Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 “Вычислительная математика и программирование”

Факультет: “Информационные технологии и прикладная математика”

Дисциплина: “Системы программирования”

Курсовой проект.

Тема: “ Разработка синтаксически управляемого перевода”

Группа: М80 – 207Б

Студент: Хренов Геннадий Николаевич

Преподаватель: Семёнов Александр Сергеевич

Оценка: _____

Дата:_____

Москва, 2020.

Содержание

Постановка задачи

Необходимо разработать синтаксически управляемый перевод алгебраических выражений, заданных в символьном виде, в алгебраические операции, на примере, операции булевой алгебры.

Метод решения

Сначала построим исходную грамматику

G = (T, V, P, S), где:

T = {a, b, c, d, ¬, ∧, ∨, (,)} – множество терминалов

V = {S, F, T, I} – множество нетерминалов

S – начальный символ

P = {
S → T
T → T∨F | T∧F | F,
F→ I | (T) | ¬F,
I→ a | b | c | d
} – правила продукции

Пронумеруем правила:

(1) S → T
(2) T → T∨F
(3) T → T∧F
(4) T → F
(5) F→ I
(6) F → (T)
(7) F → ¬F
(8) I→ a | b | c | d,

Эта грамматика порождает цепочки из множества булевой алгебры:

a∧b - конъюнкция

a∨b - дизъюнкция

¬a – отрицание

Пример порождения цепочки ¬(a ∨ ¬b ∧ c):

$S \to_{(1)} T \to_{(4)} F \to_{(7)} \neg F \to_{(6)} \neg(T) \to_{(3)} \neg(T \wedge F) \to_{(5)} \neg(T \wedge I) \to_{(8)} \neg(T \wedge c) \to_{(2)} \neg(T \vee F \wedge c) \to_{(7)} \neg(T \vee \neg F \wedge c) \to_{(5)} \neg(T \vee \neg I \wedge c) \to_{(8)} \neg(T \vee \neg b \wedge c) \to_{(4)} \neg(F \vee \neg b \wedge c) \to_{(5)} \neg(I \vee \neg b \wedge c) \to_{(8)} \neg(a \vee \neg b \wedge c)$

Далее строим грамматику перевода:

G' = (T, V, P', S), где:

T,V,S аналогичны G

P' = {

S → T
T → TF∨ | TF∧ | F,
F→ I | T | F¬,
I→ a | b | c | d

}

Пронумеруем правила:

(1) S → T
(2) T → TF∨
(3) T → TF∨
(4) T → F
(5) F→ I
(6) F → T
(7) F → F¬
(8) I→ a | b | c | d,

Перевод осуществляется с помощью транслирующей грамматики.

Транслирующей грамматикой называется пятерка объектов $G^T = (N, \Sigma_i, \Sigma_a, P, S)$, где Σ_i — словарь входных символов, Σ_a — словарь операционных символов, N — нетерминальный словарь, $S \in N$ — начальный символ транслирующей грамматики, P — конечное множество правил вывода вида $A \to \alpha$, в которых $A \in N$, а $\alpha \in (\Sigma_i \cup \Sigma_a \cup N)^*$.

Построим транслирующую грамматику $G^T=(N,\Sigma_i,\Sigma_a,P,S)$:

N = {T, F, I, S}

Σ_i = { a, b, c, d, ¬, ∧, ∨, (,)}

Σ_a = {{∨}, {∧}, {¬}, {a}, {b}, {c}, {d}}

P = {

(1) S → T
(2) T → T∨F {∨}
(3) T → T∧F {∧}
(4) T → F
(5) F→ I
(6) F → (T)
(7) F → ¬F {¬}
(8) I→ i {i}, где i ∈{a, b, c, d}

}

Цепочки языка, порождаемого транслирующей грамматикой, называют активными цепочками. Входной частью активной цепочки называется последовательностью входных символов, полученной из активной цепочки после удаления всех операционных символов. Операционной частью активной цепочки называется последовательность операционных символов, полученная путем вычёркивания из активной цепочки всех входных символов.

Программная реализация

Для реализации мы преобразуем алгоритм разбора для LL(1)-грамматик, добавив в него действия, обеспечивающие перевод входной цепочки, порождаемой входной грамматикой, в цепочку операционных символов и запись этой цепочки на выходную ленту. Преобразованный таким образом алгоритм называется нисходящим детерминированным процессором с магазинной памятью (нисходящий ДМП-процессор). После получения цепочки операционных символов подсчитаем значение булева выражения.

Модернизируем LL-разбор из фреймворка п.5. Добавим обработку операционных символов и изменим символы для удобства их набора:

∧ = &
∨ = |
¬ = -

Также добавим функцию для выполнения булевых операций.
Модернизированный фреймворк:

```
public bool Parse(string input)
        {
            ArrayList output = new ArrayList();
            ArrayList oprs = new ArrayList() { "{|}", "{&}", "{-}", "{0}",
"{1}" };
            Stack.Push("EoS"); // символ окончания входной последовательности
            Stack.Push(G.S0);
            int i = 0;
            string currInputSymbol = input[i].ToString();
            string currStackSymbol;
            do
            {
                currStackSymbol = Stack.Peek();
                if (G.T.Contains(currStackSymbol) &&
!oprs.Contains(currStackSymbol)) // в вершине стека находится терминал
                {
                    if (currInputSymbol == currStackSymbol) // распознанный
символ равен вершине стека
                    {
                        // Извлечь из стека верхний элемент и распознать
символ входной последовательности (ВЫБРОС)
                        Stack.Pop();
                        if (i != input.Length)
                        {
                            i++;
                        }
                        currInputSymbol = (i == input.Length) ? "EoI" :
input[i].ToString();
                    }
                    else
                    {
                        // ERROR
                        Console.WriteLine("Не успех. Строка не соответствует
грамматике.");
                        return false;
                    }
                }
                else if (G.V.Contains(currStackSymbol)) // если в вершине
стека нетерминал
                {
                    DataRow custRows = Table.Select("VT = '" +
currStackSymbol.Replace(@"'", @"''") + "'", null)[0];
                    if (!custRows.IsNull(currInputSymbol)) // в
клетке[вершина стека, распознанный символ] таблицы разбора существует правило
                    {
                        //  извлечь из стека элемент и занести в стек все
терминалы и нетерминалы найденного в таблице правила в стек в порядке
обратном порядку их следования в правиле
                        Stack.Pop();
                        foreach (var chainSymbol in
((Prule)custRows[currInputSymbol]).RightChain.Cast<string>().Reverse().ToList
())
                        {
                            if (chainSymbol != "")
                            {
                                Stack.Push(chainSymbol);
                            }
                        }
```

```
                        OutputConfigure +=
(((Prule)custRows[currInputSymbol]).Id);
                    }
                    else
                    {
                        // ERROR
                        Console.WriteLine("Не успех. Строка не соответствует
грамматике.");
                        return false;
                    }
                }
                else
                {
                    output.Add(Stack.Pop()[1]);
                }
            } while (Stack.Peek() != "EoS"); // вершина стека не равна концу
входной последовательности

            if (i != input.Length) // распознанный символ не равен концу
входной последовательности
            {
                // ERROR
                Console.WriteLine("Не успех. Строка не соответствует
грамматике.");
                return false;
            }
            Console.WriteLine("Успех. Строка соответствует грамматике.");
            Console.WriteLine("Выходная лента:");
            foreach (object o in output)
            {
                Console.Write(o);
            }
            Console.WriteLine();

            Stack<string> st = new Stack<string>();
            string str;
            foreach (object o in output)
            {
                if (o.ToString() == "-")
                {
                    str = DoOper(st.Pop(), "0", o.ToString());
                    st.Push(str);
                }
                else if (o.ToString() == "&" || o.ToString() == "|")
                {
                    str = DoOper(st.Pop(), st.Pop(), o.ToString());
                    st.Push(str);
                }
                else
                {
                    st.Push(o.ToString());
                }

            }
            Console.WriteLine("Значение булева выражения:");
            Console.WriteLine(st.Peek());

            return true;
        }

        public string DoOper(string a, string b, string op)
        {
            if (op == "&")
            {
```

```
            if (a == "1" && b == "1")
            {
                return ("1");
            } else {
                return ("0");
            }
        }
        if (op == "|")
        {
            if (a == "0" && b == "0")
            {
                return ("0");
            } else {
                return ("1");
            }
        }
        if (op == "-")
        {
            if (a == "0")
            {
                return ("1");
            } else {
                return ("0");
            }
        }
        return ("Error");
    }
```

Пример работы программы:

Введите строку:
-0
Успех. Строка соответствует грамматике.
Выходная лента:
0-
Значение булева выражения:
1

Введите строку:
1&0
Успех. Строка соответствует грамматике.
Выходная лента:
10&
Значение булева выражения:
0

Введите строку:
-(1&0|0)
Успех. Строка соответствует грамматике.
Выходная лента:
10&0|-

Значение булева выражения:
1

Введите строку:
-(1&0&-(0|0))
Успех. Строка соответствует грамматике.
Выходная лента:
10&00|-&-
Значение булева выражения:
1

Вывод

Для перевода требуется каждой входной цепочке (программе на входном языке) поставить в соответствие другую цепочку (программу на выходном языке). Для решения проблемы задания бесконечного перевода конечными средствами можно использовать транслирующие грамматики. Так как множество их терминалов разбито на множества входных и операционных символов, их удобно применять для трансляций различных алгебраических выражений.