PART 3:

Automata:

Turing Machines

Formal Languages & Computability Theory:

Church-Turing Thesis
Unsolvability/Undecidability of the Halting Problem
Decidable & Non-Decidable Languages
Semi-Decidable & Non-Semi-Decidable Languages

Grammar:

Unrestricted Grammars

The Church-Turing Thesis

Are We Done?

So far FSM $\Rightarrow$ PDA $\Rightarrow$ TM

Are there still problems we cannot solve?

- There is a countably infinite number of Turing machines since we can lexicographically enumerate all the strings that correspond to syntactically legal Turing machines.

- There is an uncountably infinite number of languages over any nonempty alphabet.

✓ So there are more languages than there are Turing machines!

Any New Computational Models?

- There are languages that cannot be recognized by any Turing Machine!
- Can we do better by creating **some new formal models** for the real-computers?

The Entscheidungsproblem (Decision Problem)

The Quest to Decide All Mathematical Questions!

- Does there exist an algorithm to decide, given an arbitrary sentence *w* in first order logic, whether *w* is valid?
- Given a set of axioms *A* and a sentence *w*, does there exist an algorithm to decide whether *w* is entailed by *A*?
- Given a set of axioms, *A*, and a sentence, *w*, does there exist an algorithm to decide whether *w* can be proved from *A*?

Definition of Algorithm

To answer the question, in any of these forms, requires formalizing the definition of an **algorithm**:

- **Turing**: **Turing machines.**
- **Church**: **Lambda calculus.**

✓ Turing proved that **Turing machines and the lambda calculus are equivalent in power**!

✓ Any problem that can be solved in one can be solved in the other!

The Church-Turing Thesis

"**All formalisms** powerful enough to describe everything we think of as a computational **algorithm** are **equivalent**."

- This isn't a formal statement, so we can't prove it.
- But many different computational models have been proposed and they all turn out to be equivalent!

Examples of equivalent formalisms

- **Modern computers** (with unbounded memory)
- **Turing Machines**
- **Lambda calculus**
- **Unrestricted grammars**

Examples of equivalent formalisms

- **Partial recursive functions**
- **Tag systems (Post machine** = FSM plus FIFO queue)
- **Post production systems (Post system)**
- Markov algorithms
- Conway's Game of Life
- One dimensional cellular automata
- DNA-based computing
- Lindenmayer systems

Lambda Calculus

In the pure lambda calculus, there is no built in data type number. *All expressions are functions.*

The successor function:

$(\lambda\, x.\, x + 1)$
$(\lambda\, x.\, x + 1)\ 3 = 4$
$(\lambda\, x.\, \lambda\, y.\, x + y)\ 3\ 4$

This expression is evaluated by binding 3 to *x* to create the new function $(\lambda\, y.\, 3 + y)$, which is applied to 4 to return 7.

Unrestricted Grammars

An unrestricted or type 0 or phrase structure grammar G is a quadruple (V, Σ, R, S) where:

- V is an alphabet,
- Σ (the set of terminals) is a subset of V,
- R (the set of rules) is a finite subset of $(V^+ \times V^*)$,
- S (the start symbol) is an element of $V - \Sigma$.

The language generated by G is: $\{w \in \Sigma^* : S \Rightarrow_G{}^* w\}$.

Tag Systems

A Tag system (or a Post machine) is an FSM augmented with a FIFO queue.

Tag systems are equivalent in power to Turing machines because the TM's tape can be simulated with the FIFO queue.

Post Production Systems

A Post (production) system P is a quintuple (V, Σ, X, R, S):

- V is the rule alphabet,
- Σ is a subset of V,
- X is a set of variables whose values are drawn from V^*,
- R (the set of rules) is a finite subset of:
  $(V \cup X)^* \times (V \cup X)^*$
  Every variable on the RHS must also be on the LHS.
  $A \rightarrow B$ becomes $XAY \rightarrow XBY$
- S can be any element of $V - \Sigma$.

Reading Assignment

Chapter 18:

Sections
18.1
18.2

In-Class Exercises

Chapter 18:

1 – a & b

Unsolvability (Undecidability) of the Halting Problem

Computability Theory

- **Computability**?
  - What are the fundamental capabilities and limitations of computers?
  - Classify problems as **solvable** and **unsolvable**.
  - **Unsolvability/Undecidability Theory**

Formal Models of Computation

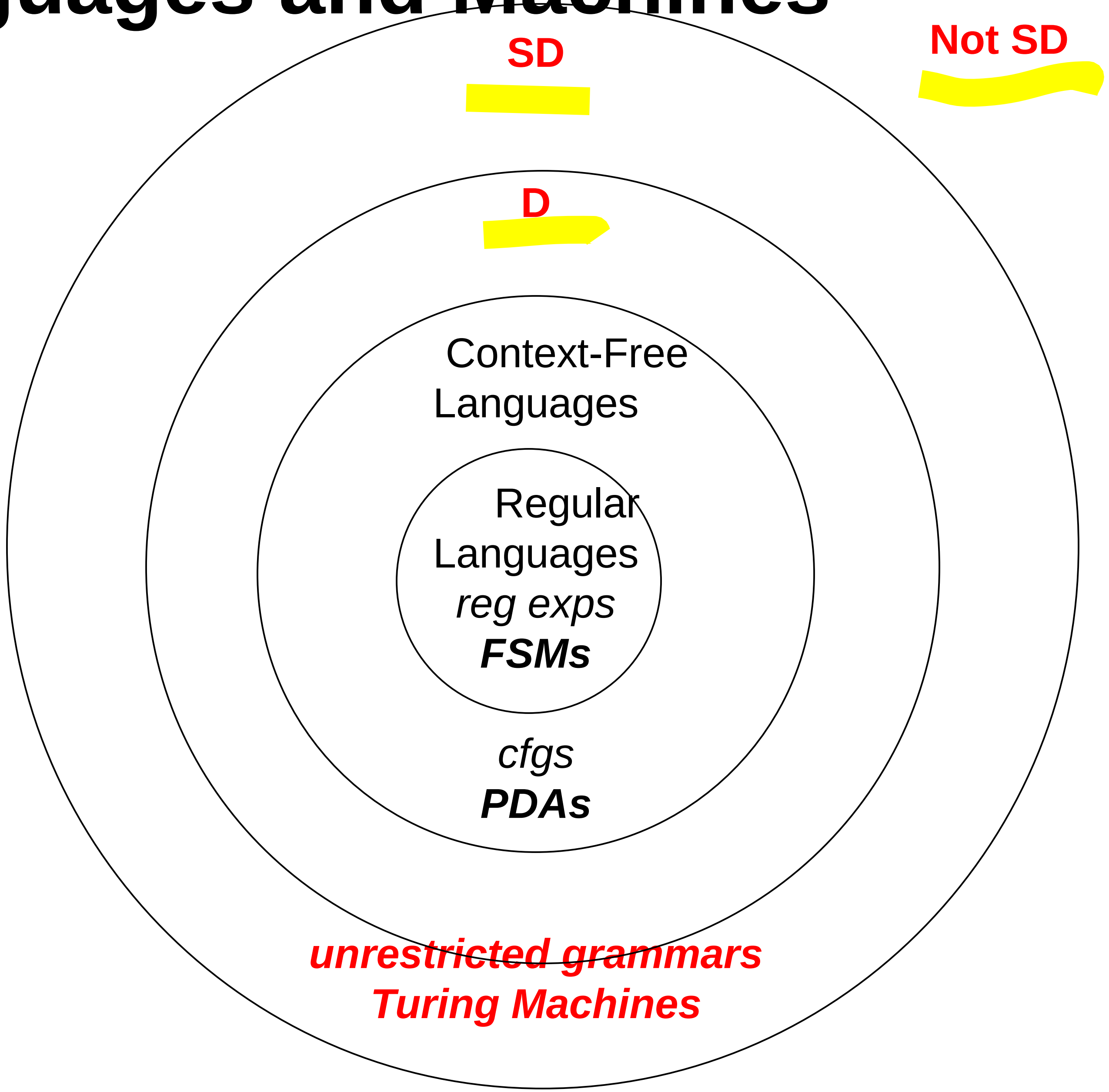
- Both deal with formal models of computation:
  - **Turing machines**
  - Lambda calculus

Computability Hierarchy

- Decidable Languages D
  - Solvable Languages
  - Computable Languages
  - Recursive Languages
  - Turing-Decidable Languages
- $\neg$ D Turing Undecidable Languages

- Semi-Decidable Languages SD
  - Recursively Enumerable (R.E.) Languages
  - Partially Decidable Languages
  - Turing Recognizable Languages
- $\neg$ SD Turing Unrecognizable Languages

Languages and Machines



Deciding a Language

TM M ***decides*** a language $L \subseteq \Sigma^*$ iff for any string $w \in \Sigma^*$:

if $w \in L$ then TM *M* accepts *w*, and

if $w \notin L$ then TM *M* rejects *w*.

TM M will always halt on all inputs!

Decidable Languages D

A language *L* is ***decidable*** *or* ***Turing-decidable*** *or* ***recursive*** iff there is a Turing Machine *M* that decides it.

We say that *L* is in ***D*** ***(****or* ***R)*** the set of all decidable languages.

Decidable Languages D

- Decidable Languages
- Solvable Languages
- Computable Languages
- Recursive Languages
- Turing-Decidable Languages

Decidable Languages/ Problems/ Functions

- A language is decidable!
- A problem is solvable!
- A function is computable!

Semideciding a Language

TM M ***semidecides (*or *recognizes)*** $L \subseteq \Sigma_M^*$ iff for any string $w \in \Sigma_M^*$:

if $w \in L \rightarrow$ TM M accepts w

if $w \notin L \rightarrow$ TM M does not accept w.

TM M may either reject or fail to halt (loop)!

Semi-Decidable Languages SD

A language *L* is ***semidecidable*** *or* ***Turing-recognizable*** *or* ***recursively-enumerable*** iff there is a Turing Machine that semidecides it.

We say that ***SD (or RE) -*** the set of all semidecidable languages.

Semi-Decidable Languages SD

- Semi-Decidable Languages
- Recursively Enumerable (R.E.) Languages
- Partially-Decidable Languages
- Turing-Recognizable Languages

Unsolvability, Undecidability & Uncomputability

- Problems that Computers Cannot Solve
- Languages that Computers Cannot Decide
- Functions that Computers Cannot Compute

Languages/ Problems/ Functions

- **Languages: decidable? vs undecidable?**
- **Problems: solvable? vs unsolvable?**
- **Functions: computable vs uncomputable?**

There Exist Languages that Are Not Decidable

Theorem There are languages that are not in D.

Proof Idea: Assume any nonempty alphabet Σ.

Lemma: There is a countably infinite number of D languages over Σ.
Lemma: There is an uncountably infinite number of languages over Σ.

So there are more languages than there are languages in D.
Thus there must exist at least one language that is in ¬D.

The Halting Problem

The Halting Problem Language:

H = {<*M*, *w*> : TM *M* halts on input string *w*}

The language H is **semidecidable (**or **Turing-recognizable)**, but is **not decidable**.

Semidecidability of The Halting Problem

Theorem 19.1 The language **H** = {<M,w> : TM M halts on input string w} is **semidecidable (**or **Turing-recognizable)**.

Proof Idea:

Proof by Construction

The TM M_H :

M_H(<M, w>) =

1. Run M on w.
2. Accept

M_H acccepts iff M halts on w. Thus, M_H semidecides H.

Undecidability of the Halting Problem

Theorem 19.2 The language **H** = {<*M*, *w*> : TM *M* halts on input string *w*} is **not decidable**.

Proof Idea:

Proof by Contraction

If H were decidable, then some TM M_H would decide it. The TM M_H would implement the specification:

halts(<*M*: string, *w*: string>) =
If <*M*> is a Turing machine description
and *M* halts on input *w*
then accept.
else reject.

Undecidability of the Halting Problem

Consider the TM *Trouble:*

Trouble(x: string) = if *halts*(x, x) then loop forever
else halt.

If there exists an M_H that computes the function *halts*, the TM *Trouble* exists.

Undecidability of the Halting Problem

Consider *Trouble*(<*Trouble*>)?

- Invoke M_H (<*Trouble*, *Trouble*>) , i.e., *halts*(<*Trouble*, *Trouble*>)

- If *halts* reports that *Trouble*(<*Trouble*>) halts, *Trouble* loops.
- But if *halts* reports that *Trouble*(<*Trouble*>) does not halt, then *Trouble* halts.

Contradiction!
Thus, there exists no TM M_H.

So, H is not decidable!

Enumerating Turing Machines

There exists an infinite lexicographic enumeration of:

- All syntactically valid TMs.
- All syntactically valid TMs with specific input alphabet Σ.
- All syntactically valid TMs with specific input alphabet Σ and specific tape alphabet Γ.

Viewing the Halting Problem as Diagonalization

- Lexicographically enumerate all Turing machines.
- Lexicographically enumerate all possible input strings.
- Let 1 mean TM halting on the input, blank mean non halting.

	i_1	i_2	i_3	…	<Trouble>	…
$machine_1$	1					
$machine_2$		1				
$machine_3$					1	
…				1		
Trouble			1			1
…	1	1	1			
…				1		

H is the Key to the Difference Between D and SD

Theorem 19.3 If H were in D then every SD language would be in D.

Proof Idea:

Proof by Construction

Let L be any SD language. There exists a TM M_L that semidecides it.

If H were also in D, then there would exist an O that decides it.

If H were in D then Every SD is in D

To decide whether w is in $L(M_L)$:

TM $M'(w$: string) =

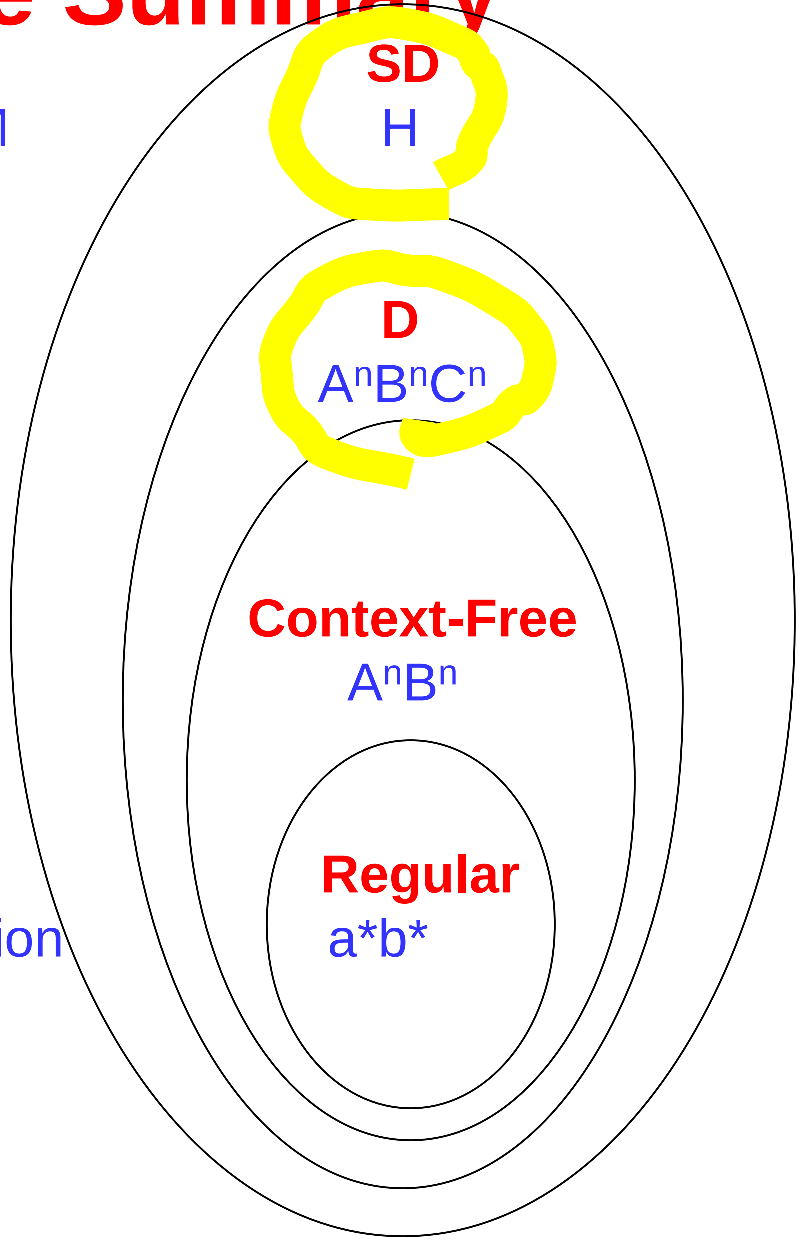
1. Run O on <M_L, w>.
2. If O accepts (i.e., M_L will halt), then:
    2.1. Run M_L on w.
    2.2. If it accepts, accept. Else reject.
3. Else reject.

So, if H were in D, all SD languages would be decidable!

The Entscheidungsproblem is Unsolvable

Theorem The Entscheidungsproblem is unsolvable.

Language Summary



IN	OUT
Semideciding TM	Reduction
Deciding TM	Diagonalize Reduction
CF grammar PDA	Pumping Closure
Regular Expression FSM	Pumping Closure

Reading Assignment

Chapter 19:

Sections
19.1
19.2
19.3

In-Class Exercises

Chapter 19:

1
2