Scout
A film by Moorebot.
digital trends
Meet Scout: A tiny, waterproof, autonomous robot that can wander
Moorebot

Regular Grammar in Discrete Mathematics

The regular languages can be generated by regular grammar. In regular grammar, the left-hand side always consists of a single non-terminal. The left side cannot have more than one non-terminal or any terminal variable. There can be a single terminal or a single terminal followed by a non-terminal on the right-hand side.

The production of regular grammar will be in the following form:

> X → xY
>
> X → x
>
> X → Yx

Where

X and Y are used to indicate the Variable (V).

'x' is used to indicate the string of terminals (T*)

Types of Regular Grammar

There are two types of regular grammar, which are described as follows:

- Left Linear grammar (LLG)
- Right Linear Grammar (RLG)

Left Linear Grammar

The production must be in the following form in the left linear

> X → xY

Scout
A film by Moorebot.
digital trends
Meet Scout: A tiny, waterproof, autonomous robot that can wander
Moorebot

X → x

Where

X, Y belongs to a variable (V), and 'x' belongs to T*.

Right Linear Grammar

The production must be in the following form in the right linear grammar.

X → Yx

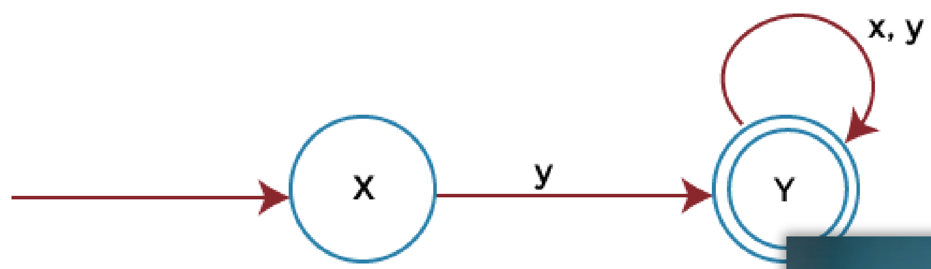
X → x

Where

X, Y belongs to a variable (V), and 'x' belongs to T*.

The regular language can be described as a language that is generated by the type 3 grammar and for which finite automata can be designed. It is able to convert FA into type 3 grammar.

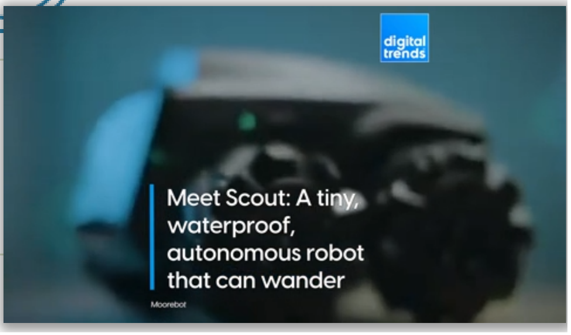
Example: In this example, we will show the finite automata accepting a string that begins with the symbol y.



∑ = {x, y}

Initial State (q0) = X

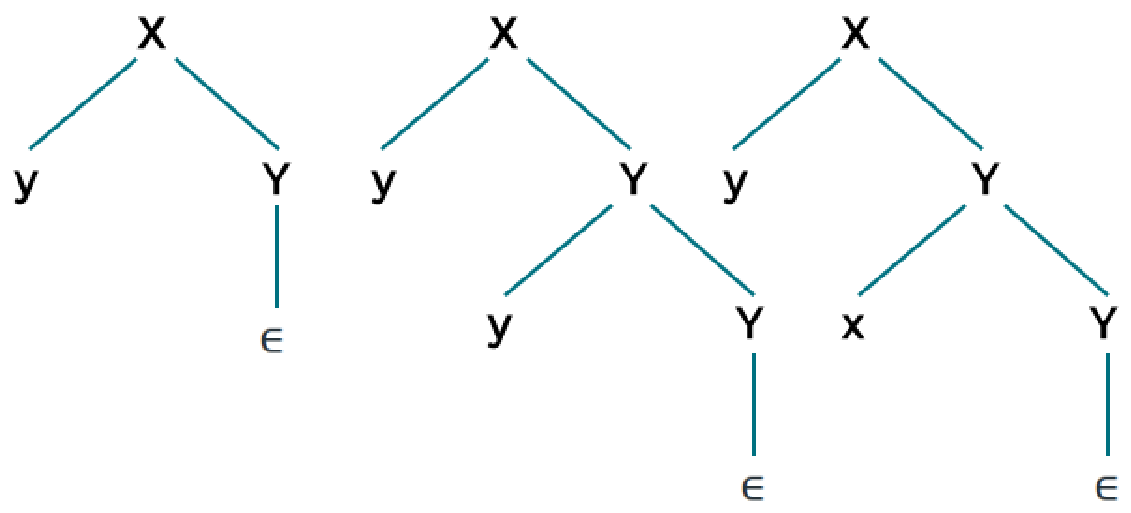
Final State (F) = Y


digital trends
Meet Scout: A tiny, waterproof, autonomous robot that can wander

The right linear grammar corresponding to FA is described as follows:

> X → yY
> Y → ∈/xY/yY

As we can see that the above grammar is right-linear grammar. This grammar can be written directly through finite automata.



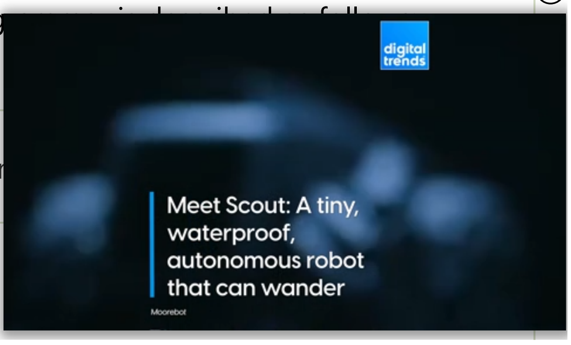
This grammar derives strings which are stating with B

A string can be generated with the help of above RLG, which begins with b. After that b, this string can accept any input symbol (i.e., ∑ = {x, y}).

> The regular language which corresponds to the right linear g
> L = {y, yx, yy, yxx, yxy, yyx, yyy ......}

We will get the following grammar if we reverse the productio

> X → Yy


digital trends
Meet Scout: A tiny, waterproof, autonomous robot that can wander
Moorebot

> Y → ∈/Yx/Yy

This grammar derives the language that has all the strings that end with b, which is described as follows:

> L' = {y, yy, xy, xxy, yxy, xyy, yyy ....}

In conclusion, we can say that suppose we have finite automata which represent language L, and we convert it into the right linear grammar, which also represents the same language L. In this case, when we reverse the RLG (right linear grammar), we will get the left linear grammar representing language L', which is the reverse of L.

Conversion of RLG into LLG

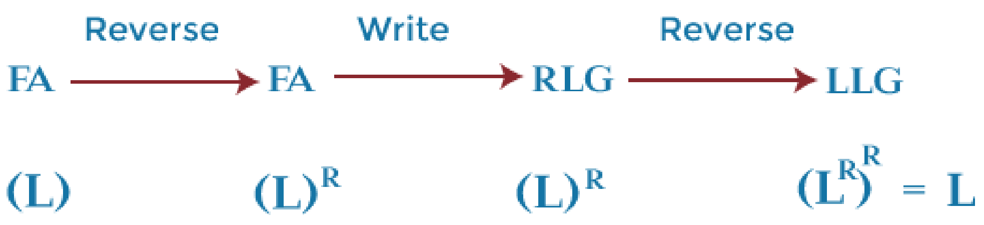
The following steps will be used when we convert the right linear grammar into the left linear grammar for language L.

Step 1: For language L, we have to reverse the FA (Finite automata).

Step 2: After that, we have to write the right linear grammar for it.

Step 3: Now, we will reverse the RLG.

After step 3, we will get the grammar that generates the language, which is able to represent the left linear grammar for the same language L.

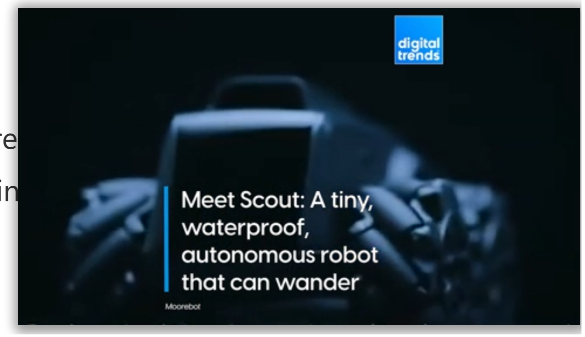


This represents the same procedure as above for converting RLG to LLG

Here L is used to describe the language for FA (finite automata), and L^R is used to describe the reversal of language L.

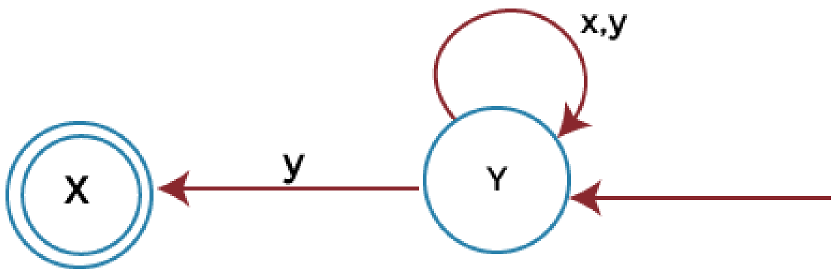
Example:

In this example, we will use the above finite automata repre
string that begins with the symbol y. This language contains in
convert this automata into the left linear grammar.


digital trends
Meet Scout: A tiny, waterproof, autonomous robot that can wander
Moorebot

Solution:

Step 1: The reversal of finite automata is described as follows:



The reversal of FA that represent all strings starting with b.

Step 2: For this reversal FA, the corresponding right linear grammar is described as follows:

> Y → xY / yY / yX
>
> X → ∈

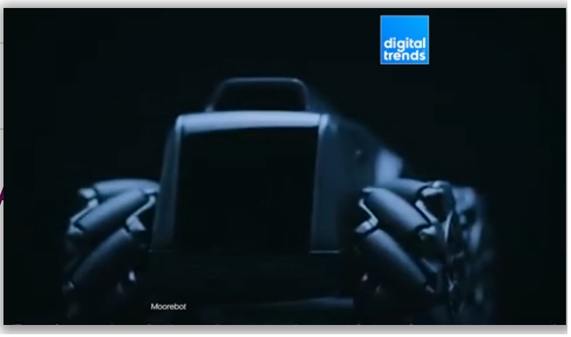
Step 3:

Now we will reverse the above right linear grammar and get the following grammar:

> Y → Yx / Yy / Xy
>
> X → ∈

So this grammar is left linear grammar, which is used to represent all string that begins with the symbol y. So

> L = {y, yx, yy, yxx, yxy, yyx, yyy ......}



Conversion of RLG (Right linear grammar) to F

We will begin from the first production.

From every left variable/alphabet, we will go to the symbol followed by it.

Start state: The first production state is known as the start state.

Final state: All the states which end up with terminals without following any further non-terminal.

Example:

In this example, we have the right linear grammar for language L. This grammar shows all the strings which end with 0.

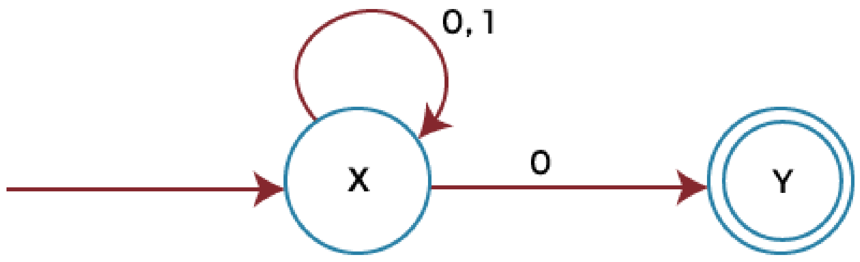
> X → 0X / 1Y / 0Y
>
> Y → ∈

So the finite automata which correspond to right linear grammar are described as follows:

Here we will begin with variable X and use its production like this:

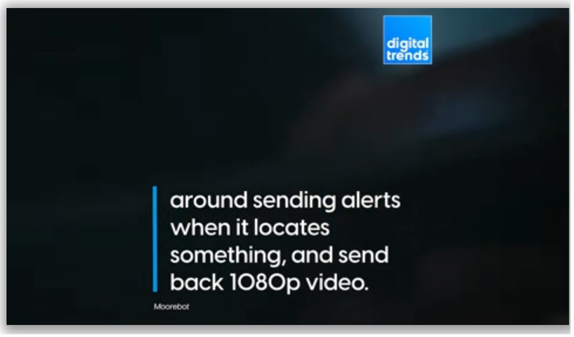
- The production X → 0X means that when the transition gets '0' as an input symbol, then state transition will always remain in the same state X.
- The production X → 1Y means that when the transition gets '1' as an input symbol, then the state transition will go to state Y from state X.
- The production X → 0Y means that when the transition gets '0' as an input symbol, then the state transition will go to state Y from state X.
- The production Y → ∈ means that this state does not require any type of state transitions. In the corresponding FA (finite automata), this type of production would be the final state because its RHS is terminal.

So for the corresponding right linear grammar, the final NFA (nonfinite automat) is described as follows:



Set of all strings that end with 0

Conversion of LLG to FA:


digital trends
around sending alerts when it locates something, and send back 1080p video.
Moorebot

$$\begin{tikzcd}
LLG \arrow[r, "Reverse"] & RLG \arrow[r] & FA \arrow[r, "Reverse"] & FA \\
(L) & (L)^R & (L)^R & (L^R)^R = L
\end{tikzcd}$$

Explanation:

Here we have first converted the LLG into the RLF grammar. Here LLG (Left linear grammar) represents the language L, which RLG (Right linear grammar) represents the reversal of language L, i.e., L^R. After that, we will design the FA corresponding to this reversal language (L^R). After that, we will reverse the finite automata (FA). The reversal FA is the final FA for language L.

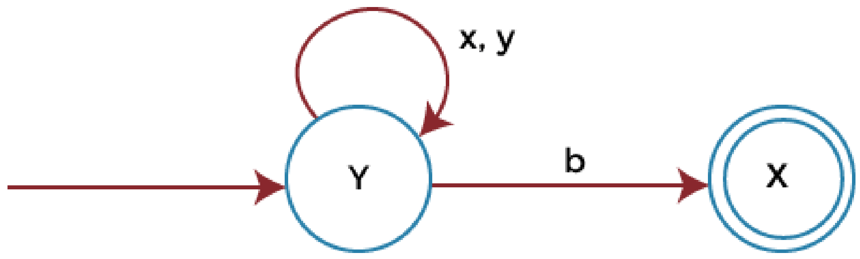
Conversion of LLG into RLG

To explain this, we will take the above grammar, which is used to represent the language L, and accept all sets of strings that begin with variable y. The left linear grammar for this grammar is described as follows:

```
Y → Yx / Yy / Xy
X → ∈
```

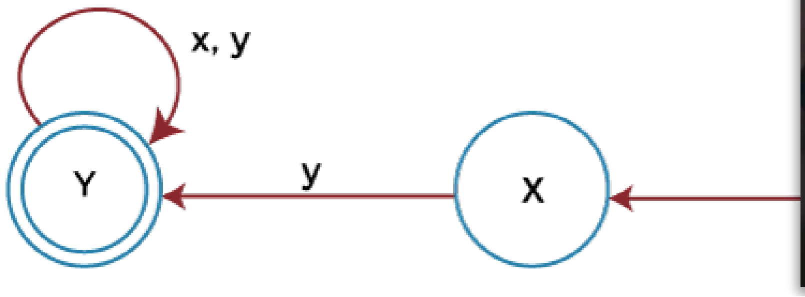
Step 1:

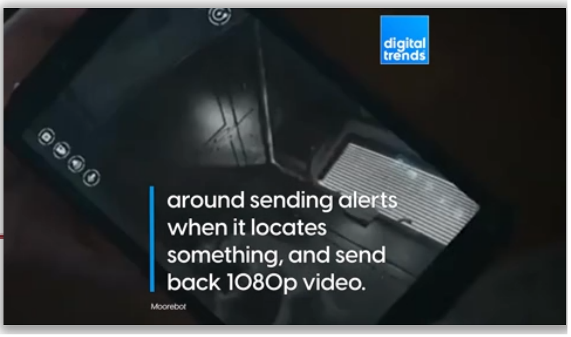
In this step, we will convert the LLG into FA. There is the same process of conversion as described above. So



Step 2:

In this step, we will reverse the FA. That means we will convert the initial state into the final state and the final state into starting state. We will also reverse all the edges like this:

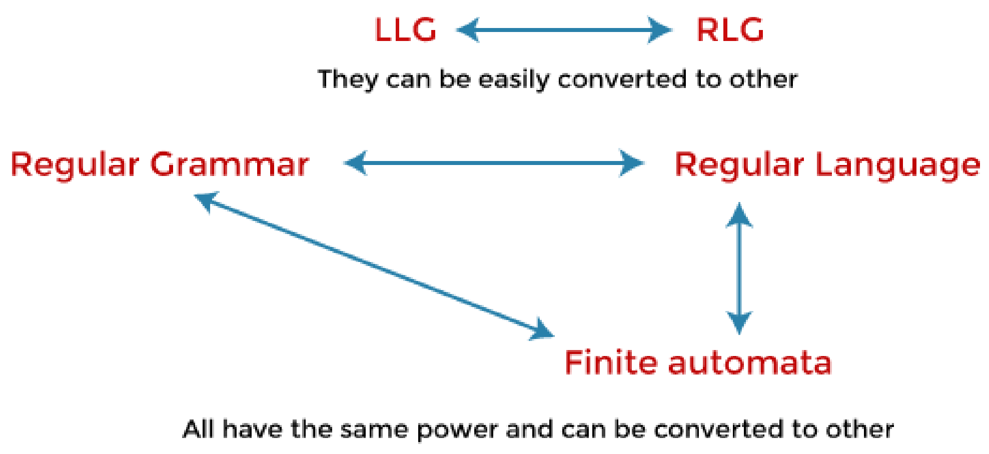



digital trends
around sending alerts when it locates something, and send back 1080p video.

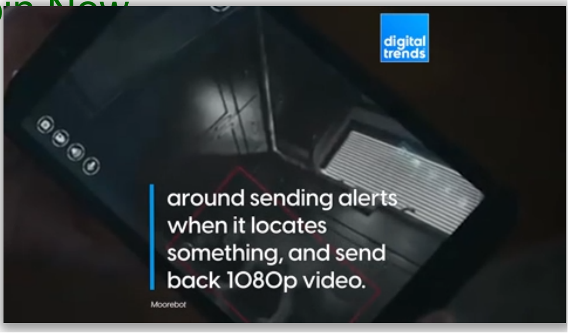
Step 3:

In this step, we will write RLG corresponding to reversed finite automata.

```
X → yY
Y → xY/yY/ ∈
```


LLG
RLG
They can be easily converted to other
Regular Grammar
Regular Language
Finite automata
All have the same power and can be converted to other

← Prev Next →


digital trends
around sending alerts when it locates something, and send back 1080p video.
Moorebot

Feedback

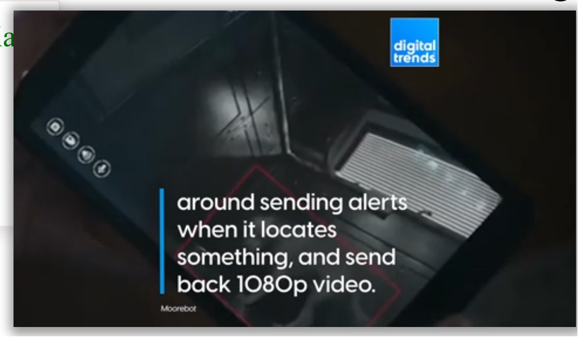
- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

Learn more

Learn Latest Tutorials

Splunk	SPSS	Swagger	Transact-SQL
Tumblr	ReactJS	Regex	Reinforcement Learning
R Programming tutorial R Programming	RxJS tutorial RxJS	React Native tutorial React Native	Python Design Patterns Python Design Patterns
Python Pillow tutorial Python Pillow	Python Turtle tutorial Python Turtle	Keras tutoria Keras	


digital trends
around sending alerts when it locates something, and send back 1080p video.
Moorebot

Preparation

Aptitude
Aptitude

Logical Reasoning
Reasoning

Verbal Ability
Verbal Ability

Interview Questions
Interview Questions

Company Interview Questions
Company Questions

Trending Technologies

Artificial Intelligence Tutorial
Artificial Intelligence

AWS Tutorial
AWS

Selenium tutorial
Selenium

Cloud Computing tutorial
Cloud Computing

Hadoop tutorial
Hadoop

ReactJS Tutorial
ReactJS

Data Science Tutorial
Data Science

Angular 7 Tutorial
Angular 7

Blockchain Tutorial
Blockchain

Git Tutorial
Git

Machine Learning Tutorial
Machine Learning

DevOps Tutorial
DevOps

B.Tech / MCA

DBMS tutorial
DBMS

Data Structures tutorial
Data Structures

DAA tutorial
DAA

Operating System tutorial
Operating System

Computer Network tutorial
Computer Network

Compiler Design tutorial
Compiler Design

Computer Organization and Architecture
Computer Organization

Discrete Mathematics Tutorial
Discrete Mathematics

Ethical Hacking Tutorial
Ethical Hacking

Computer Graphics Tutorial
Computer Graphics

Software Engineering Tutorial
Software Engineering

html tutorial
Web Technology

Cyber Security tutorial
Cyber Security

Automata Tutorial
Automata

C Language tutorial
C Programming

C++ tutorial
C++

Java tutorial
Java

.Net Framework tutorial
.Net

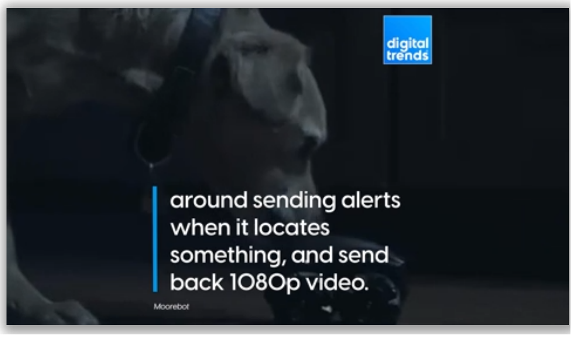
Python tutorial
Python

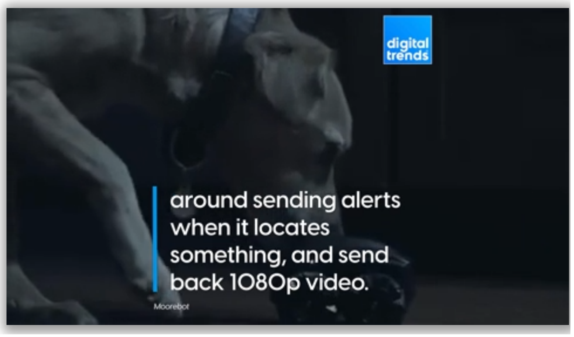
List of Programs
Programs

Control Systems tutorial
Control System

Data Mining Tutorial
Data Mining

Data Warehouse Tutorial
Data Warehouse


digital trends
around sending alerts when it locates something, and send back 1080p video.
Moorebot


digital trends
around sending alerts
when it locates
something, and send
back 1080p video.
Moorebot