Recursive and Recursively Enumerable Languages. Properties of Recursive and Recursively Enumerable Languages.

+

Decidability and Undecidability, Halting Problem, Rice's Theorem, Greibach's Theorem, Post Correspondence Problem, Context Sensitivity and Linear Bound Automata.

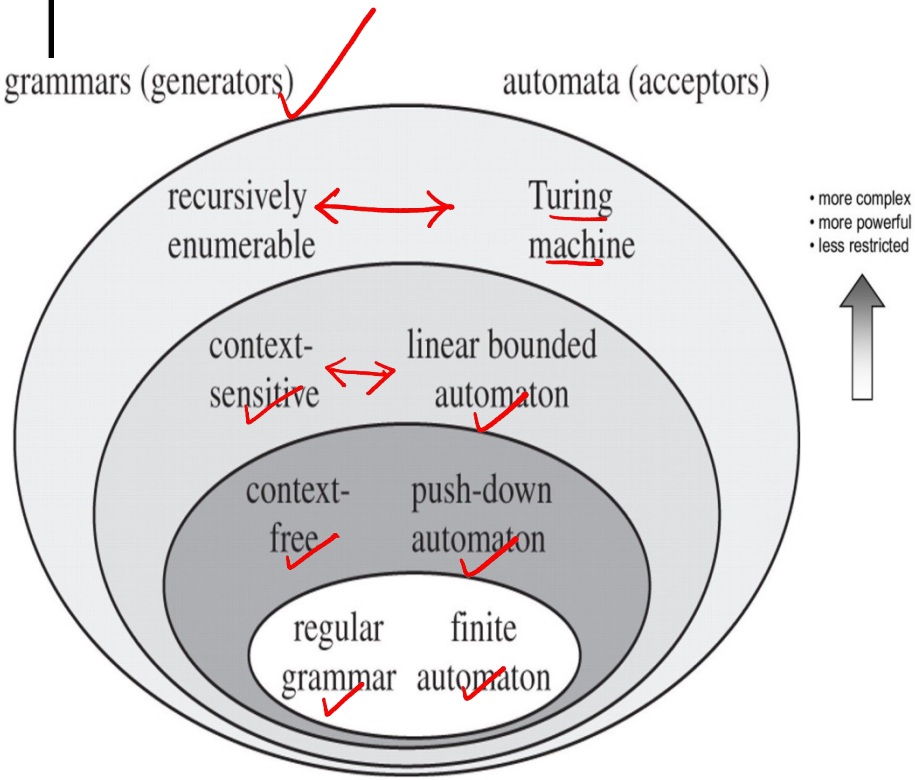
Module 6.1 and 6.2

Turing Machine

- Turing Machine was invented by Alan Turing in 1936
- Accepts Recursive Enumerable Languages (generated by Type-0 Grammar).

The Hierarchy

Class	Grammars	Languages	Automaton
Type-0	Unrestricted	Recursive Enumerable	Turing Machine
Type-1	Context Sensitive	Context Sensitive	Linear-Bound
Type-2	Context Free	Context Free	Pushdown
Type-3	Regular	Regular	Finite



Recursive Enumerable (RE) or Type -0 Language

- RE languages or type-0 languages are generated by type-0 grammars.

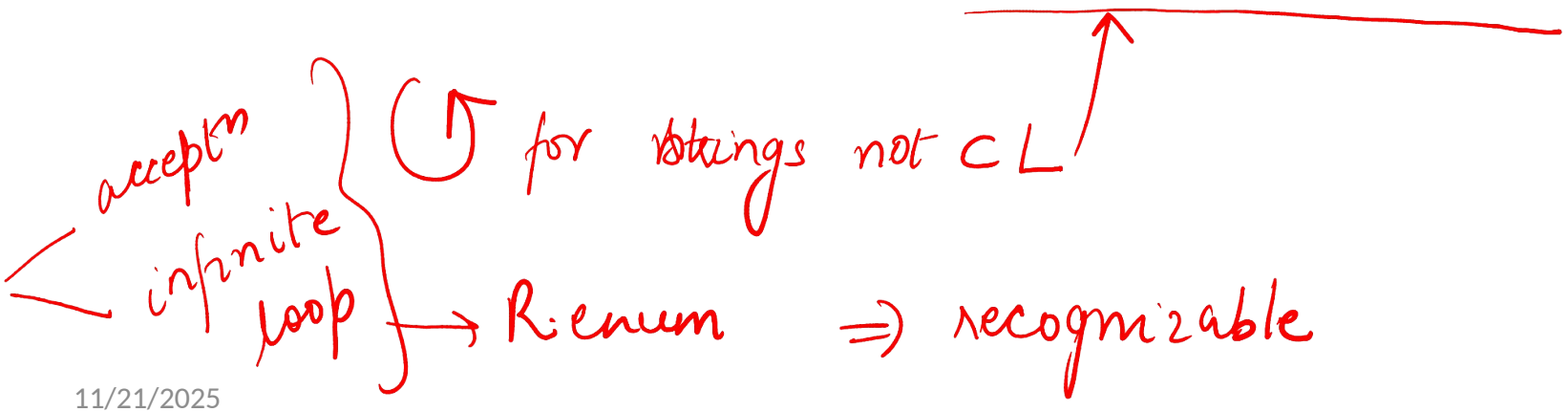
RE
REC

The Hierarchy

Class	Grammars	Languages	Automaton
Type-0	Unrestricted	Recursive Enumerable	Turing Machine
Type-1	Context Sensitive	Context Sensitive	Linear-Bound
Type-2	Context Free	Context Free	Pushdown
Type-3	Regular	Regular	Finite

Recursive Enumerable (RE) or Type -0 Language

- An RE language can be **accepted or recognized** by Turing machine which means :
- **It will enter into final state for the strings of language and may or may not enter into rejecting state for the strings which are not part of the language.**
- It means TM can **loop forever** for the strings which are not a part of the language.
- RE languages are also called as **Turing recognizable languages.**



Recursive Language (REC)

- A recursive language (subset of RE) can be decided by Turing machine which means :
- **It will enter into final state for the strings of language and rejecting state for the strings which are not part of the language.**

E.g.

L= $\{a^n b^n c^n | n>=1\}$ is recursive

- We can construct a turing machine which will move to final state if the string is of the form $a^n b^n c^n$ else move to non-final state.
- So the TM will **always halt** in this case.
- REC languages are also called as **Turing decidable languages.**

accepting
rejecting
} Recursive ⇒ decidable

Some Confusing Terms

Language Decided

- We say a TM decide a language *L*
- if it accepts all strings in *L* and rejects all strings not in *L*.

Language Recognized

- We say a TM recognizes language L if it accepts all strings in L. It may or may not reject any string not in L
- i.e., it might go into an infinite loop in case of non-acceptance. (It will never accept a string not in L)

Nomenclature

- **Recursive Enumerable (**RE) languages =**Turing recognizable languages.**
- **Recursive (**REC) languages = **Turing decidable languages.**

Recursive Language (REC)

- The relationship between RE and REC languages can be shown in Figure 1.

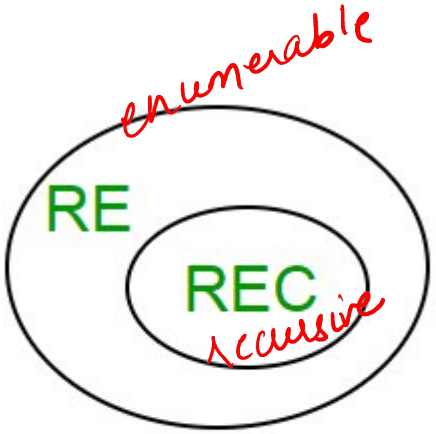


Figure 1

Closure Properties of Recursive Languages

Union:

- If L1 and If L2 are two recursive languages, their union L1∪L2 will also be recursive because if TM halts for L1 and halts for L2, it will also halt for L1∪L2.

Closure Properties of Recursive Languages

Concatenation:

- If L1 and If L2 are two recursive languages, their concatenation L1.L2 will also be recursive.

For Example:

- L1= $\{a^n b^n c^n | n>=0\}$
- L2= $\{d^m e^m f^m | m>=0\}$
- L3= L1.L2 = $\{a^n b^n c^n d^m e^m f^m | m>=0$ and $n>=0\}$ is also recursive.
- L1 says n no. of a's followed by n no. of b's followed by n no. of c's.
- L2 says m no. of d's followed by m no. of e's followed by m no. of f's.
- Their concatenation first matches no. of a's, b's and c's and then matches no. of d's, e's and f's. So it can be decided by TM.

Closure Properties of Recursive Languages

Kleene Closure:

- If L1 is recursive, its Kleene closure $L1^*$ will also be recursive.

For Example:

- $L1 = \{a^n b^n c^n \mid n>=0\}$
- $L1^* = \{a^n b^n c^n \mid \mid n>=0\}^*$ is also recursive.

$(a^n b^n)^*$ $n=2$

aabbaabb

Closure Properties of Recursive Languages

Intersection :

- If L1 and If L2 are two recursive languages, their intersection L1 ∩ L2 will also be recursive.

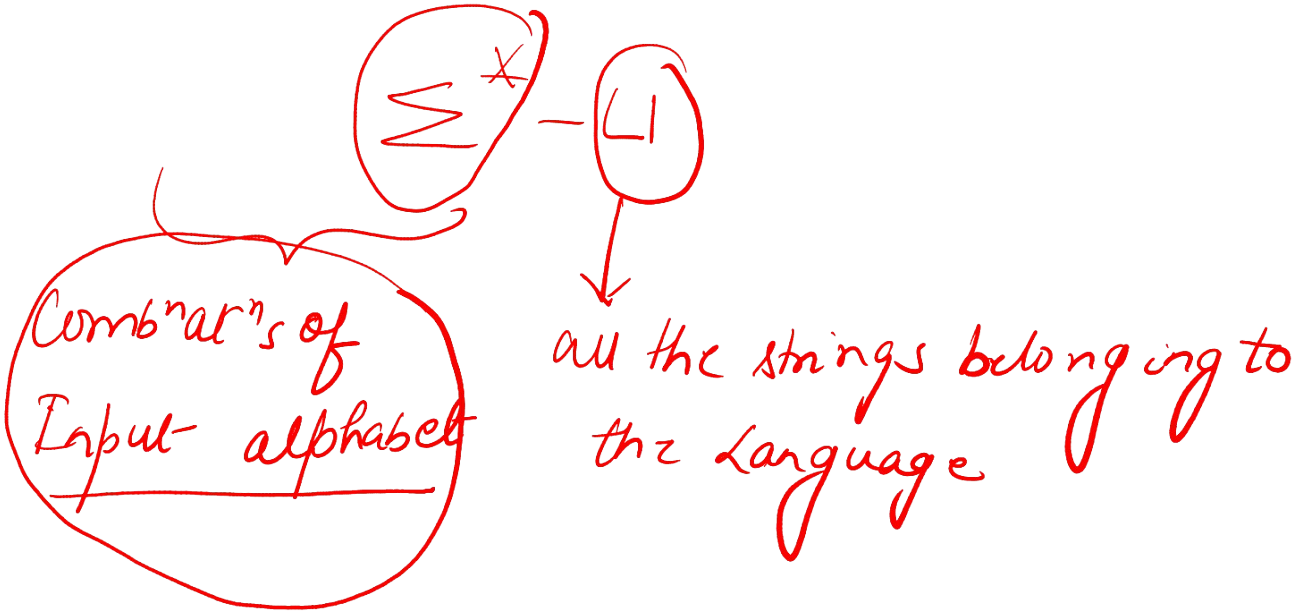
Example:

- L1= $\{a^n b^n c^n d^m \mid n>=0 \text{ and } m>=0\}$
- L2= $\{a^n b^n c^n d^n \mid n>=0 \text{ and } m>=0\}$
- L3=L1 ∩ L2 = $\{a^n b^n c^n d^n \mid n>=0\}$ will be recursive.
- L1 says n no. of a's followed by n no. of b's followed by n no. of c's and then any no. of d's.
- L2 says any no. of a's followed by n no. of b's followed by n no. of c's followed by n no. of d's.
- Their intersection says n no. of a's followed by n no. of b's followed by n no. of c's followed by n no. of d's.
- So it can be decided by turing machine, hence recursive.

Closure Properties of Recursive Languages

Complement:

- Similarly, complement of recursive language L1 which is $\sum^*$ - L1, will also be recursive.



Closure Properties of Recursive Enumerable Languages

Complement:

- Note: As opposed to REC languages, RE languages are not closed under complementation which means complement of RE language need not be RE

Decidability and Undecidability

Decidable language –

- A decision problem P is said to be decidable (i.e., have an algorithm) if the language L of all yes instances to P is decidable.

Example-

(I) (Acceptance problem for DFA) Given a DFA does it accept a given word?

(II) (Emptiness problem for DFA) Given a DFA does it accept any word?

(III) (Equivalence problem for DFA) Given two DFAs, do they accept the same language?

https://www.geeksforgeeks.org/decidability-and-undecidability-in-toc/

Decidability and Undecidability

Undecidable language –

- A decision problem P is said to be undecidable if the language L of all yes instances to P is not decidable

or

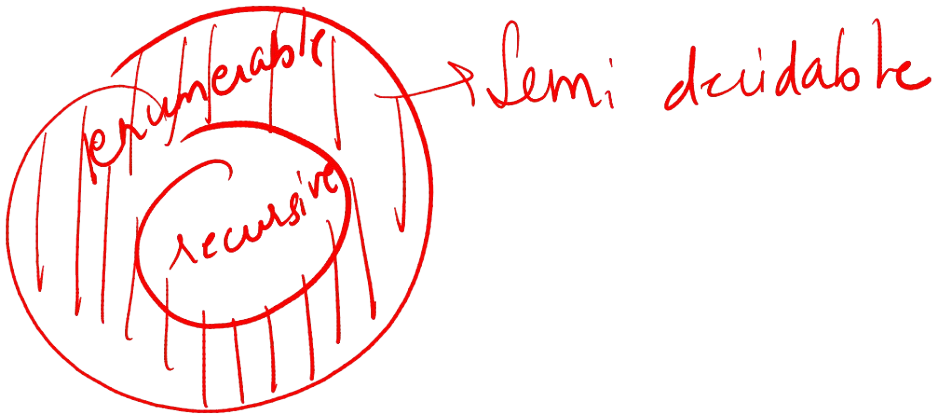
- A language is undecidable if it is not decidable.

- An undecidable language maybe a partially decidable language but not decidable.
- If a language is not even partially decidable , then there exists no Turing machine for that language.

Decidability and Undecidability

Partially decidable or Semi-Decidable Language –

- A decision problem P is said to be semi-decidable (i.e., have a semi-algorithm) if the language L of all yes instances to P is RE but not REC language.



The Halting problem

- Given a program/algorithm will ever halt or not?

The Halting problem

- Given a program/algorithm will ever halt or not?
- Halting means that the program on certain input will
  - accept it and halt or
  - reject it and halt and
  - it would never go into an infinite loop.
- Basically halting means terminating.

The Halting problem

- Can we have an algorithm that will tell that the given program will halt or not?
- In terms of Turing machine, will it terminate when run on some machine with some particular given input string.

The Halting problem

- The answer is no
- We cannot design a generalized algorithm which can appropriately say that given a program will ever halt or not?
- The only way is to run the program and check whether it halts or not.

The Halting problem

- We can refrain the halting problem question in such a way also:
- Given a program written in some programming language(c/c++/java) will it ever get into an infinite loop(loop never stops) or will it always terminate(halt)?
- This is an undecidable problem because we cannot have an algorithm which will tell us whether a given program will halt or not in a generalized way i.e by having specific program/algorithm.
- The best possible way is to run the program and see whether it halts or not.

Nomenclature revisited

- **Recursive Enumerable (**RE) languages =**Turing recognizable languages.**
- **Recursive (**REC) languages = **Turing decidable languages.**

The Rice Theorem

Part 1 (For some undecidable languages)

- Any non-trivial property of the LANGUAGE recognizable by a Turing machine (recursively enumerable language) is undecidable

Part 2 (For some unrecognizable language)

- Any non-monotonic property of the LANGUAGE recognizable by a Turing machine (recursively enumerable language) is unrecognizable

https://gatecse.in/rices-theorem/

The Rice Theorem

- A property is called to be **trivial** if either it is not satisfied by any recursively enumerable languages, or if it is satisfied by all recursively enumerable languages.

- A **non-trivial** property is satisfied by some recursively enumerable languages and are not satisfied by others.

https://www.tutorialspoint.com/automata_theory/rice_theorem.htm

The Rice Theorem

- For a property of recursively enumerable set to be **non-trivial**, there should exist at least two recursively enumerable languages, (hence two Turing machines),
- The property holding for one (*Tyes* being its TM) and not holding for the other (*Tno* being its TM).

https://gatecse.in/rices-theorem/

The Rice Theorem

- For a property of recursively enumerable set to be **non-monotonic,** there should exist at least two recursively enumerable languages (hence two Turing machines),
- The property holding for one (Tyes being its TM) and not holding for the other (Tno being its TM) and
- The property holding set (language of Tyes) must be a proper subset of the set not having the property (language of Tno).

$T_{yes} \subset T_{no}$

https://www.tutorialspoint.com/automata_theory/rice_theorem.htm

Post Correspondence Problem

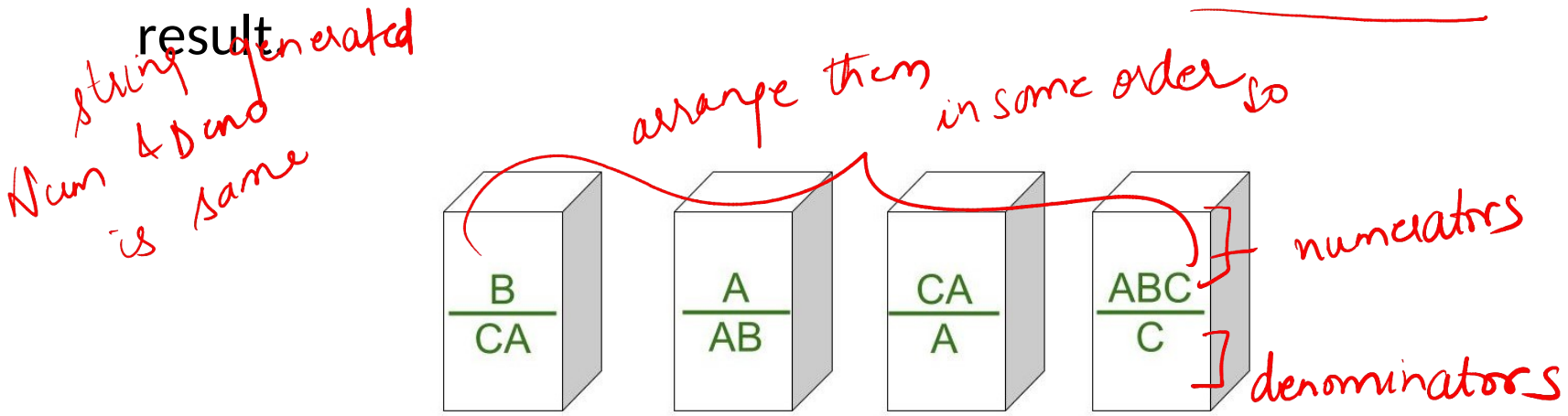
- A popular undecidable problem
- Introduced by Emil Leon Post in 1946.
- It is simpler than Halting Problem.

https://www.geeksforgeeks.org/post-correspondence-problem/

Post Correspondence Problem

- In this problem we have N number of **Dominos** (tiles).
- The aim is to arrange tiles in such order that string made by Numerators is same as string made by Denominators.

- In simple words, lets assume we have two lists both containing N words, aim is to find out concatenation of these words in some sequence such that both lists yield same result.



Post Correspondence Problem

- Let's try understanding this by taking two **lists A** and **B**

A=[aa, bb, abb]

B=[aab, ba, b]

aabb aa abb S1

aab ba aab b S2

S1 = S2

- Now for sequence 1, 2, 1, 3

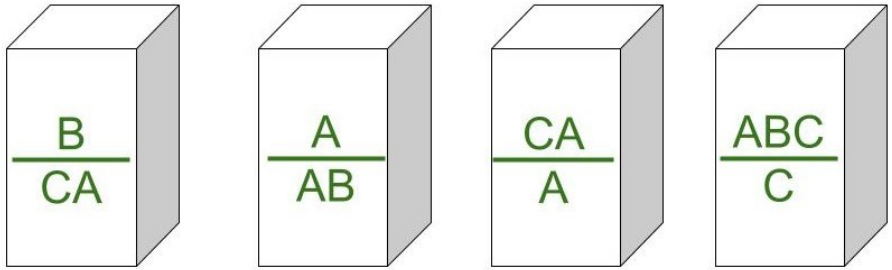
- First list will yield aabbaaabb
- Second list will yield same string aabbaaabb.
- So the solution to this PCP becomes 1, 2, 1, 3.

https://www.geeksforgeeks.org/post-correspondence-problem/

Post Correspondence Problem

- Post Correspondence Problems can be represented in two ways:

1. Domino's Form :



tiles

2. Table Form :

table

	Numerator	Denominator
1	B	CA
2	A	AB
3	CA	A
4	ABC	C

Formal Definition of Context Sensitive Grammar

- A context sensitive grammar G = (N, Σ, P, S), where
- V is a set of non-terminal symbols
- Σ is a set of terminal symbols
- S is the start symbol, and
- P is a set of production rules, of the form $\alpha \boldsymbol{A} \beta \rightarrow \alpha \boldsymbol{\gamma} \beta$, where A in V, $\alpha, \beta \in (V \cup \Sigma)$ and $\gamma \in (V \cup \Sigma)^{+}$

Formal Definition of Context Sensitive Grammar

- P is a set of production rules, of the form $\alpha \boldsymbol{A} \beta \rightarrow \alpha \boldsymbol{\gamma} \beta$, where A in V, α, $\beta \in (V \cup \Sigma)$ and $\gamma \in (V \cup \Sigma)^{+}$
- With γ nonempty
- To derive the empty string, $S \rightarrow \Lambda$ must also be included in the productions, with *S* not occurring on the right-hand side of any production.

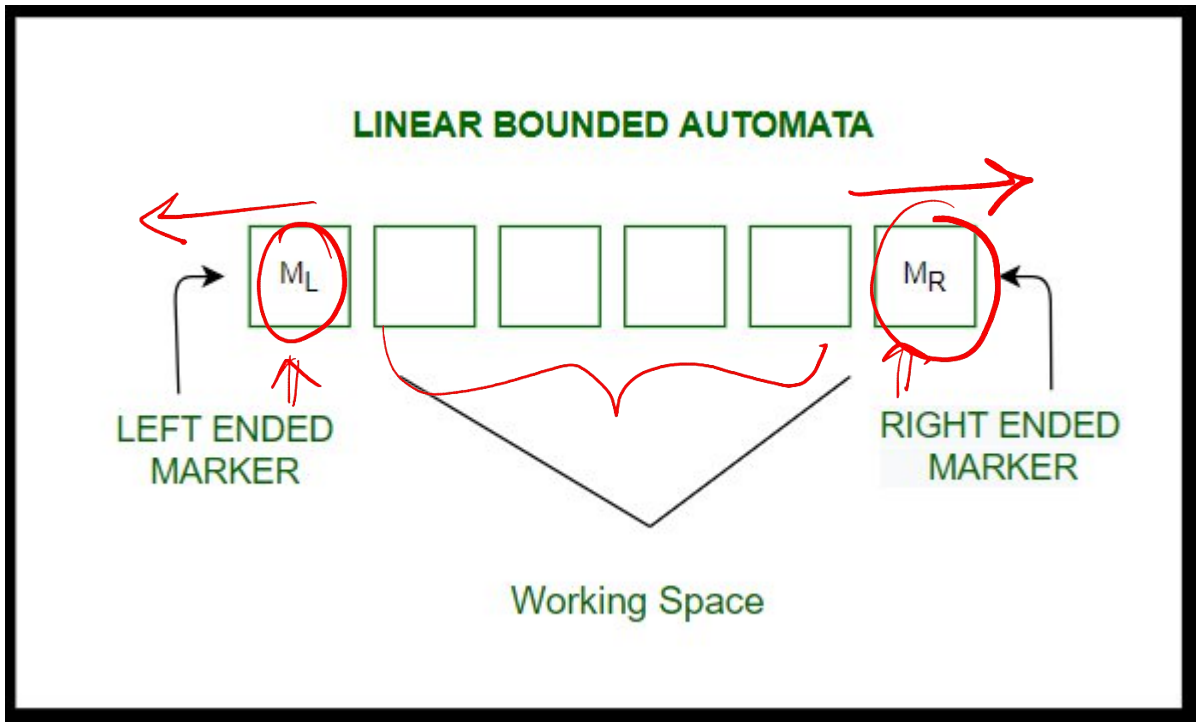
Formal Definition of Context Sensitive Grammar

- During derivation non-terminal ***A*** will be changed to **γ** only when it is present in the context of **α** and **β.**
- $\alpha A \beta \rightarrow \alpha \gamma \beta$

Linear Bound Automata

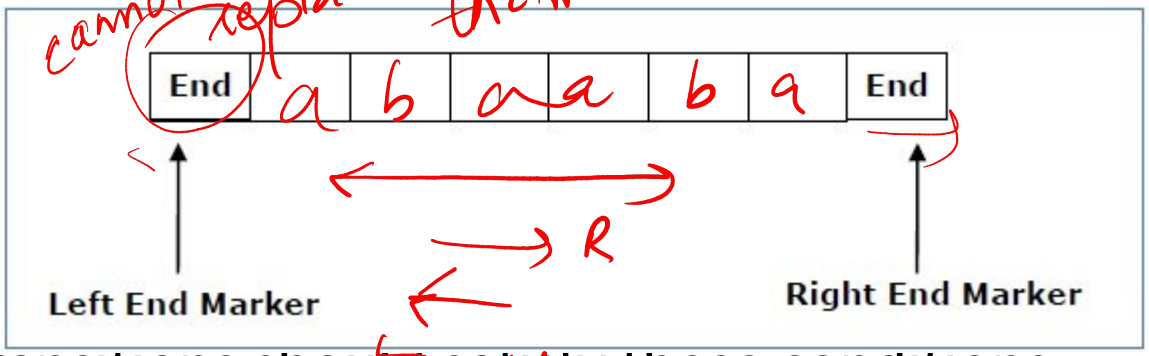
Turing ⇒ tape ⇒ unlimited size ⇒ extendable on both ends
↳ infinite size

- Linear Bounded Automata (LBA) is a single tape Turing Machine with two special tape symbols call them left marker **<** and the right marker **>.**



Linear Bound Automata

- The transitions neither move to the left of the left end marker nor to the right of the right end marker of the tape.



- The transitions should satisfy these conditions:
  - It should not replace the marker symbols by any other symbol.
  - It should not write on cells beyond the marker symbols.

Formal Definition of Linear Bound Automata

- Formally Linear Bounded Automata is a non-deterministic Turing Machine , M = (Q, Σ, Γ, δ, q0, ML, MR, F)
- Q is set of all states
- Σ is set of all terminals
- Γ is set of all tape symbols, Σ ⊂ Γ
- δ is set of transitions
- q0 is the initial state
- ML is left marker = Left bound of tape
- MR is right marker =Right bound of tape where MR !=ML
- F is finite set of final states

Formal Definition of Linear Bound Automata

- **δ** is a transition function which maps each pair (state, tape symbol) to (state, tape symbol, {L,R,N})

https://www.tutorialspoint.com/automata_theory/linear_bounded_automata.htm

Greibach's theorem

- In theoretical computer science, in particular in formal language theory, Greibach's theorem states that certain properties of formal language classes are undecidable.
- It is named after the computer scientist Sheila Greibach, who first proved it in 1963.

https://www.tutorialspoint.com/automata_theory/linear_bounded_automata.htm

Greibach's theorem

- Formal statement of the theorem

- Considers a set C of formal languages over an alphabet $\Sigma \cup \{\#\}$ such that
  - each language in C has a finite description,
  - each regular language over $\Sigma \cup \{\#\}$ is in C
  - given descriptions of languages L1, L2 $\in$ C and of a regular language R $\in$ C, a description of the products L1R and RL1, and of the union $L1 \cup L2$ can be effectively computed, and
  - it is undecidable for any member language L $\in$ C with $L \subseteq \Sigma^*$ whether $L = \Sigma^*$.

https://www.tutorialspoint.com/automata_theory/linear_bounded_automata.htm

Greibach's theorem

- Formal statement of the theorem

- Let P be any nontrivial subset of C that contains all regular sets over Σ∪{#} and is closed under quotient by each single symbol in Σ∪{#}.
- Then the question whether L ∈ P for a given description of a language L ∈ C is undecidable.

https://www.tutorialspoint.com/automata_theory/linear_bounded_automata.htm