МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Курсовая работа по дисциплине

“Теория формальных языков и компиляторов”

Десятичные константы языка PASCAL

Факультет: АВТФ

Преподаватель: Шорников Ю. В.

Группа: АВТ-613

Выполнил: Гардер Алексей

Новосибирск, 2019 г.

Содержание

1 Постановка задачи

2 Подбор порождающей грамматики

< Цифра 0..9 > → 0|1|2|3|4|5|6|7|8|9

< Цифра 1..9 > → 1|2|3|4|5|6|7|8|9

< Знак > → + | −

< Экспонента > → *E*|*e*

< Целое допускающее 0 > → < Цифра 0..9 > | < Цифра 0..9 >< Целое без знака >

< Целое допускающее *E* > → < Цифра 0..9 > | < Цифра 0..9 >< Целое без знака >

< Целое допускающее *E* > → < Цифра 0..9 >< *E* >

< Дробная часть > → . < Целое допускающее *E* > |. < *E* > | Λ

< *E* > → < Экспонента >< Целое со знаком > | < Экспонента >< Целое >

< Целое со знаком > → < Знак >< Целое >

< Целое > → < Цифра 0..9 > | < Цифра 1..9 >< Целое допускающее 0 >

< Десятичная константа допускающая 0 > → < Цифра 0..9 >< Дробная часть >

< Десятичная константа допускающая 0 > → < Цифра 0..9 >< *E* >

< Десятичная константа допускающая 0 > → < Цифра 0..9 >< Десятичная константа >

< Десятичная константа без знака > → < Цифра 0..9 >< Дробная часть >

< Десятичная константа без знака > → < Цифра 1..9 >< *E* >

< Десятичная константа без знака > → < Цифра 1..9 >< Десятичная константа допускающая 0 >

< Десятичная константа > → < Знак >< Десятичная константа без знака >

< Десятичная константа > → < Десятичная константа без знака >

3 Классификация Хомского

4 Выбор метода

5 Реализация метода

Перечень состояний конечного автомата:

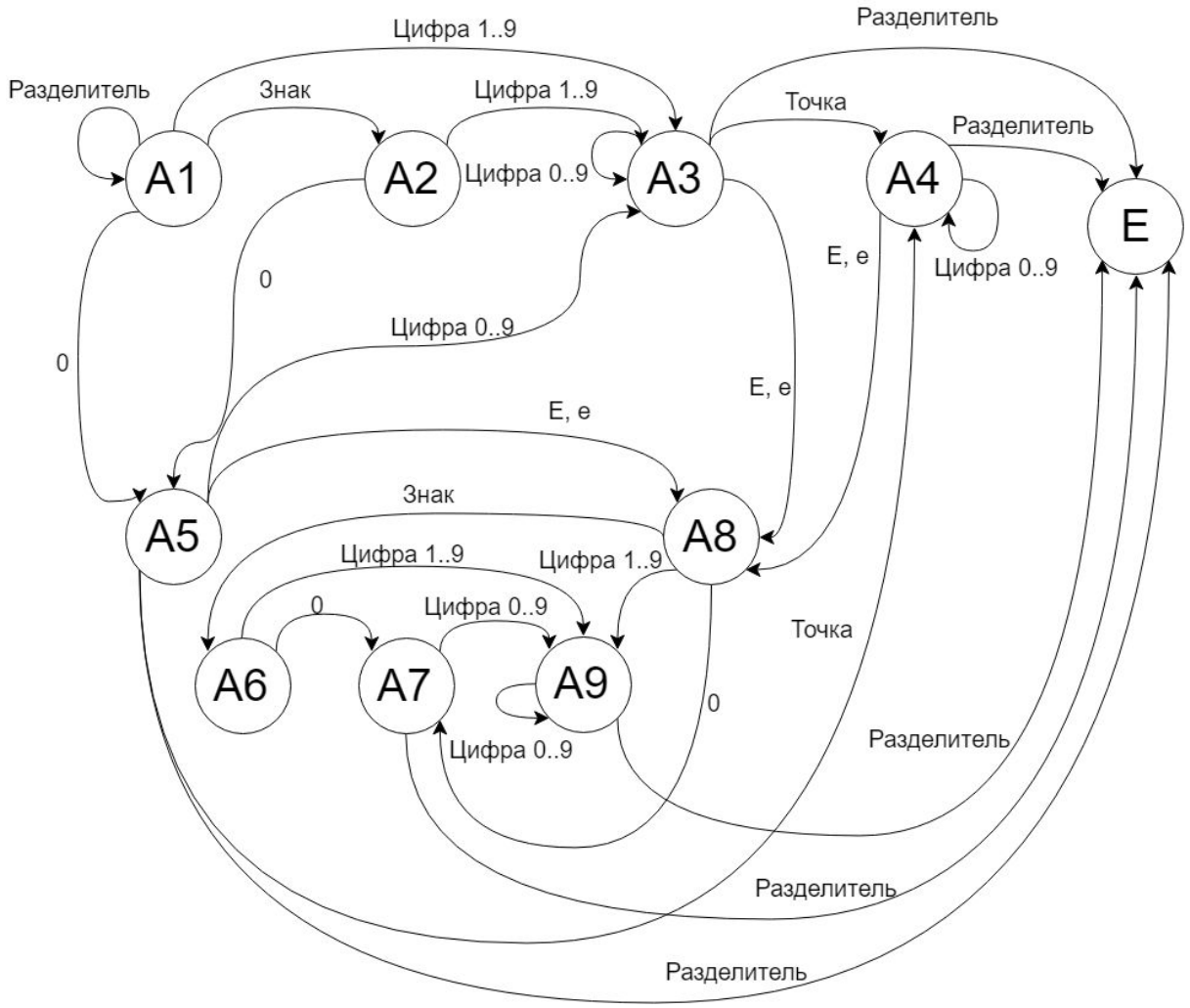

Таблица. Состояния конечного автомата

Обозначение	Описание
A1	Чтение десятичной константы
A2	Чтение десятичной константы без знака
A3	Чтение десятичной константы, допускающей ноль в начале
A4	Чтение беззнакового целого, допускающего экспоненциальную форму записи
A5	Чтение десятичной константы, содержащей ноль вначале (нужно для вывода предупреждения)
A6	Чтение беззнакового целого

A7	Чтение беззнакового целого, содержащего ноль вначале (нужно для вывода предупреждения)
A8	Чтение целого со знаком
A9	Чтение беззнакового целого, допускающего ноль в начале
E	Конечное состояние

A1 - начальное состояние.

Таблица. Обозначения на схеме

Обозначение	Описание
Цифра 0..9	Цифры в диапазоне от 0 до 9
Цифра 1..9	Цифры в диапазоне от 1 до 9
Точка	Символ-разделитель в записи десятичной дроби: точка
Разделитель	Символ, отделяющий число от основного текста: пробел, конец строки, табуляция, конец строки
Знак	Знак десятичной константы: + или -

6 Тестирование

7 Листинг (10 страниц? Ну нет...)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TeorForm_lab1.Lexer;


namespace TeorForm_lab1.RecursiveDescent
{
    class DecimalParser
    {
        private DecimalParseMode mode;
        private List<Warning> warnings;
        private TextData textData;
        private StringBuilder resultString;


        public bool ParseDecimalConst(TextData data, out List<Warning> warningsCollection, out string result)
        {
            mode = DecimalParseMode.DecimalConst;
            warnings = new List<Warning>();
            textData = data;
            resultString = new StringBuilder();


            while (true)
            {
                switch (mode)
```

```
{
    case DecimalParseMode.DecimalConst:
        ParseDecimal();
        break;
    case DecimalParseMode.UnsignedDecimalConst:
        ParseUnsignedDecimal();
        break;
    case DecimalParseMode.DecimalConstWithNull:
        ParseUnsignedDecimalWithNull();
        break;
    case DecimalParseMode.UnsignedIntegerWithExponent:
        ParseUnsignedIntegerWithExponent();
        break;
    case DecimalParseMode.NullStartDecimal:
        ParseNullStartDecimal();
        break;
    case DecimalParseMode.Ending:
        warningsCollection = warnings;
        result = resultString.ToString();
        return warnings.All(x => x.WarningType != WarningType.Error);
    case DecimalParseMode.UnsignedInteger:
        ParseUnsignedInteger();
        break;
    case DecimalParseMode.NullStartInteger:
        ParseNullStartInteger();
        break;
    case DecimalParseMode.SignedInteger:
        ParseSignedInteger();
        break;
```

```
            case DecimalParseMode.UnsignedIntegerWithNull:
                ParseUnsignedIntegerWithNull();
                break;
            default:
                throw new NotImplementedException();
        }
    }
}

void ParseDecimal()
{
    while (true)
    {
        switch (textData.PeekChar())
        {
            case ' ':  case '\t': case '\n':
                //Here we ignore whitespace
                textData.AdvanceChar();
                break;
            case '1':  case '2': case '3': case '4': case '5': case '6': case '7': case '8': case '9':
                mode = DecimalParseMode.DecimalConstWithNull;
                SaveCharacter();
                textData.AdvanceChar();
                return;
            case '0':
                mode = DecimalParseMode.NullStartDecimal;
                SaveCharacter();
                textData.AdvanceChar();
                return;
```

```
            case '+': case '-':
                mode = DecimalParseMode.UnsignedDecimalConst;
                SaveCharacter();
                textData.AdvanceChar();
                return;
            case '\0':
                MakeWarningMinimal(
                    "Value cannot be empty",
                    textData.PeekChar(),
                    textData.Position,
                    WarningType.Error);
                mode = DecimalParseMode.Ending;
                return;
            default:
                MakeWarning(
                    "Unknown character! There can only be digit or sign.",
                    textData.PeekChar(),
                    textData.Position,
                    WarningType.Error);
                textData.AdvanceChar();
                break;
        }
    }
}

void ParseUnsignedDecimal()
{
    while (true)
    {
```

```
switch (textData.PeekChar())
{
    case '1': case '2': case '3': case '4': case '5': case '6': case '7': case '8': case '9':
        mode = DecimalParseMode.DecimalConstWithNull;
        SaveCharacter();
        textData.AdvanceChar();
        return;
    case '0':
        mode = DecimalParseMode.NullStartDecimal;
        SaveCharacter();
        textData.AdvanceChar();
        return;
    case '\0': case ' ': case '\t': case '\n':
        MakeWarningMinimal(
            "Value cannot be empty",
            textData.PeekChar(),
            textData.Position,
            WarningType.Error);
        mode = DecimalParseMode.Ending;
        return;
    default:
        MakeWarning(
            "Unknown character! There can only be digit from 1 to 9",
            textData.PeekChar(),
            textData.Position,
            WarningType.Error);
        textData.AdvanceChar();
        break;
}
```

```
    }
}

void ParseUnsignedDecimalWithNull()
{
    while (true)
    {
        switch (textData.PeekChar())
        {
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                SaveCharacter();
                textData.AdvanceChar();
                break;
            case '.':
                mode = DecimalParseMode.UnsignedIntegerWithExponent;
                SaveCharacter();
                textData.AdvanceChar();
                return;
            case ',':
                mode = DecimalParseMode.UnsignedIntegerWithExponent;
```

```
            SaveCharacter('.');
            MakeWarning(
                "There can only be digit from 0 to 9 or '.' character",
                textData.PeekChar(),
                textData.Position,
                WarningType.Warning);
            textData.AdvanceChar();
            return;
        case 'E':
        case 'e':
            mode = DecimalParseMode.SignedInteger;
            SaveCharacter();
            textData.AdvanceChar();
            return;
        case '\0':
        case ' ':
        case '\t':
        case '\n':
            mode = DecimalParseMode.Ending;
            return;
        default:
            MakeWarning(
                "Unknown character! There can only be digit from 0 to 9 or '.' character",
                textData.PeekChar(),
                textData.Position,
                WarningType.Error);
            textData.AdvanceChar();
            break;
    }
```

```
    }
}

void ParseUnsignedIntegerWithExponent()
{
    while (true)
    {
        switch (textData.PeekChar())
        {
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                SaveCharacter();
                textData.AdvanceChar();
                break;
            case 'E':
            case 'e':
                mode = DecimalParseMode.SignedInteger;
                SaveCharacter();
                textData.AdvanceChar();
                return;
            case '\0':
```

```
            case ' ':
            case '\t':
            case '\n':
                mode = DecimalParseMode.Ending;
                return;
            default:
                MakeWarning(
                    "Unknown character! There can only be digit from 0 to 9",
                    textData.PeekChar(),
                    textData.Position,
                    WarningType.Error);
                textData.AdvanceChar();
                break;
        }
    }
}

void ParseNullStartDecimal()
{
    while (true)
    {
        switch (textData.PeekChar())
        {
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
```

```
case '6':
case '7':
case '8':
case '9':
    mode = DecimalParseMode.DecimalConstWithNull;
    SaveCharacter();
    MakeWarning(
        "Null at the beginning is excess",
        textData.PeekChar(),
        textData.Position,
        WarningType.Warning);
    textData.AdvanceChar();
    return;
case 'E':
case 'e':
    mode = DecimalParseMode.SignedInteger;
    SaveCharacter();
    MakeWarning(
        "Null with exponent equal null",
        textData.PeekChar(),
        textData.Position,
        WarningType.Warning);
    textData.AdvanceChar();
    return;
case '.':
    mode = DecimalParseMode.UnsignedIntegerWithExponent;
    SaveCharacter();
    textData.AdvanceChar();
    return;
```

```
            case ',':
                mode = DecimalParseMode.UnsignedIntegerWithExponent;
                SaveCharacter('.');
                MakeWarning(
                    "There can only be digit from 0 to 9 or '.' character",
                    textData.PeekChar(),
                    textData.Position,
                    WarningType.Warning);
                textData.AdvanceChar();
                return;
            case '\0':
            case ' ':
            case '\t':
            case '\n':
                mode = DecimalParseMode.Ending;
                return;
            default:
                MakeWarning(
                    "Unknown character! There can only be digit from 0 to 9",
                    textData.PeekChar(),
                    textData.Position,
                    WarningType.Error);
                textData.AdvanceChar();
                break;
        }
    }
}
```

```
void ParseSignedInteger()
{
    while (true)
    {
        switch (textData.PeekChar())
        {
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                mode = DecimalParseMode.UnsignedIntegerWithNull;
                SaveCharacter();
                textData.AdvanceChar();
                return;
            case '0':
                mode = DecimalParseMode.NullStartInteger;
                SaveCharacter();
                textData.AdvanceChar();
                return;
            case '+': case '-':
                mode = DecimalParseMode.UnsignedInteger;
                SaveCharacter();
                textData.AdvanceChar();
                return;
```

```
            case '\0':
            case ' ':
            case '\t':
            case '\n':
                MakeWarningMinimal(
                    "Exponent cannot be empty",
                    textData.PeekChar(),
                    textData.Position,
                    WarningType.Error);
                mode = DecimalParseMode.Ending;
                return;
            default:
                MakeWarning(
                    "Unknown character! There can only be digit from 0 to 9 or sign",
                    textData.PeekChar(),
                    textData.Position,
                    WarningType.Error);
                textData.AdvanceChar();
                break;
        }
    }
}

void ParseNullStartInteger()
{
    while (true)
    {
        switch (textData.PeekChar())
        {
```

```
case '0':
case '1':
case '2':
case '3':
case '4':
case '5':
case '6':
case '7':
case '8':
case '9':
    mode = DecimalParseMode.UnsignedIntegerWithNull;
    SaveCharacter();
    MakeWarning(
        "Null at the beginning is excess",
        textData.PeekChar(),
        textData.Position,
        WarningType.Warning);
    textData.AdvanceChar();
    return;
case '\0':
case ' ':
case '\t':
case '\n':
    mode = DecimalParseMode.Ending;
    return;
default:
    MakeWarning(
        "Unknown character! There can only be digit from 0 to 9 or sign",
        textData.PeekChar(),
```

```
                textData.Position,
                WarningType.Error);
            textData.AdvanceChar();
            break;
        }
    }
}

void ParseUnsignedInteger()
{
    while (true)
    {
        switch (textData.PeekChar())
        {
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                mode = DecimalParseMode.UnsignedIntegerWithNull;
                SaveCharacter();
                textData.AdvanceChar();
                return;
            case '0':
                mode = DecimalParseMode.NullStartInteger;
```

```
                SaveCharacter();
                textData.AdvanceChar();
                return;
            case '\0':
            case ' ':
            case '\t':
            case '\n':
                MakeWarningMinimal(
                  "Value cannot be empty",
                  textData.PeekChar(),
                  textData.Position,
                  WarningType.Error);
                mode = DecimalParseMode.Ending;
                return;
            default:
                MakeWarning(
                   "Unknown character! There can only be digit from 0 to 9 or sign",
                   textData.PeekChar(),
                   textData.Position,
                   WarningType.Error);
                textData.AdvanceChar();
                break;
        }
    }
}

void ParseUnsignedIntegerWithNull()
{
    while (true)
```

```
{
    switch (textData.PeekChar())
    {
        case '0':
        case '1':
        case '2':
        case '3':
        case '4':
        case '5':
        case '6':
        case '7':
        case '8':
        case '9':
            SaveCharacter();
            textData.AdvanceChar();
            break;
        case '\0':
        case ' ':
        case '\t':
        case '\n':
            mode = DecimalParseMode.Ending;
            return;
        default:
            MakeWarning(
                "Unknown character! There can only be digit from 0 to 9",
                textData.PeekChar(),
                textData.Position,
                WarningType.Error);
            textData.AdvanceChar();
```

```
                break;
            }
        }
    }

    void SaveCharacter(char value)
    {
        resultString.Append(value);
    }

    void SaveCharacter() => SaveCharacter(textData.PeekChar());

    void MakeWarning(string text, char character, int position, WarningType warningType)
    {
        warnings.Add(new Warning(text, character, position, warningType));
    }

    void MakeWarningMinimal(string text, char character, int position, WarningType warningType)
    {
        warnings.Add(new WarningMinimal(text, character, position, warningType));
    }
}

class Warning
{
    public Warning(string text, char character, int position, WarningType warningType)
    {
        Text = text;
        Position = position;
```

```
        WarningType = warningType;
        Character = character;
    }

    public string Text { get; }
    public int Position { get; }
    public WarningType WarningType { get; }
    public char Character { get; }

    public override string ToString()
    {
        return $"{WarningType}: Chartacter '{Character}' at position {Position};\nInfo: {Text};";
    }
}

class WarningMinimal : Warning
{
    public WarningMinimal(string text, char character, int position, WarningType warningType)
        : base(text, character, position, warningType)
    {
    }

    public override string ToString()
    {
        return $"{WarningType}: Position {Position};\nInfo: {Text};";
    }
}

enum DecimalParseMode:byte
```

```
    {
        DecimalConst,
        UnsignedDecimalConst,
        DecimalConstWithNull,
        NullStartDecimal,
        UnsignedIntegerWithExponent,
        UnsignedIntegerWithNull,
        UnsignedInteger,
        NullStartInteger,
        SignedInteger,
        Ending,
    }

    enum WarningType:byte
    {
        Error,
        Warning,
    }
}
```