CSE211 – Formal Languages and Automata Theory

U4L10_Rewriting Systems

Dr. P. Saravanan
School of Computing
SASTRA Deemed to be University

Agenda

- Recap of previous class
  - Function
  - Types of function
  - Other models of computation
    - Recursive functions
    - Post systems
- Rewriting Systems
  - Types of Rewriting systems

Models of computation:

- Turing Machines
- Recursive Functions
- Post Systems
- Rewriting Systems

Basic Primitive Recursive Functions

Zero function: $z(x) = 0$

Successor function: $s(x) = x + 1$

Projection functions: $p_1(x_1, x_2) = x_1$

$p_2(x_1, x_2) = x_2$

Building complicated functions:

Composition: $f(x,y) = h(g_1(x,y), g_2(x,y))$

Primitive Recursion:

$$f(x,0) = g_1(x)$$

$$f(x,y+1) = h(g_2(x,y), f(x,y))$$

Any function built from
the basic primitive recursive functions
is called:

Primitive Recursive Function

A Primitive Recursive Function: $add(x, y)$

$$add(x,0) = x$$

(projection)

$$add(x, y+1) = add(x, y) + 1$$

(successor function)

$$add(x,0) = x$$

$$add(x, y+1) = add(x, y) + 1$$

$$\begin{aligned} add(3,2) &= add(3,1) + 1 \\ &= (add(3,0) + 1) + 1 \\ &= (3+1) + 1) \\ &= 4 + 1 \\ &= 5 \end{aligned}$$

Post Systems

- Have Axioms
- Have Productions

Very similar with unrestricted grammars

Example: Unary Addition

Axiom: $1+1=11$

Productions:

$$V_1+V_2=V_3 \rightarrow V_1 1+V_2=V_3 1$$

$$V_1+V_2=V_3 \rightarrow V_1+V_2 1=V_3 1$$

A production:

$$V_1 + V_2 = V_3 \;\rightarrow\; V_1 1 + V_2 = V_3 1$$

$$1+1=11 \Rightarrow 11+1=111 \Rightarrow 11+11=1111$$

$$V_1 + V_2 = V_3 \;\rightarrow\; V_1 + V_2 1 = V_3 1$$

Rewriting Systems

They convert one string to another

- Matrix Grammars
- Markov Algorithms
- Lindenmayer-Systems

Very similar to unrestricted grammars

Matrix Grammar

- A matrix grammar is a formal grammar in which instead of individual productions, productions are grouped into finite sequences.
- A production cannot be applied separately, it must be applied in sequence.
- In the application of said sequence of productions, the rewriting is carried out according to each production in sequence, the first, the second, etc. until the last production has been used for rewriting.
- The sequences are known as matrices .

Matrix Grammars

Example:

$$P_1: \quad S \rightarrow S_1 S_2$$

$$P_2: \quad S_1 \rightarrow a S_1, \;\; S_2 \rightarrow b S_2 c$$

$$P_3: \quad S_1 \rightarrow \lambda, \;\; S_2 \rightarrow \lambda$$

Derivation:

$$S \Rightarrow S_1 S_2 \Rightarrow a S_1 b S_2 c \Rightarrow aa S_1 bb S_2 cc \Rightarrow aabbcc$$

A set of productions is applied simultaneously

$$P_1: \quad S \to S_1 S_2$$

$$P_2: \quad S_1 \to aS_1, \quad S_2 \to bS_2 c$$

$$P_3: \quad S_1 \to \lambda, \quad S_2 \to \lambda$$

$$L = \{a^n b^n c^n : n \geq 0\}$$

Theorem:

A language is recursively enumerable
if and only if
a Matrix grammar generates it

Matrix Grammar Ex:2

- We consider the matrix grammar G3 = ({S, A, B}, {a, b}, S, {m0, m1, m2, m3, m4}, ∅)

$L(G) = \{ww \mid w \in \{a, b\}^+\}$.

- where
  - m0 = (S → AB),
  - m1 = (A → aA, B → aB),
  - m2 = (A → bA, B → bB),
  - m3 = (A → a, B → a),
  - m4 = (A → b, B → b).
- What is the language generated by this grammar?

Markov Algorithms

- In theoretical computer science, a Markov algorithm is a string rewriting system
- It uses grammar-like rules to operate on strings of symbols
- Markov algorithms have been shown to be Turing-complete,
  - which means that they are suitable as a general model of computation and can represent any mathematical expression from its simple notation.
- Markov algorithms are named after the Soviet mathematician Andrey Markov, Jr.

Markov Algorithms

Grammars that produce λ

Example:

$$ab \to S$$

$$aSb \to S$$

$$S \to .\lambda$$

Derivation:

$$aaabbb \Rightarrow aaSbb \Rightarrow aSb \Rightarrow S \Rightarrow \lambda$$

$$ab \rightarrow S$$

$$aSb \rightarrow S$$

$$S \rightarrow .\lambda$$

$$L = \{a^n b^n : n \geq 0\}$$

In general: $L = \{w : \quad w \overset{*}{\Rightarrow} \lambda\}$

Theorem:

A language is recursively enumerable
if and only if
a Markov algorithm generates it

Markov Algorithm Ex: 2

- **Rules**
  - "|0" -> "0||"
  - "1" -> "0|"
  - "0" -> “”
- **Symbol string**
  - "101"

1. "101"
2. "0|01"
3. "00||1"
4. "00||0|"
5. "00|0|||"
6. "000||||"
7. "00|||||"
8. "0||||"
9. "|||||"

Markov Algorithm Ex:3

- **Rules**
  - "A" -> "apple"
  - "B" -> "bag"
  - "S" -> "shop"
  - "T" -> "the"
  - "the shop" -> "my brother"
  - "a never used" -> ."terminating rule"
- **Symbol string**
  - "I bought a B of As from T S."

Execution
the Symbol string will change in the following manner.

"I bought a B of As from T S."
"I bought a B of apples from T S."
"I bought a bag of apples from T S."
"I bought a bag of apples from T shop."
"I bought a bag of apples from the shop."
"I bought a bag of apples from my brother."

The algorithm will then terminate.

Lindenmayer-Systems

- An L-system or Lindenmayer system is a parallel rewriting system and a type of formal grammar
- It consists of
  - an alphabet of symbols that can be used to make strings,
  - a collection of production rules that expand each symbol into some larger string of symbols,
  - an initial "axiom" string from which to begin construction, and
  - a mechanism for translating the generated strings into geometric structures.
- L-systems were introduced and developed in 1968 by Aristid Lindenmayer,

Lindenmayer-Systems

They are parallel rewriting systems

Example: $a \rightarrow aa$

Derivation: $a \Rightarrow aa \Rightarrow aaaa \Rightarrow aaaaaaaa$

$$L = \{a^{2^n} : n \geq 0\}$$

Lindenmayer-Systems Ex: 2

- variables : A B
- constants : none
- axiom : A
- rules : (A → AB), (B → A)

which produces:

n = 0 : A
n = 1 : AB
n = 2 : ABA
n = 3 : ABAAB
n = 4 : ABAABABA
n = 5 : ABAABABAABAAB
n = 6 : ABAABABAABAABABAABABA
n = 7 : ABAABABAABAABABAABABAABAABABAABAAB

Ex: 3 - Fractal (binary) tree

- variables : 0, 1
- constants: “[”, “]”
- axiom : 0
- rules : (1 → 11), (0 → 1[0]0)

axiom: 0
1st recursion: 1[0]0
2nd recursion: 11[1[0]0]1[0]0
3rd recursion: 1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0
…

Real world applications

- This string can be drawn as an image by using turtle graphics,

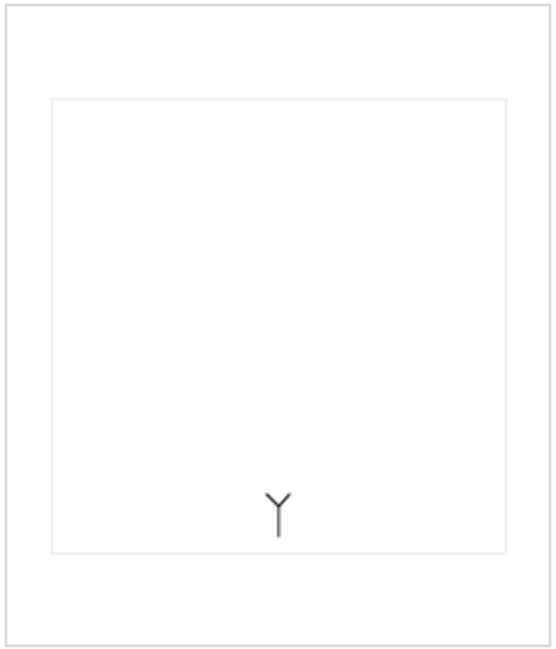
the turtle may be given the following instructions:

- 0: draw a line segment ending in a leaf
- 1: draw a line segment
- [: push position and angle, turn left 45 degrees
-]: pop position and angle, turn right 45 degrees

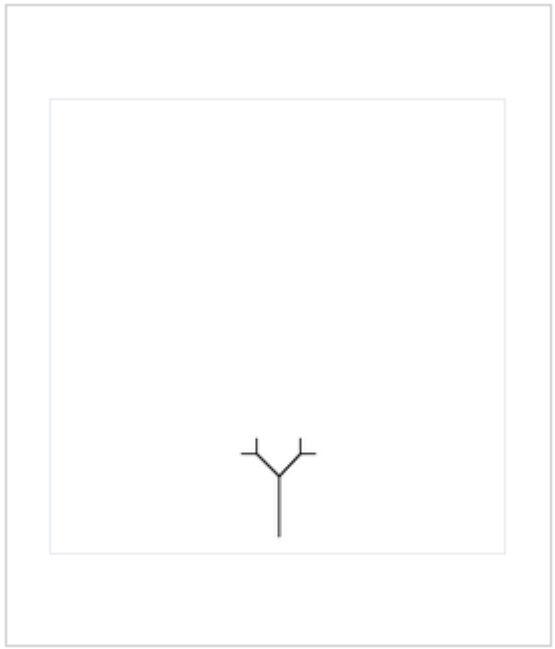
In real world..

axiom: 0
1st recursion: 1[0]0
2nd recursion: 11[1[0]0]1[0]0
3rd recursion: 1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0
…

Axiom



First recursion



Second recursion