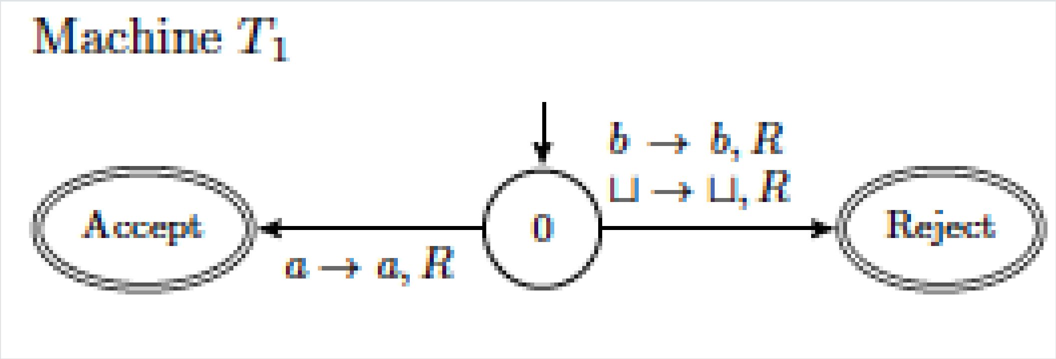
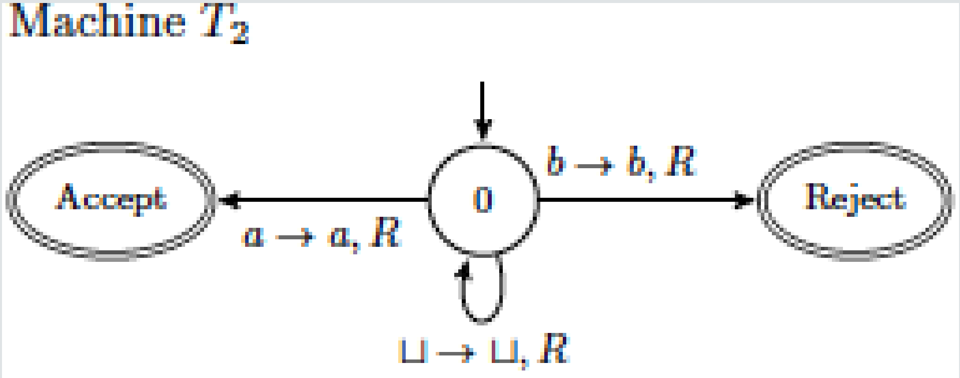
CHAPTER 4

Decidability

Equivalence of TMs





- T1 and T2 are equivalent because the accepts the same strings - recognize the same L
- However T1 and T2 might loop on different strings or reject different strings
  - T1 never loops
  - T2 loops on the blank symbol
- T1 is a special type of a TM – **decider** (program that terminates on each input)
- T2 is just a TM (a general program that might or might not terminate on strings not in L)

Decidability

- Computability Problem
  - Can a problem be solved by a computer?
- Decision Problem
  - Can we design a computer to recognize the language of the corresponding decision problem?
- Turing Computability/decidability Problem
  - Can we design a TM (or an algorithm) to recognize the language corresponding to a decision problem
  - Remember that TM can
    - Halt and accept: return **YES**
    - Halt and reject: return **NO**
    - Never halt, i.e., **LOOP**, and return on answer

Decidability

- Solving a problem
  - Requires constructing a TM that always returns an answer (i.e., never loops on input)
  - If it can be done:
    - The decision problem is **solvable** (or decidable)
    - TM that achieves this goal is called **decider**
    - The language of such TM is a decidable language (or **recursive**, or Turing decidable)

Languages of TM and Algorithm

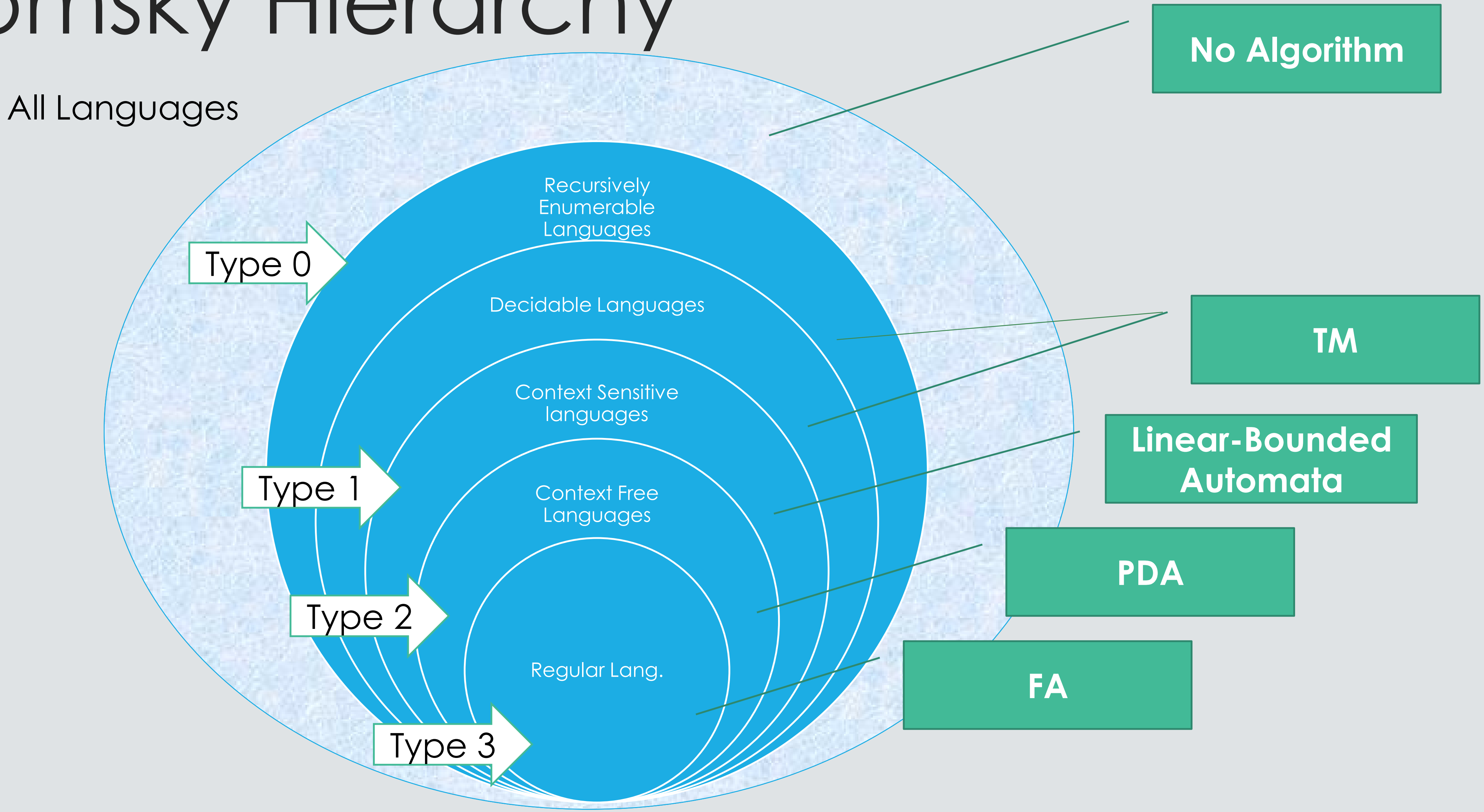
A language recognized by a TM is a **recursively enumerable** language (or TM-recognizable)

A language that is recognized by a TM that *halts* for all input strings is said to be **recursive**(or TM-decidable)

It is often accepted that any *algorithm* that can be carried out at all (by humans, a computer, or a computation model) can be carried out by a TM. (**Church –Turing Thesis**)

Definition: An **algorithm** is a procedure that can be executed on a TM. (*If a problem cannot be solved by an algorithm, i.e., no TM can be designed for it, then a real computer cannot solve it.*)

Chomsky Hierarchy



Decidable Problems of Regular Languages - DFA

- Acceptance problem:

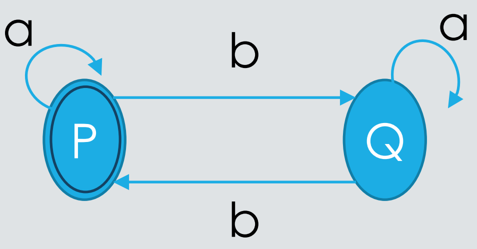
  Does a deterministic FA D accept a given string w?

  - This is a decidable problems since we can *always* answer **Yes** or **No**
  - But… how do we show (i.e., proof) that the problem is decidable
    - Step 1: Define the language corresponding to the decision problem using an encoding method
    - Step 2: Show that the language is decidable

Decidable Problems of Regular Languages - DFA

- Defining the problem in terms of an encoding
  - **A_{DFA}**= {<D,w> | D is a DFA that accepts input string w}
    - <D,w> is a string that encodes both D and w
    - Basically, the language A_{DFA} contains all the encodings of D and w, such that D accepts w
  - Encoding:
    - List all the elements in the formal description of a DFA: $(Q, \Sigma, \delta, q_0, F)$
    - Use "#" and "," as delimiters
  - Example:

DFA:



String: abb

< D,w >= #p, q#a, b#p, a, p#p, b, q#q, a, q#q, b, p#p#p##abb#

#Q#symbols#trans. p to p#trans. p to q#trans. q to q#trans. q to p#q_0#F##w#

Decidable Problems of Regular Languages - DFA

- Showing $\mathbf{A_{DFA}}$ is decidable by constructing a decider T for A_{DFA}

```
on input string < D,w >
T checks whether < D,w > is a valid encoding of a
DFA and a string w
if invalid then
      T rejects < D,w >
else
      T simulates D on string w
      if D accepts string w then
            T accepts < D,w >
      else
            T rejects < D,w >
      end if
end if
```

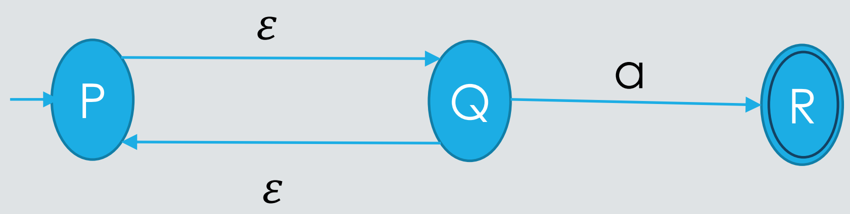
TM M= On input <D,w>, where D is an DFA and w is a string

1. Simulate D on input w
2. If the simulation ends in an accept state, *accept*. If it ends in a non-accepting state, *reject*

  - Since *D* is a DFA, it **halts on every input**, then *T* **halts on every input**. Therefore, T is a decider and A_{DFA} is decidable. Finally, we can conclude the DFA acceptance problem is solvable

Decidable Problems of Regular Languages - NFA

- Decision Problem: Does an NFA N accept a string w?
- The corresponding language is: $\mathbf{A_{NFA}}$= {<N,w>| N is a NFA that accepts input string w}
- $\mathbf{A_{NFA}}$ is decidable, but simulating NFA N on a TM is not possible, since it might not halt on every input
  - Example:



- To prove $\mathbf{A_{NFA}}$ is decidable, the decider needs to first convert the NFA to a DFA and then simulate the DFA on w

Decidable Problems of Regular Languages - NFA

- Proof

TM T = On input <N,w>, where N is an NFA and w is a string
1. Convert N to a equivalent DFA D, using Theorem 1.39
2. Run TM M from Theorem 4.1 on input <D, w>
3. It M accepts, *accept*, otherwise, *reject*

Running TM M in step 2 means incorporating M into the design of N as a subprocedure

Decidable Problems of Regular Languages – Regular Expression

- Decision Problem: Does a RegEx R describe a string w?
- The corresponding language is: $\mathbf{A}_{\mathbf{RegEx}}$= {<R,w> | R is a RegEx that describes string w}
- $\mathbf{A}_{\mathbf{RegEx}}$ is decidable, which we prove by constructing a TM that converts the RegEx into an equivalent DFA and the simulates the DFA on string w
- Proof

TM P = On input <R,w>, where R is a RegEx and w is a string
1. Convert T to a equivalent NFA N by using Theorem 1.54
2. Run TM T from Theorem 4.2 on input <N, w>
3. It T accepts, *accept*, otherwise, *reject*

Decidable Problems of Regular Languages – Emptiness

- Decision Problem: Does a DFA D reject all strings?
- The corresponding language is: $\mathbf{E_{DFA}}$= {<D>| D is a DFA and L(D)=∅}
- $\mathbf{E_{DFA}}$ is decidable, which we prove by constructing a TM S that verifies whether reaching an accept state from the start state: if true reject <D>, otherwise accept<D>
- Proof

TM S = On input <D>, where D is a DFA
1. Mark the start state of D
2. Repeat until no new states get marked
    1. Mark any state that has a transition coming into it from any state that is already marked
3. If no accept state is marked, *accept*, otherwise, *reject*

Decidable Problems of Regular Languages – Equivalence

- Decision Problem: Two DFA are equivalent, i.e., recognize the same language?
- The corresponding language is: $\mathbf{EQ_{DFA}}$= {<D_1, D_2> | D_1 and D_2 are DFAs and L(D_1)= L(D_2)}
- $\mathbf{EQ_{DFA}}$ is decidable, which we prove by constructing a TM V that first construct a new DFA D which recognizes the following language L(D)=$\left(L(D_1) \cap \overline{L(D_2)}\right) \cup \left(\overline{L(D_1)} \cap L(D_2)\right)$. Then V checks whether D rejects all strings, i.e., whether L(D)= ∅, then V accepts <D_1, D_2>, otherwise, reject.
- Proof

TM V = On input <D_1, D_2>, where D_1 and D_2 are DFAs
1. Construct DFA D as described
2. Run TM S from Theorem 4.4 on <D>
3. If S accepts, *accept*. If S rejects, *reject*

Decidable Problems of Context-Free Languages – CFG

- Decision Problem: Does a CFG G generate w?
- The corresponding language is: $\mathbf{A_{CFG}}$= {<G,w>| G is a CFG that generates string w}
- $\mathbf{A_{CFG}}$ is decidable
  - We could construct a TM to generate all the strings of L(G), but G might generate infinite strings, causing the TM not to halt, thus the TM is not a decider
  - A valid alternative involves considering the Chomsky Normal form of CFG, i.e., G' (p. 108-110 in the textbook, not covered in class)
    - In Chomsky normal form every rule is of the form $A \rightarrow BC$ or $A \rightarrow a$ and B, C cannot be starting variables
    - An important property of Chomsky normal form is that if a string w can be derived by a CFG G', then string w can be derived by G' in exactly 2n-1 steps, where n is the length of the string w, i.e., $n = |w|$
    - There is an algorithm (in your book) that converts G to G'
      - The downside of this conversion is the size of G', which can range from $|G|^2$, i.e., polynomial in the size of the original grammar, to $2^{2|G|}$, i.e., exponential size of the original grammar

Decidable Problems of Context-Free Languages – CFG

```
on input string < G,w >
T checks whether < G,w > is a valid encoding of a CFG and a string
if invalid then
    T rejects < G,w >
else
    T constructs Chomsky normal form grammar G’ such
    that L(G’) = L(G)
    T finds all string of length n = |w| in L(G’) by simulating G’
    for exactly 2n-1 steps, at each step trying all possible rules
    if T finds string w then
        T accepts < G,w >
    else
        T rejects < G,w >
    end if
end if
```

TM T= On input <G,w>, where G is an CFG and w is a string

1. Convert G to an equivalent grammar in Chomsky normal form
2. List all derivations with 2n-1 steps, where n is the length of w, except is n=0, then list all derivations with one step instead
3. If any of these derivations generate w, *accept*, otherwise, *reject*

Decidable Problems of Context-Free Languages – Emptiness

- Decision Problem: Does a CFG G generate no strings ?
- The corresponding language is: $\mathbf{E_{CFG}}$= {<G>| G is a CFG and L(G)=∅}
- $\mathbf{E_{CFG}}$ is decidable, which we verify by constructing a TM T which tests whether the start variable can generate a string of terminals
- Proof

TM T = On input <G>, where G is a CFG
1. Mark all terminal symbols in G
2. Repeat until no new variables get marked
   1. Mark any variable A where G has a rule $A \rightarrow U_1 U_2 U_3 \ldots U_k$ and each symbol $U_1, U_2, U_3, \ldots U_k$ has already been marked
3. If the start variable is not marked, *accept*, otherwise, *reject*

Decidable Problems of Context-Free Languages – Emptiness

```
on input string < G >
T checks whether < G > is a valid encoding of a CFG
if invalid then
        T rejects < G,w >
else
        T defines set X that contains all symbols of G that can finally derive some string with only terminals or ε
        Step 1: T initializes X with all terminals of G and ε
        Step 2: for all rules A →α  ∈ G do
                        if all symbols of α  ∈  X and A ∉ X then
                                add variable A to X
                        end if
                   end for
        Step 3: if Step 2 adds new variables to X
                        then go to step 2
                  else
                        stop
                   end if
        if start variable S ∈  X then
                T rejects < G >
        else
                T accepts < G >
        end if
end if
```

Decidable Problems of Context-Free Languages – PDA

- Decision Problem: Does a PDA P generate string w?
- The corresponding language is: $\mathbf{A_{PDA}}$= {<P,w>| P is a PDA and w is a string}
- $\mathbf{A_{PDA}}$ is decidable, which we prove by constructing a TM that converts P to an equivalent CFG, and then determines whether the CFG can generate w
  - We do not simulate directly P on a TM since it might loop, which would make the TM **not** a decider

Decidable Problems of Context-Free Languages – CFL

- Theorem 4.9 : Every context-free language is decidable
- Proof idea:
  - Take advantage of Theorem 4.7, which decides de acceptance of a context-free grammar
- Proof

Let G be a CFG for a CFL A.
Design a TM MG that decides A
TM MG= On input w:

1. Run TM S (from Theorem 4.7) on input <G,w>
2. If S accepts, *accept*; if S rejects, *reject*

What about TMS?

- Problem: Does a TM accepts a given string w?
  - Is this a decidable problem?
  - Can we create a decider to solve this problem?
  - Remember: saying we cannot create a decider is not enough; we need to prove it

Stay tuned.....

Undecidability

- What sorts of problems are unsolvable by a computer?
  - Automating the problem of software verification, where a program and the specifications for the program are the input of another computer program
- Problem: Does a TM accepts a given string w?
- The corresponding language is: $\mathbf{A_{TM}}$= {<M,w>| M is a TM and w is a string}
- $\mathbf{A_{TM}}$ is undecidable 

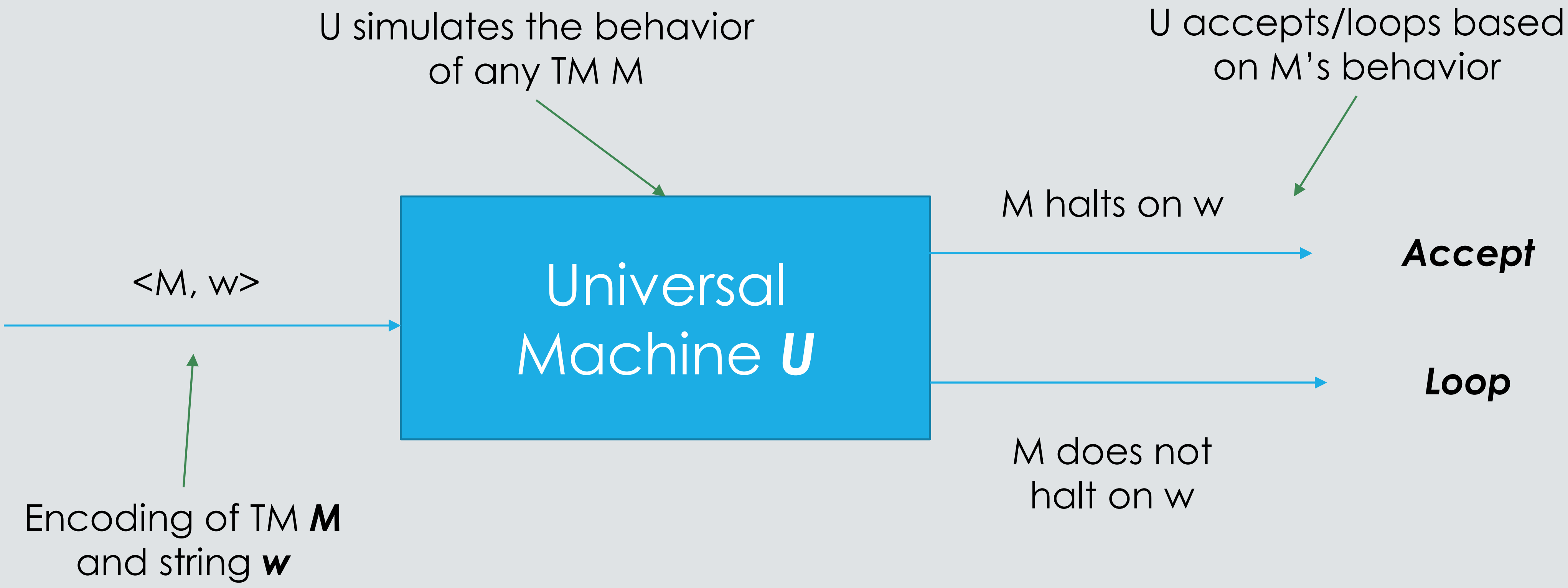
Undecidability

- Theorem 4.11: A_{TM} is undecidable
- Proof idea

> U = On input <M,w>, where M is a TM and w
> 1. Simulate M on input w
> 2. If M ever enters its accept state, *accept*, if M ever enters its reject state, *reject*

- Issues with this proof idea:
  - U *might loop* on input <M, w> if M loops on w, which shows why U does not decide A_{TM}
- Remember
  - A_{TM} is *Turing-recognizable,* which highlights the fact that recognizers are ***more powerful*** than deciders
  - U is also called a ***Universal Turing Machine***, since it is capable of simulating any other TM from the description of that machine
- Next stop: the proof of the halting problem

Universal Turing Machine



Decidable and Turing-Recognizable Languages

- Theorem: A language A is decidable iff itself and its complement are Turing-recognizable
  - Proof

Let M_1 be the recognizer for A
Let M_2 be the recognizer for $\bar{A}$
M = On input w

1. Run both M_1 and M_2 on input w in parallel
2. If M_1 accepts, *accept*, if M_2 accepts, *reject*

  - Remember:
    - Running in parallel means that M has two tapes, one simulating M_1 and another simulating M_2
  - Basically,
    - Every string w is either in A or $\bar{A}$. Therefore, either M_1 or M_2 will accept w.
    - Because M halts whenever M_1 or M_2 halt, M always halts, which makes it a decider.
    - Because M accepts all strings in A and rejects all strings not in A, M is a decider for A and A is decidable.

Turing-Unrecognizable Languages

- Some languages are not decidable or even Turing-recognizable
  - There are **uncountably** many languages yet only **countably** many Turing machines.
- Since each Turing Machine can recognize a single language and there are more languages than Turing Machines, *some languages are not recognized by any Turing Machine*. These languages are **Turing-Unrecognizable**.
  - We need only to show that the set of all Turing Machines is countable and the set of all languages is uncountable.

Turing-Unrecognizable Languages

- The set of all *Turing Machines* is **countable**:
  - Let S be the set of all strings over an alphabet Σ
    - We can prove that S is countable, since all the strings in S can be sorted in lexicographical order, and thus there is a correspondence with ℕ
  - Let M denote the set of encodings of TM
    - M is infinite and M is a subset of S, since S also contains strings that are encodings of invalid TMs

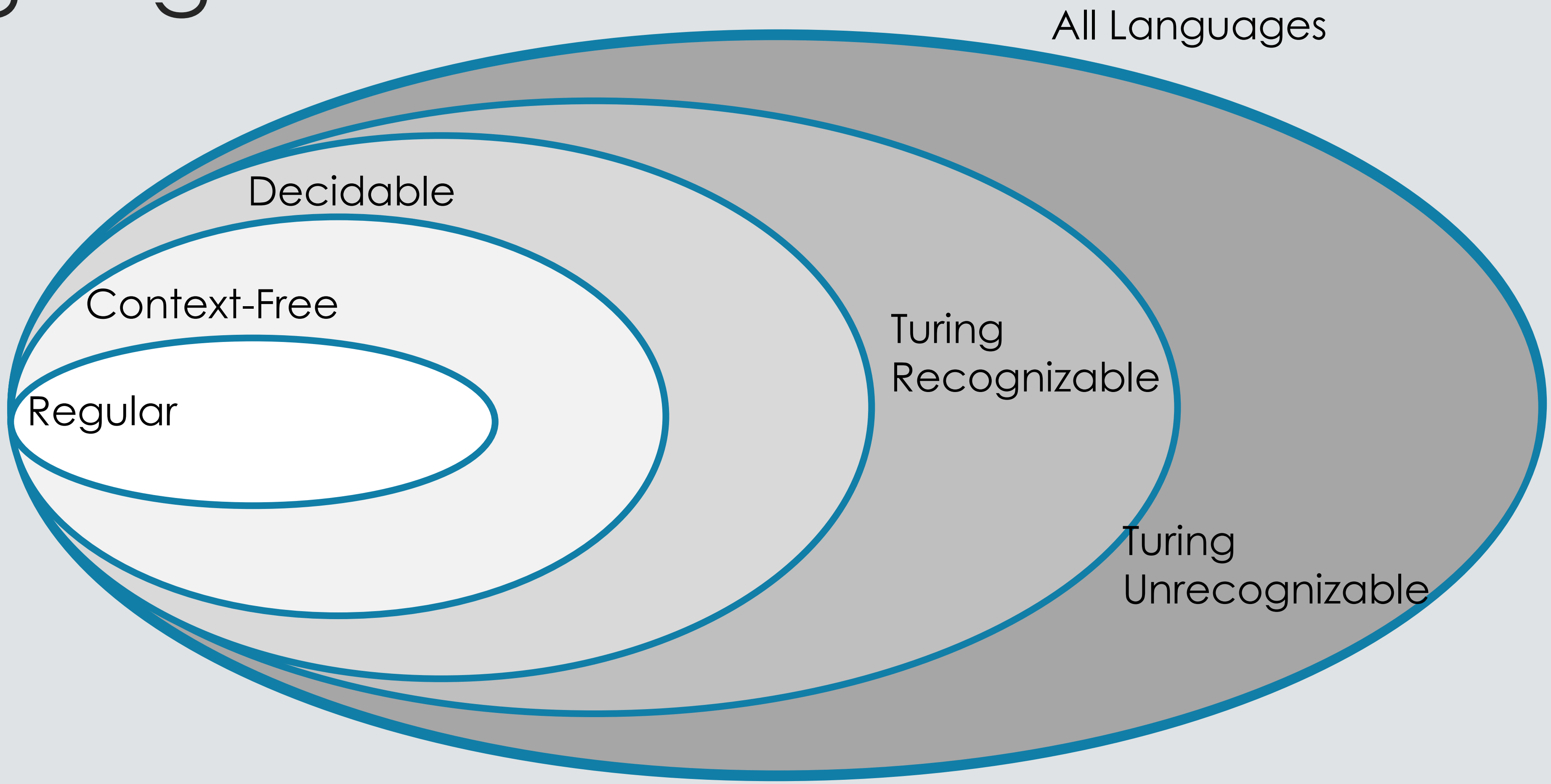
M is an infinite subset of a countable set, thus M is also countable

Turing-Unrecognizable Languages

- The set of all languages is **uncountable**
  - Every language in Σ can be described as an infinite binary sequence
  - Let the countable set of all finite strings will be {w1, w2, w3, … }
  - Let's order those those strings, e.g., lexicographical order (ε, w3, w1, w4, w7 …)
  - Then a language L ={ε, w1, w7, …} can be represented as (1,0, 1,0,1,…), i.e, an infinite binary sequence
  - If we assume that the number of all languages in countable, then we can always construct one more language that is different from the previous one, thus makes it **uncountable**

Since the number of TM is countable and the number of language is uncountable, therefore there are languages that no TM can recognize

Relationship Among Classes of Languages



Insolvability

- Why worry about identifying problems that cannot be solved?
  - If a problem is algorithmically unsolvable, we must think of
    - Simplified/altered versions of the problem , or
    - Approximation of a solution
- Why bother with solvable vs unsolvable problems?
  - To become aware of capabilities and limitations of computers
  - Exercise creativity in finding solutions to problems
  - Explore a different perspective on computation