CENTRO *UNIVERSITÁRIO* DE BARRA MANSA
ACADEMIC PRO-RECTORY

COMPUTER ENGINEERING COURSE

FORTRAN NUMERICAL CONSTANTS RECOGNIZER

By:

Leniel Braz de Oliveira Macaferi
Wellington Magalhães Leite

Barra Mansa
March 7, 2006

CENTRO *UNIVERSITÁRIO* DE BARRA MANSA
ACADEMIC PRO-RECTORY

COMPUTER ENGINEERING COURSE

FORTRAN NUMERICAL CONSTANTS RECOGNIZER

By:

Leniel Braz de Oliveira Macaferi
Wellington Magalhães Leite

Paper presented to the Computer Engineering course at Centro Universitário de Barra Mansa, as a partial requisite to the obtention of the first grade related to the Compilers Construction discipline, under prof. José Nilton Cantarino Gil supervision.

Barra Mansa
March 7, 2006

ABSTRACT

A didactic method for the construction of a compiler front-end is the one substantiated in transition diagrams. So, its construction helps with the familiarization regarding the tasks of a compiler project.
This paper presents a Fortran numerical constants recognizer. It is developed based on a state transition diagram and its implementation follows the standards of the C# programming language.

Keywords: fortran, compiler construction, state transition diagram, C# programming language

LIST OF FIGURES

CONTENTS

Page

1 INTRODUCTION

1.1 Objective

Our objective in the present work is to implement a FORTRAN numerical constants recognizer using the C# programming language. The recognizer must cover the specifications of a state transition diagram. For that purpose we'll create a console project within the Microsoft Visual C# Express Edition 2005 development environment.

1.2 Definition

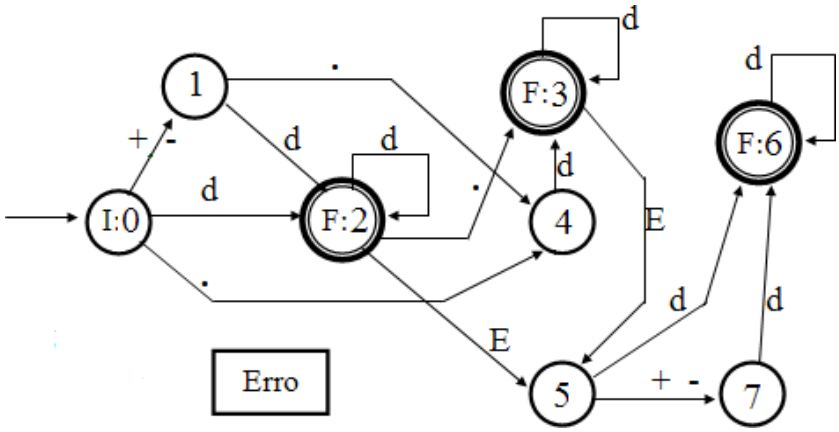

Figure 1 - State Transition Diagram for a FORTRAN numerical constants recognizer

Alphabet ∑ = {d, +, - , . , E} (d is any digit)

The arches of any vertex to an Error vertex aren't showed but they exist.

Expressions like the following are recognized: 13, +13, -13, 13., 1.25, .25, -.25, -32.43, 13E-15, 13.E-15, -13.25E+72, .75E5, etc.

Expressions like the following aren't recognized: ++13, .E13, etc.

The vertex with a 0 (zero) value is the initial state.

The vertexes with bold circles are the final states.

2 DEVELOPMENT

2.1 Mapping the constants

```
/// <summary>
/// Enumeration where each item corresponds to one state in the State
Transition Diagram.
/// </summary>
public enum PossibleStates
{
  s0 = 0,
  s1,
  s2,
  s3,
  s4,
  s5,
  s6,
  s7,
  s8,
  error
}
```

```
/// <summary>
/// Array of type PossibleStates, which contains the finalStates
acceptable by the State Transition Diagram.
/// </summary>  public static possibleStates[] finalStates = {
possibleStates.s2, possibleStates.s3, possibleStates.s6 };
```

2.2 Mapping the functions

```
/// <summary>
/// Recognizes the current state and the character "label" being analysed,
values passed as parameters. After, the function does the transition of
state case some condition is satisfied, otherwise, the function will
return an error flag.
/// </summary>
public static PossibleStates Recognizer(PossibleStates currentState, char
c)
{
  switch(currentState)
  {
    case PossibleStates.s0:
    {
      if(c == '+' || c == '-')
        return PossibleStates.s1;

      if(char.IsDigit(c))
        return PossibleStates.s2;

      if(c == '.')
        return PossibleStates.s4;

      break;
    }

    case PossibleStates.s1:
    {
      if(char.IsDigit(c))
```

```
        return PossibleStates.s2;

      if(c == '.')
        return PossibleStates.s4;

      break;
    }

    case PossibleStates.s2:
    {
      if(char.IsDigit(c))
        return PossibleStates.s2;

      if(c == '.')
        return PossibleStates.s3;

      if(c == 'E')
        return PossibleStates.s5;

      break;
    }

    case PossibleStates.s3:
    {
      if(char.IsDigit(c))
        return PossibleStates.s3;

      if(c == 'E')
        return PossibleStates.s5;

      break;
    }

    case PossibleStates.s4:
    {
      if(char.IsDigit(c))
        return PossibleStates.s3;

      break;
    }

    case PossibleStates.s5:
    {
      if(char.IsDigit(c))
        return PossibleStates.s6;

      if(c == '+' || c == '-')
        return PossibleStates.s7;

      break;
    }

    case PossibleStates.s6:
    {
      if(char.IsDigit(c))
        return PossibleStates.s6;

      break;
    }

    case PossibleStates.s7:
```

```
    {
      if(char.IsDigit(c))
        return PossibleStates.s6;

      break;
    }
  }
  return PossibleStates.error;
}

/// <summary>
/// Reads an input expression, recognizes its characters, changes the
states accordingly to those characters and hence validates the entry.
/// </summary>
public static void SingleExpression()
{
  do
  {
    // The machine points to the initial state of the State Transition
Diagram.
    PossibleStates currentState = PossibleStates.s0;

    Console.Write("\n\nEnter the expression to be evaluated: ");

    // strExpression receives the entry typed by the user.
    string strExpression = Console.ReadLine();

    /* For each string's character (label), calls the function Recognizer
that
    on the other hand changes the machine state accordingly. */
    for(int i = 0; strExpression.Length > i; ++i)
      if(currentState != PossibleStates.error)
        currentState = Recognizer(currentState, strExpression[i]);
      else
        break;

    /* Calls the function IsFinalState to verify if the state where the
machine
     stopped is a final state or not. */
    if(IsFinalState(currentState))
      Console.WriteLine("\n Valid expression.\n");
    else
      Console.WriteLine("\n Invalid expression!\n");

    Console.Write("Do you wanna try again? (y\\n) ");
  }
  while(Console.ReadKey().KeyChar == 'y');
}

/// <summary>
/// Reads an input file, recognizes its lines, expression by expression
and changes the states accordingly to each expression. In other words,
validates the entire list.
/// </summary>
public static void SetOfExpressions()
{
  do
  {
    Console.Write("\n\nEnter the file path: ");

    // Obtains the file name.
```

```
    string fileName = Console.ReadLine();

    // Verifies if the file exists.
    if(!File.Exists(fileName))
      Console.Write("\n File not found!\n\n");
    else
    {
      // Reads all the file's lines and stores them.
      StreamReader sReader = new StreamReader(fileName);

      string expression;

      // Evaluates each line until achieve the EOF (end of file).
      while((expression = sReader.ReadLine()) != null)
      {
        // The machine points to the initial state of the State Transition
Diagram.
        PossibleStates currentState = PossibleStates.s0;

        /* For each expression's character (label), calls the function
Recognizer that
        on the other hand changes the machine state accordingly. */
        for(int i = 0; expression.Length > i; ++i)
          if(currentState != PossibleStates.error)
            currentState = Recognizer(currentState, expression[i]);
          else
            break;

        /* Calls the function IsFinalState to verify if the state where
the machine
         stopped for the expression is a final state or not. */
        if(IsFinalState(currentState))
          Console.WriteLine("\n{0} is a valid expression.", expression);
        else
          Console.WriteLine("\n{0} is an invalid expression!",
expression);
      }
      sReader.Close();
    }
    Console.Write("\nDo you wanna try again? (y\\n) ");
  }
  while(Console.ReadKey().KeyChar == 'y');
}
```

2.3 Application main entry point

```
public static void Main(string[] args)
{
  Console.Title = "State Transition System for recognizing numeric
constants in FORTRAN";

  Console.BackgroundColor = ConsoleColor.White;
  Console.ForegroundColor = ConsoleColor.Black;

  char ch;

  do
  {
    Console.Clear();
```

```
    // Print startup banner
    Console.Write("\nState Transition System C# Sample Application\n");
    Console.Write("Copyright ©2006 Leniel Braz de Oliveira Macaferi &
Wellington Magalhães Leite.\n\n");
    Console.Write("UBM COMPUTER ENGINEERING - 7TH SEMESTER
[http://www.ubm.br/]\n\n");

    // Describes program function
    Console.Write("This program example demonstrates the State Transition
Diagram's algorithm for\n");
    Console.Write("numeric constants validation in FORTRAN.\n\n");

    // Describes program's options
    Console.Write("You can validate expressions by two different ways as
follow:\n\n");
    Console.Write("1 - A single expression by providing an entry.\n\n");
    Console.Write("2 - A set of expressions by providing an input
file.\n");
    Console.Write("  * Notice: the expressions must be separeted in-
lines.\n");

    Console.Write("\n\nEnter your choice: ");

    ch = Console.ReadKey().KeyChar;

    switch(ch)
    {
      case '1':
      {
        SingleExpression();
        break;
      }
      case '2':
      {
        SetOfExpressions();
        break;
      }
    }
  }
  while(ch == '1' || ch == '2');
}
```

3 APPLICATION

3.1 Validating expressions

Validation of expressions is carried out upon expressions provided by the user with the aid of standard computer input output, that is, a keyboard and a monitor.

Validation can be done on a single expression or on a set of expressions written in a text file. In such a case the expressions must be separated in lines.

3.1.1 Single expression

Constants described in section 1.2 typed one by one.

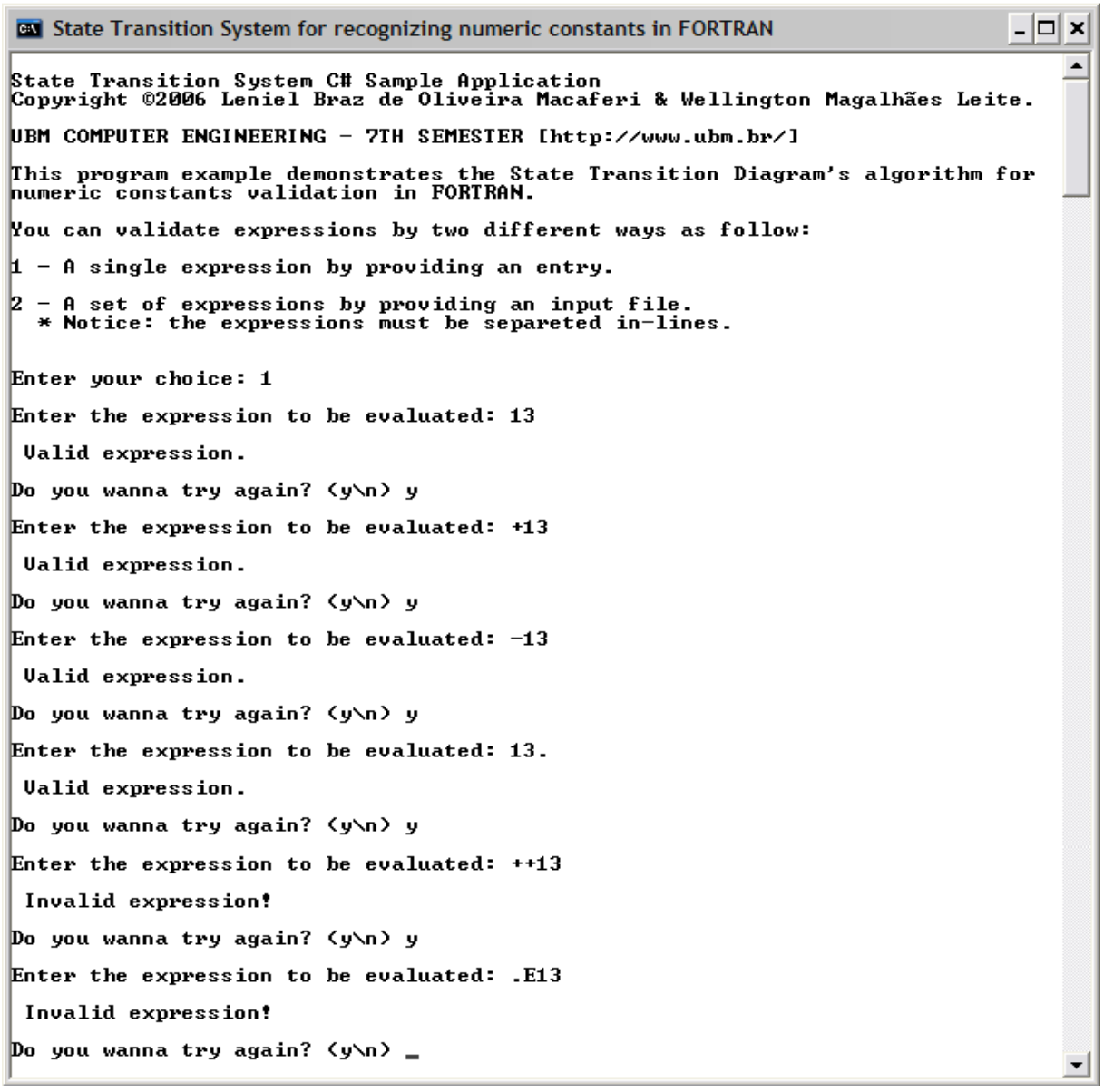

Figure 2 - Single expression - constants typed one by one

3.1.2 Set of expressions

Constants described in section 1.2 written in a text file.

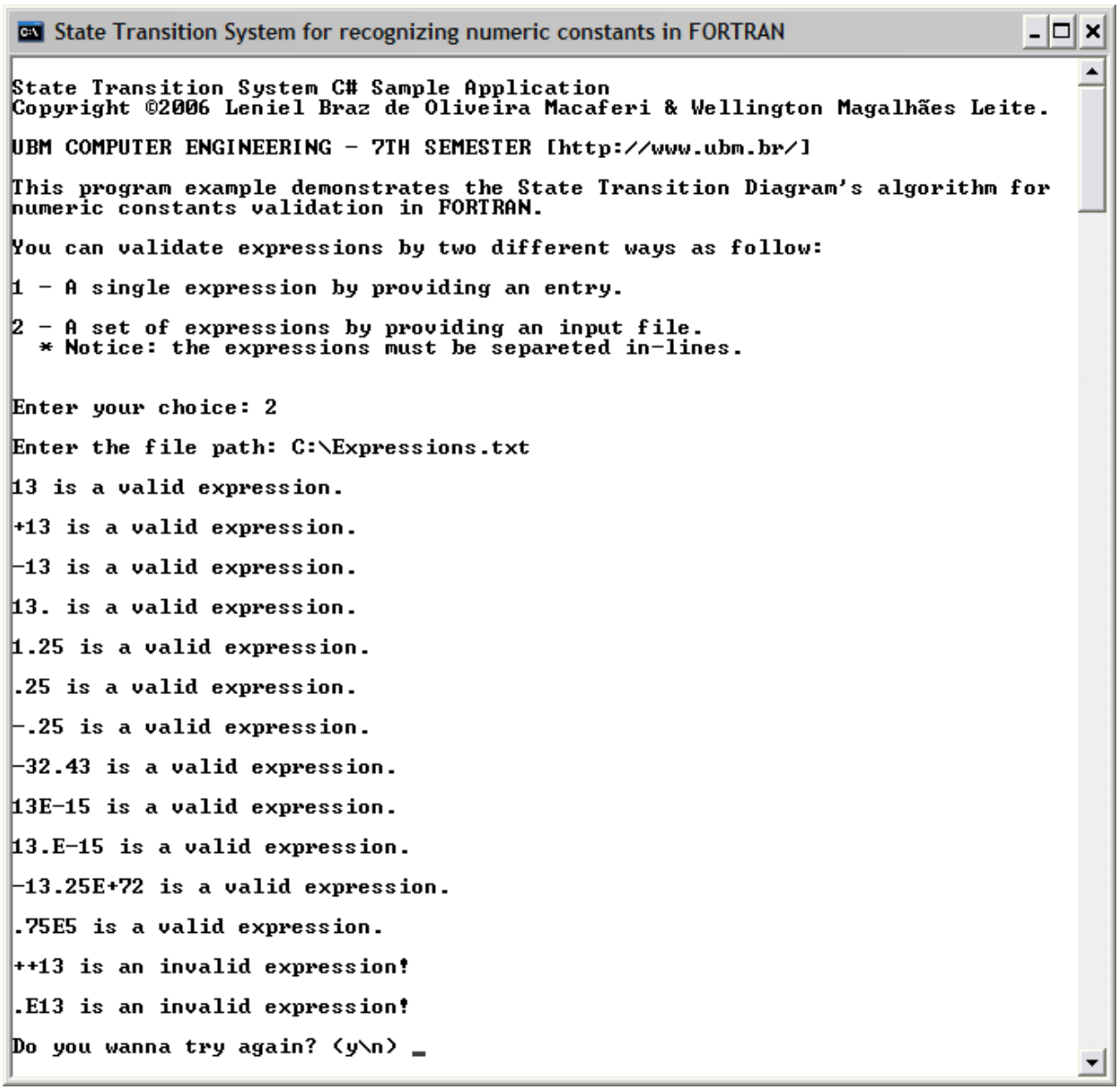

Figure 3 - Set of expressions - constants are read from a text file

4 CONCLUSION

Developing this work we had the opportunity to take a step further in our study about compilers construction, even knowing that the lexical state transition diagram shown has few states (note: lexical state transition diagrams are finite automatons).

One of the interesting points was the fact that we could explore the potential of a programming language, in our case the C# language so that we could develop an efficient program. This way we got in contact with tools that we hadn't used yet.

All the tests we've done with the console application satisfy the lexical state transition diagram to recognize Fortran numerical constants as shown in the Application section of this document.

The most important thing is that this work gives us an overview of how we need to proceed in other more complex cases, for it helps us to understand the tasks that make part of a compiler project. Therefore, we can master the concepts of compiler construction what is of great value for a computer engineer.

This paper and the Fortran numerical constants recognizer files can be downloaded at:
http://leniel.net/

5 REFERENCES

[1] ROQUE, K. **Microsoft® C# Segredos da Linguagem**. 1ª ed. Rio de Janeiro : Campus, 2001.

[2] Microsoft. **Microsoft Visual C# 2005 Express Edition**. Available at <http://msdn.microsoft.com/vstudio/express/visualcsharp/default.aspx>. Accessed February 28, 2006.

[3] Mokarzel, Fabio Carneiro. **Class notes about compilers**. Available at <http://www.comp.ita.br/professores/fabio.htm>. Accessed February 28, 2006.

6 ADDENDUM

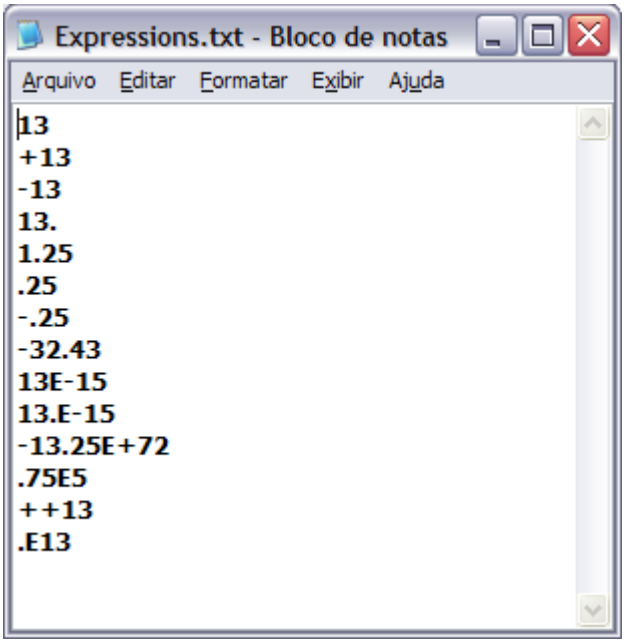

Figure 4 - File used to store the set of expressions described in section 1.2.