# Lecture 11

# Limits of Algorithmic Computation

COSE215: Theory of Computation

Seunghoon Woo

Fall 2023

# Contents

- **Decidability**
- **Chomsky Hierarchy**

# Limits of Algorithmic Computation

- **Turing machines can do anything that computers can do**
- **Some problems cannot be solved by Turing machines**
  - A problem that cannot be done by a Turing machine

    = A problem that is not in the power of even the most powerful computer

# Decidability

- **Let $\boldsymbol{M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)}$ be a Turing machine**
  - The language accepted by $M$ is the set

$$L(M) = \{w \in \Sigma^* : q_0 w \vdash^* x_1 q_f x_2 \nvdash,\ \ q_f \in F, x_1, x_2 \in \Gamma^*\}$$

# Decidability

- A function $f$ (domain $D$) is said to be **Turing-computable** if there exists some Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ such that

$$q_0 w \vdash^* q_f f(w) \nvdash, \text{ where } q_f \in F \text{ and all } w \in D$$

# Decidability

- **A language $L$ is decidable (recursive) if there is a TM $M$ such that**
  1. $L(M) = L$
  2. $M$ halts on all inputs

# Decidability

- **Revisit: A language $L$ is recursively enumerable if there exists a TM $M$ such that $L = L(M)$**
  - If $w \in L$, then $M$ halts on $w$ and accepts $w$
  - If $w \notin L$, then the following two cases are possible
    - ❖ $M$ halts on $w$ and rejects $w$
    - ❖ $M$ does not halt on $w$ (e.g., infinite loop)

# Decidability

- **A language $L$ is decidable (recursive) if there is a TM $M$ such that $L = L(M)$**
  - If $w \in L$, then $M$ halts on $w$ and accepts $w$
  - If $w \notin L$, then $M$ halts on $w$ and rejects $w$

# Decidability

- **Halting problem**
  - Given the description of a Turing machine $M$ and an input $w$, does $M$, when started in the initial configuration $q_0w$, perform a computation that eventually halts?
    - $(M, w)$ halts or does not halt
    - Domain of this problem is the set of all Turing machines and all $w$

# Decidability

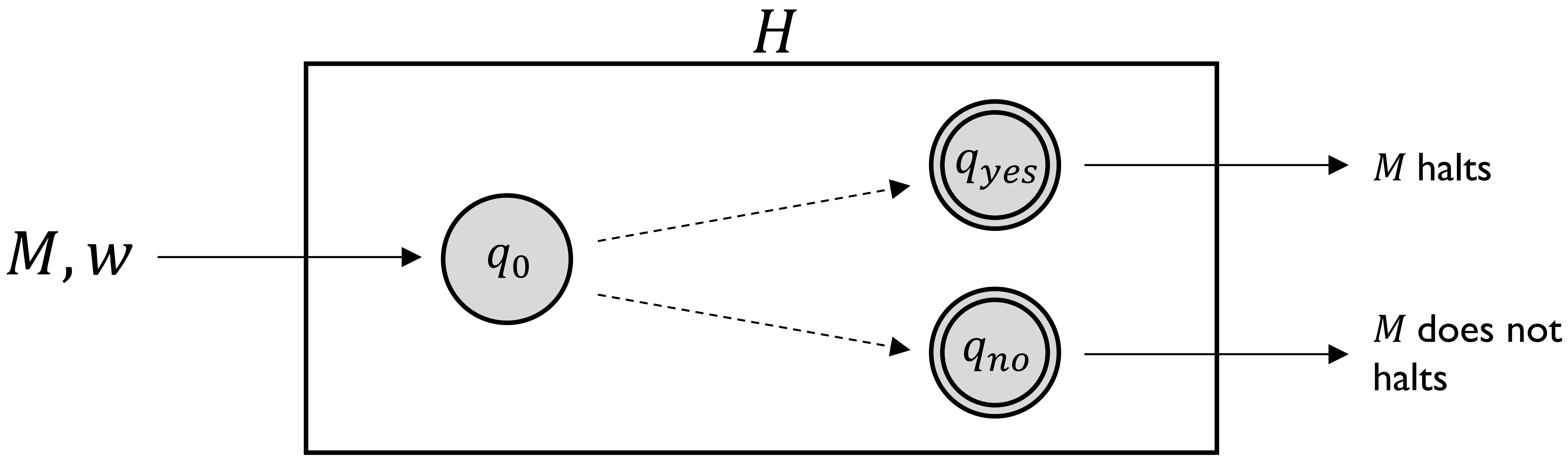
- **Halting problem**
  - Given the description of a Turing machine $M$ and an input $w$, does $M$, when started in the initial configuration $q_0 w$, perform a computation that eventually halts?
    - $(M, w)$ halts or does not halt
    - Domain of this problem is the set of all Turing machines and all $w$
  - We cannot find the answer by simply simulating $M$ with respect to $w$
    - Actually entering an infinite loop vs. very long calculation

# Decidability

- **Halting problem (undecidable)**
  - Given the description of a Turing machine $M$ and an input $w$, does $M$, when started in the initial configuration $q_0 w$, perform a computation that eventually halts?
    - $(M, w)$ halts or does not halt
    - Domain of this problem is the set of all Turing machines and all $w$
  - We cannot find the answer by simply simulating $M$ with respect to $w$
    - Actually entering an infinite loop vs. very long calculation
  - Actually, this is a undecidable problem!

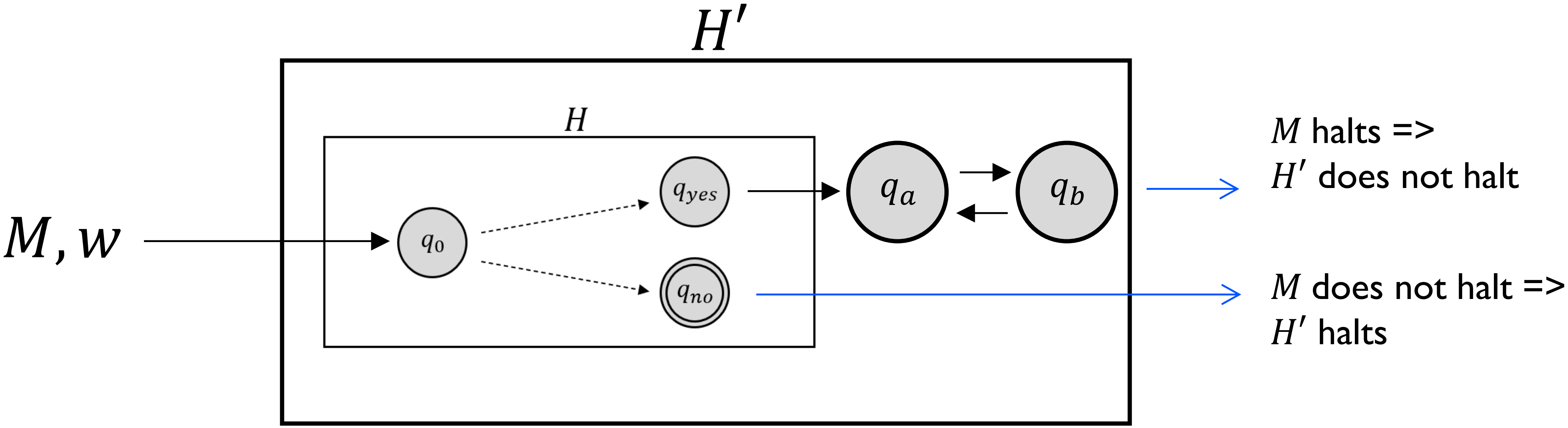
# Decidability

- **Proof by Contradiction**
  - Assume we can create a Turing machine $H(M, w)$
    - After receiving $M$ and $w$, $H$ will output whether or not the Turing machine $M$ halts



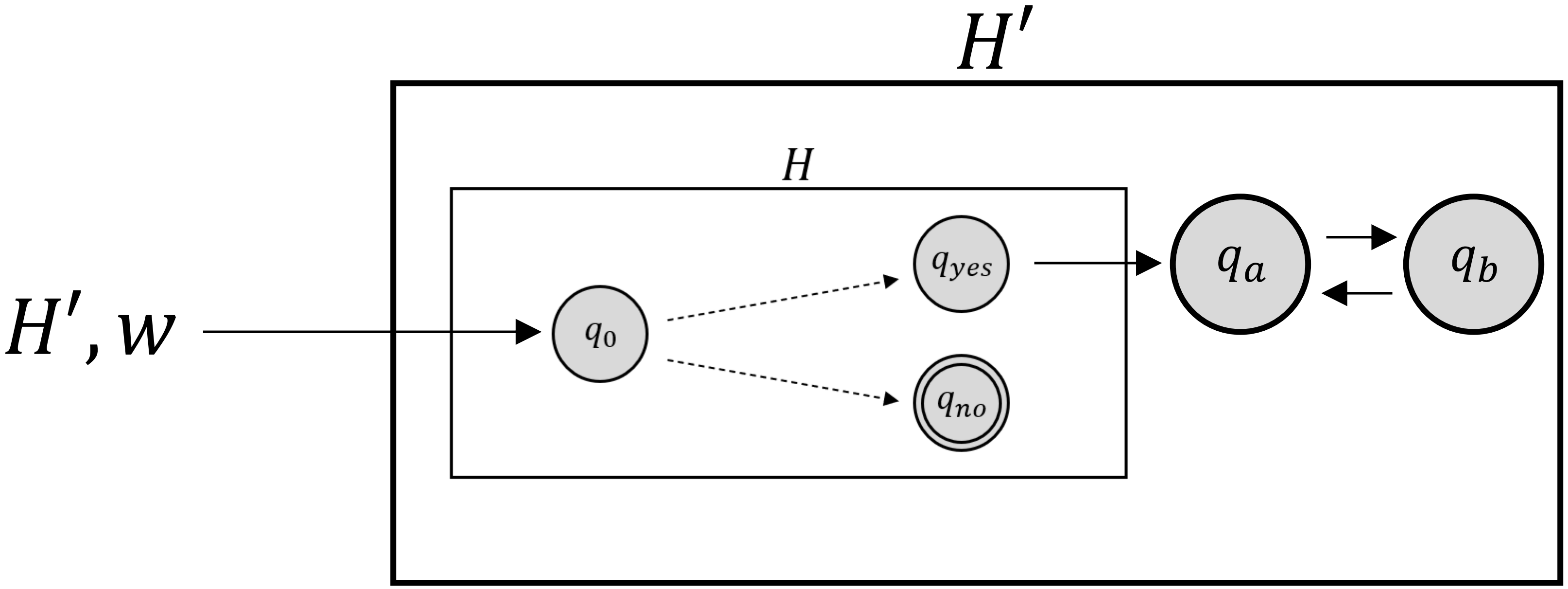
# Decidability

- **Proof by Contradiction**
  - Consider an inverted Turing machine $H'(M,w)$
    - ❖ If $H(M,w)$ returns yes, then $H'$ falls into an infinite loop
    - ❖ If $H(M,w)$ returns no, then $H'$ halts



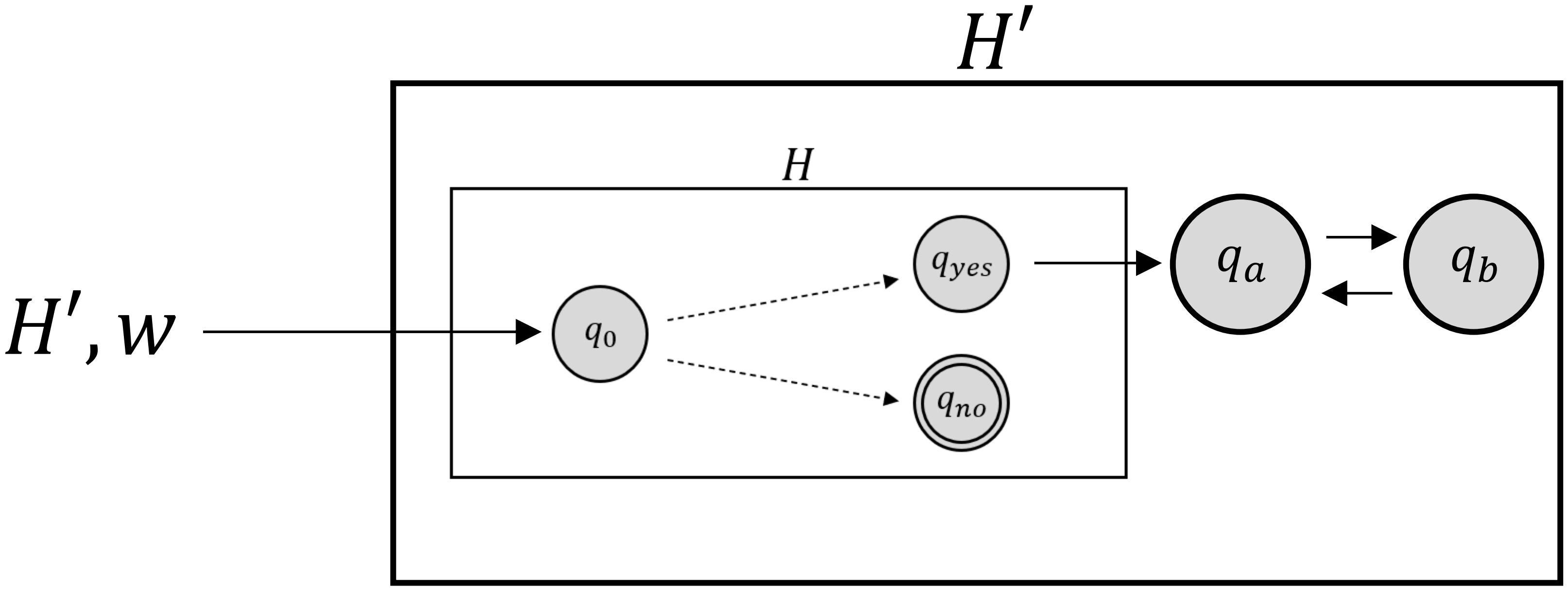
# Decidability

- **Proof by Contradiction**
  - Then, consider $H'(H', w)$
    - ❖ If $H'$ finally halts, then $H'$ falls into an infinite loop
    - ❖ If $H'$ does not halt, then $H'$ halts



# Decidability

- **Proof by Contradiction**
  - Then, consider $H'(H', w)$
    - If $H'$ finally halts, then $H'$ falls into an infinite loop
    - If $H'$ does not halt, then $H'$ halts




CONTRADICTION

# Decidability

- **Proof by Contradiction (using pseudocode)**
  - Assume that there is an algorithm that can decide the halting problem
  - Consider the function `exit(a, i)`
    - `a:` an arbitrary program to be used
    - `i:` an arbitrary input to be used

# Decidability

- **Proof by Contradiction (using pseudocode)**
  - Assume that there is an algorithm that can decide the halting problem
  - Consider the function `exit(a, i)`
    - `a:` an arbitrary program to be used
    - `i:` an arbitrary input to be used

  - `exit` returns True if `a` stops after finite steps for input `i` and returns a result
  - `exit` returns False if `a` does not stop with an input `i` (e.g., infinite loop)

# Decidability

- **Proof by Contradiction (using pseudocode)**
  - Let consider another function `test(s)`

```
function test(s) {
  if exit(s,s) == false
    return True
  else
    loop forever
}
```

# Decidability

- **Proof by Contradiction (using pseudocode)**
  - Let consider another function `test(s)`

```
function test(s) {
  if exit(s,s) == false
    return True
  else
    loop forever
}
```

Is `exit(test, test)` True?

# Decidability

- **Proof by Contradiction (using pseudocode)**
  - Let consider another function `test(s)`

```
function test(s) {
  if exit(s,s) == false
    return True
  else
    loop forever
}
```

`exit(test, test)` == True

$\Rightarrow$ `test(test)` should be halted

$\Rightarrow$ But because `exist(test, test)` is True, thus `test(test)` does not be halted

**CONTRADICTION**

# Decidability

- **Proof by Contradiction (using pseudocode)**
  - Let consider another function `test(s)`

```
function test(s) {
  if exit(s,s) == false
    return True
  else
    loop forever
}
```

`exit(test, test)` == False

⇒ `test(test)` should not be halted

⇒ But because `exist(test, test)` is False, thus `test(test)` halts

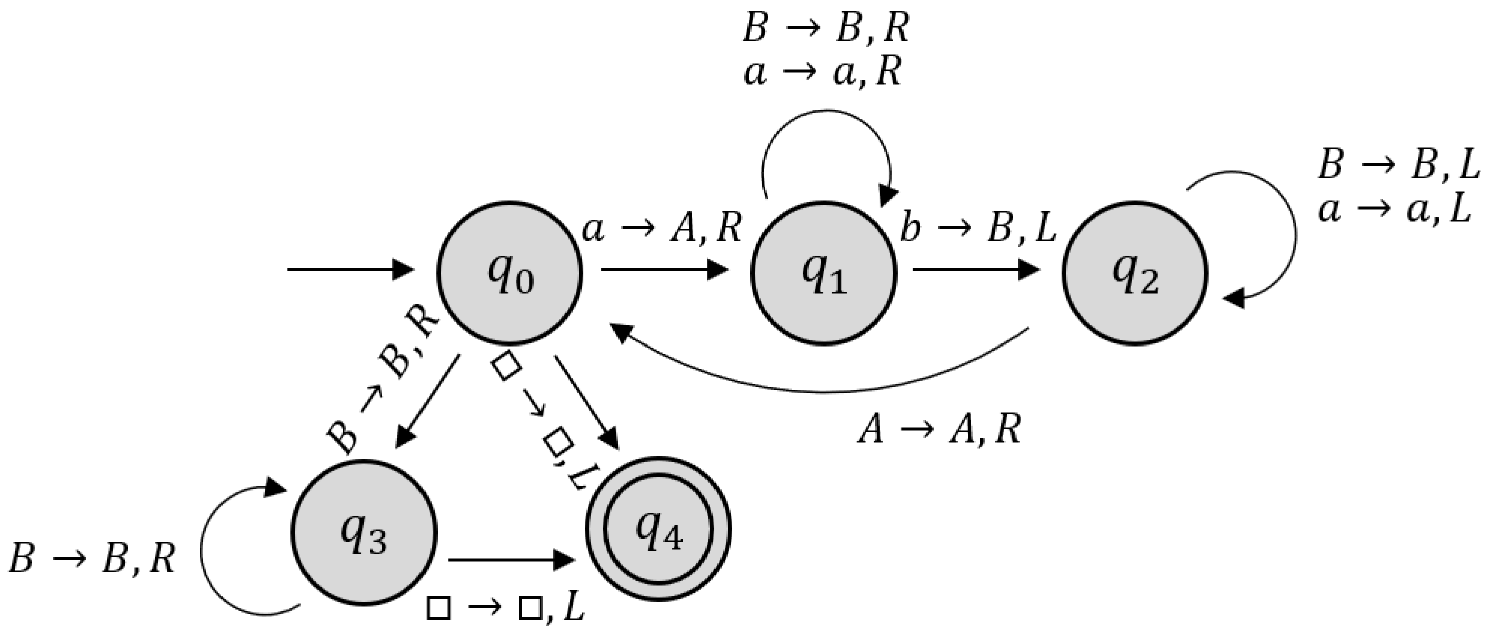
**CONTRADICTION**

# Efficiency

- **Certain problems can be solved by different automata**
  - There may be a difference in terms of efficiency
  - E.g., Standard Turing machine vs. Multitape Turing machine

# Efficiency

- **Example**
  - Standard Turing machine for $L = \{a^n b^n \mid n \geq 0\}$
    - $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, \{a, b, A, B, \square\}, \delta, q_0, \square, \{q_4\})$

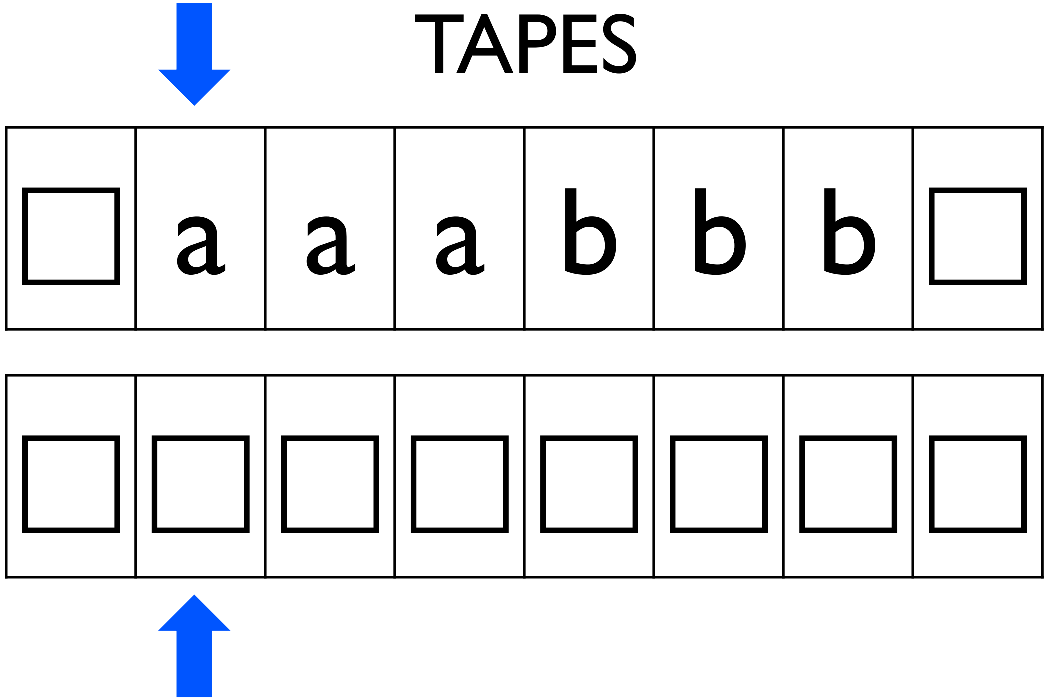


# Efficiency

- **Example**
  - Standard Turing machine for $L = \{a^n b^n \mid n \geq 0\}$
    - Roughly $2n$ steps are required to match each $a$ with the corresponding $b$
    - Complexity: $O(n^2)$

# Efficiency

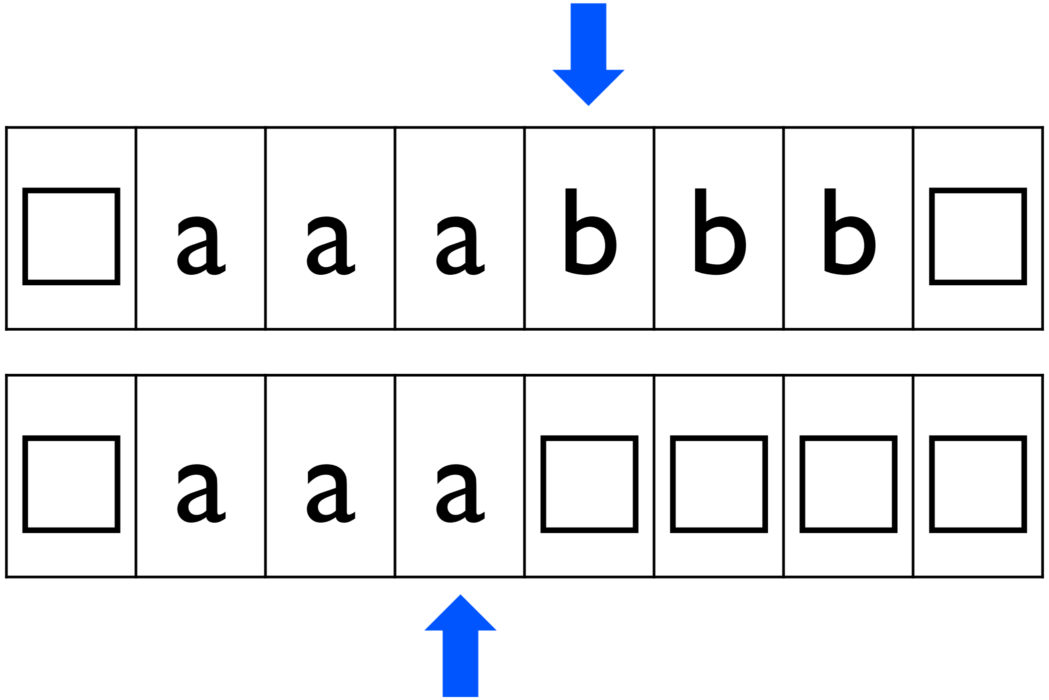
- **Example**
  - Multitape Turing machine for $L = \{a^n b^n \mid n \geq 0\}$




Copy a's

# Efficiency

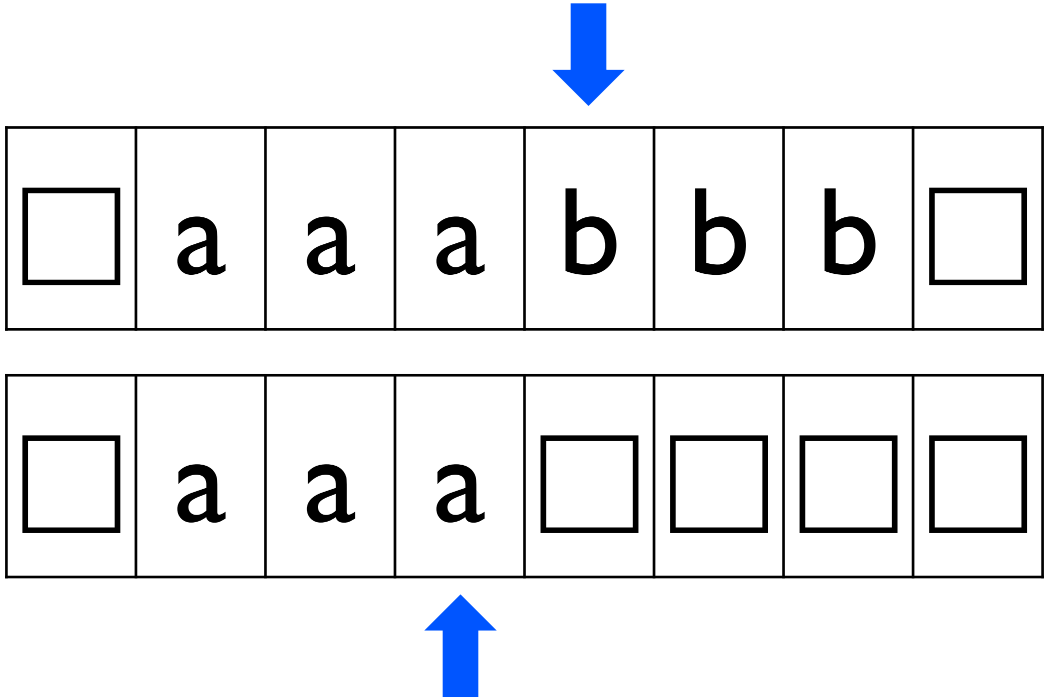
- **Example**
  - Multitape Turing machine for $L = \{a^n b^n \mid n \geq 0\}$




Compare
Tape1 and Tape2

# Efficiency

- **Example**
  - Multitape Turing machine for $L = \{a^n b^n \mid n \geq 0\}$




Complexity
= $O(n)$

# Unrestricted grammar

- **Definition**
  - $G = (V, T, S, P)$ is said to be unrestricted if all productions are of the form
    - $x \to y$

    where $x \in (V \cup T)^+$ and $y \in (V \cup T)^*$

# Unrestricted grammar

- **Definition**
  - $G = (V, T, S, P)$ is said to be unrestricted if all productions are of the form
    - $x \to y$

    where $x \in (V \cup T)^+$ and $y \in (V \cup T)^*$

- Unrestricted grammar generates exactly the family of recursively enumerable languages!
  - For every recursively enumerable language $L$, there exists an unrestricted grammar $G$, such that $L = L(G)$

# Context-sensitive grammar

- **Definition**
  - $G = (V, T, S, P)$ is said to be context-sensitive if all productions are of the form
    - $x \to y$

    where $x, y \in (V \cup T)^+$ and $|x| \leq |y|$

# Context-sensitive grammar

- **Definition**
  - $G = (V, T, S, P)$ is said to be context-sensitive if all productions are of the form
    - $x \to y$

    where $x, y \in (V \cup T)^{+}$ and $|x| \leq |y|$

- Context-sensitive grammar generates the family of context-sensitive languages!
  - For every context-sensitive language $L$, there exists a context-sensitive grammar $G$, such that $L = L(G)$ or $L = L(G) \cup \{\lambda\}$

# Context-sensitive grammar

- **CSL can be recognized by linear bounded automata**
  - A nondeterministic Turing machine that limits the size of tape that can be used
    - Size limits vary depending on input
    - e.g., exactly equal to the length of the input, or we can use as multiples of input
  - Use '[' and ']' to restrict tape cells
    - Immutable tape alphabets (we cannot convert them to other alphabets)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\dots$ | [ | $a$ | $a$ | $b$ | $b$ | ] | $\dots$ |

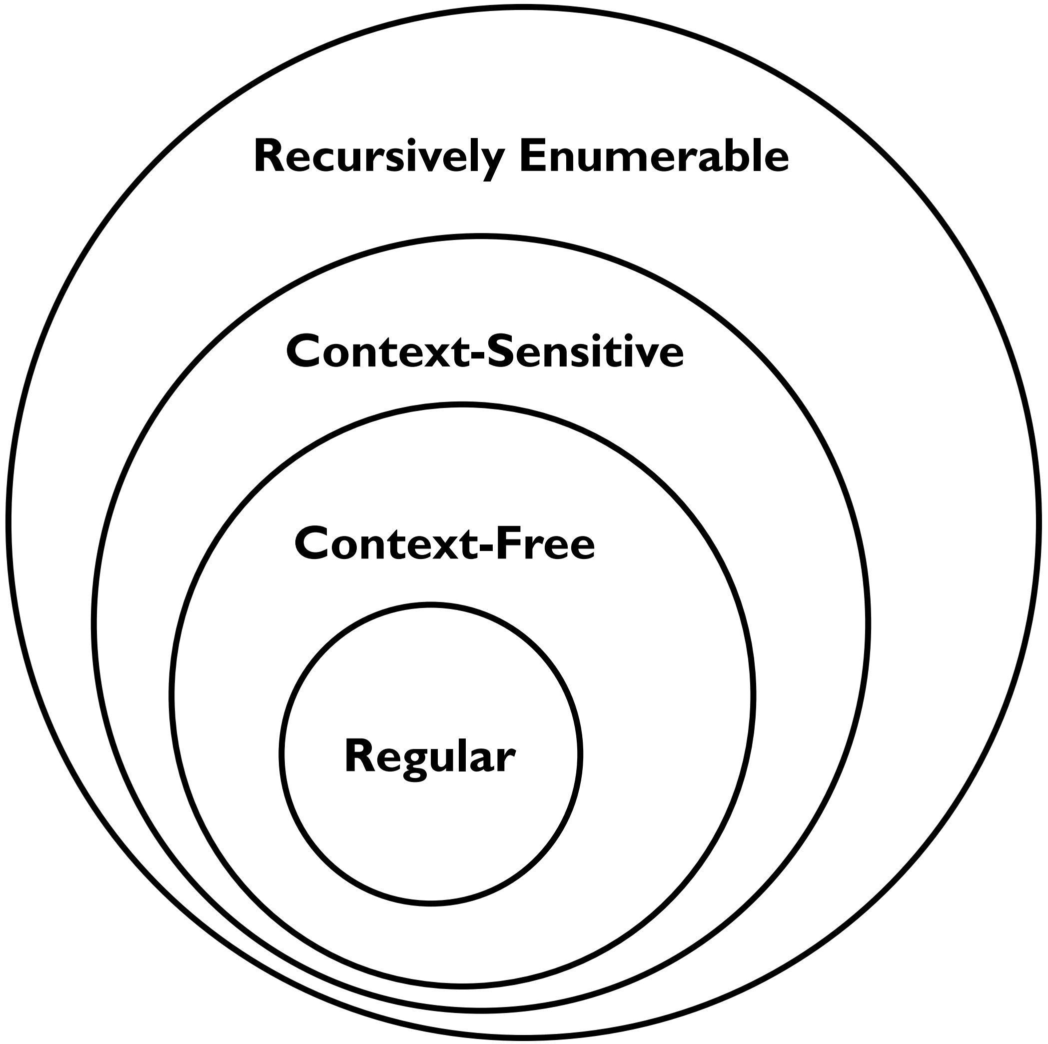
# Context-sensitive grammar

- **CSL can be recognized by linear bounded automata**
  - A nondeterministic Turing machine that limits the size of tape that can be used
    - ❖ Size limits vary depending on input
    - ❖ e.g., exactly equal to the length of the input, or we can use as multiples of input
  - Use '[' and ']' to restrict tape cells
    - ❖ Immutable tape alphabets (we cannot convert them to other alphabets)

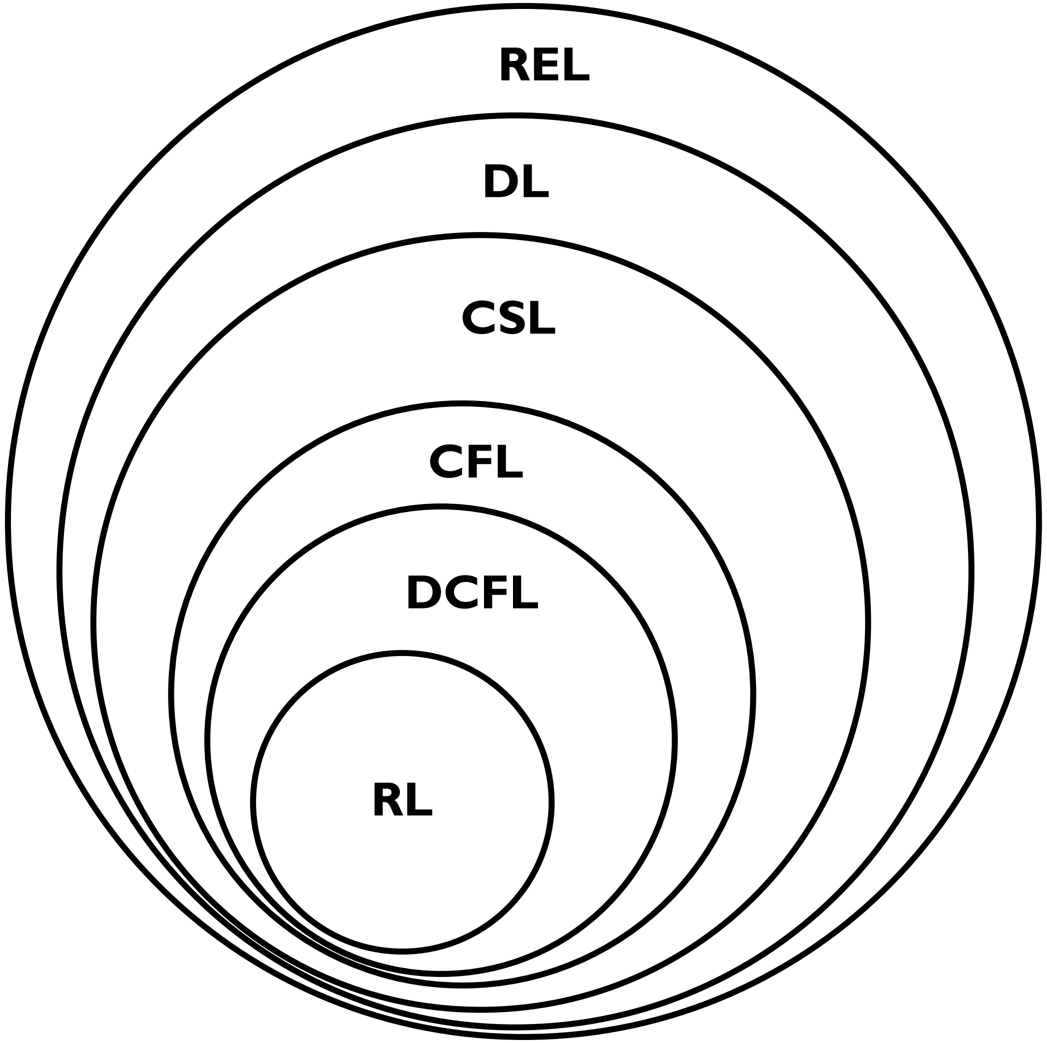
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\ldots$ | [ | $a$ | $a$ | $b$ | $b$ | ] | $\ldots$ |

**Since CSL and LBA are not covered in detail in the ToC textbooks, these will be mentioned briefly**

# Chomsky Hierarchy



# Language Relationships



# Next Lecture

$\emptyset$

Small seminar..!