless state* in a Turing machine, an automaton, or a PDA is one that is never entered on any input string. Show that the problem of determining whether a DFA has a useless state is decidable, whereas determining if a Turing machine has a useless state is undecidable.
6. Show that for any two disjoint co-Turing-recognizable languages L and L' there exists a decidable language L'' such that $L \subseteq L''$ and $L' \subseteq \overline{L''}$

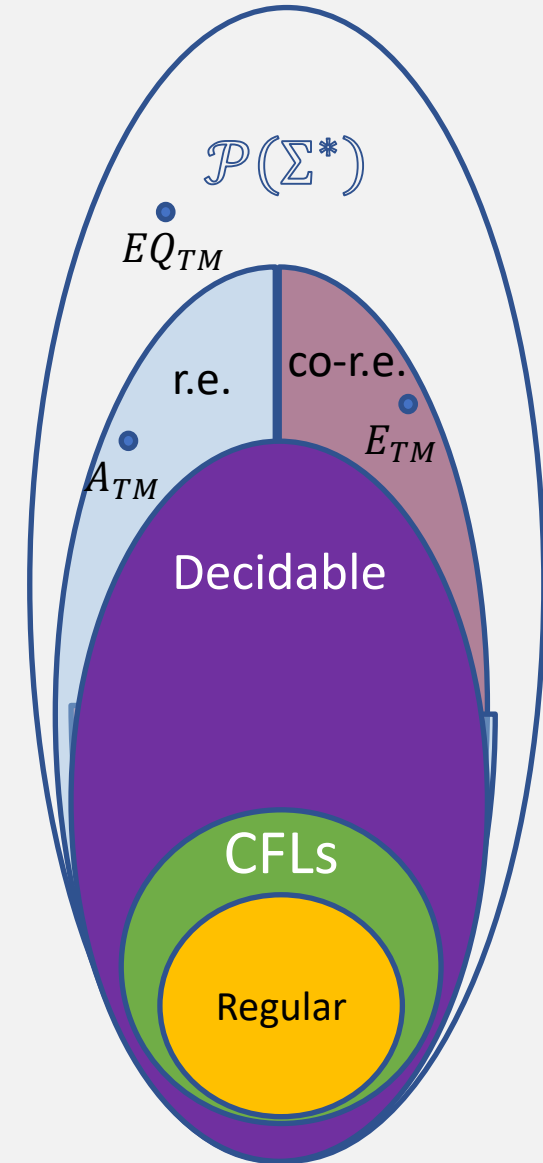
Questions/comments?



Let predicate $Check_A(M, w, y)$ be true iff y encodes an accepting computation of M on w .

– $Check_R(M, w, y), Check_H(M, w, y)$ for checking a rejecting/halting computation respectively

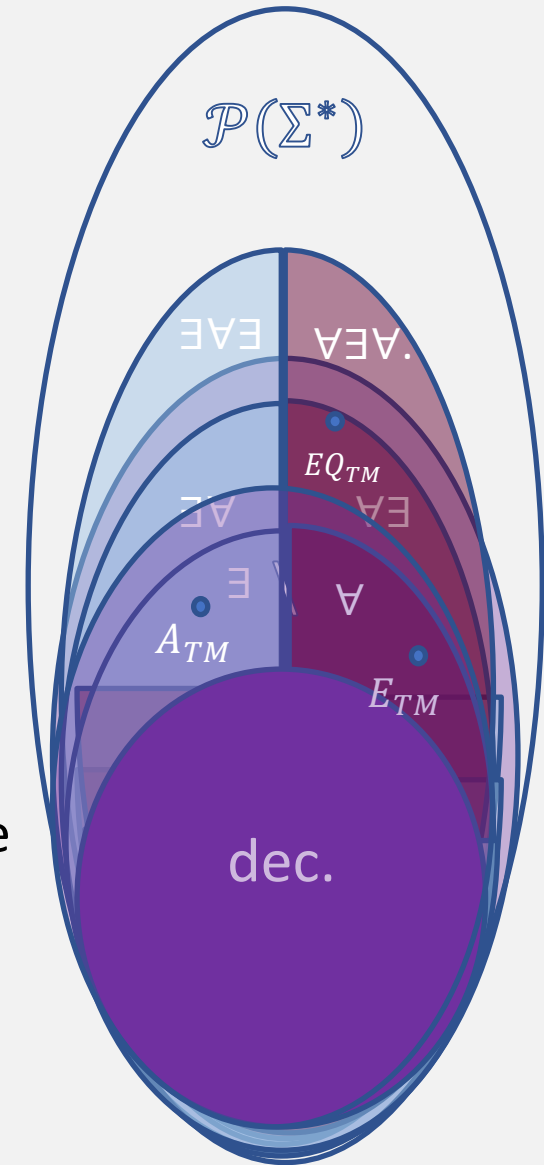
- $A_{TM} = \{ \langle M, w \rangle \mid \exists y \text{ } Check_A(M, w, y) \}$
- $HaltB = \{ \langle M \rangle \mid M \text{ halts on } \lambda \}$
 - $\{ \langle M \rangle \mid \exists y \text{ } Check_H(M, \lambda, y) \}$
- $E_{TM} = \{ \langle M \rangle \mid \mathcal{L}(M) = \emptyset \}$
 - $\{ \langle M \rangle \mid \forall w \forall y \neg Check_A(M, w, y) \}$
 - $\overline{E_{TM}} = \{ x \mid x \neq \langle M \rangle \} \cup \{ \langle M \rangle \mid M \text{ accepts some string } w \}$
 - $\{ x \mid x \neq \langle M \rangle \} \cup \{ \langle M \rangle \mid \exists (w, y) Check_A(M, w, y) \}$
 - Recall our $M_{\overline{E_{TM}}}$ iterating over pairs (w, y) !
- $EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid \mathcal{L}(M_1) = \mathcal{L}(M_2) \}$
 - $\{ \langle M_1, M_2 \rangle \mid \forall w (\exists y_1 Check_A(M_1, w, y_1)) \leftrightarrow (\exists y_2 Check_A(M_2, w, y_2)) \}$
 - $$= \{ \langle M_1, M_2 \rangle \mid \forall w \left(\begin{array}{c} \forall z_1 \forall z_2 \neg Check_A(M_1, w, z_1) \wedge \neg Check_A(M_2, w, z_2) \\ \vee \\ \exists y_1 \exists y_2 Check_A(M_1, w, y_1) \wedge Check_A(M_2, w, y_2) \end{array} \right) \}$$
- $ALL_{TM} = \{ \langle M \rangle \mid M \text{ accepts everything} \}$





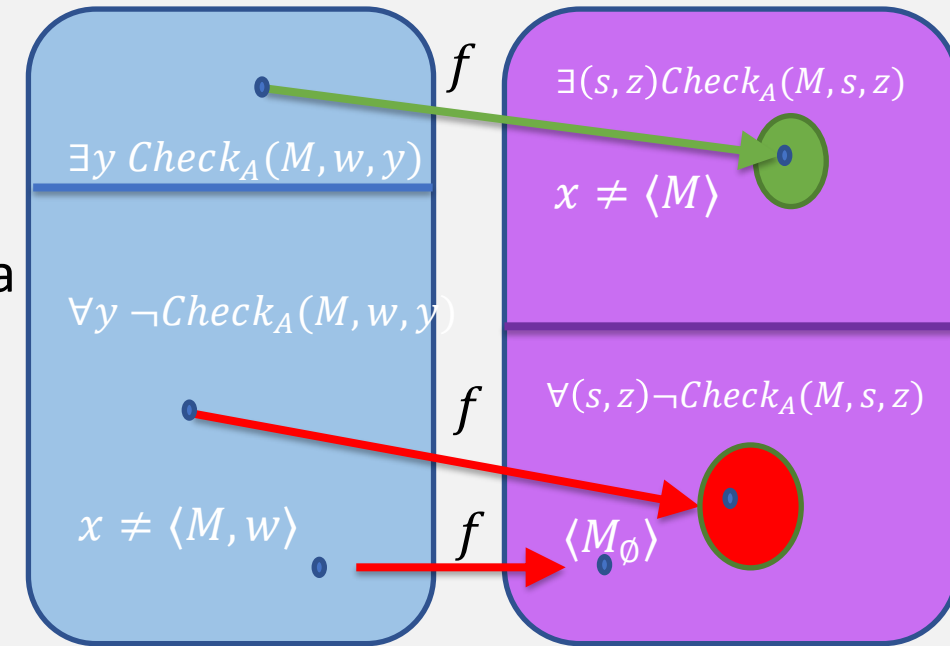
Arithmetic hierarchy

- Idea: a level for each # of quantifier alternations
 - Bottom (0^{th}) level: decidable
 - 1^{st} level: $\exists R, \forall R$ where R is decidable.
 - 2^{nd} level: $\exists \forall R, \forall \exists R$ where R is decidable.
 - Equivalently, $\exists \varphi, \forall \varphi$ where φ from 1^{st} level with opposite quantifier
 - ... i^{th} level: $\exists \varphi, \forall \varphi$ where φ is from level $i-1$
 - With opposite quantifiers
- Each subsequent layer contains all the previous ones.
 - It is possible to prove that each contains languages not appearing below.
- Some languages can be written both as $\exists x_1 \varphi_1(x_1)$ and $\forall x_2 \varphi_2(x_2)$ where $\varphi_1 = \forall \dots$, $\varphi_2 = \exists \dots$ from the previous level.
 - Eg: $L = \{\langle M \rangle \mid M \text{ accepts } 0 \text{ and loops on } 1\} =$
 $= \{\langle M \rangle \mid \exists y_1 \text{Check}_A(M, 0, y_1) \wedge \forall y_2 \neg \text{Check}_H(M, 1, y_2)\}$
 is in both $\forall \exists$ and $\exists \forall$ (though not in $\exists$, and not in $\forall$)



Quantifiers and reductions

- Reductions often implicitly map quantifiers of the description of L to (some of) quantifiers of the description of L'
 - If $f(x) \in L'$, then it should be possible to recover witnesses for $x \in L$ from witnesses for $f(x) \in L'$
 - When reducing $L \leq_m L'$, quantifier prefix of L has to be a substring of quantifier prefix of L'
- $A_{TM} = \{ \langle M, w \rangle \mid \exists y \text{ Check}_A(M, w, y) \}$
- $\overline{E_{TM}} = \{x \mid x \neq \langle M \rangle\} \cup \{\langle M \rangle \mid \exists(s, z) \text{ Check}_A(M, s, z)\}$
- Reduction: $A_{TM} \leq_m \overline{E_{TM}}$
 - Relates $\exists y$ for $\langle M, w \rangle$ and $\exists(s, z)$ for $f(\langle M, w \rangle)$
 - This is the proof that the reduction works correctly when $x = \langle M, w \rangle$
 - Also need to argue that the “garbage” is taken care of correctly.



$$f(x) = \begin{cases} \langle M' \rangle & \text{if } x = \langle M, w \rangle \\ \langle M_{\emptyset} \rangle & \text{otherwise} \end{cases}$$

M' = “ On input s
 Simulate M on input w
 If M ever enters accept state, **accept**.
 If M ever enters reject state, **reject**. ”

- Reduction: $A_{TM} \leq_m \overline{E_{TM}}$
- Main idea: computation of M' on any input consists of
 1. writing w on the tape (erase input, write w , come back to the beginning of w)
 2. Computation of M on w

⇒ Suppose $\langle M, w \rangle \in A_{TM}$, that is, $\exists y \text{ Check}_A(M, w, y)$

From this y , we construct (s, z) such that $\text{Check}_A(M', s, z)$ is true, showing that $\langle M' \rangle \in \overline{E_{TM}}$

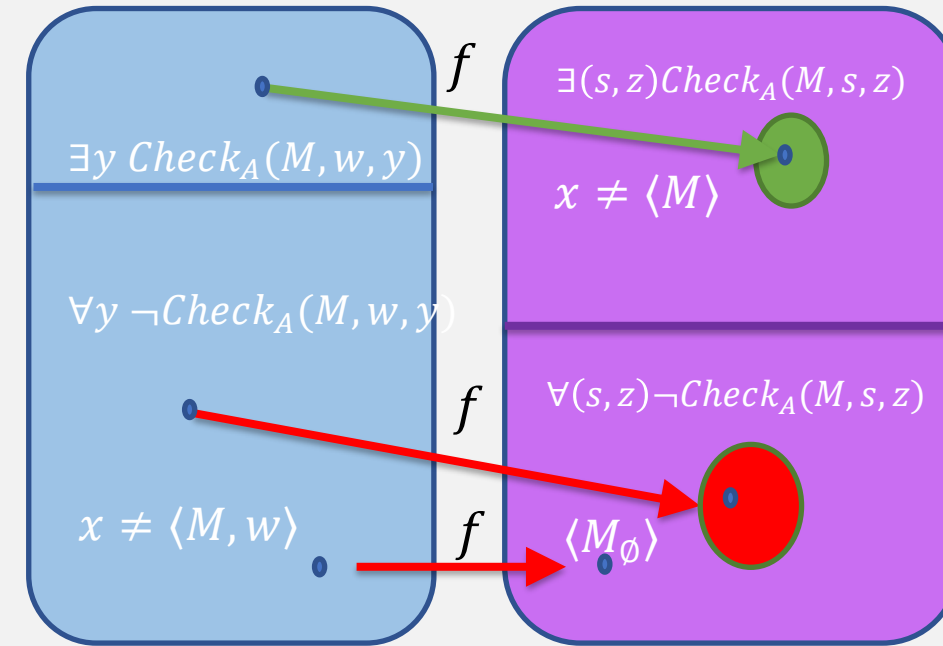
- Let $s = \lambda$ (since M' behaves the same on all inputs, s can be any)
- Let $z = (\text{"write } w", y)$.
 - A sequence of configurations of y with a list of configurations to write w before it.
- If y is an accepting computation of M on w , then $(\lambda, (\text{"write } w", y))$ is (s, z) satisfying $\exists(s, z) \text{Check}_A(M', s, z)$. Therefore, $\langle M' \rangle \in \overline{E_{TM}}$

⇐ Suppose $f(\langle M, w \rangle) = \langle M' \rangle \in \overline{E_{TM}}$, so $\exists(s, z) \text{Check}_A(M', s, z)$

- **We only care for the image of f , not arbitrary $\langle M \rangle \in \overline{E_{TM}}$**
- Then $z = (\text{"write } w", y)$ is an accepting computation
 - Where “write w ” depends on s that needs erasing
- So can recover y from z
 - Where y will also be an accepting computation, of M on w .

$$A_{TM} = \{ \langle M, w \rangle \mid \exists y \text{ Check}_A(M, w, y) \}$$

$$\overline{E_{TM}} = \{ x \mid x \neq \langle M \rangle \} \cup \{ \langle M \rangle \mid \exists (s, z) \text{Check}_A(M, s, z) \}$$



$$f(x) = \begin{cases} \langle M' \rangle & \text{if } x = \langle M, w \rangle \\ \langle M_\emptyset \rangle & \text{otherwise} \end{cases}$$

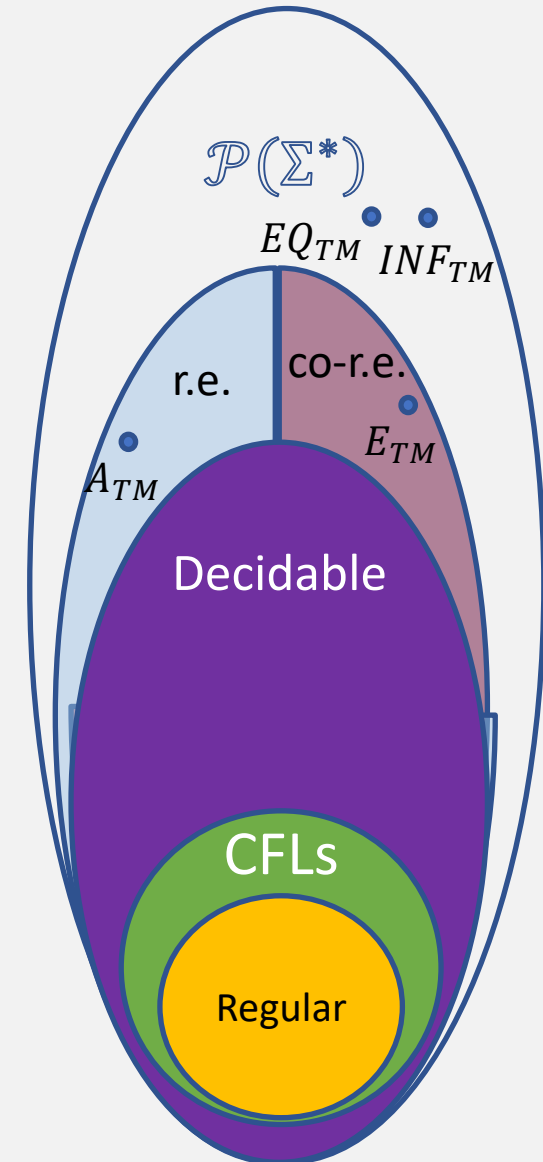
$M' =$ “ On input s
 Simulate M on input w
 If M ever enters accept state, **accept**.
 If M ever enters reject state, **reject**. ”

INF_{TM}

- $INF_{TM} = \{\langle M \rangle \mid \mathcal{L}(M) \text{ is infinite}\}$
 - To say that a set S is infinite, write $\forall w \exists w' (w' > w) \wedge (w' \in S)$
 - Here, “>” is some order on elements of S
 - string order on strings, “>” for numbers,..
- $INF_{TM} = \{\langle M \rangle \mid \forall w \exists w' (w' > w \wedge M \text{ accepts } w')\}$
- $= \{\langle M \rangle \mid \forall w \exists w' \exists y (w' > w \wedge Check_A(M, w', y))\}$

Claim: INF_{TM} is neither Turing-recognizable nor co-Turing-recognizable, but in the 2nd level of the arithmetic hierarchy, $\forall\exists$ level.

- INF_{TM} is not co-Turing-recognizable
 - Show that $A_{TM} \leq_m INF_{TM}$
- INF_{TM} is not Turing-recognizable.
 - Show that $\overline{A_{TM}} \leq_m INF_{TM}$, or, equivalently, $A_{TM} \leq_m \overline{INF_{TM}}$



- $INF_{TM} = \{\langle M \rangle \mid \forall s' \exists s \exists z (s > s' \wedge Check_A(M, s, z))\}$
- $\overline{INF_{TM}} = \{x \mid x \neq \langle M \rangle\} \cup \{\langle M \rangle \mid \exists s' \forall s \forall z (s > s' \rightarrow \neg Check_A(M, s, z))\}$

- INF_{TM} is not co-Turing-recognizable

- Let's show that $A_{TM} \leq_m INF_{TM}$

$$f(x) = \begin{cases} \langle M' \rangle & \text{if } x = \langle M, w \rangle \\ \langle M_\emptyset \rangle & \text{otherwise} \end{cases}$$

M' = “ On input u

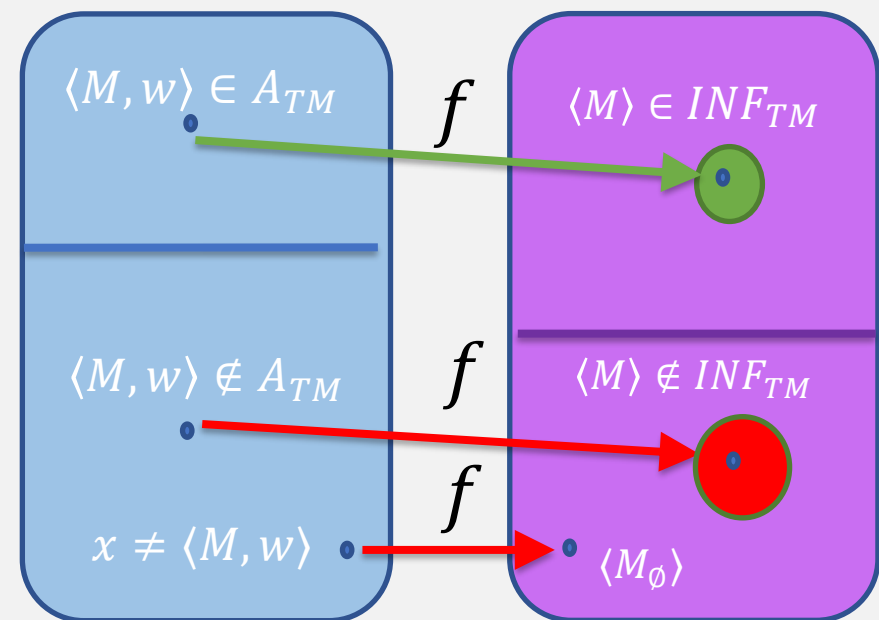
Simulate M on input w

If M ever enters accept state, *accept*.

If M ever enters reject state, *reject*. “

If M accepts w , $\mathcal{L}(M') = \Sigma^*$, infinite.

If M does not accept w , $\mathcal{L}(M') = \emptyset$, finite.



An accepting computation y of M on w turns into accepting computations on all strings.

So can always satisfy $\forall s' \exists s \exists z (s > s' \wedge Check_A(M, s, z))$:

- for any s' , take s to be the next string in string order
- Take the respective $z = (\text{"write } w", y)$, where “write w ” might depend on s needed to be erased.

And from an accepting computation of M' on any string can extract an accepting computation of M on w

- $INF_{TM} = \{\langle M \rangle \mid \forall s' \exists s \exists z (s > s' \wedge Check_A(M, s, z))\}$
- $\overline{INF_{TM}} = \{x \mid x \neq \langle M \rangle\} \cup \{\langle M \rangle \mid \exists s' \forall s \forall z (s > s' \rightarrow \neg Check_A(M, s, z))\}$

- INF_{TM} is not Turing-recognizable

- Let's show that $A_{TM} \leq_m \overline{INF_{TM}}$

Construct a computable function f such that

- If M accepts w , $\mathcal{L}(M')$ is finite.
- If M does not accept w , $\mathcal{L}(M')$ is infinite.
 - Match $\exists y$ from the description of A_{TM} with $\exists s$ in $\overline{INF_{TM}}$

$$f(x) = \begin{cases} \langle M' \rangle & \text{if } x = \langle M, w \rangle \\ \langle M_{\Sigma^*} \rangle & \text{otherwise} \end{cases}$$

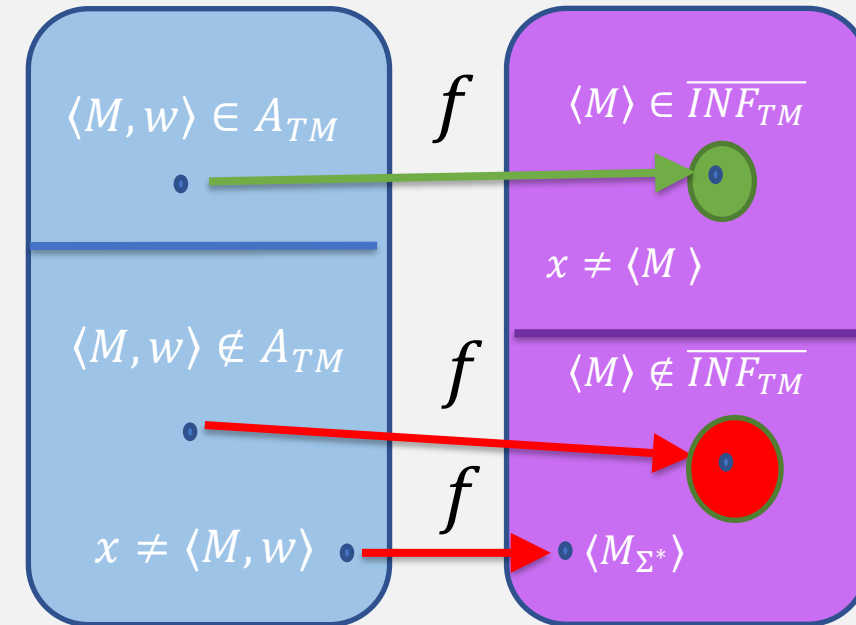
M' = “ On input u

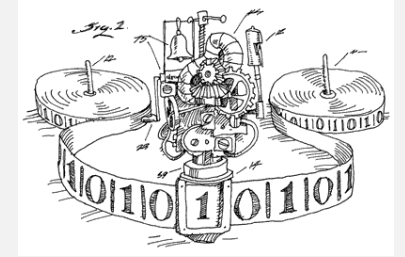
Simulate M on input w **for u steps**

If M enters **accept** state **within this time**, **reject**. “

Else **accept**

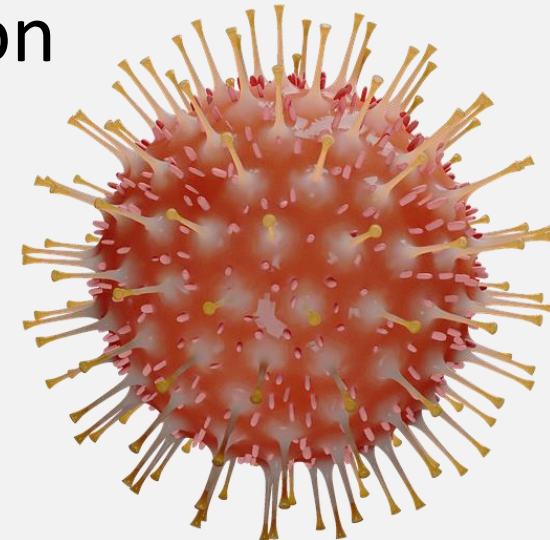
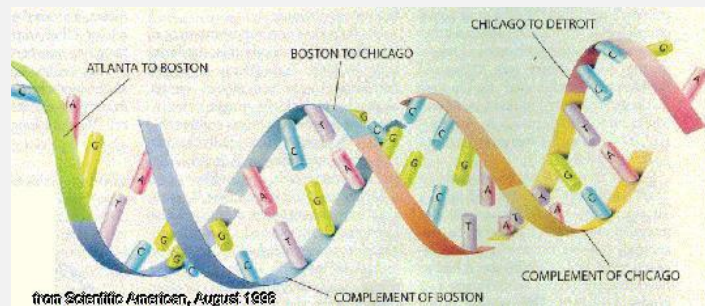
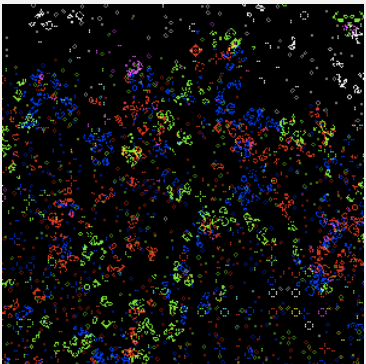
- If M accepts w with computation $y = (C_0, \dots, C_t)$ for some t , then $\mathcal{L}(M') = \{s_0 \dots s_{t-1}\}$ is finite.
- If M does not accept w , $\mathcal{L}(M') = \Sigma^*$ is infinite.
 - So steps of computation of M on w are matched with input strings of M' .

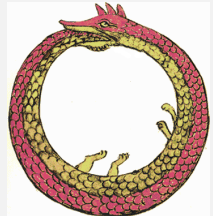




Viruses and virus checkers

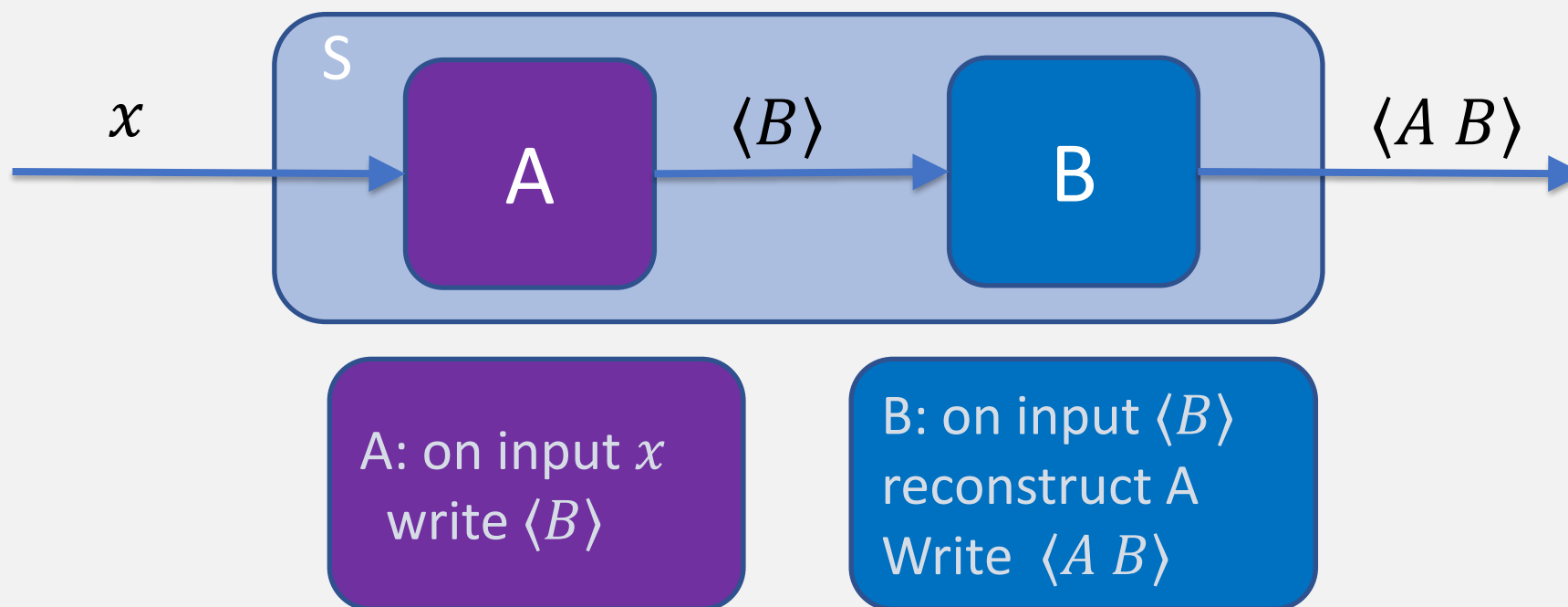
How to build self-replicating programs
-- and how hard to check for self-replication



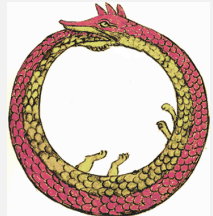


A self-replicating program (quine)

- TM S ignores its input and prints (outputs) $\langle S \rangle$



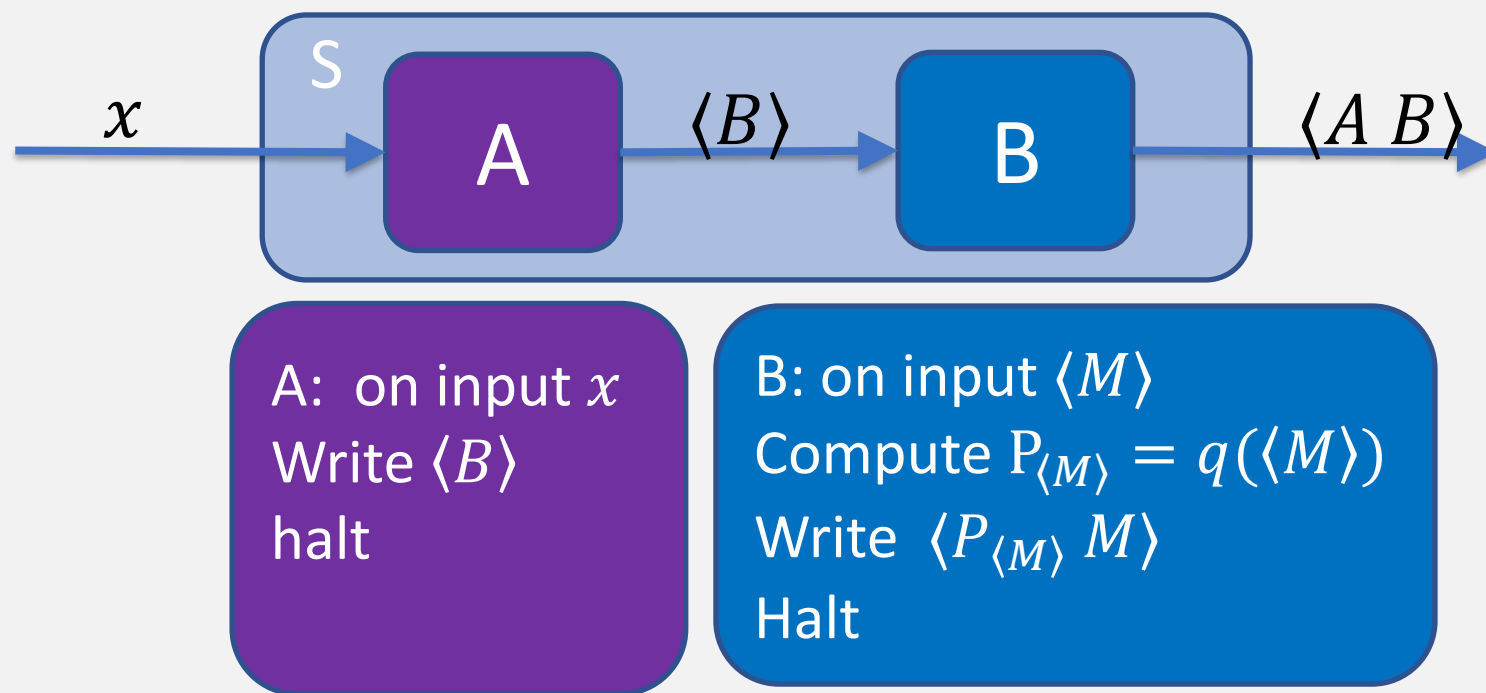
- Note: assume encoding of TMs is reasonable enough that it is possible to compute $\langle A B \rangle$ from $\langle A \rangle, \langle B \rangle$ without much trouble

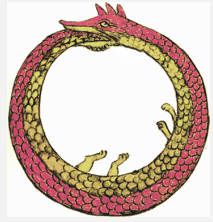


A self-replicating program (quine)

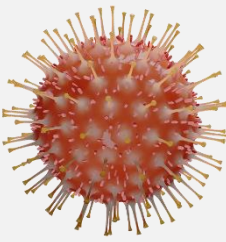
- Let $q(w) = \langle P_w \rangle$ be a function that on input w produces a code of a TM which prints (outputs) w
 - $q(w)$ is computable:

Q = “ On input w
Construct P_w as follows:
 P_w = ‘On input x
Erase input
Write w on the tape
halt’
Write $\langle P_w \rangle$ on the tape
halt”



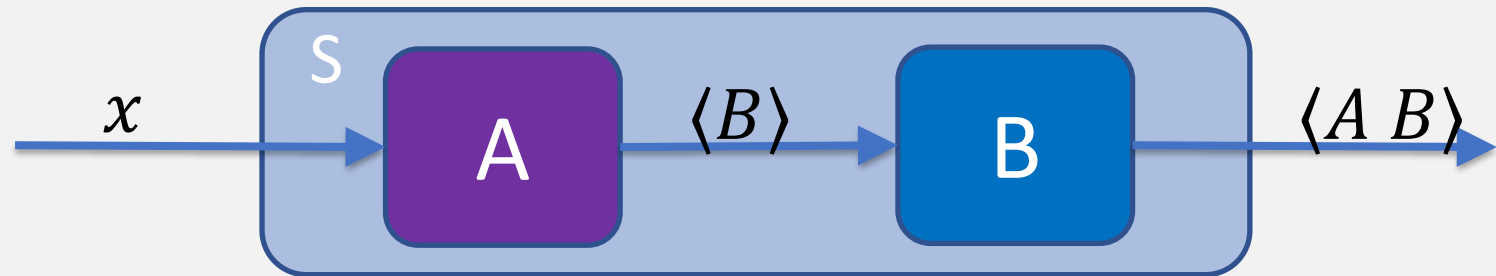


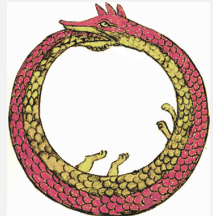
A self-replicating program in Python



```
a = "print('a = ' + chr(34) + a + chr(34)+ chr(10) + a)"  
print('a = ' + chr(34) + a + chr(34) + chr(10) + a)
```

- Here, `chr(10)` is a newline, and `chr(34)` is “

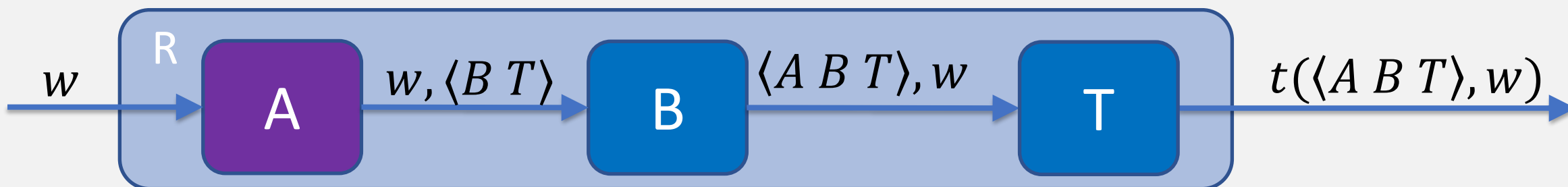


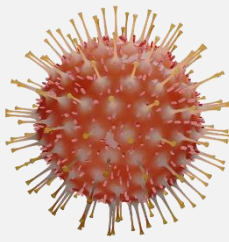
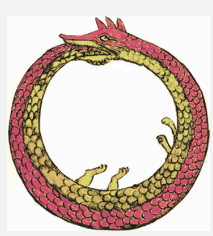


Recursion theorem

Theorem: Let $t: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ be a computable function. Then there is a Turing machine R that computes a function $r: \Sigma^* \rightarrow \Sigma^*$ such that for every w , $r(w) = t(\langle R \rangle, w)$

- That is, “get own code” can be used in any function r , as long as a variant of r which gets code as an extra input, t , is computable.
 - Proof idea: let T be a TM computing t
 - Since t is computable, such a T exists
 - Build $R = \langle A B T \rangle$ using ideas similar to $S = \langle A B \rangle$

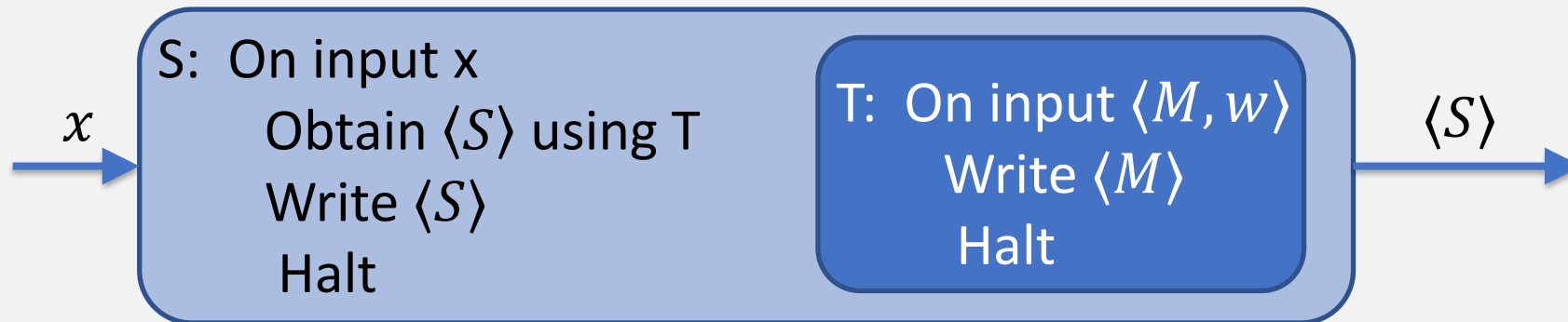


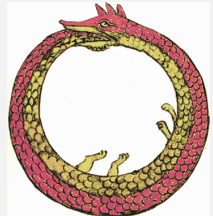


Recursion theorem

– Let $t: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ be a computable function. There is a Turing machine R computing a function $r: \Sigma^* \rightarrow \Sigma^*$ s.t. $\forall w, r(w) = t(\langle R \rangle, w)$

- Application: self-print again

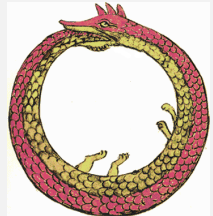




Fixed-point theorem

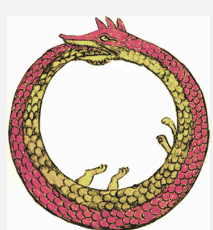
- Recall: two TMs M_1, M_2 are called equivalent if $\mathcal{L}(M_1) = \mathcal{L}(M_2)$
- *Definition.* Let f be a function. A *fixed point* of f is a value x such that $f(x) = x$
 - E.g.: $f(x) = x^2$ has fixed points 0 and 1.
 - $f(x) = x + 1$ does not have a fixed point.
- Let t be a computable function on the set of encodings of TMs. That is, $t(\langle M \rangle) = \langle M' \rangle$.
 - Let's relax the definition of a fixed point: rather than requiring that $\langle M' \rangle = \langle M \rangle$, we merely ask that $\mathcal{L}(M) = \mathcal{L}(M')$
 - If you want $t: \Sigma^* \rightarrow \Sigma^*$, just treat “garbage” as equivalent to $M_\emptyset$
- Theorem. Any computable function t that transforms TMs to TMs has a (relaxed) fixed point.
 - Proof: given t , we construct F such that $\mathcal{L}(t(\langle F \rangle)) = \mathcal{L}(F)$

F: On input x
Obtain $\langle F \rangle$
Compute $G = t(\langle F \rangle)$
Simulate G on w



Minimal TM

- *Definition:* the *length (size)* of a TM M is $|\langle M \rangle|$
 - Here, as before, we assume a fixed underlying encoding.
- A TM M is minimal iff $\forall M' \mathcal{L}(M) = \mathcal{L}(M') \rightarrow |\langle M' \rangle| \geq |\langle M \rangle|$
 - That is, M has the smallest length of all TMs computing the same language.
- Let $MIN_{TM} = \{|\langle M \rangle| \mid M \text{ is a minimal TM}\}$
 - We will show that MIN_{TM} is undecidable using Recursion theorem.

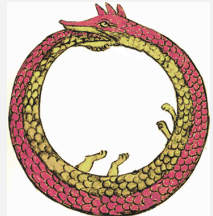


Enumerators

- Recall that Turing-recognizable is also called “recursively enumerable”
- Let’s define what “enumerable” means.

Consider a TM model with an output tape separate from work tape.

- A TM M outputs a string by writing it on the output tape.
- *Definition:* A TM E (with an output tape) enumerates a language L iff, when started on a blank tape, E eventually outputs all $x \in L$
 - Call such TM E an enumerator
 - When L is infinite, E will run forever.
 - E may output the same string multiple times.
- *Theorem:* A language L is Turing-recognizable iff it is recursively enumerable.
 - $\Leftarrow$ M_L on x : run the enumerator E_L , see if the input string x is ever outputted
 - $\Rightarrow$: Enumerate L by running M_L on strings $s_0 \dots s_i$ for i steps in each round
 - Output a string if M_L accepts it



MIN_{TM} is not Turing-recognizable

- Proof by contradiction
 - Assume that some TM E enumerates MIN_{TM}

C = “ On input w

Get own code $\langle C \rangle$

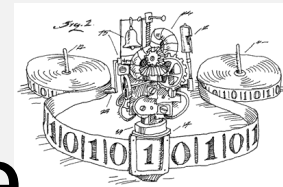
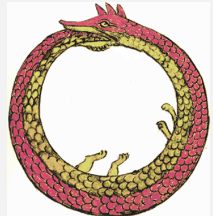
Run E until some D appears such that $|\langle D \rangle| > |\langle C \rangle|$

Simulate D on w ”

Since M_{TM} is infinite, such a D must eventually appear.

Note that $\mathcal{L}(C) = \mathcal{L}(D)$, but $|\langle D \rangle| > |\langle C \rangle|$, so D cannot be minimal.

Contradiction!



Checking for self-replication is undecidable

- Last time we built a TM which printed its own code.
- Consider a language $\text{SELFPRINT} = \{\langle M \rangle \mid M \text{ prints its own code}\}$

Theorem: SELFPRINT is undecidable.

Proof: $A_{TM} \leq_m \text{SELFPRINT}$

$$f(x) = \begin{cases} \langle M' \rangle & \text{if } x = \langle M, w \rangle \\ \langle M_\emptyset \rangle & \text{otherwise} \end{cases}$$

M' = “ On input u

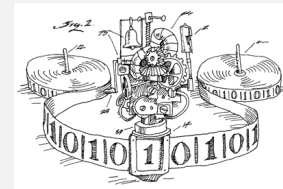
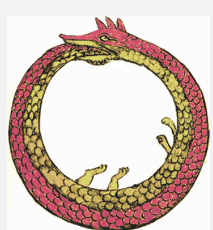
 get $\langle M' \rangle$ using recursion theorem

 Simulate M on input w

 If M ever enters accept state, print $\langle M' \rangle$ and *accept*.

 If M rejects, *reject*.

Corollary: Virus checking is undecidable.



Rice's theorem

- A *property* P of Turing machines is a set of Turing machine descriptions such that if $\mathcal{L}(M_1) = \mathcal{L}(M_2)$, then either both $\langle M_1 \rangle, \langle M_2 \rangle$ are in P , or neither.
 - A property is *non-trivial* if $\exists M_{in} \langle M_{in} \rangle \in P$, and $\exists M_{out} \langle M_{out} \rangle \notin P$
- *Rice's theorem*: Any non-trivial property of TMs is undecidable.
 - Proof idea: by contradiction. Suppose there is a non-trivial property P decidable by a TM D
 - Since P is non-trivial, $\exists M_{in} \langle M_{in} \rangle \in P$ and $\exists M_{out} \langle M_{out} \rangle \notin P$

$R =$ "On input u

 - get $\langle R \rangle$ using recursion theorem
 - Run $D(\langle R \rangle)$
 - If D **accepts** $\langle R \rangle$, **simulate** M_{out} on u
 - If D **rejects** $\langle R \rangle$, **simulate** M_{in} on u

Is $\langle R \rangle \in P$? Contradiction!
- Note: Rice's theorem is only applicable to undecidability of properties, not any languages!
 - It works for, e.g., E_{TM} , but not for languages such as MIN_{TM}
- *Intuition*: The only thing to do with a Turing machine to understand what it does is to run it.