Automata & Complexity

Jörg Endrullis

Vrije Universiteit Amsterdam

2018

Looking Back

Previous subjects relevant for this lecture:

- Not all languages are context-free:
  pumping lemma for context-free languages.

- Turing machines: automata with unlimited memory (tape)
  can describe languages that are not context-free.

Variations of Turing Machines

We consider the following variations of Turing machines:

- Turing machines with multiple tapes
- nondeterministic Turing machines
- universal Turing machines

Turing Machines with Multiple Tapes

Theorem

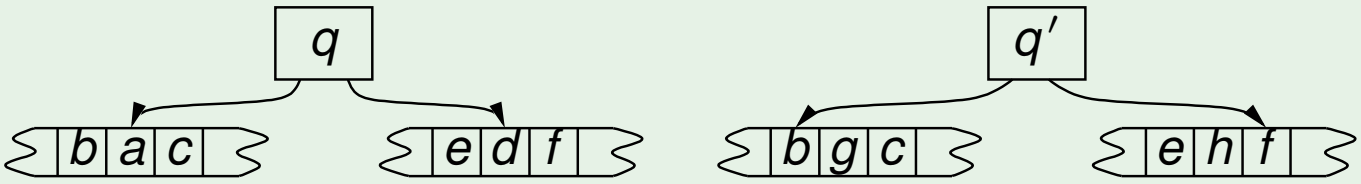
A TM with two tapes can be simulated by a TM with one tape.

Turing Machines with Multiple Tapes

Theorem

A TM with two tapes can be simulated by a TM with one tape.

Consider $\delta(q, a, d) = (q', g, h, L, R)$

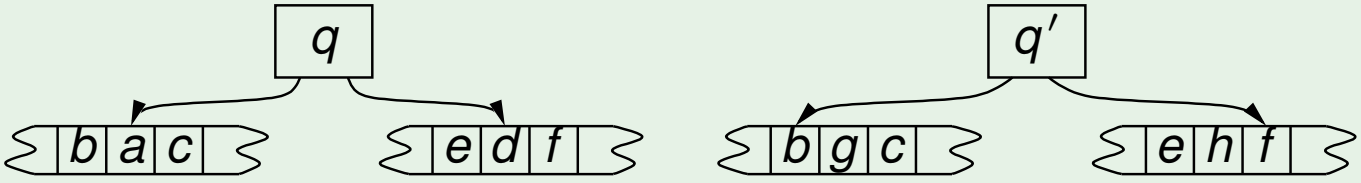


Turing Machines with Multiple Tapes

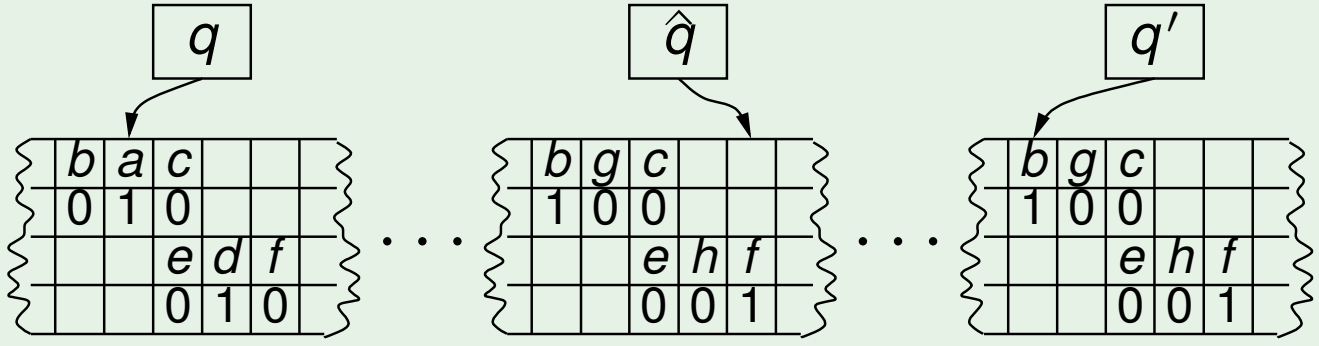
Theorem

A TM with two tapes can be simulated by a TM with one tape.

Consider $\delta(q, a, d) = (q', g, h, L, R)$



The transition translates to multiple transitions on one tape:

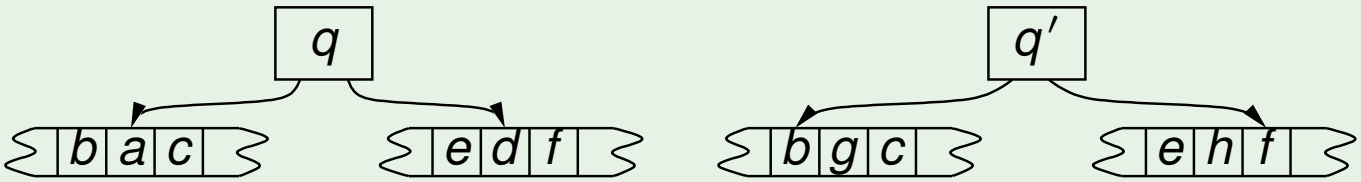


Turing Machines with Multiple Tapes

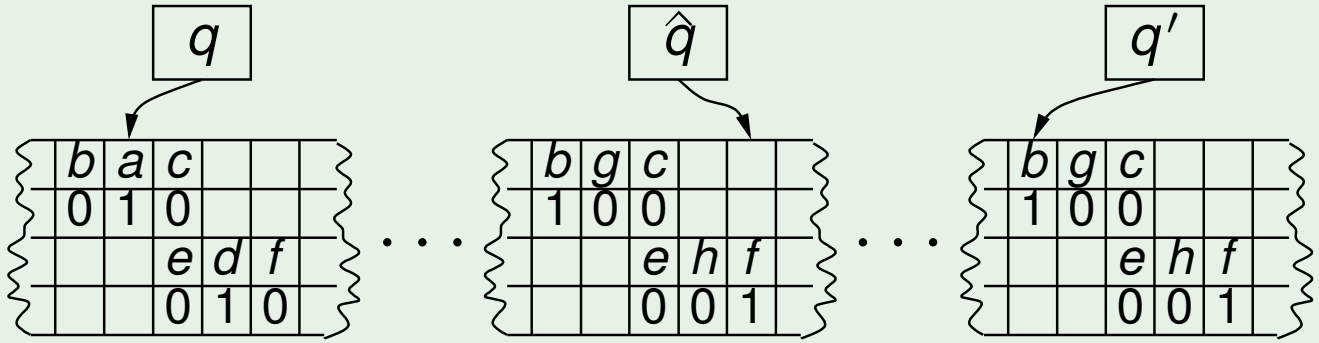
Theorem

A TM with two tapes can be simulated by a TM with one tape.

Consider $\delta(q, a, d) = (q', g, h, L, R)$



The transition translates to multiple transitions on one tape:



The difference in **time complexity** between a Turing machine with one tape and multiple tapes is a **polynomial** factor.

Nondeterministic Turing Machines

A nondeterministic Turing machine has as transition function

$$\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L,R\}}$$

Nondeterministic Turing Machines

A nondeterministic Turing machine has as transition function

$$\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L,R\}}$$

Theorem

A nondeterministic TM can be simulated by a deterministic TM.

Nondeterministic Turing Machines

A nondeterministic Turing machine has as transition function

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$$

Theorem

A nondeterministic TM can be simulated by a deterministic TM.

Proof.

The deterministic TM can use

- **breadth-first search**

to simulate all executions of the nondeterministic TM in parallel.

The branches of the breadth-first search can be stored on the tape in the form **queue**. □

Nondeterministic Turing Machines

A nondeterministic Turing machine has as transition function

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$$

Theorem

A nondeterministic TM can be simulated by a deterministic TM.

Proof.

The deterministic TM can use

- **breadth-first search**

to simulate all executions of the nondeterministic TM in parallel.

The branches of the breadth-first search can be stored on the tape in the form **queue**. □

The difference in **time complexity** between nondeterministic and deterministic TM is, as far as known, an **exponential** factor.

Universal Turing Machine

A computer can execute any program on any input.

Universal Turing Machine

A computer can execute any program on any input.

A TM is called **universal** if it can simulate every TM.

A universal TM gets as input

- a Turing machine M (described as a word w),
- an input word u,

and then executes (simulates) M on u.

Universal Turing Machine

A computer can execute any program on any input.

A TM is called **universal** if it can simulate every TM.

A universal TM gets as input

- a Turing machine M (described as a word w),
- an input word u,

and then executes (simulates) M on u.

These two inputs can either

- be written on two different tapes, or
- be written behind each other ($w\#u$) on one tape.

Universal Turing Machine

A computer can execute any program on any input.

A TM is called **universal** if it can simulate every TM.

A universal TM gets as input

- a Turing machine M (described as a word w),
- an input word u,

and then executes (simulates) M on u.

These two inputs can either

- be written on two different tapes, or
- be written behind each other ($w\#u$) on one tape.

Theorem

There exists a universal Turing machine.

Unrestricted Grammars

What class of grammars corresponds to Turing machines?

Unrestricted Grammars

What class of grammars corresponds to Turing machines?

An unrestricted grammar G contains rules

$$x \rightarrow y$$

where $x \neq \lambda$.

Note that there is no restriction other than x being non-empty.

Unrestricted Grammars

What class of grammars corresponds to Turing machines?

An unrestricted grammar G contains rules

$$x \rightarrow y$$

where $x \neq \lambda$.

Note that there is no restriction other than x being non-empty.

Theorem

A language L is generated by an unrestricted grammar
$\iff$ L is accepted by a Turing machine.

From Unrestricted Grammars to Turing Machines

Theorem

For every unrestricted G there is a Turing machine M such that

$$L(M) = L(G)$$

From Unrestricted Grammars to Turing Machines

Theorem

For every unrestricted G there is a Turing machine M such that

$$L(M) = L(G)$$

Construction

Input for M is a word w (written on the tape).

From Unrestricted Grammars to Turing Machines

Theorem

For every unrestricted G there is a Turing machine M such that

$$L(M) = L(G)$$

Construction

Input for M is a word w (written on the tape).

M can do a **breadth-first search** for a derivation of w from S.

From Unrestricted Grammars to Turing Machines

Theorem

For every unrestricted G there is a Turing machine M such that

$$L(M) = L(G)$$

Construction

Input for M is a word w (written on the tape).

M can do a **breadth-first search** for a derivation of w from S.

If a derivation is found, then w is accepted by M.

From Unrestricted Grammars to Turing Machines

Theorem

For every unrestricted G there is a Turing machine M such that

$$L(M) = L(G)$$

Construction

Input for M is a word w (written on the tape).

M can do a **breadth-first search** for a derivation of w from S.

If a derivation is found, then w is accepted by M.

Then $L(M) = L(G)$.

From Turing Machines to Unrestricted Grammars

Theorem

For every TM M there is a grammar G with $L(G) = L(M)$.

From Turing Machines to Unrestricted Grammars

Theorem

For every TM M there is a grammar G with $L(G) = L(M)$.

Construction

The variables are S, T, $\square$

and V_γ^α, $V_{q\gamma}^\alpha$ for every $\alpha \in \Sigma \cup \{\square\}$, $\gamma \in \Gamma$ and $q \in Q$.

From Turing Machines to Unrestricted Grammars

Theorem

For every TM M there is a grammar G with $L(G) = L(M)$.

Construction

The variables are S, T, $\square$

and V_γ^α, $V_{q\gamma}^\alpha$ for every $\alpha \in \Sigma \cup \{\square\}$, $\gamma \in \Gamma$ and $q \in Q$.

Step 1: guessing the word w

$$S \to V_\square^\square S \mid S V_\square^\square \mid T$$

$$T \to T V_a^a \mid V_{q_0 a}^a \qquad \text{for every } a \in \Sigma$$

From Turing Machines to Unrestricted Grammars

Theorem

For every TM M there is a grammar G with $L(G) = L(M)$.

Construction

The variables are S, T, $\square$

and V_γ^α, $V_{q\gamma}^\alpha$ for every $\alpha \in \Sigma \cup \{\square\}$, $\gamma \in \Gamma$ and $q \in Q$.

Step 1: guessing the word w

$$S \to V_\square^\square S \mid S V_\square^\square \mid T$$

$$T \to T V_a^a \mid V_{q_0 a}^a \qquad \text{for every } a \in \Sigma$$

After step 1, we have derived something of the form

$$V_\square^\square \cdots V_\square^\square V_{q_0 a_1}^{a_1} V_{a_2}^{a_2} V_{a_3}^{a_3} \cdots V_{a_n}^{a_n} V_\square^\square \cdots V_\square^\square$$

where $w = a_1 a_2 \cdots a_n$.

From Turing Machines to Unrestricted Grammars

Theorem

For every TM M there is a grammar G with $L(G) = L(M)$.

Construction

The variables are S, T, $\square$

and V_γ^α, $V_{q\gamma}^\alpha$ for every $\alpha \in \Sigma \cup \{\square\}$, $\gamma \in \Gamma$ and $q \in Q$.

Step 1: guessing the word w

$$S \to V_\square^\square S \mid S V_\square^\square \mid T$$
$$T \to T V_a^a \mid V_{q_0 a}^a \qquad \text{for every } a \in \Sigma$$

After step 1, we have derived something of the form

$$V_\square^\square \cdots V_\square^\square V_{q_0 a_1}^{a_1} V_{a_2}^{a_2} V_{a_3}^{a_3} \cdots V_{a_n}^{a_n} V_\square^\square \cdots V_\square^\square$$

where $w = a_1 a_2 \cdots a_n$.

Next, the TM is simulated using the lower line (the subscripts).

From Turing Machines to Unrestricted Grammars

Construction continued

Step 2: simulating the TM (in the subscripts)

$$V^{\alpha}_{qc} V^{\beta}_{\gamma} \to V^{\alpha}_{d} V^{\beta}_{q'\gamma} \qquad \text{if } \delta(q, c) = (q', d, R)$$

$$V^{\beta}_{\gamma} V^{\alpha}_{qc} \to V^{\beta}_{q'\gamma} V^{\alpha}_{d} \qquad \text{if } \delta(q, c) = (q', d, L)$$

for every $\alpha, \beta \in \Sigma \cup \{\square\}$ and $\gamma \in \Gamma$.

From Turing Machines to Unrestricted Grammars

Construction continued

Step 2: simulating the TM (in the subscripts)

$$V_{qc}^{\alpha} V_{\gamma}^{\beta} \to V_{d}^{\alpha} V_{q'\gamma}^{\beta} \qquad \text{if } \delta(q,c) = (q',d,R)$$

$$V_{\gamma}^{\beta} V_{qc}^{\alpha} \to V_{q'\gamma}^{\beta} V_{d}^{\alpha} \qquad \text{if } \delta(q,c) = (q',d,L)$$

for every $\alpha, \beta \in \Sigma \cup \{\square\}$ and $\gamma \in \Gamma$.

Step 3: If TM reaches accepting state, then generate w.
(From the superscripts left unchanged in step 2.)

$$V_{q\gamma}^{\alpha} \to \alpha \qquad \text{for every } q \in F$$

$$\beta V_{\gamma}^{\alpha} \to \beta\alpha$$

$$V_{\gamma}^{\alpha} \beta \to \alpha\beta$$

$$\square \to \lambda$$

for every $\alpha, \beta \in \Sigma \cup \{\square\}$ and $\gamma \in \Gamma$.

From Turing Machines to Unrestricted Grammars

Construction continued

Step 2: simulating the TM (in the subscripts)

$$V_{qc}^{\alpha} V_{\gamma}^{\beta} \to V_{d}^{\alpha} V_{q'\gamma}^{\beta} \qquad \text{if } \delta(q,c) = (q',d,R)$$
$$V_{\gamma}^{\beta} V_{qc}^{\alpha} \to V_{q'\gamma}^{\beta} V_{d}^{\alpha} \qquad \text{if } \delta(q,c) = (q',d,L)$$

for every $\alpha, \beta \in \Sigma \cup \{\square\}$ and $\gamma \in \Gamma$.

Step 3: If TM reaches accepting state, then generate w.
(From the superscripts left unchanged in step 2.)

$$V_{q\gamma}^{\alpha} \to \alpha \qquad \text{for every } q \in F$$
$$\beta V_{\gamma}^{\alpha} \to \beta\alpha$$
$$V_{\gamma}^{\alpha} \beta \to \alpha\beta$$
$$\square \to \lambda$$

for every $\alpha, \beta \in \Sigma \cup \{\square\}$ and $\gamma \in \Gamma$.

Then $L(G) = L(M)$.

Example

Consider the TM with $\Sigma = \{a, b, c\}$, $\Gamma = \Sigma \cup \{\Box\}$, $F = \{q_2\}$ and

$$\delta(q_0, a) = (q_0, c, R) \qquad \delta(q_0, c) = (q_1, b, L)$$
$$\delta(q_0, b) = (q_0, b, R) \qquad \delta(q_1, b) = (q_2, a, R)$$

This TM accepts the language $L((a + b)^* bc(a + b + c)^*)$.

Example

Consider the TM with $\Sigma = \{a, b, c\}$, $\Gamma = \Sigma \cup \{\Box\}$, $F = \{q_2\}$ and

$$\delta(q_0, a) = (q_0, c, R) \qquad \delta(q_0, c) = (q_1, b, L)$$
$$\delta(q_0, b) = (q_0, b, R) \qquad \delta(q_1, b) = (q_2, a, R)$$

This TM accepts the language $L((a + b)^* bc(a + b + c)^*)$.

The resulting grammar is:

Example

Consider the TM with $\Sigma = \{a, b, c\}$, $\Gamma = \Sigma \cup \{\Box\}$, $F = \{q_2\}$ and

$$\delta(q_0, a) = (q_0, c, R) \qquad \delta(q_0, c) = (q_1, b, L)$$
$$\delta(q_0, b) = (q_0, b, R) \qquad \delta(q_1, b) = (q_2, a, R)$$

This TM accepts the language $L((a+b)^* bc(a+b+c)^*)$.

The resulting grammar is:

$$S \to V_{\Box}^{\Box} S \mid S V_{\Box}^{\Box} \mid T$$
$$T \to T V_a^a \mid T V_b^b \mid T V_c^c \mid V_{q_0 a}^a \mid V_{q_0 b}^b \mid V_{q_0 c}^c$$

$$V_{q_0 a}^{\alpha} V_{\gamma}^{\beta} \to V_c^{\alpha} V_{q_0 \gamma}^{\beta} \qquad V_{q_2 \gamma}^{\alpha} \to \alpha$$
$$V_{q_0 b}^{\alpha} V_{\gamma}^{\beta} \to V_b^{\alpha} V_{q_0 \gamma}^{\beta} \qquad \beta V_{\gamma}^{\alpha} \to \beta \alpha$$
$$V_{\gamma}^{\beta} V_{q_0 c}^{\alpha} \to V_{q_1 \gamma}^{\beta} V_b^{\alpha} \qquad V_{\gamma}^{\alpha} \beta \to \alpha \beta$$
$$V_{q_1 b}^{\alpha} V_{\gamma}^{\beta} \to V_a^{\alpha} V_{q_2 \gamma}^{\beta} \qquad \Box \to \lambda$$

with $\alpha, \beta \in \Sigma \cup \{\Box\}$ and $\gamma \in \Gamma$.

Example

Consider the TM with $\Sigma = \{a, b, c\}$, $\Gamma = \Sigma \cup \{\Box\}$, $F = \{q_2\}$ and

$$\delta(q_0, a) = (q_0, c, R) \qquad \delta(q_0, c) = (q_1, b, L)$$
$$\delta(q_0, b) = (q_0, b, R) \qquad \delta(q_1, b) = (q_2, a, R)$$

This TM accepts the language $L((a+b)^* bc(a+b+c)^*)$.

The resulting grammar is:

$$S \to V_\Box^\Box S \mid S V_\Box^\Box \mid T$$
$$T \to T V_a^a \mid T V_b^b \mid T V_c^c \mid V_{q_0 a}^a \mid V_{q_0 b}^b \mid V_{q_0 c}^c$$

$$V_{q_0 a}^\alpha V_\gamma^\beta \to V_c^\alpha V_{q_0 \gamma}^\beta \qquad V_{q_2 \gamma}^\alpha \to \alpha$$
$$V_{q_0 b}^\alpha V_\gamma^\beta \to V_b^\alpha V_{q_0 \gamma}^\beta \qquad \beta V_\gamma^\alpha \to \beta\alpha$$
$$V_\gamma^\beta V_{q_0 c}^\alpha \to V_{q_1 \gamma}^\beta V_b^\alpha \qquad V_\gamma^\alpha \beta \to \alpha\beta$$
$$V_{q_1 b}^\alpha V_\gamma^\beta \to V_a^\alpha V_{q_2 \gamma}^\beta \qquad \Box \to \lambda$$

with $\alpha, \beta \in \Sigma \cup \{\Box\}$ and $\gamma \in \Gamma$. Exercise: derive abc.

Context-Sensitive Grammars

A grammar is **context-sensitive** if for every rule $x \to y$ it holds:

$$|x| \leq |y| \quad \text{(and } x \neq \lambda\text{)}$$

Note: the words cannot get shorter during derivation.

Context-Sensitive Grammars

A grammar is **context-sensitive** if for every rule $x \to y$ it holds:

$$|x| \leq |y| \quad \text{(and } x \neq \lambda\text{)}$$

Note: the words cannot get shorter during derivation.

For every context-sensitive grammar G_1
there exists a grammar G_2 with rules of the form

$$xAy \to xvy \quad \text{with} \quad v \neq \lambda$$

such that $L(G_1) = L(G_2)$.

(Compare with the shape of rules in a context-free grammar.)

Context-Sensitive Grammars

A grammar is **context-sensitive** if for every rule $x \to y$ it holds:

$$|x| \leq |y| \quad \text{(and } x \neq \lambda\text{)}$$

Note: the words cannot get shorter during derivation.

For every context-sensitive grammar G_1
there exists a grammar G_2 with rules of the form

$$xAy \to xvy \quad \text{with} \quad v \neq \lambda$$

such that $L(G_1) = L(G_2)$.

(Compare with the shape of rules in a context-free grammar.)

A language L is **context-sensitive** if there exists a context-sensitive grammar G with $L(G) = L \setminus \{\lambda\}$.

Example

The language

$$\{ a^n b^n c^n \mid n \geq 1 \}$$

is generated by the context-sensitive grammar:

$$S \rightarrow aAbc \mid abc$$
$$A \rightarrow aAB \mid aB$$
$$Bb \rightarrow bB$$
$$Bc \rightarrow bcc$$

Example derivation:

$$\begin{aligned} S &\Rightarrow aAbc \Rightarrow aaABbc \Rightarrow aaAbBc \\ &\Rightarrow aaAbbcc \Rightarrow aaaBbbcc \Rightarrow aaabBbcc \\ &\Rightarrow aaabbBcc \Rightarrow aaabbbccc \end{aligned}$$

Linear Bounded Automata

A **linear bounded automaton**, short **LBA**,
is a nondeterministic TM $(Q, \Sigma, \Gamma, \delta, q_0, F)$.

Note that there is no $\square$!

Instead, we have symbols $[$ and $]$, and

- $[$ and $]$ are placed around the input word
- for every $q \in Q$, $\delta(q, [\,)$ is of the form $(q', [\,, R)$
- for every $q \in Q$, $\delta(q,]\,)$ is of the form $(q',]\,, L)$

The head can only move within the bounds of the input word!

Linear Bounded Automata

A **linear bounded automaton**, short **LBA**,
is a nondeterministic TM $(Q, \Sigma, \Gamma, \delta, q_0, F)$.

Note that there is no $\square$!

Instead, we have symbols [and], and

- [and] are placed around the input word
- for every $q \in Q$, $\delta(q, [\,)$ is of the form $(q', [\,, R)$
- for every $q \in Q$, $\delta(q,]\,)$ is of the form $(q',]\,, L)$

The head can only move within the bounds of the input word!

So the memory is restricted by the length of the input word.

Linear Bounded Automata

A **linear bounded automaton**, short **LBA**,
is a nondeterministic TM $(Q, \Sigma, \Gamma, \delta, q_0, F)$.

Note that there is no $\square$!

Instead, we have symbols $[$ and $]$, and

- $[$ and $]$ are placed around the input word
- for every $q \in Q$, $\delta(q, [\,)$ is of the form $(q', [\,, R)$
- for every $q \in Q$, $\delta(q,]\,)$ is of the form $(q',]\,, L)$

The head can only move within the bounds of the input word!

So the memory is restricted by the length of the input word.

The language $L(M)$ accepted by LBA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is

$$\{\, w \in \Sigma^+ \mid q_0[w] \vdash^+ [uqv] \text{ for some } q \in F, u, v \in \Gamma^* \,\}$$

From Context-Sensitive Grammars to LBA's

Theorem

For every context-sensitive grammar G there exists an LBA M such that $L(M) = L(G)$.

From Context-Sensitive Grammars to LBA's

Theorem

For every context-sensitive grammar G there exists an LBA M such that $L(M) = L(G)$.

Proof.

A derivation of $w \in L(G)$ contains only words of length $\leq |w|$.

A nondeterministic Turing machine can simulate (guess) this derivation without leaving the bounds of w. $\square$

From LBA's to Context-Sensitive Grammars

Theorem

For every LBA M, the language $L(M)$ is context-sensitive.

From LBA's to Context-Sensitive Grammars

Theorem

For every LBA M, the language $L(M)$ is context-sensitive.

Proof sketch.

As before, build an unrestricted grammar G with $L(G) = L(M)$.

From LBA's to Context-Sensitive Grammars

Theorem

For every LBA M, the language $L(M)$ is context-sensitive.

Proof sketch.

As before, build an unrestricted grammar G with $L(G) = L(M)$.

All productions rules are context-sensitive, except for:

$$\square \rightarrow \lambda$$

From LBA's to Context-Sensitive Grammars

Theorem

For every LBA M, the language $L(M)$ is context-sensitive.

Proof sketch.

As before, build an unrestricted grammar G with $L(G) = L(M)$.

All productions rules are context-sensitive, except for:

$$\square \to \lambda$$

However, a linear bounded automaton does not use $\square$!
(It never leaves the borders of the input word.)

From LBA's to Context-Sensitive Grammars

Theorem

For every LBA M, the language $L(M)$ is context-sensitive.

Proof sketch.

As before, build an unrestricted grammar G with $L(G) = L(M)$.

All productions rules are context-sensitive, except for:

$$\square \to \lambda$$

However, a linear bounded automaton does not use $\square$!
(It never leaves the borders of the input word.)

Therefore, we can drop

- the rule $\square \to \lambda$

From LBA's to Context-Sensitive Grammars

Theorem

For every LBA M, the language $L(M)$ is context-sensitive.

Proof sketch.

As before, build an unrestricted grammar G with $L(G) = L(M)$.

All productions rules are context-sensitive, except for:

$$\square \to \lambda$$

However, a linear bounded automaton does not use $\square$!
(It never leaves the borders of the input word.)

Therefore, we can drop

- the rule $\square \to \lambda$, and
- the rules $S \to V_{\square}^{\square} S \mid S V_{\square}^{\square}$.

From LBA's to Context-Sensitive Grammars

Theorem

For every LBA M, the language $L(M)$ is context-sensitive.

Proof sketch.

As before, build an unrestricted grammar G with $L(G) = L(M)$.

All productions rules are context-sensitive, except for:

$$\square \to \lambda$$

However, a linear bounded automaton does not use $\square$!
(It never leaves the borders of the input word.)

Therefore, we can drop

- the rule $\square \to \lambda$, and
- the rules $S \to V_{\square}^{\square} S \mid S V_{\square}^{\square}$.

(In step 1, we derive from S a word $V_{q_0[a_1}^{a_1} V_{a_2}^{a_2} \cdots V_{a_{n-1}}^{a_{n-1}} V_{a_n]}^{a_n}$.)

Basic Properties of Context-Sensitive Languages

Theorem

If L_1 and L_2 are context-sensitive, then so are

$$L_1 \cup L_2 \qquad L_1 \cap L_2 \qquad L_1^R \qquad L_1 L_2 \qquad L_1^* \qquad \overline{L_1} \qquad L_1 \setminus L_2$$

Basic Properties of Context-Sensitive Languages

Theorem

If L_1 and L_2 are context-sensitive, then so are

$$L_1 \cup L_2 \qquad L_1 \cap L_2 \qquad L_1^R \qquad L_1 L_2 \qquad L_1^* \qquad \overline{L_1} \qquad L_1 \setminus L_2$$

Proof.

Basic Properties of Context-Sensitive Languages

Theorem

If L_1 and L_2 are context-sensitive, then so are

$$L_1 \cup L_2 \qquad L_1 \cap L_2 \qquad L_1^R \qquad L_1 L_2 \qquad L_1^* \qquad \overline{L_1} \qquad L_1 \setminus L_2$$

Proof.

- $L_1 \cup L_2$, L_1^R, $L_1 L_2$, L_1^*: using grammars (or automata)

Basic Properties of Context-Sensitive Languages

Theorem

If L_1 and L_2 are context-sensitive, then so are

$$L_1 \cup L_2 \qquad L_1 \cap L_2 \qquad L_1^R \qquad L_1 L_2 \qquad L_1^* \qquad \overline{L_1} \qquad L_1 \setminus L_2$$

Proof.

- $L_1 \cup L_2$, L_1^R, $L_1 L_2$, L_1^*: using grammars (or automata)
- $L_1 \cap L_2$: using linear bounded automata

Basic Properties of Context-Sensitive Languages

Theorem

If L_1 and L_2 are context-sensitive, then so are

$$L_1 \cup L_2 \qquad L_1 \cap L_2 \qquad L_1^R \qquad L_1 L_2 \qquad L_1^* \qquad \overline{L_1} \qquad L_1 \setminus L_2$$

Proof.

- $L_1 \cup L_2$, L_1^R, $L_1 L_2$, L_1^*: using grammars (or automata)
- $L_1 \cap L_2$: using linear bounded automata
- $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$

Basic Properties of Context-Sensitive Languages

Theorem

If L_1 and L_2 are context-sensitive, then so are

$$L_1 \cup L_2 \qquad L_1 \cap L_2 \qquad L_1^R \qquad L_1 L_2 \qquad L_1^* \qquad \overline{L_1} \qquad L_1 \setminus L_2$$

Proof.

- $L_1 \cup L_2$, L_1^R, $L_1 L_2$, L_1^*: using grammars (or automata)
- $L_1 \cap L_2$: using linear bounded automata
- $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$
- $\overline{L_1}$: proven by Immerman and Szelepcsényi (1987) $\square$

Basic Properties of Context-Sensitive Languages

Theorem

If L_1 and L_2 are context-sensitive, then so are

$$L_1 \cup L_2 \qquad L_1 \cap L_2 \qquad L_1^R \qquad L_1 L_2 \qquad L_1^* \qquad \overline{L_1} \qquad L_1 \setminus L_2$$

Proof.

- $L_1 \cup L_2$, L_1^R, $L_1 L_2$, L_1^*: using grammars (or automata)
- $L_1 \cap L_2$: using linear bounded automata
- $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$
- $\overline{L_1}$: proven by Immerman and Szelepcsényi (1987) □

It is unknown whether **deterministic** LBA's are equally expressive as **nondeterministic** LBA's.

Recursively Enumerable Languages

A language L is **recursively enumerable** if L is accepted by a Turing machine.

Recursively Enumerable Languages

A language L is **recursively enumerable** if L is accepted by a Turing machine.

Equivalently, a language L is recursively enumerable if there exists a Turing machine enumerates all words in the language.

Recursively Enumerable Languages

A language L is **recursively enumerable** if L is accepted by a Turing machine.

Equivalently, a language L is recursively enumerable if there exists a Turing machine enumerates all words in the language.

That is, the Turing machine outputs words $w_1, w_2, \ldots$ such that

$$L = \{ w_1, w_2, \ldots \}$$

Recursively Enumerable Languages

A language L is **recursively enumerable** if L is accepted by a Turing machine.

Equivalently, a language L is recursively enumerable if there exists a Turing machine enumerates all words in the language.

That is, the Turing machine outputs words $w_1, w_2, \ldots$ such that

$$L = \{ w_1, w_2, \ldots \}$$

Then $w_1, w_2, \ldots$ is also called a **recursive enumeration** of L.

Turing Machines are Recursively Enumerable

Theorem

Turing machines are recursively enumerable.

Turing Machines are Recursively Enumerable

Theorem

Turing machines are recursively enumerable.

Proof.

- A Turing machine can be represented as a word.
- A parser can check whether a word represents a TM. (If so, accept.)

□

Turing Machines are Recursively Enumerable

Theorem

Turing machines are recursively enumerable.

Proof.

- A Turing machine can be represented as a word.
- A parser can check whether a word represents a TM. (If so, accept.)

□

Thus: there is a recursive enumeration $M_1, M_2, \ldots$ of all TMs.

Properties of Recursively Enumerable Languages

Theorem

The class of recursively enumerable languages is closed under

$\cup$ and $\cap$

Properties of Recursively Enumerable Languages

Theorem

The class of recursively enumerable languages is closed under

$$\cup \qquad \text{and} \qquad \cap$$

There exist recursively enumerable languages L, for which the complement $\overline{L}$ is not recursively enumerable.

Properties of Recursively Enumerable Languages

Theorem

The class of recursively enumerable languages is closed under

$$\cup \qquad \text{and} \qquad \cap$$

There exist recursively enumerable languages L, for which the complement $\overline{L}$ is not recursively enumerable.

Proof.

Let $M_1, M_2, M_3, \ldots$ be a recursive enumeration of all TMs.

Properties of Recursively Enumerable Languages

Theorem

The class of recursively enumerable languages is closed under

$$\cup \qquad \text{and} \qquad \cap$$

There exist recursively enumerable languages L, for which the complement $\overline{L}$ is not recursively enumerable.

Proof.

Let $M_1, M_2, M_3, \ldots$ be a recursive enumeration of all TMs.

Define $L = \{ a^i \mid a^i \in L(M_i), i \geq 1 \}$.

Properties of Recursively Enumerable Languages

Theorem

The class of recursively enumerable languages is closed under

$\cup$ and $\cap$

There exist recursively enumerable languages L, for which the complement $\overline{L}$ is not recursively enumerable.

Proof.

Let $M_1, M_2, M_3, \ldots$ be a recursive enumeration of all TMs.

Define $L = \{ a^i \mid a^i \in L(M_i), i \geq 1 \}$. L is recursively enumerable.

Properties of Recursively Enumerable Languages

Theorem

The class of recursively enumerable languages is closed under

$\cup$ and $\cap$

There exist recursively enumerable languages L, for which the complement $\overline{L}$ is not recursively enumerable.

Proof.

Let $M_1, M_2, M_3, \ldots$ be a recursive enumeration of all TMs.

Define $L = \{ a^i \mid a^i \in L(M_i), i \geq 1 \}$. L is recursively enumerable.

If $\overline{L}$ was recursively enumerable

Properties of Recursively Enumerable Languages

Theorem

The class of recursively enumerable languages is closed under

$\cup$ and $\cap$

There exist recursively enumerable languages L, for which the complement $\overline{L}$ is not recursively enumerable.

Proof.

Let $M_1, M_2, M_3, \ldots$ be a recursive enumeration of all TMs.

Define $L = \{ a^i \mid a^i \in L(M_i), i \geq 1 \}$. L is recursively enumerable.

If $\overline{L}$ was recursively enumerable: $\overline{L} = L(M_k)$ for some $k \geq 1$.

Properties of Recursively Enumerable Languages

Theorem

The class of recursively enumerable languages is closed under

$$\cup \qquad \text{and} \qquad \cap$$

There exist recursively enumerable languages L, for which the complement $\overline{L}$ is not recursively enumerable.

Proof.

Let $M_1, M_2, M_3, \ldots$ be a recursive enumeration of all TMs.

Define $L = \{ a^i \mid a^i \in L(M_i), i \geq 1 \}$. L is recursively enumerable.

If $\overline{L}$ was recursively enumerable: $\overline{L} = L(M_k)$ for some $k \geq 1$.

Then $a^k \in \overline{L} \iff a^k \in L(M_k)$

Properties of Recursively Enumerable Languages

Theorem

The class of recursively enumerable languages is closed under

$\cup$ and $\cap$

There exist recursively enumerable languages L, for which the complement $\overline{L}$ is not recursively enumerable.

Proof.

Let $M_1, M_2, M_3, \ldots$ be a recursive enumeration of all TMs.

Define $L = \{ a^i \mid a^i \in L(M_i), i \geq 1 \}$. L is recursively enumerable.

If $\overline{L}$ was recursively enumerable: $\overline{L} = L(M_k)$ for some $k \geq 1$.

Then $a^k \in \overline{L} \iff a^k \in L(M_k) \iff a^k \in L$.

Properties of Recursively Enumerable Languages

Theorem

The class of recursively enumerable languages is closed under

$$\cup \qquad \text{and} \qquad \cap$$

There exist recursively enumerable languages L, for which the complement $\overline{L}$ is not recursively enumerable.

Proof.

Let $M_1, M_2, M_3, \ldots$ be a recursive enumeration of all TMs.

Define $L = \{ a^i \mid a^i \in L(M_i), i \geq 1 \}$. L is recursively enumerable.

If $\overline{L}$ was recursively enumerable: $\overline{L} = L(M_k)$ for some $k \geq 1$.

Then $a^k \in \overline{L} \iff a^k \in L(M_k) \iff a^k \in L$.

Contradiction. Hence $\overline{L}$ is not recursively enumerable. $\square$

Recursive Languages

A language L is **recursive** if

- L is recursively enumerable, and
- $\overline{L}$ is recursively enumerable.

Recursive Languages

A language L is **recursive** if

- L is recursively enumerable, and
- $\overline{L}$ is recursively enumerable.

Not every recursively enumerable language is recursive!

Recursive Languages

A language L is **recursive** if

- L is recursively enumerable, and
- $\overline{L}$ is recursively enumerable.

Not every recursively enumerable language is recursive!

Proof.

See last slide. □

Recursive Languages

A language L is **recursive** if

- L is recursively enumerable, and
- $\overline{L}$ is recursively enumerable.

Recursive Languages

A language L is **recursive** if

- L is recursively enumerable, and
- $\overline{L}$ is recursively enumerable.

Theorem

A language L is recursive $\iff$ L is accepted by a deterministic TM M that reaches for every input a halting state.

Recursive Languages

A language L is **recursive** if

- L is recursively enumerable, and
- $\overline{L}$ is recursively enumerable.

Theorem

A language L is recursive $\iff$ L is accepted by a deterministic TM M that reaches for every input a halting state.

Proof.

($\Leftarrow$) We show that $\overline{L}$ is recursively enumerable.

Recursive Languages

A language L is **recursive** if

- L is recursively enumerable, and
- $\overline{L}$ is recursively enumerable.

Theorem

A language L is recursive $\iff$ L is accepted by a deterministic TM M that reaches for every input a halting state.

Proof.

($\Leftarrow$) We show that $\overline{L}$ is recursively enumerable.

From M we construct a Turing machine N as follows:

- Add fresh state q_f.
- From all non-accepting halting states, add transition to q_f.
- Make q_f the only final state.

Recursive Languages

A language L is **recursive** if

- L is recursively enumerable, and
- $\overline{L}$ is recursively enumerable.

Theorem

A language L is recursive $\iff$ L is accepted by a deterministic TM M that reaches for every input a halting state.

Proof.

($\Leftarrow$) We show that $\overline{L}$ is recursively enumerable.

From M we construct a Turing machine N as follows:

- Add fresh state q_f.
- From all non-accepting halting states, add transition to q_f.
- Make q_f the only final state.

Then $L(N) = \overline{L(M)}$, thus $L(M)$ is recursive. $\square$

Recursive Languages

A language L is **recursive** if

- L is recursively enumerable, and
- $\overline{L}$ is recursively enumerable.

Theorem

A language L is recursive $\iff$ L is accepted by a deterministic TM M that reaches for every input a halting state.

Proof.

($\Rightarrow$) L and $\overline{L}$ accepted by deterministic TMs M_1 and M_2.

Recursive Languages

A language L is **recursive** if

- L is recursively enumerable, and
- $\overline{L}$ is recursively enumerable.

Theorem

A language L is recursive $\iff$ L is accepted by a deterministic TM M that reaches for every input a halting state.

Proof.

($\Rightarrow$) L and $\overline{L}$ accepted by deterministic TMs M_1 and M_2.

Construct a TM M executes M_1 and M_2 in parallel:

- M accepts when M_1 accepts
- M has non-accepting halting state when M_2 accepts

Recursive Languages

A language L is **recursive** if

- L is recursively enumerable, and
- $\overline{L}$ is recursively enumerable.

Theorem

A language L is recursive $\iff$ L is accepted by a deterministic TM M that reaches for every input a halting state.

Proof.

($\Rightarrow$) L and $\overline{L}$ accepted by deterministic TMs M_1 and M_2.

Construct a TM M executes M_1 and M_2 in parallel:

- M accepts when M_1 accepts
- M has non-accepting halting state when M_2 accepts

Then $L(M) = L(M_1) = L$, and M halts for every input. $\square$

Properties of Recursive Languages

Theorem

If L, L_1 and L_2 are recursive, then so are

$$L_1 \cup L_2 \qquad L_1 \cap L_2 \qquad \overline{L} \qquad L^* \qquad L_1 L_2 \qquad L_1 \setminus L_2$$

Properties of Recursive Languages

Theorem

If L, L_1 and L_2 are recursive, then so are

$$L_1 \cup L_2 \qquad L_1 \cap L_2 \qquad \overline{L} \qquad L^* \qquad L_1 L_2 \qquad L_1 \setminus L_2$$

Proof.

- $L_1 \cup L_2$, $\overline{L}$, $L_1 L_2$, L^*: use (non)deterministic TMs
- $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$
- $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$

Context-Sensitive Languages are Recursive

Theorem

Context-sensitive langauges are recursive.

Context-Sensitive Languages are Recursive

Theorem

Context-sensitive langauges are recursive.

Proof.

Let G be a context-sensitive grammar.

Context-Sensitive Languages are Recursive

Theorem

Context-sensitive langauges are recursive.

Proof.

Let G be a context-sensitive grammar.

We argue that there exists a Turing machine M accepting $L(G)$.

Context-Sensitive Languages are Recursive

Theorem

Context-sensitive langauges are recursive.

Proof.

Let G be a context-sensitive grammar.

We argue that there exists a Turing machine M accepting $L(G)$.

Let $w \in T^*$ be the input word.

Context-Sensitive Languages are Recursive

Theorem

Context-sensitive langauges are recursive.

Proof.

Let G be a context-sensitive grammar.

We argue that there exists a Turing machine M accepting $L(G)$.

Let $w \in T^*$ be the input word.

The are finitely words over $V \cup T$ of length $\leq |w|$:

Context-Sensitive Languages are Recursive

Theorem

Context-sensitive langauges are recursive.

Proof.

Let G be a context-sensitive grammar.

We argue that there exists a Turing machine M accepting $L(G)$.

Let $w \in T^*$ be the input word.

The are finitely words over $V \cup T$ of length $\leq |w|$:

- M can compute the set of all words u with

$$S \Rightarrow^* u \qquad \text{and} \qquad |u| \leq |w|$$

Context-Sensitive Languages are Recursive

Theorem

Context-sensitive langauges are recursive.

Proof.

Let G be a context-sensitive grammar.

We argue that there exists a Turing machine M accepting $L(G)$.

Let $w \in T^*$ be the input word.

The are finitely words over $V \cup T$ of length $\leq |w|$:

- M can compute the set of all words u with

$$S \Rightarrow^* u \qquad \text{and} \qquad |u| \leq |w|$$

- M accepts w if w is among these words.
  (Otherwise M halts in a non-accepting state.)

Context-Sensitive Languages are Recursive

Theorem

Context-sensitive langauges are recursive.

Proof.

Let G be a context-sensitive grammar.

We argue that there exists a Turing machine M accepting $L(G)$.

Let $w \in T^*$ be the input word.

The are finitely words over $V \cup T$ of length $\leq |w|$:

- M can compute the set of all words u with

$$S \Rightarrow^* u \qquad \text{and} \qquad |u| \leq |w|$$

- M accepts w if w is among these words.
  (Otherwise M halts in a non-accepting state.)

Then M accepts $L(G)$ and always reaches a halting state. □

Context-Sensitive versus Recursive Languages

Theorem

Not every recursive language is context-sensitive.

Context-Sensitive versus Recursive Languages

Theorem

Not every recursive language is context-sensitive.

Proof.

$\Sigma = \{0, 1\}$. There exists an **injective, computable function**

$$h : \{\, G \mid G \text{ context-sensitive} \,\} \to \{\, 0, 1 \,\}^*$$

such that the **image** of h is **recursive**. For example:

$$h(0) = 010 \qquad h(\to) = 01110 \qquad h(A_i) = 01^{i+4}0$$
$$h(1) = 0110 \qquad h(;) = 011110$$

Context-Sensitive versus Recursive Languages

Theorem

Not every recursive language is context-sensitive.

Proof.

$\Sigma = \{0, 1\}$. There exists an **injective, computable function**

$$h : \{\, G \mid G \text{ context-sensitive} \,\} \to \{\, 0, 1 \,\}^*$$

such that the **image** of h is **recursive**. For example:

$$h(0) = 010 \qquad h(\to) = 01110 \qquad h(A_i) = 01^{i+4}0$$
$$h(1) = 0110 \qquad h(;) = 011110$$

Define $L = \{\, h(G) \mid G \text{ context-sensitive } \wedge\ h(G) \notin L(G) \,\}$.

Context-Sensitive versus Recursive Languages

Theorem

Not every recursive language is context-sensitive.

Proof.

$\Sigma = \{0, 1\}$. There exists an **injective, computable function**

$$h : \{\, G \mid G \text{ context-sensitive} \,\} \to \{\, 0, 1 \,\}^*$$

such that the **image** of h is **recursive**. For example:

$$h(0) = 010 \qquad h(\to) = 01110 \qquad h(A_i) = 01^{i+4}0$$
$$h(1) = 0110 \qquad h(;) = 011110$$

Define $L = \{\, h(G) \mid G \text{ context-sensitive } \wedge\ h(G) \notin L(G) \,\}$.
Then L is recursive (by the above assumptions on h).

Context-Sensitive versus Recursive Languages

Theorem

Not every recursive language is context-sensitive.

Proof.

$\Sigma = \{0, 1\}$. There exists an **injective, computable function**

$$h : \{\, G \mid G \text{ context-sensitive} \,\} \to \{\, 0, 1 \,\}^*$$

such that the **image** of h is **recursive**. For example:

$$h(0) = 010 \qquad h(\to) = 01110 \qquad h(A_i) = 01^{i+4}0$$
$$h(1) = 0110 \qquad h(;) = 011110$$

Define $L = \{\, h(G) \mid G \text{ context-sensitive } \wedge\ h(G) \notin L(G) \,\}$.
Then L is recursive (by the above assumptions on h).

Assume $L = L(G_0)$ for a context-free grammar G_0.

Context-Sensitive versus Recursive Languages

Theorem

Not every recursive language is context-sensitive.

Proof.

$\Sigma = \{0, 1\}$. There exists an **injective, computable function**

$$h : \{\, G \mid G \text{ context-sensitive} \,\} \to \{\, 0, 1 \,\}^*$$

such that the **image** of h is **recursive**. For example:

$$h(0) = 010 \qquad h(\to) = 01110 \qquad h(A_i) = 01^{i+4}0$$
$$h(1) = 0110 \qquad h(;) = 011110$$

Define $L = \{\, h(G) \mid G \text{ context-sensitive } \wedge\ h(G) \notin L(G) \,\}$.
Then L is recursive (by the above assumptions on h).

Assume $L = L(G_0)$ for a context-free grammar G_0. Then

$$h(G_0) \in L \iff h(G_0) \notin L(G_0) \iff h(G_0) \notin L$$

Context-Sensitive versus Recursive Languages

Theorem

Not every recursive language is context-sensitive.

Proof.

$\Sigma = \{0, 1\}$. There exists an **injective, computable function**

$$h : \{\, G \mid G \text{ context-sensitive} \,\} \to \{\, 0, 1 \,\}^*$$

such that the **image** of h is **recursive**. For example:

$$h(0) = 010 \qquad h(\to) = 01110 \qquad h(A_i) = 01^{i+4}0$$
$$h(1) = 0110 \qquad h(;) = 011110$$

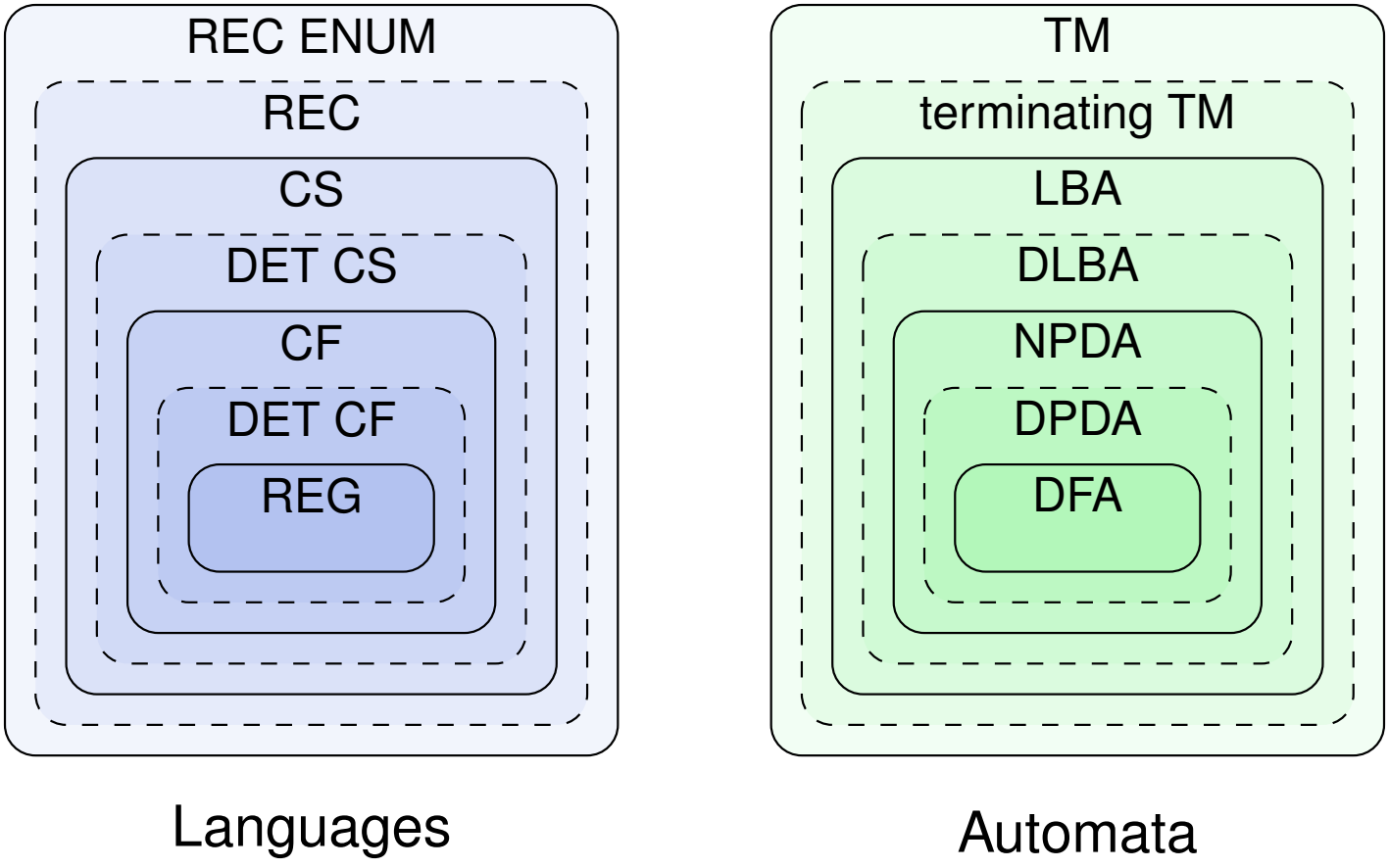
Define $L = \{\, h(G) \mid G \text{ context-sensitive } \wedge\ h(G) \notin L(G) \,\}$.
Then L is recursive (by the above assumptions on h).

Assume $L = L(G_0)$ for a context-free grammar G_0. Then

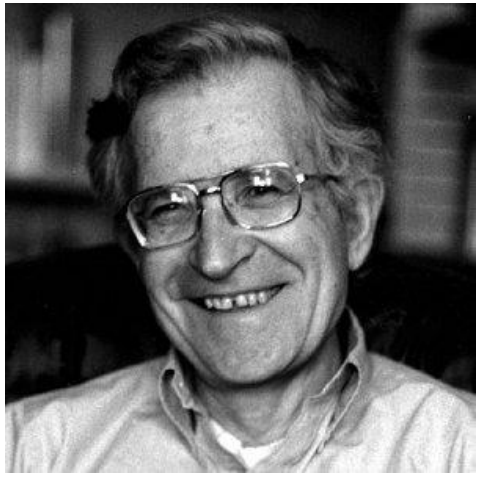
$$h(G_0) \in L \iff h(G_0) \notin L(G_0) \iff h(G_0) \notin L$$

Contradiction! □

Chomsky Hierarchy



Noam Chomsky



Noam Chomsky (born 1928) is a world famous

- language expert,
- philosopher, and
- political activist.

Turing Machines are Countable, Languages not

Theorem

There are countably many TMs over an input alphabet Σ.

Theorem

There are uncountable many languages over Σ.

Turing Machines are Countable, Languages not

Theorem

There are countably many TMs over an input alphabet Σ.

Theorem

There are uncountable many languages over Σ.

Proof.

Let $a \in \Sigma$.

Assume $L_0, L_1, L_2, \ldots$ is enumeration of all languages over $\{a\}$.

Turing Machines are Countable, Languages not

Theorem

There are countably many TMs over an input alphabet Σ.

Theorem

There are uncountable many languages over Σ.

Proof.

Let $a \in \Sigma$.

Assume $L_0, L_1, L_2, \ldots$ is enumeration of all languages over $\{a\}$.

Define a language L as follows: for every $i \geq 0$.

$$a^i \in L \iff a^i \notin L_i$$

Turing Machines are Countable, Languages not

Theorem

There are countably many TMs over an input alphabet Σ.

Theorem

There are uncountable many languages over Σ.

Proof.

Let $a \in \Sigma$.

Assume $L_0, L_1, L_2, \ldots$ is enumeration of all languages over $\{a\}$.

Define a language L as follows: for every $i \geq 0$.

$$a^i \in L \iff a^i \notin L_i$$

Then for every $i \geq 0$, we have $L \neq L_i$.

Turing Machines are Countable, Languages not

Theorem

There are countably many TMs over an input alphabet Σ.

Theorem

There are uncountable many languages over Σ.

Proof.

Let $a \in \Sigma$.

Assume $L_0, L_1, L_2, \ldots$ is enumeration of all languages over $\{a\}$.

Define a language L as follows: for every $i \geq 0$.

$$a^i \in L \iff a^i \notin L_i$$

Then for every $i \geq 0$, we have $L \neq L_i$.

Thus L is **not** part of the above enumeration. Contradiction. $\square$

Turing Machines are Countable, Languages not

Theorem

There are countably many TMs over an input alphabet Σ.

Theorem

There are uncountable many languages over Σ.

Proof.

Let $a \in \Sigma$.

Assume $L_0, L_1, L_2, \ldots$ is enumeration of all languages over $\{a\}$.

Define a language L as follows: for every $i \geq 0$.

$$a^i \in L \iff a^i \notin L_i$$

Then for every $i \geq 0$, we have $L \neq L_i$.

Thus L is **not** part of the above enumeration. Contradiction. $\square$

Conclusion: not all languages are recursively enumerable.

Looking Back

- Extensions of TMs do not add expressivity
- Unrestricted grammars are equivalent to TMs
- Recursively enumerable languages are described by TMs
- Recursive languages: TM always reaches a halting state
- Linear bounded automaton (LBA) is a TM without $\square$
- Context-sensitive grammars are equivalent to LBA's
- Chomsky hierarchy

Looking Forward

Read:

- Linz 10.2–10.3, 10.5, 11.1–11.4

Do the following exercises:

- Linz 11.1: 3, 6, 7, 8, 10
- Linz 11.2: 6, 7
- Linz 11.3: 1b,d, 3
- Linz 10.5: 4c,d

Following lecture:

- Decidability and undecidability
- Halting problem
- Theorem of Rice