Unit 5

Unrestricted Grammars

- $G = (\Sigma, N, S, P)$, where
  - Σ is the set of terminal symbols,
  - N is the set of non-terminal symbols ($N \cap \Sigma = \emptyset$),
  - $S \in N$ is the start symbol, and
  - P is a finite set of rules or productions.
- Each production is of the form

$$\alpha \rightarrow \beta$$

for any $\alpha, \beta \in (N \cup \Sigma)^*$ with α containing at least one non-terminal symbol.

- Such a production can also be written as

$$\gamma A \delta \rightarrow \beta$$

for any $\beta, \gamma, \delta \in (N \cup \Sigma)^*$, and for any $A \in N$.

- $L(G) = \{w \in \Sigma^* \mid S \xrightarrow[G]{}^* w\}$.

Example 1

- $L_1 = \{a^{2^n} \mid n$ “ $0\}$.
- Productions:

  $S \to TaU$

  $U \to \varepsilon \mid AU \quad a\,A \to$

  Aaa

  $T\,A \to T$

  $T \to \varepsilon$

- Derivation of a^8 using these productions:

  $S \to TaU \to TaAU \to TaAAU \to TaAAAU \to$

  $TaAAA$

  $\to TAaaAA \to TaaAA$

  $\to TaAaaA \to TAaaaaA \to TaaaaA$

  $\to TaaaAaa \to TaaAaaaa \to TaAaaaaaa \to$

  $TAaaaaaaaa \to Taaaaaaaa$

  $\to aaaaaaaa$

Example 2

- $L_2 = \{a^n b^n c^n \mid n$ “ $0\}$.
- Productions:

$$S \rightarrow UT$$

$$U \rightarrow \varepsilon \mid aUbC$$

$$Cb \rightarrow bC$$

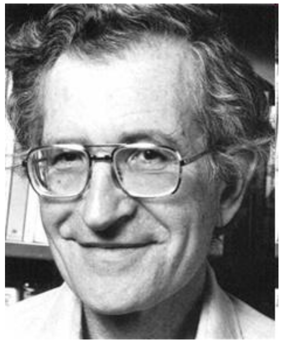
$$CT \rightarrow Tc$$

$$T \rightarrow \varepsilon$$

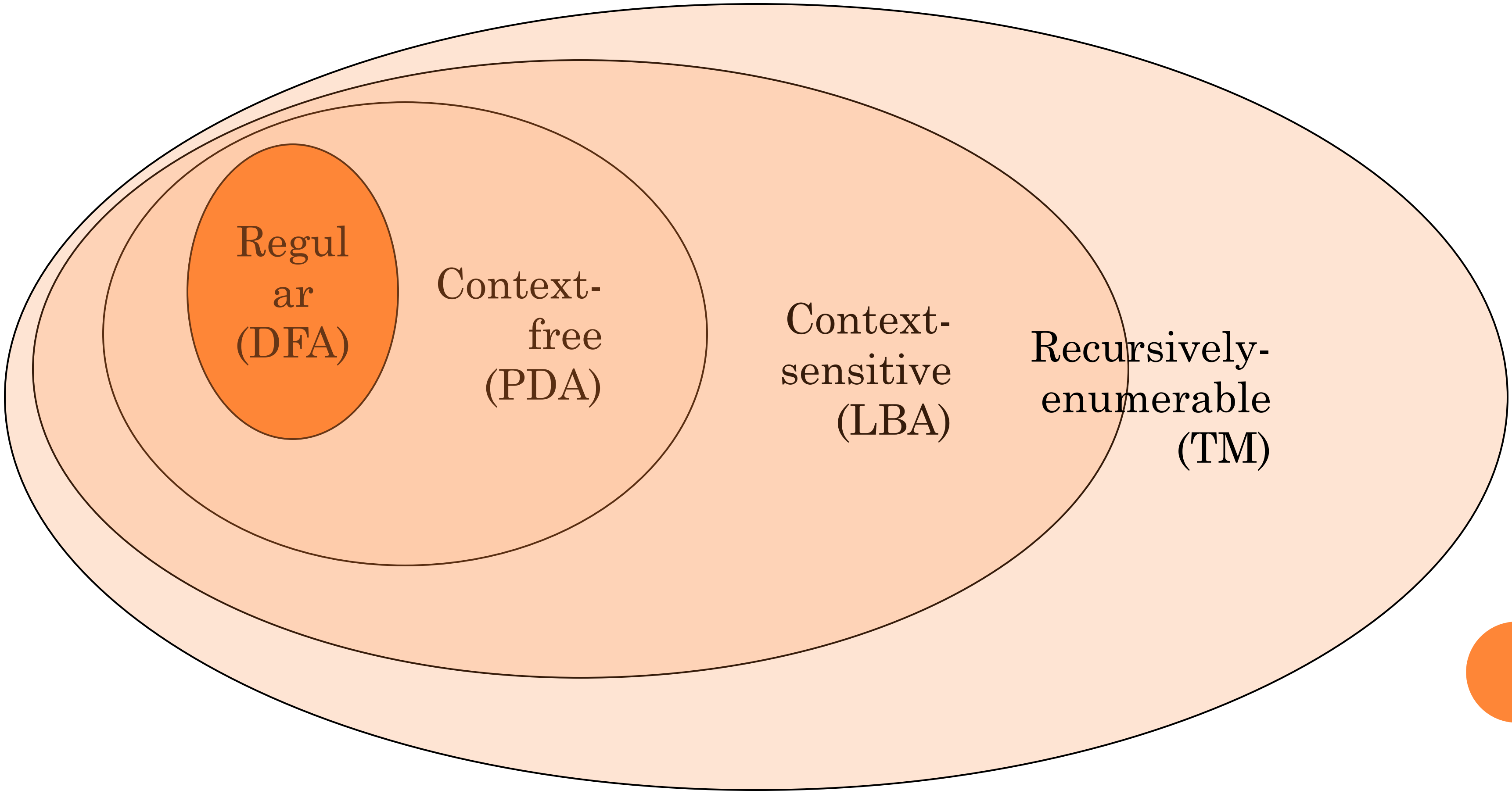
- Derivation of $a^3b^3c^3$ using these productions:

$$S \rightarrow UT \rightarrow aUbCT \rightarrow aaUbCbCT \rightarrow$$
$$aaaUbCbCbCT \rightarrow aaabCbCbCT$$
$$\rightarrow aaabCbbCCT \rightarrow aaabbCbCCT \rightarrow$$
$$aaabbbCCCT$$
$$\rightarrow aaabbbCCTc \rightarrow aaabbbCTcc \rightarrow$$

The Chomsky Hierachy



- A containment hierarchy of classes of formal languages



- Comprises four types of languages and their associated grammars and machines.
- Type 3: Regular Languages
- Type 2: Context-Free Languages
- Type 1: Context-Sensitive Languages
- Type 0: Recursively Enumerable Languages
- These languages form a strict hierarchy

Language	**Grammar**	**Machine**	**Example**
Regular Language	Regular Grammar ▪ Right-linear grammar ▪ Left-linear grammar	Deterministic or Nondeterministic Finite-state acceptor	a*
Context-free Language	Context-free grammar	Nondeterministic Pushdown automaton	a^nb^n
Context-sensitiv e	Context-sensitive grammar	Linear-bounded automaton	$a^nb^nc^n$
Recursively enumerable	Unrestricted grammar	Turing machine	Any computable function

Turing-Machine Definition

- A TM is described by:

1. A finite set of *states* (Q, typically).
2. An *input alphabet* (Σ, typically).
3. A *tape alphabet* (Γ, typically; contains Σ), which includes a blank symbol, B, not in Σ.
   - Entire tape except for the input is initially blank.
4. A *transition function* (δ, typically).
5. A *start state* (q_0, in Q, typically).
6. An *final (accept) state* (f or q_{accept}, typically).
7. A *reject state* (r or q_{reject}, typically).

The Transition Function

- Takes two arguments:
  1. A state, in Q.
  2. A tape symbol in Γ.
- δ(q, Z) is either undefined or a triple of the form (p, Y, D).
  - p is a state.
  - Y is the new tape symbol.
  - D is a *direction*, L or R.

Actions of the TM

- If δ(q, Z) = (p, Y, D) then, in state q, scanning Z under its tape head, the TM:

1. Changes the state to p.
2. Replaces Z by Y on the tape.
3. Moves the head one square in direction D.
   - D = L: move left; D = R; move right.

Conventions

- a, b, … are input symbols.
- …, X, Y, Z are tape symbols.
- …, w, x, y, z are strings of input symbols.
- α, β,… are strings of tape symbols.

Language of a Turing Machine

- Once a TM has entered either the accept state or reject state, it **halts**.
- Initially, the input for a TM, M, is on its tape, its head is pointing to the first character of the input (or B if it is null), and M is in its start state
- An input string, w, is in the **language** of M if the actions of M with w as its input results in it halting in the accept state.

Example: Turing Machine

- This TM scans its input right, turning each 0 into a 1.
- If it ever finds a 1, it goes to final reject state r, goes right on square, and halts.
- If it reaches a blank, it changes moves left and accepts.
- Its language is 0*

Example: Turing Machine – (2)

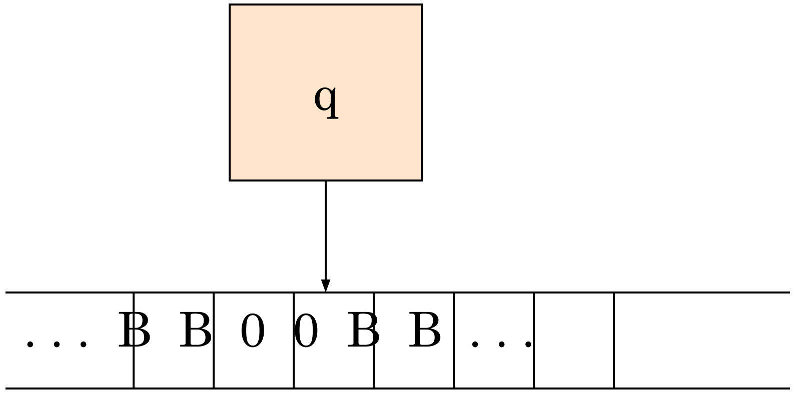
- States = {q (start), f (accept), r (reject)}.
- Input symbols = {0, 1}.
- Tape symbols = {0, 1, B}.
- $\delta(q, 0) = (q, 1, R)$.
- $\delta(q, 1) = (r, 1, R)$.
- $\delta(q, B) = (f, B, L)$.

Simulation of TM

δ(q, 0) = (q, 1, R)

δ(q, 1) = (r, 1, R)

δ(q, B) = (f, B, L)

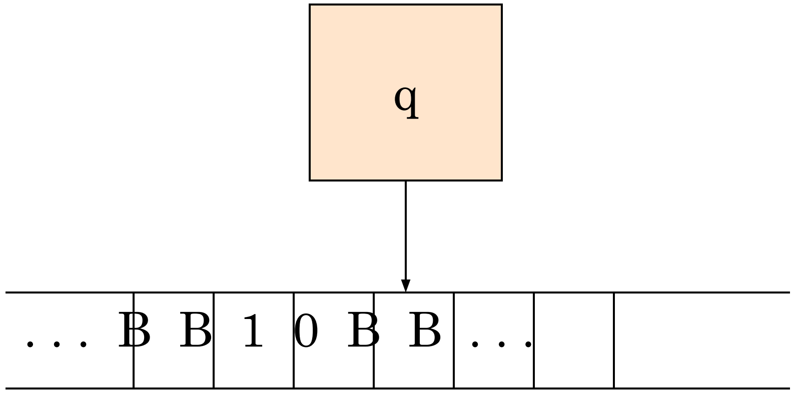


Simulation of TM

δ(q, 0) = (q, 1, R)

δ(q, 1) = (r, 1, R)

δ(q, B) = (f, B, L)

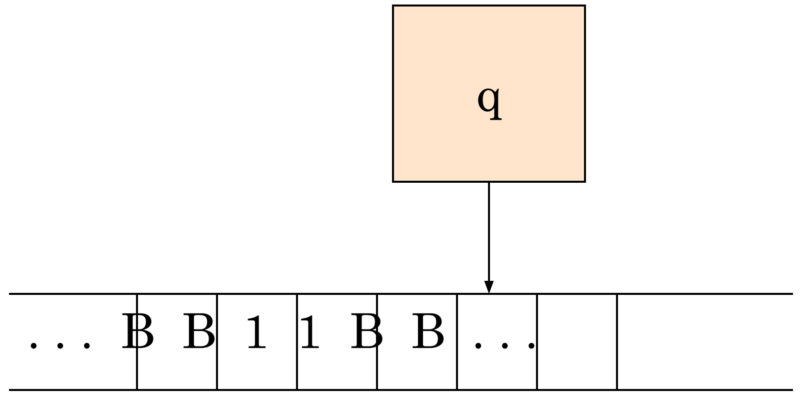


Simulation of TM

$\delta(q, 0) = (q, 1, R)$

$\delta(q, 1) = (r, 1, R)$

$\delta(q, B) = (f, B, L)$

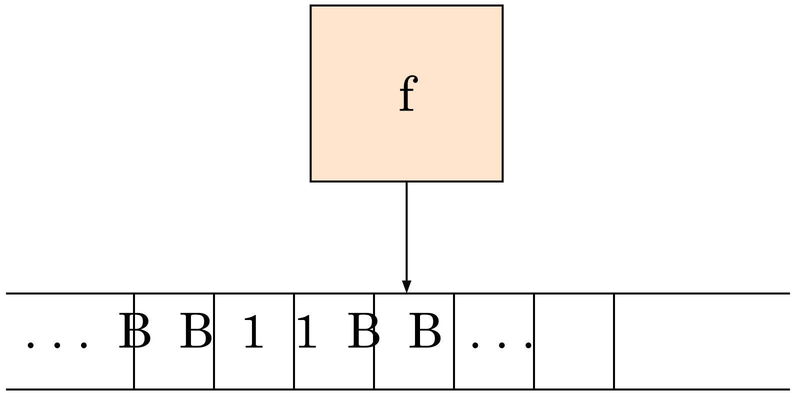


Simulation of TM

δ(q, 0) = (q, 1, R)

δ(q, 1) = (r, 1, R)

δ(q, B) = (f, B, L)



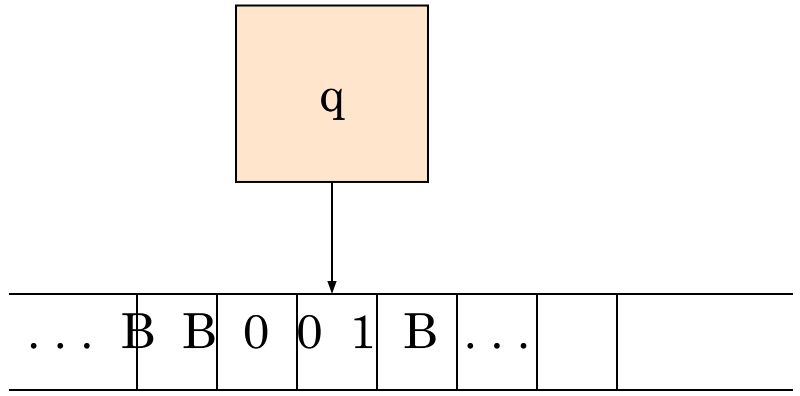
The TM halts and accepts. (So "00" is in its language.)

Simulation of TM – on 001

$\delta(q, 0) = (q, 1, R)$

$\delta(q, 1) = (r, 1, R)$

$\delta(q, B) = (f, B, L)$

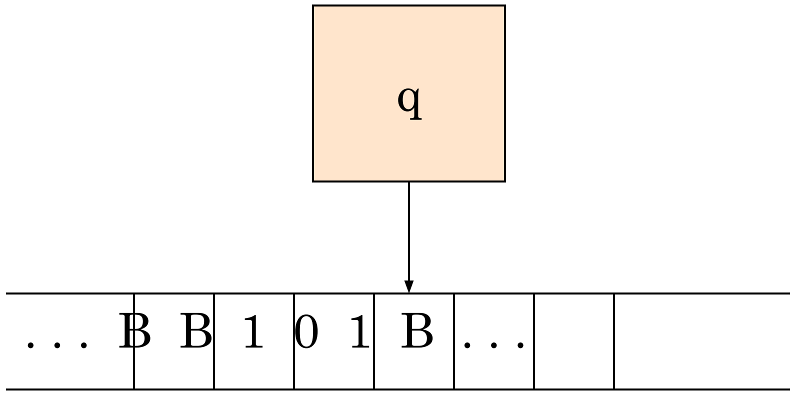


Simulation of TM – on 001

δ(q, 0) = (q, 1, R)

δ(q, 1) = (r, 1, R)

δ(q, B) = (f, B, L)

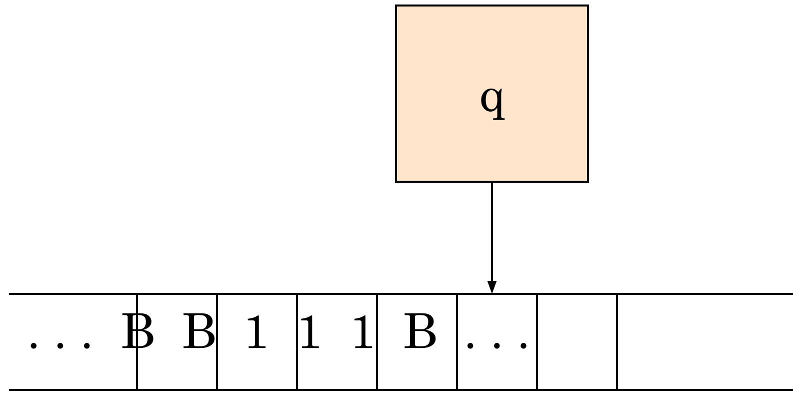


Simulation of TM – on 001

δ(q, 0) = (q, 1, R)

δ(q, 1) = (r, 1, R)

δ(q, B) = (f, B, L)

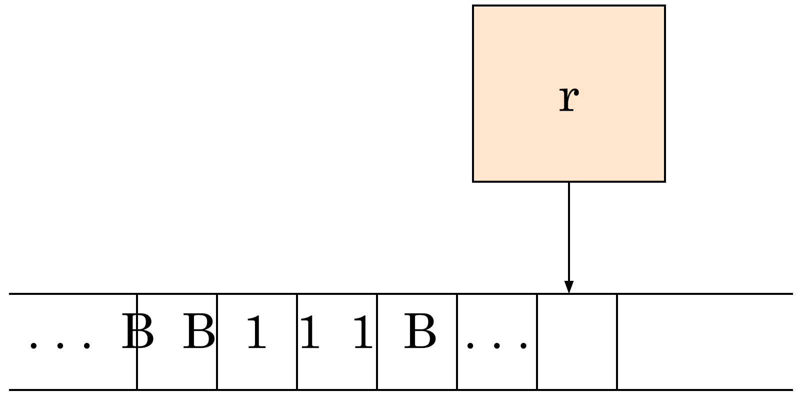


Simulation of TM

δ(q, 0) = (q, 1, R)

δ(q, 1) = (r, 1, R)

δ(q, B) = (f, B, L)



The TM halts and rejects. (So “001” is not in its language.)

Instantaneous Descriptions of a Turing Machine

- Initially, a TM has a tape consisting of a string of input symbols surrounded by an infinity of blanks in both directions.
- The TM is in the start state, and the head is at the leftmost input symbol.

TM ID's – (2)

- An ID is a string αqβ, where αβ is the tape between the leftmost and rightmost nonblanks (inclusive).
- The state q is immediately to the left of the tape symbol scanned.
- If q is at the right end, it is scanning B.
  - If q is scanning a B at the left end, then consecutive B's at and to the right of q are part of α.

TM ID's – (3)

- As for PDA's we may use symbols $\vdash$ and $\vdash^*$ to represent "becomes in one move" and "becomes in zero or more moves," respectively, on ID's.
- Example: The moves of the previous TM are $q00 \vdash 0q0 \vdash 00q \vdash 0q01 \vdash 00q1 \vdash 000f$

Formal Definition of Moves

1. If $\delta(q, Z) = (p, Y, R)$, then
   - $\alpha q Z \beta \vdash \alpha Y p \beta$
   - If Z is the blank B, then also $\alpha q \vdash \alpha Y p$
2. If $\delta(q, Z) = (p, Y, L)$, then
   - For any X, $\alpha X q Z \beta \vdash \alpha p X Y \beta$
   - In addition, $q Z \beta \vdash p B Y \beta$

Formal Definition of the Language of a TM

- Recall that once a TM has entered either the accept state or reject state, it halts.
- If M is a Turing Machine, the **language accepted** by M is:

  $L(M) = \{w \mid q_0 w \vdash^* I$, where I is an ID with the accept state$\}$.

Turing-Recognizable Languages

- A language accepted by a TM.
- But the TM might loop for strings not in its language.
- This class of languages is also called the *recursively enumerable languages*.
  - Why? The term actually predates the Turing machine and refers to another notion of computation of functions.

Turing-Decidable Language

- A languages accepted by a TM that always halts.
- An *algorithm* is a TM that is guaranteed to halt whether or not it accepts.
- If L = L(M) for some TM M that is an algorithm, we also say L is a *recursive language*.
  - Why? It's a term with a history…

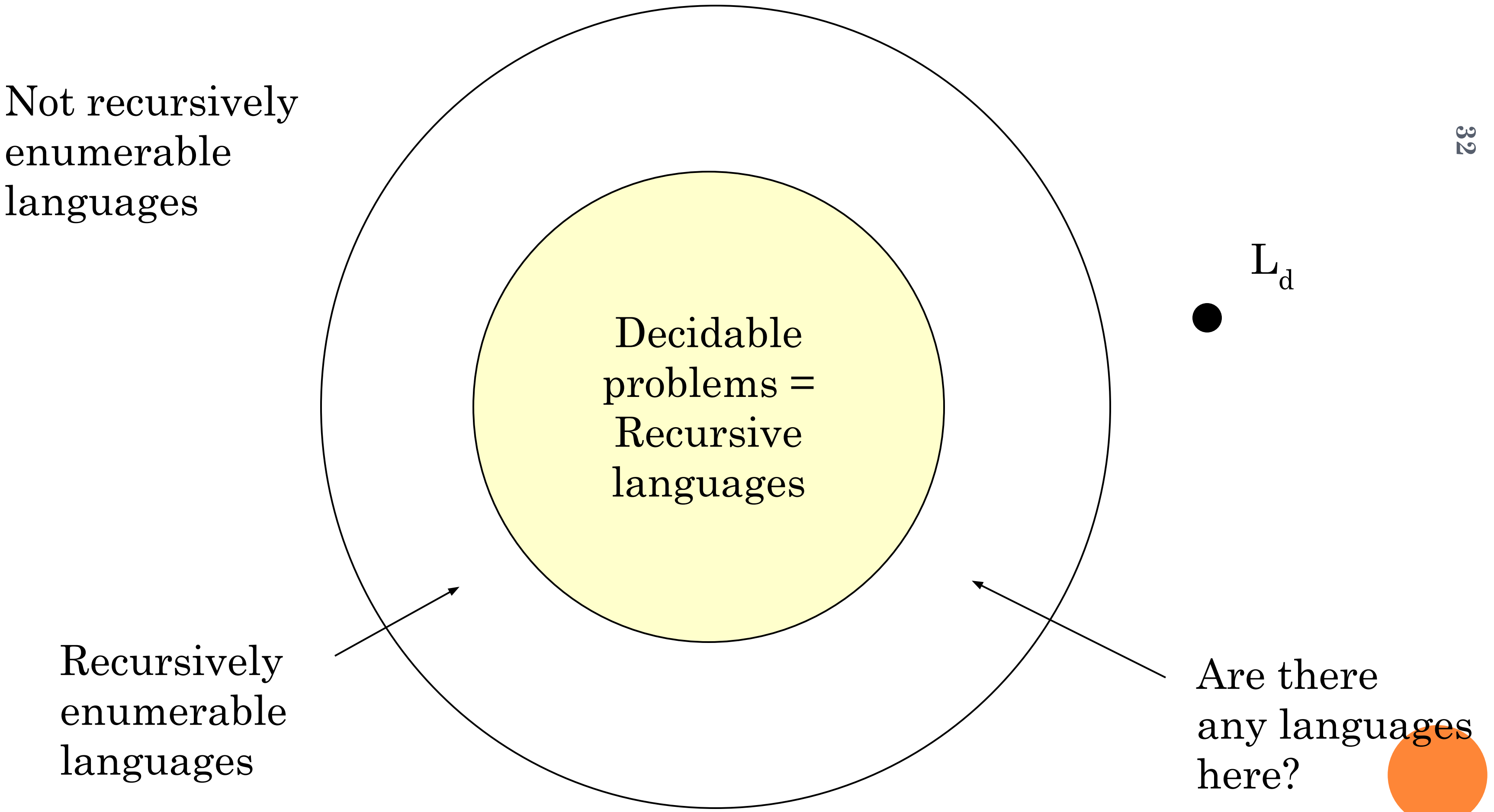
Example: Turing-decidable Languages

- Every CFL is a Turing-decidable language.
  - Use the CYK algorithm.
- Every regular language is a Turing-decidable language.
  - Simulate its DFA.
- Almost anything you can think of is Turing-decidable.

Decidable Problems

- A problem is *decidable* if there is an algorithm to answer it.
  - Recall: An "algorithm," formally, is a TM that halts on all inputs, accepted or not.
  - Put another way, "decidable problem" = "recursive language."
- Otherwise, the problem is *undecidable*.

Bullseye Picture


Not recursively enumerable languages
L_d
Decidable problems = Recursive languages
Recursively enumerable languages
Are there any languages here?

From the Abstract to the Real

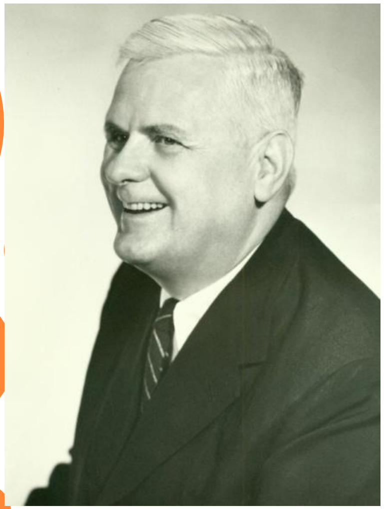
- While the fact that L_d is undecidable is interesting intellectually, it doesn't impact the real world directly.
- We first shall develop some TM-related problems that are undecidable, but our goal is to use the theory to show some real problems are undecidable.

Examples: Undecidable Problems

- Can a particular line of code in a program ever be executed?
- Is a given context-free grammar ambiguous?
- Do two given CFG's generate the same language?

The Church-Turing Thesis and Turing-completeness

Michael T. Goodrich
Univ. of California, Irvine

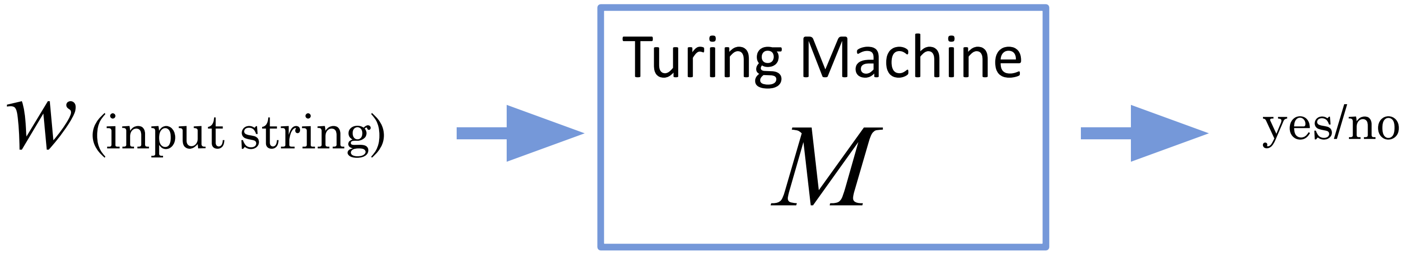

Alonzo Church (1903-1995)

Alan Turing (1912-1954)

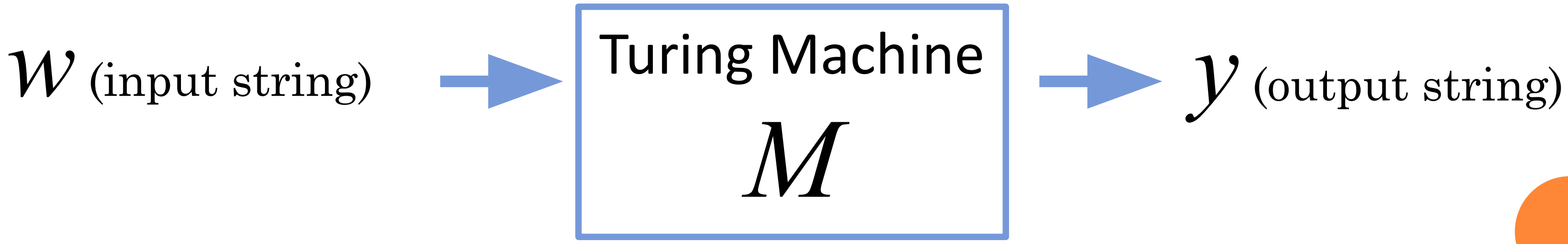
Some slides adapted from David Evans, Univ. of Virginia, and Antony Galton, Univ. of Exeter

Types of Turing Machines

- **Decider/acceptor**:



- **Transducer** (more general, computes a function):



Church

- Alonzo Church, 1936, *An unsolvable problem of elementary number theory*.
- Introduced **recursive functions** and **λ-definable functions** and proved these classes equivalent.

"We … define the notion … of an *effectively calculable* function of positive integers by identifying it with the notion of a recursive function of positive integers."

Turing

- Alan Turing, 1936, *On computable numbers, with an application to the Entscheidungs-problem.*
- Introduced the idea of a **Turing machine computable number**

"The [Turing machine] computable numbers include all numbers which could naturally be regarded as computable."

The Church-Turing Thesis

"Every effectively calculable function can be computed by a Turing-machine transducer."

"Since a precise mathematical definition of the term effectively calculable (effectively decidable) has been wanting, we can take this **thesis** ... as a definition of it..." – Kleene, 1943.

That is, for every definition of "effectively computable" functions that people have come up with so far, a Turing machine can compute all such such functions.

Equivalent Statements of the Church-Turing Thesis

- "Intuitive notion of algorithms equals Turing machine algorithms." Sipser, p. 182.
- Any mechanical computation can be performed by a Turing Machine
- There is a TM-*n* corresponding to every computable problem
- We can model any mechanical computer with a TM
- The set of languages that can be decided by a TM is identical to the set of languages that can be decided by any mechanical computing machine
- If there is no TM that decides problem P, there is no algorithm that solves problem P.

All of these statements are equivalent to the Church-Turing thesis

Examples of the Church-Turing Thesis

- With respect to computational power (i.e., what can be computed):
  - Making the tape infinite in both directions adds no power
  - [Soon] Adding more tapes adds no power
  - [Church] Lambda Calculus is equivalent to TM
  - [Chomsky] Unrestricted replacement grammars are equivalent to TM
  - Random-Access Machine (RAM) model is equivalent to a TM

“Some of these models are very much like Turing machines, but others are quite different.”

End