

# Uncomputability

Robert M. Keller  
Harvey Mudd College  
April 2010

## Generality of Decision Problems

- Decision problems are about existence of algorithms to test membership in a language that is the subset of all problem encodings.
- Because Turing machines accept languages, it is possible to ask the question about whether a TM exists that does a certain **algorithm**.
- The fundamental question is whether there is an **algorithm**. We are not generally interested in answering the question for just one specific case.

## Decidability

- A problem is called **decidable** if there is an **algorithm** that will determine a **yes/no** answer for every instance of the problem.
- Equivalently, is there an algorithm that determines membership in the **language** of instances for which the answer is yes.
- The language is said to be a **decidable language**.

## There Are Undecidable Languages

- Every decidable language corresponds to a Turing machine that always halts and gives a yes or no answer, per the Church-Turing thesis.
- The set of **encodings** is countably infinite.
- The set of **languages** is equivalent to the power set of  $\Sigma^*$ , which is **uncountable**.
- So there are languages L for which there is no Turing machine that accepts L.

## Meaning of "Enumerate"

- A set is **enumerable** (or "denumerable") iff it can be put into one-to-one correspondence with the natural numbers or a subset thereof.
- Examples:
  - The set of all **pairs** of natural numbers **is** enumerable.
  - The set of all **subsets** of natural numbers **is not** enumerable.
  - The set of all **finite subsets** of natural numbers **is** enumerable.

## Cantor's Diagonal Argument

- Suppose the set of all languages can be enumerated.

Enumeration of  $\Sigma^*$

	$x_0$	$x_1$	$x_2$	$x_3$	$\dots$
$L_0$					
$L_1$					
$L_2$					
$L_3$					
$\vdots$					
$\vdots$					

Supposed enumeration of **all** languages

Have a 1 in row  $i$  column  $j$  if  $x_j \in L_i$ . Have a 0 otherwise.

## Diagonal Argument

- Have a 1 in row  $i$  column  $j$  if  $x_j \in L_i$ .  
Have a 0 otherwise.

		Enumeration of $\Sigma^*$					
		$x_0$	$x_1$	$x_2$	$x_3$	$\dots$	
Supposed enumeration of <b>all</b> languages	$L_0$	0	1	0	1	0	1
	$L_1$	1	0	1	0	1	0
	$L_2$	0	0	1	1	0	1 ...
	$L_3$						
	$\vdots$						

## Diagonal Argument

- Where is the flattened flipped diagonal in the enumeration of languages?

		Enumeration of $\Sigma^*$					
		$x_0$	$x_1$	$x_2$	$x_3$	$\dots$	
Supposed enumeration of <b>all</b> languages	$L_0$	0	1	0	1	0	1
	$L_1$	1	0	1	0	1	0
	$L_2$	0	0	1	1	0	1 ...
	$L_3$						
	$\vdots$						

flattened flipped diagonal

1 1 0 ...

## Diagonal Argument

- The flattened flipped diagonal can't be anywhere in the enumeration. It disagrees with each language in at least one bit.

		Enumeration of $\Sigma^*$					
		$x_0$	$x_1$	$x_2$	$x_3$	$\dots$	
Supposed enumeration of <b>all</b> languages  Busted!	$L_0$	0	1	0	1	0	1
	$L_1$	1	0	1	0	1	0
	$L_2$	0	0	1	1	0	1 ...
	$L_3$						
	$\vdots$						

flattened flipped diagonal

1 1 0 ...

## Diagonal argument in symbols

- From the supposed enumeration of all languages  $L_0, L_1, L_2, \dots$  and the known enumeration of  $\Sigma^*$ ,  $x_0, x_1, x_2, \dots$ , we constructed a new language not in the enumeration after all:  
 $K = \{x_i \mid x_i \notin L_i\}$   
 For, if  $K$  were in the enumeration it would be  $L_k$  for some  $k$ , thus:  
 $L_k = \{x_i \mid x_i \notin L_i\}$   
 which is absurd,  
 because it implies  $x_k$  is in  $L_k$  iff it isn't in  $L_k$ .

## Summary so far, and direction

- The set of all languages cannot be enumerated. There are "too many".
- The set of all Turing machines *can* be enumerated. Each one can be encoded into a *single element* of  $\Sigma^*$  (the machine's description).
- Thus there must be *some* languages (many, in fact) for which there is no Turing machine.
- That is, *some* languages are not effectively computable.

## Can't we simply add $K$ to the list?

- We could add  $K$  to the list, in principle.
  - This would yield a new list.
  - The diagonal argument can then be repeated on the new list
- and so on, until we get tired.

## A Specific Undecidable Language

- We know that for each decidable language there is a Turing machine accepting that language.
- Some Turing machines don't always halt, so we also have some other machines in the mix of all machines.
- If we enumerate all machines, we'll get all decidable languages, and then some.

## A Specific Undecidable Language

- Again use the diagonal argument, only this time, **enumerate the Turing machines.**

Enumeration of  $\Sigma^*$

	$x_0$	$x_1$	$x_2$	$x_3$	$\dots$
$T_0$					
$T_1$					
$T_2$					
$T_3$					
$\vdots$					

Have a 1 in row  $i$  column  $j$  if  $x_j \in L(T_i)$ . Have a 0 otherwise.

Enumeration of **all** TM's. (This does exist.) All decidable languages are represented.

## A Specific Undecidable Language

- Form a new language  $K = \{x_i \notin L(T_i)\}$  by "flipping" the diagonal as before. This language is not accepted by any Turing machine, hence is not decidable.

Enumeration of **all** TM's. (This does exist.) All decidable languages are represented.

	$x_0$	$x_1$	$x_2$	$x_3$	$\dots$	
$T_0$	0	1	0	1	0	1
$T_1$	1	0	1	0	1	0
$T_2$	0	0	1	1	0	1
$T_3$	1	1	0	...		
$\vdots$						

*flattened flipped diagonal*

## Diagonal Argument in Symbols

- We created language  $K = \{x_i \mid x_i \notin L(T_i)\}$  by "flipping the diagonal".
- We observed that  $K$  is **not in our enumeration**, i.e. it is **not** among the enumeration of all TM-acceptable languages.

## Another Version, Same Idea

- $\langle M \rangle$  is the encoding of machine TM  $M$ , for every  $M$ .
- Consider language  $K = \{\langle M \rangle \mid M \text{ does not accept } \langle M \rangle\}$ .
- There is no Turing machine that accepts  $K$ , i.e.  $K$  is undecidable.**
- Proof: **Suppose some machine,  $N$ , does accept  $K$ .**
- Ask the question whether  $\langle N \rangle \in K$  or not.
- Then the following are equivalent:
  - $\langle N \rangle \in K$  iff, by definition of  $K$
  - $N$  does not accept  $\langle N \rangle$
  - $\langle N \rangle \notin K$  iff, by definition of  $N$  as the machine for  $K$
- Obviously, we have a **contradiction**.

## Accept vs. Recognize

- A TM **accepts** (or "**decides**") a language iff it always halts, and indicates whether its original input is in the language or not.
- A TM  $M$  **recognizes** a language  $L$  iff for each input  $x$ :
  - If  $x \in L$  then  $M$  will halt and indicate acceptance.
  - If  $x \notin L$  then  $M$  does **not** indicate acceptance. ( $M$  **may halt** and reject, or it **may diverge**, i.e. go on forever without halting.)

## Notation Refinement

- Let  $M$  be a Turing machine.
- By  $L(M)$  we will mean the language **recognized** by  $M$ .
- **If**  $M$  always halts,  $L(M)$  is **also** the language **accepted** by  $M$ .

## Recognizability vs. Decidability

- A language is (Turing-) **recognizable** iff it is recognized by some TM.
- A language is (Turing-) **decidable** iff it is accepted by some TM.
- Decidable  $\rightarrow$  recognizable.
- But is the converse true?

## $K$ is not even recognizable, (but its complement is).

- Once again, the language  $K$  defined as  $K = \{ \langle M \rangle \mid \langle M \rangle \text{ encodes a TM and } \langle M \rangle \notin L(M) \}$  (1) is not recognizable by a Turing Machine.
  - Suppose  $K$  were recognized by some TM, say  $N$ , In other words,  $L(N) = K$ . (2)
- Then explore whether  $\langle N \rangle \in K$ .
- $\langle N \rangle \in K$  iff  $\langle N \rangle \notin L(N)$  by the definition of  $K$  (1). and
  - $\langle N \rangle \in K$  iff  $\langle N \rangle \in L(N)$  by the definition of  $N$  (2). So our supposition is invalid.  $K$  is not recognizable.
  - But the **complement**  $K^c$  is recognizable (e.g. using a UTM to recognize  $\langle M \rangle \in L(M)$ ).

## Complementarity Theorem

- A language  $L$  is decidable iff both it and its complement are recognizable.
- Proof:
- ( $\rightarrow$ ) If  $L$  is decidable, it is recognizable. Furthermore, its complement is decidable by the same machine with accept and reject states interchanged. Hence the complement is decidable.
- (Continued on next page.)

## Complementarity Theorem

- Proof continued:
- ( $\leftarrow$ ) Suppose  $L$  and its complement are recognizable. Let  $M$  recognize  $L$  and  $N$  recognize its complement. Construct a deterministic TM  $P$  that decides  $L$ , as follows: For a given input,  $P$  alternates simulating a step of  $M$  on  $x$  with simulating  $N$  on  $x$ . If  $M$  accepts  $x$ , then  $P$  accepts  $x$ . If  $N$  accepts  $x$ , then  $P$  rejects  $x$ .
- One of  $M$  or  $N$  will accept  $x$  eventually (by definition of recognition), so  $P$  will decide  $x$ . Not both of  $M$  or  $N$  will accept  $x$ , because one recognizes the complement of what the other recognizes.

## co-recognizability

- A language is called **co-recognizable** iff its complement is recognizable.
- Thus, from the Complementarity Theorem:
  - $L$  decidable iff ( $L$  recognizable and  $L$  co-recognizable)
  - ( $L$  recognizable but not co-recognizable) implies  $L$  not decidable
  - ( $L$  co-recognizable but not recognizable) implies  $L$  not decidable
- It is possible for a language to be neither recognizable nor co-recognizable, as we shall see.

## More Standard Terminology

- Recall: A language is (Turing-) **decidable** iff it is accepted by some TM.

Much literature uses the term "**recursive**" as a synonym for "**decidable**".

- Recall: A language is (Turing-) **recognizable** iff it is recognized by some TM:

Most literature uses the term "**recursively-enumerable**" (**r.e.**) instead of "**recognizable**".

## Why "recursive"?

- It has to do with the Gödel/Kleene notion of **recursive functions** as being the equivalent of the effectively computable functions.
- A recursive language has a recursive characteristic function.
- Note: Do not read into this anything about the function calling itself, etc. which deals with the way in which the function is *expressed*.

## Why "recursively enumerable"?

- It means there is a recursive function that **enumerates** the language. We'll soon see that this is equivalent to the language being recognizable by a Turing machine.

## Computable Function (vs. decidable language)

A **function**  $f: \Sigma^* \rightarrow \Sigma^*$  is **computable** if there is a TM such that if M is started with x on its tape, M will eventually halt with f(x) on its tape.

Computable functions are also called **recursive functions** in the computability literature.

## Partial Functions

A **partial function** is like a function, except that it can be undefined for some or all values of arguments.

So it has the **uniqueness** property of a function:  $x = y$  implies  $f(x) = f(y)$ , but may lack the **definedness** property, that  $f(x)$  is defined for all x in the domain.

The **same notation** is usually used for function and partial function, relying on context to resolve the distinction.

Sometimes we write  $f(x) = \perp$  to designate " $f(x)$  is undefined". But be aware that  $\perp$  is **no ordinary value**.

## Computable Partial Function

A **partial function**  $f: \Sigma^* \rightarrow \Sigma^*$  is **computable** if there is a TM such that if M is started with x on its tape, M will **eventually halt** with f(x) on its tape, or **diverge** (never halt).

Common notation:

$f(x) \downarrow$  means f(x) is defined.

$f(x) \uparrow$  means f(x) diverges ( $f(x) = \perp$ ).

In general, there is no computable test for  $f(x) = \perp$ . It is just a notational convenience. Computable partial functions are also called **partial recursive functions** in the literature.

## Characteristic Functions

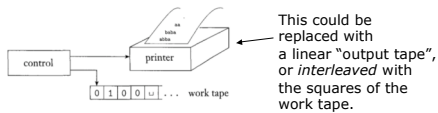
- The **characteristic function** of a language  $L \subseteq \Sigma^*$  is a function  $ch_L: \Sigma^* \rightarrow \{0, 1\}$  defined by
 
$$\forall x \in \Sigma^* \quad ch_L(x) = \begin{cases} 1 & \text{if } x \in L \\ 0 & \text{otherwise} \end{cases}$$
- Every language has one, and every function of this form determines a language.

## Observations

- A language is **decidable** iff it has a computable characteristic function.
- A language is **recognizable** iff it has a computable "partial characteristic function":
 
$$\forall x \in \Sigma^* \quad pch_L(x) = \begin{cases} 1 & \text{if } x \in L \\ \text{undefined} & \text{otherwise} \end{cases}$$

## Enumerability

- The Sipser definition of an Enumerator:
- A Turing machine with a "printer" that prints out all the elements of a language over time.



## The Language Enumerated

- An enumerator **enumerates** a language if, running autonomously, *each* element of the language will get printed at some step ("in finite time").
- If the language enumerated is **infinite**, then the enumerator can never halt.
- If the language enumerated is **finite**, the enumerator may or may not halt (it could print the same element multiple times).

## An Alternate Definition of **Enumerator**

- A Turing machine that interprets its input as an encoded natural number  $\langle i \rangle$ , and
- for each value of  $i$ , the machine **halts** with an element of  $\Sigma^*$  in a designated area of its tape. This element is declared to be the  **$i^{\text{th}}$  element**  $x_i$  of a language  $L$ .
- The language enumerated is  $L = \{x_0, x_1, x_2, \dots\}$  the set of strings produced by halting computations.
- (If  $L$  is finite, there will be repetitions in the sequence.)

## Why is the Alternate Definition Equivalent to Sipser's?

- Given a Sipser enumerator  $S$ , we can construct an Alternate enumerator  $A$ :
  - With input  $\langle i \rangle$ , where  $i \geq 0$ ,  $A$  simulates  $S$  up until  $i+1$  strings have been "printed". Then  $A$  outputs the last string printed as  $x_i$ .
- Given an alternate enumerator  $A$ , we can construct a Sipser enumerator  $S$ :  $S$  simulates calls  $A$  on different arguments:  $A(\langle 0 \rangle)$ ,  $A(\langle 1 \rangle)$ , ..., to produce the elements of the set being enumerated.

## Yet Another, More Liberal, Version of **Enumerator**

- A Turing machine that interprets its input as an encoded natural number  $\langle i \rangle$ , and
- for each value of  $i$ , the machine, **if it halts**, does so with an element of  $\Sigma^*$  in a designated area of its tape. The  $k^{\text{th}}$  element for which the machine halts is declared to be the  **$k^{\text{th}}$  element**  $x_k$  of a language  $L$ .
- The language enumerated is  $L = \{x_0, x_1, x_2, \dots\}$  the set of strings produced by halting computations.
- Since whether the machine halts can't be predicted, we use "dovetailing" to run multiple computations of the machine at the same time.

## Dovetailing

- If  $S$  were to call  $A(\langle 0 \rangle)$ , then  $A(\langle 1 \rangle)$ , then  $A(\langle 2 \rangle)$ , ... and wait until the previous computation succeeded before going on, there could be a problem:  $A(i)$  might **diverge**.
- To avoid this issue,  $S$  simulates:
  - 0 steps of  $A(\langle 0 \rangle)$ ,
  - 1 step of  $A(\langle 0 \rangle)$  then 1 step of  $A(\langle 1 \rangle)$ ,
  - ...
  - $i$  steps of  $A(\langle 0 \rangle)$ ,  $A(\langle 1 \rangle)$ , ...,  $A(\langle i \rangle)$
  - ...
- Whenever one of the  $A(\langle j \rangle)$  halts, the value on the tape is the next string in the enumeration.

## Acceptance Languages

- Sipser's notation includes languages of the form  $A_F$  where  $F$  is some formalism, such as DFA, Regular Expressions, Grammars, etc.
- Example:  $A_{\text{DFA}}$  is the language of encodings of DFAs with a string the DFA accepts.  $\langle B, w \rangle$  is in the language iff:
  - $\langle B \rangle$  encodes a DFA
  - $\langle w \rangle$  encodes an input to the DFA
  - $B$  accepts  $w$
- If  $\langle B \rangle$  or  $\langle w \rangle$  are malformed in some way, then  $\langle B, w \rangle$  is simply not in the language.

## Examples of Acceptance Languages

- $A_{\text{DFA}} = \{\langle B, w \rangle \mid B \text{ is a DFA accepting } w\}$
- $A_{\text{REX}} = \{\langle R, w \rangle \mid R \text{ is a regular expression and } w \in L(R)\}$ .
- $A_{\text{CFG}} = \{\langle G, w \rangle \mid G \text{ is a context-free grammar and } w \in L(G)\}$ .
- $A_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ is a Turing machine and } w \in L(M)\}$ .

Note:  $L(M)$  is the language *recognized* by  $M$ .

## Important Note

- One should not infer there is only one specific algorithm for such languages.
- In particular, an algorithm for deciding (or recognizing) membership will not necessarily involve **executing** a particular model.
- Thinking that it does will only lead to trouble.
- Testing whether  $\langle B, w \rangle \in A_{\text{CFG}}$ , for example, can be done in a variety of ways (such as?).

## Which Languages are Decidable? Recognizable?

- $A_{\text{DFA}} = \{\langle B, w \rangle \mid B \text{ is a DFA accepting } w\}$
- $A_{\text{REX}} = \{\langle R, w \rangle \mid R \text{ is a regular expression and } w \in L(R)\}$ .
- $A_{\text{CFG}} = \{\langle G, w \rangle \mid G \text{ is a context-free grammar and } w \in L(G)\}$ .
- $A_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ is a Turing machine and } w \in L(M)\}$ .

Note:  $L(M)$  is the language *recognized* by  $M$ .

Why  $A_{TM}$  is not decidable.

- Claim:  $A_{TM}$  decidable  $\rightarrow$  K decidable.
- But we know K is not decidable.
- If  $A_{TM}$  were decidable, we could create an algorithm for deciding K as well:
  - With input  $\langle M \rangle$ , construct  $\langle M, \langle M \rangle \rangle$  (the description of M together with its own description) and pass it to the algorithm for  $A_{TM}$ .  $\langle M, \langle M \rangle \rangle \notin A_{TM}$  iff M does not accept  $\langle M \rangle$  iff  $\langle M \rangle \in K$ .
- Thus an algorithm  $A_{TM}$  can be used to construct an algorithm for K, which we previously showed to be impossible.

Corollary:  $A_{TM}^c$  (the complement of  $A_{TM}$ ) is not recognizable.

- Why?

## The "Halting Problem"

- $HALT_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w \}$
- $HALT_{TM}$  is undecidable.

## Proof that $HALT_{TM}$ is undecidable

- Suppose  $HALT_{TM}$  were decidable. That means there is an algorithm for it. We'll use that algorithm to **construct an algorithm for  $A_{TM}$** , which we previously showed to be impossible.
- Algorithm: With input  $\langle M, w \rangle$ , first check if  $\langle M, w \rangle \in HALT_{TM}$ .
  - **If not**, then **reject**, as M cannot accept w in this case.
  - **If so**, then simulate M on w (e.g. using a universal TM), and accept iff M accepts w.

## Summary

- An algorithm for  $HALT_{TM}$  could be used to construct an algorithm for  $A_{TM}$   
thus
- $HALT_{TM}$  decidable  $\rightarrow A_{TM}$  decidable  
equivalently (contrapositive)
- $A_{TM}$  **undecidable**  $\rightarrow HALT_{TM}$  **undecidable**

## Terminology

- On the previous slide, we have **reduced  $A_{TM}$  to the halting problem**, in the sense that **if** the halting problem were solvable, so would  $A_{TM}$  be.  
But the latter was already established to be unsolvable.
- Note: It is **not correct** to say the opposite, that we have reduced the halting problem to  $A_{TM}$  (yet, but this can be done).

## Notation for Reduction

- If a problem (language) A can be reduced to a problem (language) B we write:

$$A \leq B$$

sort of suggesting that B is "at least as difficult" as A.

So if A is unsolvable, so must be B.

## Different kinds of reduction

- What we just saw was an example of a **Turing reduction**.

- Language  $A \subseteq \Sigma^*$  is **Turing reducible** to  $B \subseteq \Sigma^*$ , notated  $A \leq_T B$  provided:

an algorithm for deciding A can be implemented by calling an algorithm for deciding B (querying B as if it were an **oracle**). Any number of such calls can be used in general.

## Mapping Reducibility $\leq_m$

- Language  $A \subseteq \Sigma^*$  is **mapping reducible** to  $B \subseteq \Sigma^*$ , notated  $A \leq_m B$  provided:

there is some computable  $f: \Sigma^* \rightarrow \Sigma^*$  such that

$$x \in A \text{ iff } f(x) \in B.$$

- In this case, f is called a **reduction** of A to B.
- Mapping reductions are also called "many-to-one" reductions.
- They are a special case of Turing reductions, in which the oracle is only called once, to give the final answer. (Sort of like the **tail-recursive** version of  $\leq_T$ .)

## $A \leq_m B$ and computability

- $A \leq_m B$  implies if B were computable, so would A be:

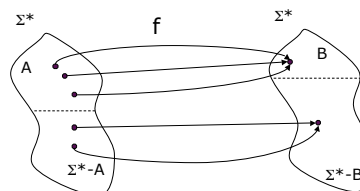
To determine if x is in A: compute  $f(x)$  by machine, determine whether  $f(x)$  is in B, to get your answer.

- It also means the contrapositive, that if A is uncomputable, so is B.

## Example

- $K \leq_m A_{TM}^c$
- $K = \{ \langle M \rangle \mid \langle M \rangle \notin L(M) \}$
- $A_{TM}^c = \{ \langle M, w \rangle \mid w \notin L(M) \}$
- The reduction mapping in this case is:  
 $f(\langle M \rangle) = \langle M, M \rangle$

## $\leq_m$ Mapping need not be 1-1



### How about $\text{HALT}_{\text{TM}} \leq A_{\text{TM}}$ ?

- Here we can use a mapping reduction.
- Assume  $A_{\text{TM}}$  is decidable. Map  $\text{HALT}_{\text{TM}}$  into it.
- Given a machine and input  $\langle M, w \rangle$ , map it to  $\langle M', w \rangle$  by creating  $\langle M' \rangle$  from  $\langle M \rangle$ :
  - Change all **halting** (accepting or rejecting) states of  $M$  into **accepting** states of  $M'$ , leaving everything else unchanged.
  - The mapping reduction  $f$  does this transformation:  
 $f(\langle M, w \rangle) = \langle M', w \rangle$ .
  - So  $M$  halts on  $w$  iff  $M'$  accepts  $w$ .
  - i.e.  $\langle M, w \rangle \in \text{HALT}_{\text{TM}}$  iff  $\langle M', w \rangle \in A_{\text{TM}}$ .

### Emptiness Problems

- $E_{\text{DFA}} = \{ \langle B \rangle \mid B \text{ is a DFA accepting no strings} \}$
- $E_{\text{REGEX}} = \{ \langle R \rangle \mid R \text{ is a regular expression and } L(R) = \emptyset \}$
- $E_{\text{CFG}} = \{ \langle G \rangle \mid G \text{ is a context-free grammar and } L(G) = \emptyset \}$ .
- $E_{\text{CSG}} = \{ \langle G \rangle \mid G \text{ is a context-sensitive grammar and } L(G) = \emptyset \}$ .
- $E_{\text{TM}} = \{ \langle M \rangle \mid M \text{ is a Turing machine and } L(M) = \emptyset \}$
- Which of these are decidable? (Not necessarily obvious).

### $E_{\text{TM}}$ is not decidable

- Use a mapping reduction  $A_{\text{TM}} \leq_m E_{\text{TM}}^c$ , the latter accepting those  $\langle M \rangle$  where  $L(M)$  is **non-empty**, in other words  $M$  accepts **some** input.
- Transformation: With input  $\langle M, w \rangle$ , construct machine  $\langle M' \rangle$  such that:
  - $\langle M' \rangle \in E_{\text{TM}}^c$   
iff
  - $\langle M, w \rangle \in A_{\text{TM}}$
- $M'$ : With input  $x$ , erase  $x$ , write  $w$  on the tape, then behave as  $M$ .

### Property of $M'$

- $M'$  does exactly the same thing on every input  $x$ .
  - It accepts  $x$  iff  $M$  accepts  $w$ .
  - It rejects  $x$  iff  $M$  rejects  $w$ .
  - It diverges on  $x$  iff  $M$  diverges on  $w$ .
- So it accepts **some** input iff  $M$  accepts  $w$ .

### Is either $E_{\text{TM}}$ or its complement recognizable?

### $A \leq_m B$ and recognizability

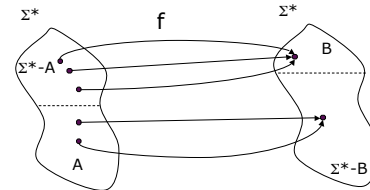
- $A \leq_m B$  also implies that if  $B$  were recognizable, so would  $A$  be.
- To recognize if  $x$  is in  $A$ : compute  $f(x)$  by machine, ask whether  $f(x)$  is in  $B$ .  
  
If the answer is affirmative, then the original  $x$  was in  $A$ .  
If we never get an answer, then we don't get answer for whether the original  $x$  was in  $A$  either.
- implies if  $A$  is *not* recognizable, neither is  $B$ .

## $A \leq_m B$ and co-recognizability

- $A \leq_m B$  similarly implies that if  $B$  were co-recognizable, so would  $A$  be.
- $A \leq_m B$  implies if  $A$  is *not* co-recognizable, neither is  $B$ .

## $A^c \leq_m B$

- $A^c \leq_m B$  means the complement of  $A$  reduces to  $B$ . This is **equivalent** to  $A \leq_m B^c$ .



## Sipser Problem 5.7

- If  $A$  is recognizable, and  $A \leq_m A^c$ , then  $A$  is decidable.
- Proof: If  $A \leq_m A^c$ , then also  $A^c \leq_m A$ , per the previous slide. But since  $A$  is recognizable, by  $A^c \leq_m A$ ,  $A^c$  is also recognizable. But then both  $A$  and  $A^c$  are recognizable, so  $A$  is decidable from the complementarity lemma.

## Example

- We already showed  $A_{TM} \leq_m E_{TM}^c$ .
- In effect, the same mapping gives  $A_{TM}^c \leq_m E_{TM}$ .
- But  $A_{TM}^c$  is not recognizable.
- Therefore  $E_{TM}$  is not recognizable.
- However,  $E_{TM}^c$  is co-recognizable.

## "ALL" Problems

- $ALL_{DFA} = \{ \langle B \rangle \mid B \text{ is a DFA accepting all strings} \}$   
("all" means all of  $\Sigma^*$  where  $\Sigma$  is the alphabet of the DFA)
- $ALL_{REG} = \{ \langle R \rangle \mid R \text{ is a regular expression and } L(R) = \Sigma^* \}$
- $ALL_{CFG} = \{ \langle G \rangle \mid G \text{ is a context-free grammar and } L(G) = \Sigma^* \}$ .
- $ALL_{CSG} = \{ \langle G \rangle \mid G \text{ is a context-sensitive grammar and } L(G) = \Sigma^* \}$ .
- $ALL_{TM} = \{ \langle M \rangle \mid M \text{ is a Turing machine and } L(M) = \Sigma^* \}$
- Which of these are decidable? (Not necessarily obvious).

## $ALL_{TM}$

- Is  $ALL_{TM}$  decidable?
- What is your intuition?
- How would you prove it?
- (Remember the tune "All or Nothing At All".)
- What about recognizable or co-recognizable, if not decidable.

## Fixed-String Acceptance

- Suppose  $x_0$  is a fixed string, such as the empty string (blank tape).
- Is there an algorithm for determining whether an arbitrary TM accepts  $x_0$ ?
- We call the corresponding language acceptance problem **Accepts- $x_0$  TM**.

Accepts- $x_0$  TM is unsolvable, for any fixed  $x_0$ .

- See if you can reduce one of the other unsolvable problems to this problem.

## Regular<sub>TM</sub> is Undecidable

- Define  $\langle M \rangle \in \text{Regular}_{\text{TM}}$  iff  $L(M)$  is regular.
- Reduce a known unsolvable problem to Regular<sub>TM</sub>.
- Try a mapping reduction.

## Reduction to Regular<sub>TM</sub>

- Reduce  $A_{\text{TM}}$  to Regular<sub>TM</sub> as follows:
- Given an instance  $\langle M, w \rangle$ , create  $\langle M' \rangle$  as follows:
  - $M'$  with input  $x$ : check whether  $x$  has the form  $0^n 1^n$  for some  $n$ .
  - If  $x$  does not have the form, reject.
  - If  $x$  does have the form, run  $M$  on  $w$ , accepting when and if  $M$  accepts  $w$ . (This may, of course, diverge, in which case  $M$  cannot accept  $w$ .)

## Reduction to Regular<sub>TM</sub>

- Thus  $M'$  accepts exactly strings of the form  $0^n 1^n$  iff  $M$  accepts  $w$ . Otherwise  $M'$  accepts  $\emptyset$ .
- So  $L(M')$  is either
  - regular ( $\emptyset$ ) or
  - non-regular ( $\{0^n 1^n \mid n \geq 0\}$ )depending on whether  $M$  accepts  $w$ .

[We've thus shown  $A_{\text{TM}} \leq_m \text{Regular}_{\text{TM}}^c$ .]

What about Regular itself?

## Functional vs. Structural Properties

- We have been using TM's to represent languages.
- When a property of a machine depends only on the **language** and not the specific TM used to represent the language, we call this a **functional property**.
- When a property depends on the **specific** details of a **machine**, it is called a **structural property**.

## Functional vs. Structural Examples

- **Functional:**
  - $L(M) = \emptyset$
  - $L(M)$  is regular
  - $L(M)$  is infinite
  - $L(M)$  is decidable
  - $L(M)$  is recognizable (always true, by definition)
- **Structural:**
  - $M$  has more than 100 reachable control states
  - $M$  writes a non-blank character on its tape when started on an empty tape.
  - $M$  reverses its direction of head travel at least once on each input.

## Decidable vs. Undecidable Structural Properties

- $M$  has more than 100 control states.
- $M$  reaches a specific control state when started on an empty tape.
- $M$  writes a non-blank character on its tape when started on an empty tape.
- $M$  uses more than a specified amount of tape when started on a certain input.
- Which of these properties is decidable?

## Rice's Theorem, Informally

- Practically **no** functional properties are decidable for TM recognizable languages.
- The caveat here is that the theorem only applies to functional properties, not structure ones.

## Trivial Functional Properties

- A functional property of a recognizable language is called **trivial** if it is either:
  - true for *no* recognizable language, or
  - true for *all* recognizable languages
- A **non-trivial property**, then, holds for **some** recognizable languages, but **not all**.

## Rice's Theorem

- **Any non-trivial functional property of the recognizable languages is not decidable.**
- Put another way:

For any non-trivial functional property, there is no algorithm that will determine whether or not the language recognized by a given TM has the property.

## Observations about Non-Trivial Properties P

- Any given language has property P, or it does not.
- A language L has property P iff L does not have  $\neg P$ .
- To decide P, it is adequate to decide the complementary property  $\neg P$ , since yes/no answers are required in both cases.

## Proof of Rice's Theorem (1 of 3)

- Suppose  $P$  is a non-trivial property.
- **Critical assumption: The empty language  $\emptyset$  does not have property  $P$ .** If the opposite is true, then interchange  $P$  and  $\neg P$  so that the assumption is true. [This would be necessary in the case of  $P = \text{Regular}$ , for example.]
- Let  $L_P$  be some arbitrary recognizable language **with** property  $P$  (which must exist, because  $P$  is non-trivial). We know that  $L_P$  is distinct from  $\emptyset$ , by the assumption above. Let  $M_P$  be a machine recognizing the chosen language  $L_P$ .
- **Plan:** Reduce the acceptance problem to that of deciding property  $P$ , which will imply that there is no algorithm for the latter. This will hinge on having the  $P$  decider **differentiate** between  $L_P$ , which has property  $P$ , and  $\emptyset$ , which does not.

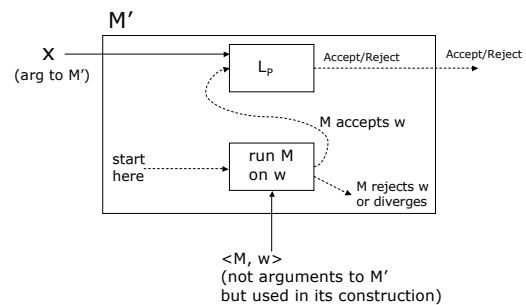
## Proof of Rice's Theorem (2 of 3)

- The reduction works as follows: Suppose we have an algorithm for testing property  $P$  for any input  $\langle M \rangle$ .
- We can then test whether an arbitrary Turing machine  $M$  accepts an arbitrary input  $w$  by constructing a machine  $M'$  with the specifications on the following page.

## Proof of Rice's Theorem (3 of 3)

- $M'$ : with input  $x$ , (temporarily set aside  $x$  and) start **behaving as**  $M$  on  $w$ .
- **If  $M$  on  $w$  accepts**, then continue by behaving as  $M_P$  on the original input  $x$ .
- Thus if and when  $M$  accepts  $w$ ,  $M'$  will accept  $x$  iff  $M_P$  accepts  $x$ , i.e.  $L(M') = L(M_P)$ . So in this case,  $L(M')$  **has** property  $P$ , because  $M_P$  was selected to have it.
- If  $M$  on  $w$  **does not terminate**, then  $L(M') = \emptyset$ , which by the critical assumption, **does not** have property  $P$ .
- So if we can test an arbitrary machine whether its language has property  $P$ , we can test  $M'$  in particular. But the answer to this test determines whether or not  $M$  accepts  $w$ .

## Diagram of $M'$ as constructed (dashed lines = control, solid = data)



## Note

- Although  $P$  was discussed as a property of **language** recognized by a Turing machine, the same proof works in the case that:
  - $P$  is a property of the **partial function** computed by a Turing machine.
- This means, for example, there is no algorithm that will decide equivalence of an arbitrary machine's partial function to that of a given machine.

## Languages neither recognizable nor co-recognizable.

- $EQ_{TM}$  is neither.
- **Proof that  $EQ_{TM}$  is not recognizable:** Use reduction from  $A_{TM}$  (which is not co-recognizable) to  $EQ_{TM}^c$ . Given  $\langle M, w \rangle$ , construct  $M_1$  and  $M_2$  thus:
  - $M_1$  **always rejects**.
  - $M_2$  on any input, behaves as  $M$  on  $w$ .
- So  $\langle M_1, M_2 \rangle \notin EQ_{TM}$  iff  $\langle M, w \rangle \in A_{TM}$ .

## Languages neither recognizable nor co-recognizable.

- $EQ_{TM}$  is neither.
- Proof that  $EQ_{TM}^c$  is not recognizable:  
Use reduction from  $A_{TM}$  (which is not co-recognizable) to  $EQ_{TM}$ . Given  $\langle M, w \rangle$ , construct  $M_1$  and  $M_2$  thus:
  - $M_1$  **always accepts**.
  - $M_2$  on any input, behaves as  $M$  on  $w$ .
- So  $\langle M_1, M_2 \rangle \in EQ_{TM}$  iff  $\langle M, w \rangle \in A_{TM}$ .

## Languages neither recognizable nor co-recognizable.

- $ALL_{TM}$  is neither.
- To show  $ALL_{TM}$  is **not co-recognizable**, give a mapping reduction from  $A_{TM}$  (which is not co-recognizable), to  $ALL_{TM}$ .  
 $f(\langle M, w \rangle) = M'$ , where  $M'(x) = M(w)$ .  
Thus  $M'$  accepts all iff  $M$  accepts  $w$ .
- But how to show  $ALL_{TM}$  is not recognizable??

## $ALL_{TM}$ is not recognizable

- We prove this by  $HALT_{TM}^c \leq_m ALL_{TM}$ .
- Assume that  $ALL_{TM}$  is recognizable.
- Let  $\langle M, w \rangle$  be an arbitrary machine with input. The reduction mapping constructs  $M'$  to behave as follows:
  - $M'(x)$  simulates  $M(w)$  for  $|x|$  steps.
  - If  $M(w)$  **fails to halt** within  $|x|$  steps, accept  $x$ .
  - Otherwise reject  $x$ .
- Hence  $M'$  accepts all inputs iff  $M(w)$  does not halt.