Тема 18. Многочлены Жегалкина в нераскрытом виде от трех переменных

Примеры:

1) ((xyz+zyx+1)*(z+xxx)+xyzzy+z+y+x)zxz+1

Грамматика

Алфавит A = {x, y, z, (), 1}

<многочлен>::=<конструкция><хвост>;

<конструкция>::=1 | <символ><type1>|(<скобка>)<умножение><type2>;

<type1>::=(<скобка>)<type2>|<символ><type1>|Λ

<type2>::=(<скобка>)<type1>|<символ><type1>

<скобка>::=<конструкция>+<конструкция><хвост>;

<хвост>::=+<конструкция><хвост> | Λ

<символ>::=x,y,z

<умножение>::=*, Λ

type1 type2

type1-выражение после которого умножение необязательно, т.к необходимость в скобках вызвана предыдущим выражением.

Пример: y(x+x+z+1)

type2- выражение после которого идет умножение, иначе скобки можно было бы опустить.

Пример: (x+y+z+1)x

Проверка на принадлежности классу LL(1)

<конструкция>::=1_{w1} | <символ><type1>$_{w2}$|(<скобка>)<умножение><type2>$_{w3}$;

$L(w_1)$={ 1 };$L(w_2)$={ x | y | z };$L(w_3)$={ (}

L(w1)^L(w2)=0;

L(w1)^L(w3)=0;

L(w2)^L(w3)=0;

<type1>::=(<скобка>)<type1>$_{w1}$|<символ><type1>$_{w2}$|Λ_{w3}

$L(w_1)$={ (};$L(w_2)$={ x | y | z };$L(w_3)$={ }

L(w1)^L(w2)=0;

L(w1)^L(w3)=0;

L(w2)^L(w3)=0;

<type2>::=(<скобка>)<type1>w1|<символ><type1>w2

$L(w_1)$={ 1 };$L(w_2)$={ x | y | z };

L(w1)^L(w2)=0;

<скобка>::=<конструкция>+<конструкция><хвост>;

<хвост>::=+<конструкция><хвост> $_{w1}$| Λ_{w2}

$L(w_1)$={ + };$L(w_2)$={ };

L(w1)^L(w2)=0;

<символ>::=x_{w1}|y_{w2}|z_{w3}

$L(w_1)$={ x };$L(w_2)$={ y };$L(w_3)$={ z }

L(w1)^L(w2)=0;

L(w1)^L(w3)=0;

L(w2)^L(w3)=0;

<умножение>::=*$_{w1}$|Λ_{w2}

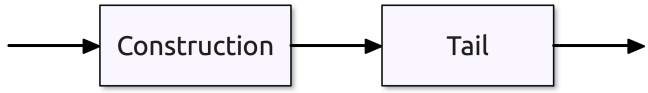
$L(w_1)$={ * };$L(w_2)$={ };

L(w1)^L(w2)=0;

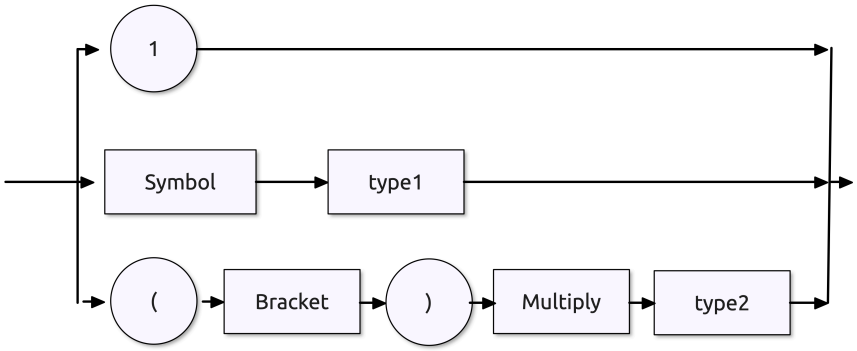
Следовательно, полученная КС-грамматика обладает однозначностью ветвления по первому символу

Синтаксические диаграммы

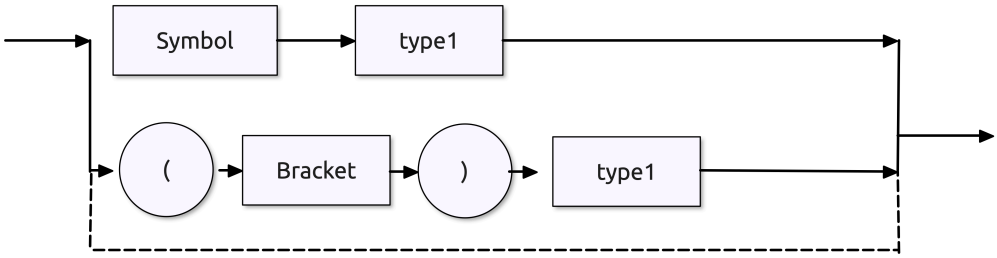
<многочлен>::=<конструкция><хвост>;



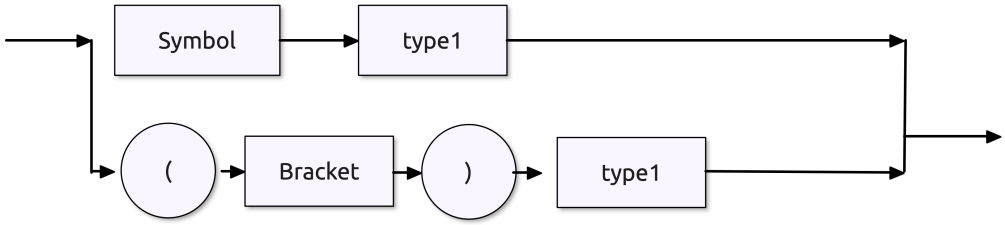
<конструкция>::=1 | <символ><type1>|(<скобка>)<умножение><type2>;



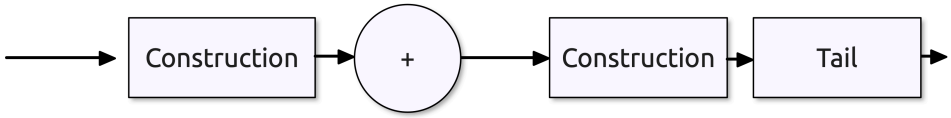
<type1>::=(<скобка>)<type1>|<символ><type1>|Λ



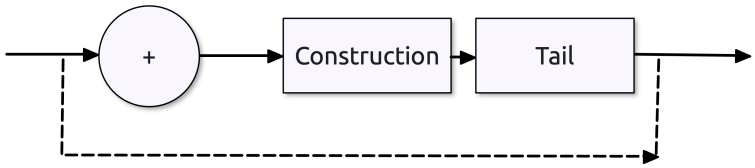
<type2>::=(<скобка>)<type1>|<символ><type1>



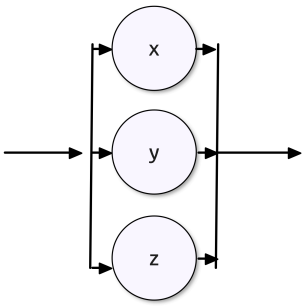
<скобка>::=<конструкция>+<конструкция><хвост>;



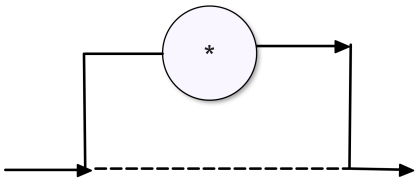
<хвост>::=+<конструкция><хвост> | Λ



<символ>::=x|y|z



<умножение>::=*|Λ



Алгоритм синтаксического анализа

Псевдокод

<многочлен>::=<конструкция><хвост>;	Polynom{
	Construct;
	Tail;
	}

<конструкция>::=1 | <символ><type1>|
(<скобка>)<умножение><type2>;
	Construct{
	If ch=1 then
	If ch=1 then read(ch) else error;
	If ch=x OR ch=y OR ch=z then
	{
	If ch=x OR ch=y OR ch=z then {
	Symbol;
	TYPE1;
	} else error;
	If ch='(' then
	{
	If ch='('then read(ch) else error;
	Bracket;
	If ch=')' then read(ch) else error;
	Multiply;
	TYPE2;
	}
	else error;
	}

<type1>::=(<скобка>)<type1>|<символ><type1>|Λ	TYPE1{
	If ch=')' then
	{
	If ch=')' then read(ch) else error;
	Bracket;
	If ch=')' then read(ch) else error;
	TYPE11;
	}
	If ch=x OR ch=y OR ch=z then

	{
	If ch=x OR ch=y OR ch=z then {
	Symbol;
	TYPE1;
	}
	}

<type2>::=(<скобка>)<type1>|<символ><type1>	TYPE2{
	If ch='(' then
	{
	If ch='(' then read(ch) else error;
	Bracket;
	If ch=')' then read(ch) else error;
	TYPE1;
	}
	If ch=x OR ch=y OR ch=z then
	{
	If ch=x OR ch=y OR ch=z then {
	Symbol;
	TYPE1;
	}
	else error;
	}

<скобка>::=<конструкция>+<конструкция><хвост>	Bracket{
	Contruct;
	If ch='+' then read(ch) else error;
	Construct;
	Tail;
	}

<хвост>::=+<конструкция><хвост> | Λ	Tail{
	If ch='+' them
	{
	If ch='+' then read(ch) else error;
	Construct;
	Tail;
	}
	}

<символ>::=x|y|z	Symbol{
	If ch=x then
	If ch=x then read(ch) else error;
	If ch=y then
	If ch=y then read(ch) else error;
	If ch=z then
	If ch=z then read(ch) else error;
	else error;
	}

<умножение>::=*|Λ	Multiply{
	If ch='*' then
	If ch='*' then read(ch) else error;
	}

Перевод в функцияя на Python

Функция на Python3	

def Polynom():
  Construct()
  Tail()
	Polynom{
	  Construct;
	  Tail;
	}

def Construct():
  global string, i, error
  print("Proceeding Construct")
  if string[i] == '1':
  i += 1
  elif string[i] in "xyz":
  Symbol()
  TYPE1()
  elif string[i] == '(':
  i += 1
  Bracket()
  if string[i] == ')':
  i += 1
  Multiply()
  TYPE2();
  else:
  error = True
  exit("Error: no closing bracket in construct")
  else:
  error = True
  exit("Error: Construct")
	Construct{
	  If ch=1 then
	  If ch=1 then read(ch) else error;
	  If ch=x OR ch=y OR ch=z then
	  {
	  If ch=x OR ch=y OR ch=z then {
	  Symbol;
	  TYPE1;
	  } else error;
	  If ch='(' then
	{
	  If ch='('then read(ch) else error;
	  Bracket;
	  If ch=')' then read(ch) else error;
	  Multiply;
	  TYPE2;
	  }
	else error;
	}

def TYPE1():
  global string, i, error
  print("Proceeding TYPE1")
  if string[i] == ')':
  i += 1
  Bracket()
  if string[i] == ')':
  i += 1
  TYPE1()
  else:
  error = True
  exit("Error: No cloign bracking in TYPE1")
  elif string[i] in "xyz":
  Symbol()
  TYPE1()
	TYPE1{
	  If ch=')' then
	{
	  If ch=')' then read(ch) else error;
	  Bracket;
	  If ch=')' then read(ch) else error;
	  TYPE11;
	  }
	  If ch=x OR ch=y OR ch=z then
	  {
	  If ch=x OR ch=y OR ch=z then {
	  Symbol;
	  TYPE1;
	  }
	}

def TYPE2():
  global string, i, error
  print("Proceeding TYPE2")
 	
  if string[i] == '(':
  i += 1
  Bracket()
  if string[i] == ')':
  i += 1
  TYPE1();
  else:
  error = True
  exit("Error: no closing bracket in TYPE2")
	TYPE2{
	  If ch='(' then
	{
	  If ch='(' then read(ch) else error;
	  Bracket;
	  If ch=')' then read(ch) else error;
	  TYPE1;
	  }
	  If ch=x OR ch=y OR ch=z then
	  {
	  If ch=x OR ch=y OR ch=z then {
	  Symbol;
	  TYPE1;

elif string[i] in "xyz":
  Symbol()
  TYPE1()
else:
  error = True
  exit("Error: unneded brackets")
	}
	else error;
	}

def Bracket():
  global string, i, error
  print("Proceeding Bracket")
  Construct()
  if string[i] == "+":
  i += 1
  Construct()
  Tail()
  else:
  error = True
  exit("Error: unneded bracket in Bracket")
	Bracket{
	Contruct;
	If ch='+' then read(ch) else error;
	Construct;
	Tail;
	}

def Tail():
  global string, i, error
  print("Proceeding tail")
 	
  if string[i] == "+":
  i += 1
  Construct()
  Tail()
	Tail{
	If ch='+' them
	{
	If ch='+' then read(ch) else error;
	Construct;
	Tail;
	}
	}

def Symbol():
  global string, i, error
  print("Proceeding Symbol")
  if string[i] in 'xyz':
  i += 1
  else:
  error = True
  exit("Error: unknown variable")
	Symbol{
	If ch=x then
	If ch=x then read(ch) else error;
	If ch=y then
	If ch=y then read(ch) else error;
	If ch=z then
	If ch=z then read(ch) else error;
	else error;
	}

def Multiply():
  global string, i, error
  print("Proceeding Multiply")
  if string[i] == "*":
  i += 1
	Multiply{
	If ch='*' then
	If ch='*' then read(ch) else error;
	}