Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра электронных вычислительных машин

Отчет

По лабораторной работе №4

«ГПСЧ в задачах инъективного преобразования последовательностей»

|  |  |  |
| --- | --- | --- |
| Выполнил:  Ст. гр. 050501  Тускенис Д. С. |  | Проверил:  Кобяк И. П. |

Минск 2014

1. **ПОСТАНОВКА ЗАДАЧИ**

Написать программу, которая генерирует состояния системы с заданным порождающим полиномом , по полученным состояниям строит матрицу переходов , матрицу биномиальных коэффициентов , и вычисляет матрицу коэффициентов полиномиального описания преобразователя .

1. **АЛГОРИТМ РЕШЕНИЯ**
2. Выбрать значения параметра в диапазоне
3. Выбрать полином, соответствующий параметру .

Например, (для )

1. Сгенерировать состояния системы с помощью LFSR с заданным в пункте 2 порождающим полиномом.
2. Построить матрицу размером , в которой единичные значения будут находиться на пересечении столбцов и строк с номерами, соответствующим соседним состояниям, сгенерированных в пункте 3.
3. Построить матрицу , используя формулу
4. Вычислить матрицу по формуле
5. Проверить строку с номером на наличие единичных значений только в столбцах с номерами .
6. Если проверка, осуществившееся в пункте 7, является удовлетворительной, то попытка линеаризации системы считается удачной
7. **ЛИСТИНГ ПРОГРАММЫ**

class LFSR

{

private String state;

private String polynom;

public LFSR(String polynom, String initialState = null)

{

if (initialState != null && initialState.Length != polynom.Length)

{

throw new ArgumentException();

}

this.state = initialState != null ? (String)initialState.Clone() :

new String('0', polynom.Length);

this.polynom = (String)polynom.Clone();

}

public String[] GenerateStates(Int32 numberOfStates)

{

List<String> states = new List<string>();

while (numberOfStates > 0)

{

states.Add(state);

Next();

numberOfStates--;

}

return states.ToArray();

}

public void Next()

{

StringBuilder nextStateBuilder = new StringBuilder();

nextStateBuilder.Append(CalculateNewValue());

for (int i = 0; i < state.Length - 1; i++)

{

nextStateBuilder.Append(state[i]);

}

state = nextStateBuilder.ToString();

}

private Char CalculateNewValue()

{

bool newValue = false;

for (int c = 0; c < polynom.Length; c++)

{

if (polynom[c] == '1')

{

newValue ^= state[c] == '1' ? true : false;

}

}

return newValue ? '1' : '0';

}

}

class TMatrix : Matrix

{

private bool[,] internalMatrix;

public TMatrix(Int32 size, String[] states = null)

{

internalMatrix = new bool[size, size];

if (states != null)

{

for (int i = 0; i < states.Length - 1; i++)

{

Int32 x = Convert.ToInt32(states[i], 2);

Int32 y = Convert.ToInt32(states[i + 1], 2);

internalMatrix[y, x] = true;

}

}

}

public override int numberOfRows

{

get { return internalMatrix.GetLength(0); }

}

public override int numberOfColumns

{

get { return internalMatrix.GetLength(1); }

}

public override Int32 this[int i, int j]

{

get { return internalMatrix[i, j] ? 1 : 0; }

set { }

}

}

class CMatrix : Matrix

{

private Int32[,] internalMatrix;

public CMatrix(Int32 size)

{

internalMatrix = new Int32[size, size];

for (int k = 0; k < size; k++)

{

for (int n = 0; n < size; n++)

{

if (n >= k)

{

internalMatrix[k, n] = (Int32)(Factorial(n) / (Factorial(k) \* Factorial(n - k)));

}

else

{

internalMatrix[k, n] = 0;

}

}

}

}

private BigInteger Factorial(BigInteger number)

{

return number < 2 ? 1 : number \* Factorial(number - 1);

}

public override int numberOfRows

{

get { return internalMatrix.GetLength(0); }

}

public override int numberOfColumns

{

get { return internalMatrix.GetLength(1); }

}

public override Int32 this[int i, int j]

{

get { return internalMatrix[i, j] % 2; }

set { }

}

}

class Program

{

static void Main(string[] args)

{

// 1 + x + x^3 => "101"

String polynom = "101";

Int32 matrixSize = (Int32)Math.Pow(2, polynom.Length);

LFSR lfsr = new LFSR(polynom, "100");

String [] states = lfsr.GenerateStates(7);

Console.WriteLine("States:");

for (int i = 0; i < 7; i++) {

Console.WriteLine((i + 1) + ". " + states[i]);

}

Console.WriteLine();

Matrix tMatrix = new TMatrix(matrixSize, states);

Console.WriteLine("Matrix T:\n");

PrintMatrix(tMatrix);

Console.WriteLine();

Matrix cMatrix = new CMatrix(matrixSize);

Console.WriteLine("Matrix C:\n");

PrintMatrix(cMatrix);

Console.WriteLine();

Matrix lMatrix = MatrixMath.Multiply(MatrixMath.Transpose(cMatrix), tMatrix);

lMatrix = MatrixMath.Multiply(lMatrix, MatrixMath.Transpose(cMatrix));

lMatrix = MatrixMath.Mod(lMatrix, 2);

Console.WriteLine("Matrix L:\n");

PrintMatrix(lMatrix);

// Check linearity

Boolean isLinear = CheckLinearity(lMatrix, polynom.Length);

Console.WriteLine("\n" + (isLinear ? "Linear" : "Not linear"));

Console.WriteLine("\n\nPress any key...");

Console.ReadKey();

}

static bool CheckLinearity(Matrix lMatrix, int l)

{

bool isLinear = true;

int rowIndex = (int)Math.Pow(2, l - 1);

List<double> twos = new List<double>();

for (int i = 0; i < l; i++)

{

twos.Add(Math.Pow(2, i));

}

for (int i = 0; i < Math.Pow(2, l); i++)

{

if ((!twos.Contains(i) && lMatrix[rowIndex, i] != 0) || (twos.Contains(i) && lMatrix[rowIndex, i] == 0))

{

isLinear = false;

break;

}

}

return isLinear;

}

}

1. **РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ**

