# Theory of Computation
# Chapter 5

## Reductions

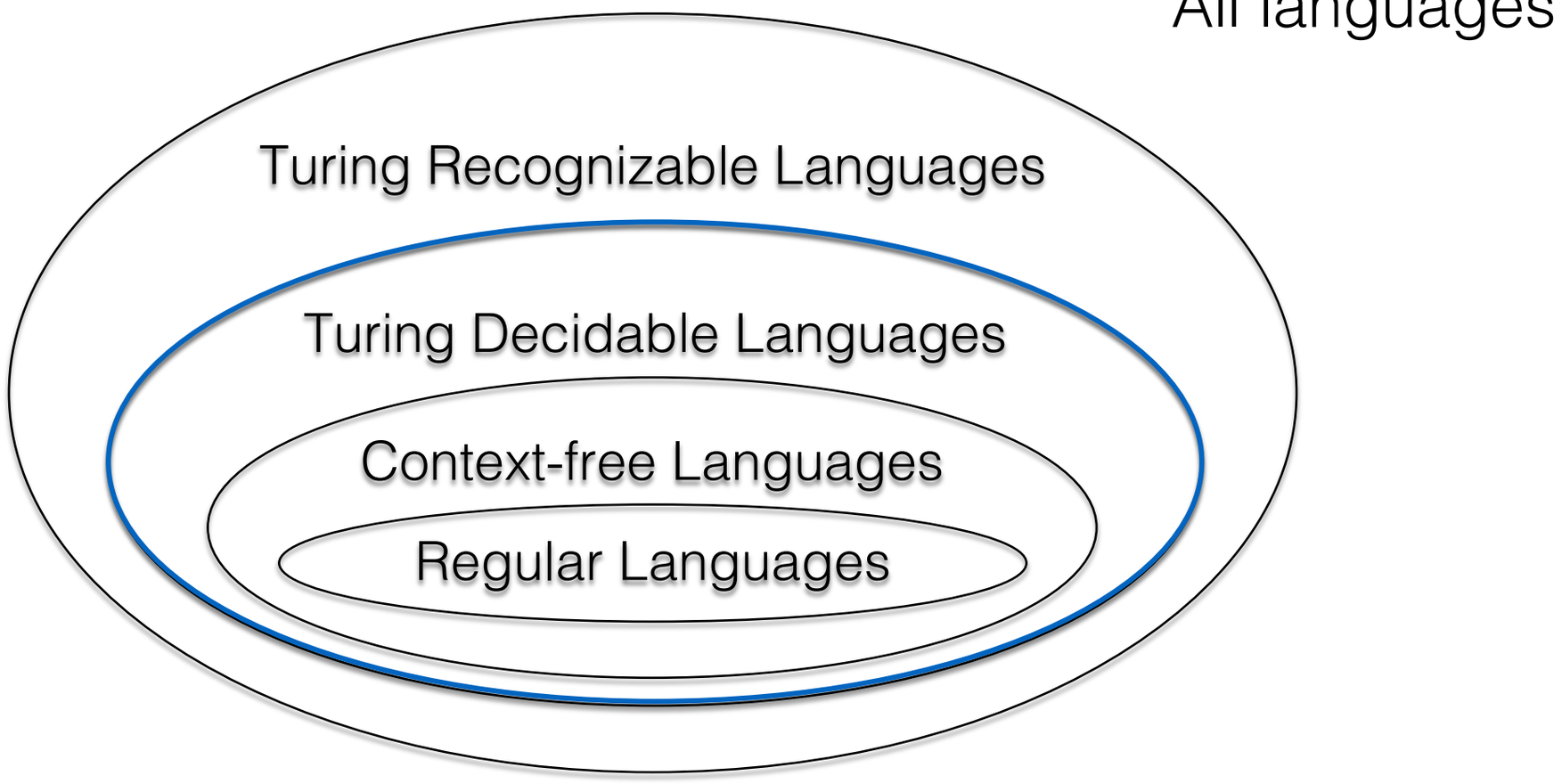
Additional Source: Hopcroft, Motwani, and Ullman,
Introduction to
Automata Theory, Languages and Computation


School of Engineering | Computer Science

# David Hilbert
# 1862 - 1943

- German mathematician and philosopher of mathematics and one of the most influential mathematicians of his time
- In 1900, he presented a collection of problems that set a course for mathematical research of the 20th century
- Contributed to establishing rigor and developed important tools used in modern mathematical physics.
- Cofounder of proof theory and mathematical logic

# Decidability



# Undecidability

- Fact: Almost all problems must be undecidable by any system that involves programming.
  - A problem is the membership of a string in a language
  - The number of different languages over any alphabet of more than one symbol is not countable
    - There is no way to assign integers to the languages such that every language has an integer, and every integer is assigned to one language
  - On the other hand, programs, being finite strings over a finite alphabet are countable, thus there are infinitely fewer programs as there are problems

# Undecidable Problem

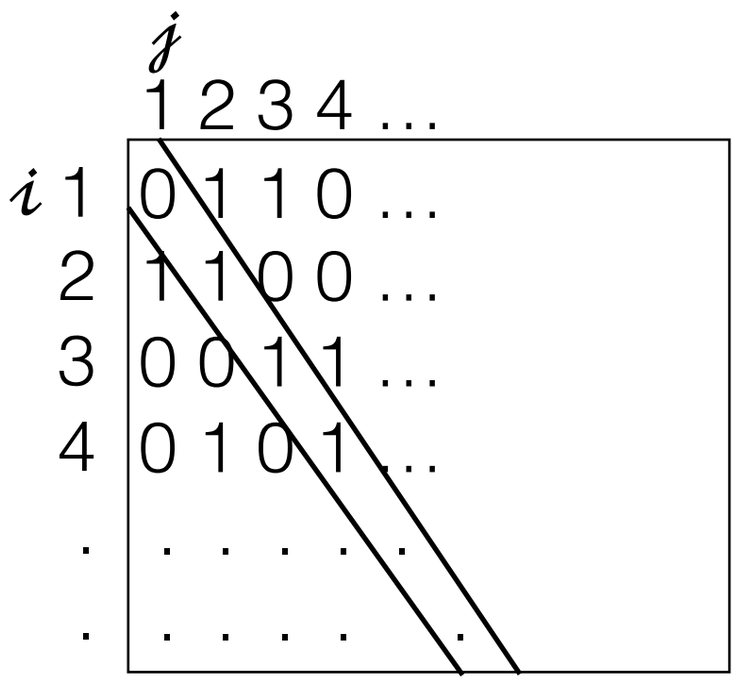
- Remember that we can encode Turing Machines as strings using $\Sigma = \{0,1\}$. If we create a formal process for doing this, we would get some strings that represent Turing Machines (TM).
- However, most strings from $\Sigma^*$ do not follow this process and do not represent Turing Machines.
- We look at some strings, $w_i$, from $\Sigma^*$ to see if they were valid Turing Machines <M>. If $w_i$ is not a valid TM encoding, we say that this encoding is a Turing Machine, $M_i$, with one state and no transitions (a TM that immediately halts on any input), so $L(M_i)$ is $\emptyset$.

# Undecidable Problem

- $L_d$ can be the Diagonalization Language, which is the set of strings $w_i$ such that $w_i$ is not in $L(M_i)$.
- $L_d$ consists of all strings $w$ such that the TM M whose encoding is $w$ does not accept when given $w$ as input.
- $L_d$ is called a Diagonalization Language because of our diagonalization argument.

# Undecidable Problem

- $L_d$ is called the Diagonalization Language because of our diagonalization argument
- The table shows whether the TM $M_i$ accepts input string $w_j$. 1 means yes, 0 means no.
- The $i$th row is the vector for the language $L(M_i)$



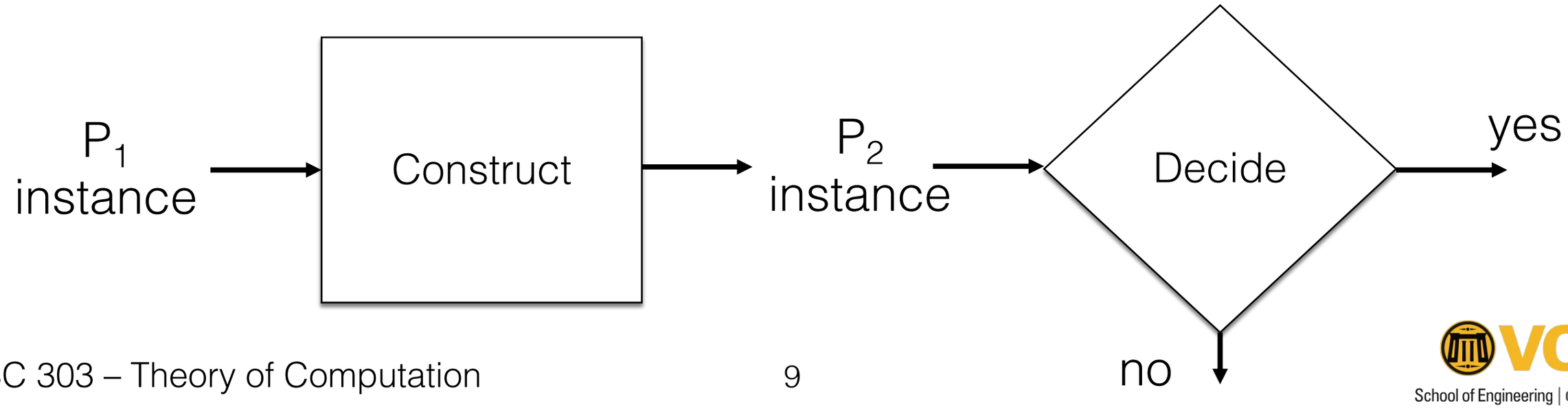

- To construct $L_d$ we take the complement of the diagonal so that the language is not in the table and not the diagonal.

# Proof $L_d$ Is not Decidable

- There is no Turing Machine to decide $L_d$.
- Proof: Suppose $L_d$ were L(M) for some TM M. Since $L_d$ is a language over $\Sigma = \{0,1\}$, M would be in the list of TM's with input $\Sigma = \{0,1\}$. Thus, there is at least one encoding for M, say $M = M_i$.
- Let's see if $w_i$, the encoding, is in $L_d$:
  - If $w_i$ is in $L_d$, then $M_i$ accepts $w_i$. But we know that $w_i$ is not in $L_d$ by the definition of $L_d$ because $L_d$ contains only those $w_j$ such that $M_j$ does not accept $w_j$.
  - Also, if $w_i$ is not in $L_d$, then $M_i$ does not accept $w_i$, thus by definition of $L_d$, $w_i$ is in $L_d$.
  - Since $w_i$ can neither be in $L_d$ nor fail to be in $L_d$, we conclude a contradiction, so $L_d$ is not decidable.

# Reductions

- Now that we have proven the existence of an undecidable problem, we can use this problem to help us prove other undecidable problems
  - Use reduction strategy of one problem to another
    - If we know problem $P_1$ is undecidable, we can reduce $P_1$ to $P_2$, to prove that $P_2$ is also undecidable

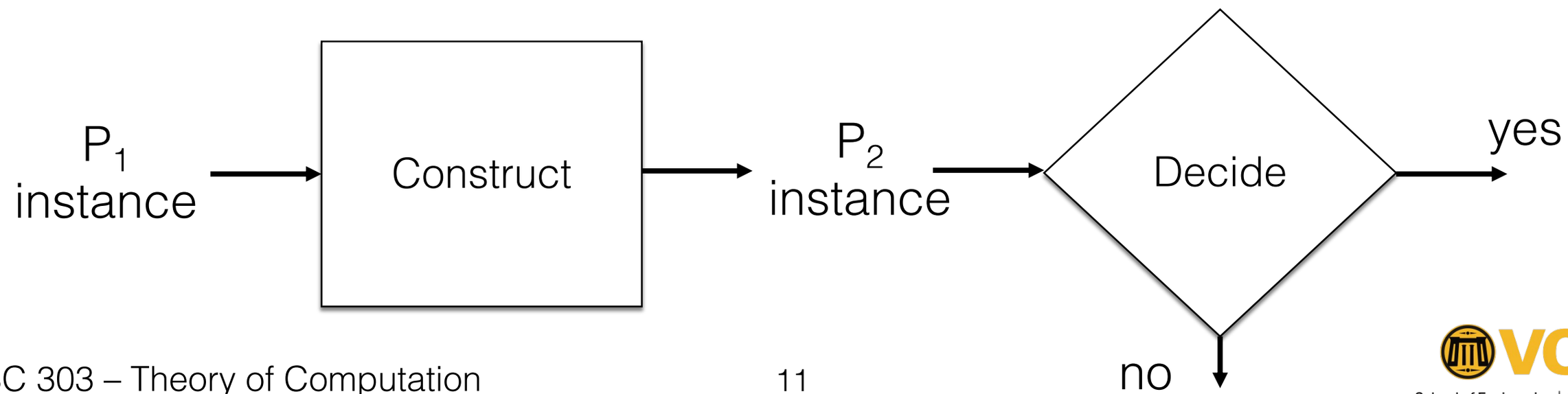


# Reductions

- The main way to prove problems are undecidable is using reductions.
- A reduction converts one problem to another problem so that a solution to the second problem helps to solve the first problem.
- Goal: Show relationship between problems.
- Let A and B be problems. We can say A reduces to B and write $A \leq B$ if "the ability to solve B implies the ability to solve A".
  - Ex: Need to find your way around a new city.

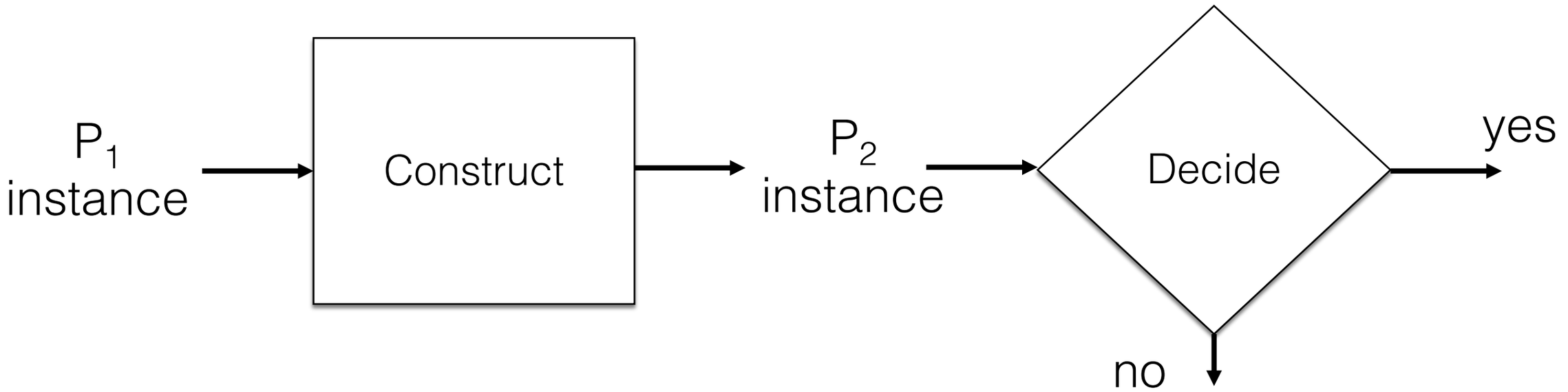
# Reductions

- Reduction: $P_1 \leq P_2$
  - Given an instance of $P_1$, a string w, that may or may not be in $P_1$, apply a construction algorithm to produce string x.
    - Test if x is in $P_2$, and give the same answer about w in $P_1$



# Reductions

- Reduction: $P_1 \leq P_2$



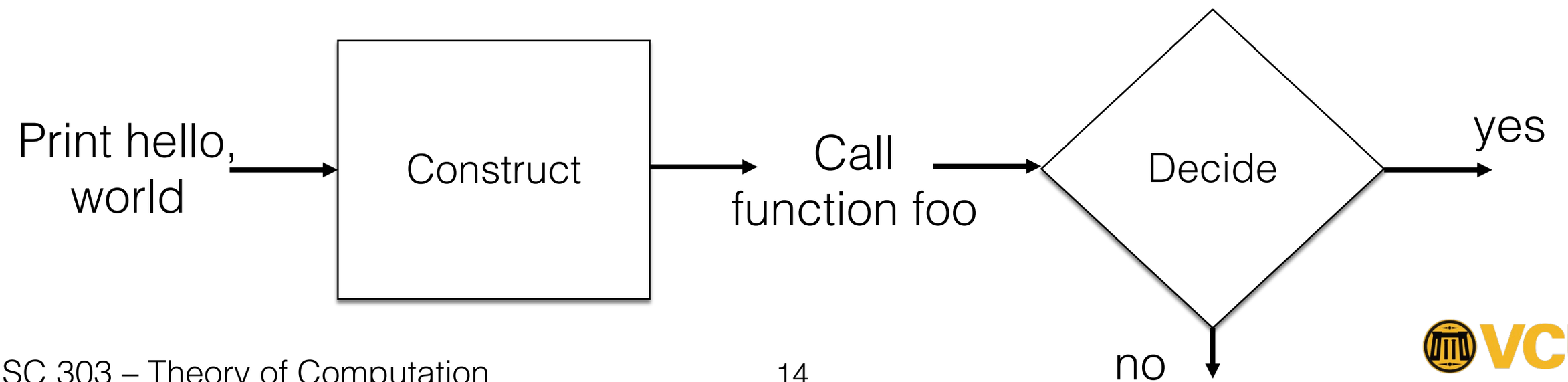

- If w is in $P_1$, then x is in $P_2$, so this algorithm says yes. If w is not in $P_1$, then x is not in $P_2$, and the algorithm says no.
- Since no algorithm exists to decide membership of a string in $P_1$, also, no algorithm for $P_2$ exists, so $P_2$ is undecidable.

# Common Reduction Mistake

- The direction of the reduction is important.
- It is a common mistake to try to prove a problem $P_2$ is undecidable by reducing it to some known undecidable problem $P_1$. This proves $P_2 \leq P_1$.
  - Proves "If $P_1$ is decidable, then $P_2$ is decidable". This is useless since hypothesis "$P_1$ is decidable" is false.
  - The only way to prove a new problem $P_2$ is undecidable is to reduce a known undecidable problem $P_1$ to $P_2$.
    - This shows "if $P_2$ is decidable, then $P_1$ is decidable".
    - The contrapositive of that statement is "If $P_1$ is undecidable, then $P_2$ is undecidable." Thus, we show $P_2$ is undecidable.

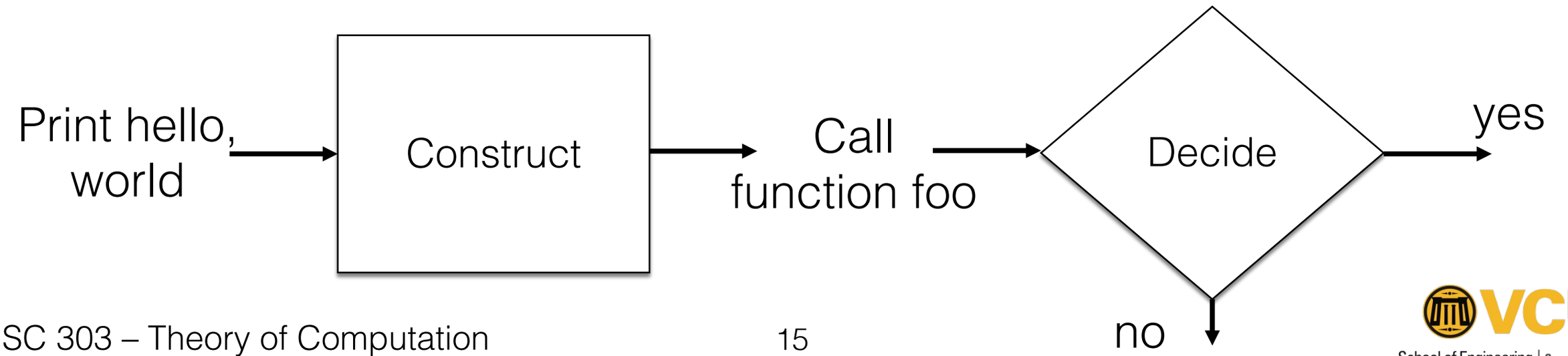
# Reduction Example

- Ex: We want to show that the question: “Does program Q, given input y, ever call function foo”
- Reduction: $P_1 \leq P_2$, this time:
  - $P_1$ is our known undecidable problem, “print hello, world”
  - $P_2$ is this problem of “call function foo”



# Reduction Example

- Ex: We want to show that the question: “Does program Q, given input y, ever call function foo”
- Given program Q and input y, we must construct a program R and an input z such that R, with input z, calls foo if and only if Q with input y prints hello, world
- How do we construct this?



# Reduction Example

- Ex: We want to show that the question: “Does program Q, given input y, ever call function foo”
- Construction:
  - If Q has a function called foo, rename it and all calls to that function → $Q_1$
  - Add to $Q_1$ a function foo. This function does nothing and is not called → $Q_2$
  - Modify $Q_2$ to remember the first 12 characters that it prints, storing them in a global array A → $Q_3$
  - Modify $Q_3$ so that whenever it executes any output statement, it then checks in the array A to see if it has written 12 characters or more and if “hello, world” are the first 12 characters. If that is the case, call the new function foo. → R
  - Thus, you now have R and input z is the same as y.

# Reduction Example

- Ex: We want to show that the question: “Does program Q, given input y, ever call function foo”
- With the Q with input y and R with input z:
  - If Q with input y prints “hello, world” as its first output, then R as constructed will call foo
  - However, if Q with input y does not print “hello, world” as its first output, then R will never call foo.
  - If we can decide R, then we can decide Q.
  - Since we know no algorithm exists to solve Q, there is no algorithm that exists to solve R.

# Common Reduction Mistake

- The direction of the reduction is important.
- It is a common mistake to try to prove a problem $P_2$ is undecidable by reducing it to some known undecidable problem $P_1$. This proves $P_2 \leq P_1$.
  - Proves "If $P_1$ is decidable, then $P_2$ is decidable". This is useless since hypothesis "$P_1$ is decidable" is false.
  - The only way to prove a new problem $P_2$ is undecidable is to reduce a known undecidable problem $P_1$ to $P_2$.
    - This shows "if $P_2$ is decidable, then $P_1$ is decidable".
    - The contrapositive of that statement is "If $P_1$ is undecidable, then $P_2$ is undecidable." Thus, we show $P_2$ is undecidable.

# Reductions

- The primary method for proving that problems are computationally unsolvable or undecidable is reducibility
- A reduction is a way of converting one problem to another problem in such a way that a solution to the second problem can be used to solve the first problem.
- The goal is to show the relationship between the different problems

# Reductions

- A reduction converts one problem to another problem so that a solution to the second problem helps to solve the first problem.
- Goal: Show relationship between problems.
- Let A and B be problems. We can say <u>A reduces to B</u> and write <u>A ≤ B</u> if "the ability to solve B implies the ability to solve A".
  - Ex: A = Need to find your way around a new city
    - B = Need to find a map of the city
    - The map solution to B can help you solve A, A ≤ B
- There is a clear connection between A and B

# $L_d$ & $\overline{L_d}$

- Remember that $L_d$ is undecidable, but what about $\overline{L_d}$? Is this language undecidable?
- Know $\overline{L_d}$ is not Turing Decidable, but it is Turing Recognizable.
- $\overline{L_d}$ is the set of strings $w_i$ such that $M_i$ accepts $w_i$, which is very similar to our $A_{TM}$ from before.
- Now we have more power for our reductions, not only can we show undecidable, but also Turing Recognizable

# The Halting Problem

- This is one of the most important problems in Computer Science Theory
- We define $HALT_{TM}$ = {<M, w> | M is a TM which halts on input w}
- Theorem 5.1: $HALT_{TM}$ is undecidable
- Proof Idea: Show that $A_{TM} \leq HALT_{TM}$, if we can decide $HALT_{TM}$ then we can decide $A_{TM}$, but $A_{TM}$ is undecidable. Therefore, $HALT_{TM}$ is undecidable.

# The Halting Problem

- Theorem 5.1: $HALT_{TM}$ is undecidable
- Proof: Let R be a TM deciding $HALT_{TM}$. Construct a TM S to decide $A_{TM}$:
  - S = "on input <M, w>,
    1. Run TM R on input <M, w> to see if M would halt on w.
    2. If R rejects, reject (w is not in the language)
    3. If R accepts, run M on w and output its answer."
  - What do we know about Step 2? TM S that decides $A_{TM}$ does not halt and reject – contradiction, so $HALT_{TM}$ is undecidable.

# TM Recognizing Ø

- Theorem 5.2: $E_{TM}$ = {<M> | M is a TM and L(M) = Ø} is undecidable
- Proof by Contradiction: Suppose we have a TM R deciding $E_{TM}$. We can show that $A_{TM}$ is decidable using R. ($A_{TM} \leq E_{TM}$)

# TM Recognizing Ø

- Theorem 5.2: $E_{TM}$ = {<M> | M is a TM and L(M) = Ø} is undecidable
- Proof by Contradiction: Suppose we have a TM R deciding $E_{TM}$. We can show that $A_{TM}$ is decidable using R. ($A_{TM} \leq E_{TM}$)
  - Step 1: Construct TM $M_w$ such that:
    - If $w \in L(M)$, then $L(M_w) \neq \emptyset$
    - If $w \notin L(M)$, then $L(M_w) = \emptyset$
  - TM $M_w$: "on input x:
    1. If $x \neq w$, reject
    2. If $x = w$, run M on w and accept if M accepts."

Modify M to guarantee that M rejects all strings except w, but on w it works as usual

# TM Recognizing Ø

- Theorem 5.2: $E_{TM}$ = {<M> | M is a TM and L(M) = Ø} is undecidable
- Proof by Contradiction cont.: Suppose we have a TM R deciding $E_{TM}$. We can show that $A_{TM}$ is decidable using R. ($A_{TM} \leq E_{TM}$)
  - What is $L(M_w)$?
    - $L(M_w) \subseteq \{w\}$
    - Two cases:
      1. If $w \in L$, then $M_w(w)$ accepts, so $L(M_w) \neq \emptyset$
      2. If $w \notin L$, then $M_w(w)$ rejects or loops forever, so $L(M_w) = \emptyset$

# TM Recognizing Ø

- Theorem 5.2: $E_{TM}$ = {<M> | M is a TM and L(M) = Ø} is undecidable
- Proof by Contradiction cont.: Suppose we have a TM R deciding $E_{TM}$. We can show that $A_{TM}$ is decidable using R. ($A_{TM} \leq E_{TM}$)
- Step 2: Use R to distinguish between Cases 1 and 2 above. Here is a TM S which decides $A_{TM}$ given R:
  - S = "On input <M, w> for $A_{TM}$,
    1. Construct $M_w$
    2. Run R on input <M, w>
    3. If R accepts (i.e. $L(M_w) \neq \emptyset$), reject

       If R rejects (i.e. $L(M_w) = \emptyset$), accept"

A decider for $A_{TM}$ does not exist, so $E_{TM}$ must be **undecidable**

# TM Recognizing $EQ_{TM}$

- Theorem 5.4: $EQ_{TM} = \{<M_1, M_2> \mid M_1$ and $M_2$ are TMs and $L(M_1) = L(M_2)\}$ is undecidable
- Proof by Contradiction: Suppose we have a TM R deciding $EQ_{TM}$. We can show that $E_{TM}$ is decidable using R. ($E_{TM} \leq EQ_{TM}$)
  - S = "On input <M>, where M is a TM:
    1. Run R on input <M, $M_1$>, where $M_1$ is a TM that rejects all inputs
    2. If R accepts, accept. If R rejects, reject."
  - If R decides $EQ_{TM}$, S decides $E_{TM}$. We know that $E_{TM}$ is undecidable, so $EQ_{TM}$ must be undecidable.

# Linkage to Turing Machines

- Remember $E_{DFA}$ = {<M> | M is a DFA and L(M) = ∅} is decidable.
- However, the corresponding language for a Turing Machine is not.
- Thus, Turing Machines are much more complex in their implementation than what we have seen so far.

# Try It

Using the modified TM concept of showing $A_{TM} \leq E_{TM}$ of:

- TM $M_w$: ”on input x:
  1. If x ≠ w, reject
  2. If x = w, run M on w:
     1. Accept if M accepts w.
     2. Reject if M rejects w.”

Use this approach above to show that L = {<M> | M is a TM and 111 ∈ L(M)} is undecidable.

# Try It

- Show that L = {<M> | M is a TM and 111 ∈ L(M)} is undecidable.

  Let $A_{TM}$ = {<M, w> | M accepts input w}, using a reduction $A_{TM} \leq L$, we can define the TM N as follows:

  - TM N: ”on input x ∈ $\{0, 1\}^*$:
    1. If x ≠ 111, reject x.
    2. Else, run M on w.
       - If M accepts w, then accept x.
       - Else, reject x.”