ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО
Доцент факультета компьютерных наук, заместитель декана по учебно-методической работе, канд. социол. наук

_____ И. Ю. Самоненко
«_____» _____ 2020 г.

УТВЕРЖДАЮ
Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук

_____ В. В. Шилов
«_____» _____ 2020 г.

Приложение для визуализации метода рекурсивного спуска

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.13-01 ТП 01-1-ЛУ

Исполнитель:
студент группы БПИ 199
_____ К. Н. Борисов
«_____» _____ 2020 г.

Москва 2020

УТВЕРЖДЕН
RU.17701729.04.13-01 ТП 01-1

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.04.13-01 ТП 01-1				

Приложение для визуализации метода рекурсивного спуска

Текст программы

RU.17701729.04.13-01 ТП 01-1

Листов 12

Москва 2020

Содержание

1 Текст программы

1.1 ParserApp/App.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Threading.Tasks;
using System.Windows;

namespace ParserApp {
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application {
    }
}
```

1.2 ParserApp/AssemblyInfo.cs

```
using System.Windows;

[assembly: ThemeInfo(
    ResourceDictionaryLocation.None, //where theme specific resource
    ↪  dictionaries are located
                                     //(used if a resource is not found in
                                     ↪  the page,
                                     // or application resource
                                     ↪  dictionaries)
    ResourceDictionaryLocation.SourceAssembly //where the generic resource
    ↪  dictionary is located
                                               //(used if a resource is not
                                               ↪  found in the page,
                                               // app, or any theme specific
                                               ↪  resource dictionaries)
)]
```

1.3 ParserLib/HistoryEntry.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace ParserLib {
    public class HistoryEntry : IEnumerable<HistoryToken> {
        public HistoryToken[] TreeRanges { get; } // can be private
        public string RtfGrammar { get; }
        public int CursorPos { get; }

        internal HistoryEntry(HistoryToken[] ranges, string rtf) {
            RtfGrammar = rtf;
            TreeRanges = ranges;
            CursorPos = TreeRanges.Max(e => Math.Max(e.StartPos, e.EndPos -
            ↪  1)) + 1;
            CalculateDisplayLevels();
        }

        private void AssignDisplayLevels() {
            var maxRecLevel = this.Max(e => e.RecLevel);
            foreach (var tok in this) {
                tok.DisplayLevel = maxRecLevel - tok.RecLevel;
            }
        }

        private void CalculateDisplayLevels() {
            int[] recLvs = new int[CursorPos];

            Span<int> Slice(HistoryToken tok) {
                var end = tok.EndPos;
                if (end == -1) end = CursorPos;
                return new ArraySegment<int>(recLvs, tok.StartPos, end -
                ↪  tok.StartPos);
            }

            foreach (var tok in this.OrderBy(e => -e.RecLevel)) {
                tok.DisplayLevel = Slice(tok).ToArray().Max();
                foreach (ref int e in Slice(tok))
                    e = tok.DisplayLevel + 1;
            }
        }

        // todo: public SortLevels
        public override string ToString() {
            return string.Join(' ', (object[])TreeRanges);
        }

        public IEnumerator<HistoryToken> GetEnumerator() {
            return TreeRanges.Where(e =>
            ↪  !e.Name.StartsWith('"')).GetEnumerator();
        }
```

```
        IEnumerator IEnumerable.GetEnumerator() {
            return GetEnumerator();
        }
    }
}
```

1.4 ParserLib/HistoryToken.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ParserLib {
    public class HistoryToken {
        public string Name { get; }
        public int StartPos { get; }
        public int EndPos { get; }

        internal int RecLevel { get; }
        public bool Trimmable { get; }
        public int DisplayLevel { get; internal set; }

        internal HistoryToken(ParserTreeToken tok) {
            Name = tok.Name;
            StartPos = tok.StartPos;
            EndPos = tok.EndPos;
            RecLevel = tok.RecLevel;
            Trimmable = tok.ChildCount == 1 && tok.EndPos >= 0;
        }

        public override string ToString() {
            if (EndPos == -1) return $"{StartPos}:-({Name}, {RecLevel})";
            return $"{StartPos}:{EndPos}({Name}, {RecLevel})";
        }
    }
}
```

1.5 ParserLib/Parser.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
```

```
using System.Diagnostics;

namespace ParserLib {
    public abstract class Parser {
        abstract public ParserHistory Run(string input);
    }
}
```

1.6 ParserLib/ParserHistory.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ParserLib {
    public class ParserHistory : IEnumerable<HistoryEntry> {
        private Stack<ParserTreeToken> stack = new Stack<ParserTreeToken>();
        private List<ParserTreeToken> state = new List<ParserTreeToken>();
        private int prevPos = -1;

        /// <summary>
        /// Inherited parser parameters.
        /// </summary>
        public string OriginalRtf { get; }
        public string InputString { get; }

        public IEnumerable<string> RuleNames =>
        ↪  RtfBuilder.GetNames(OriginalRtf);

        private List<HistoryEntry> history = new List<HistoryEntry>();

        private HistoryToken[] CopyState() {
            return state.Select(e => e.Clone()).ToArray();
        }

        internal ParserHistory(string rtf, string input) {
            OriginalRtf = rtf;
            InputString = input;
        }

        private void SaveState() {
            var tokens = CopyState();
            var rtf = RtfBuilder.Build(OriginalRtf, stack);
            var r = new HistoryEntry(tokens, rtf);
```

```
    history.Add(r);
}

public void Add(string line) {
    if (line == null) return;
    line = line.Trim();
    if (line == "") return;

    if (line.StartsWith("eval failed: SyntaxError:")) {
        // todo
        return;
    }

    var words = line.Split(' ',
    ↪  StringSplitOptions.RemoveEmptyEntries);
    var pos = int.Parse(words[0].Split(':').Last()) - 1;

    var hasFailed = prevPos > pos;
    while (prevPos > pos) {
        var t = state.Last();
        if (t.EndPos == -1) break;
        t.Parent.ChildCount--;
        state.RemoveAt(state.Count - 1);
        prevPos = t.EndPos;
    }
    prevPos = pos;

    if (hasFailed) SaveState();

    if (words[1] == "rule.enter") {
        var val = 0;
        ParserTreeToken parent = null;
        if (stack.Count != 0) {
            parent = stack.Peek();
            parent.ChildCount++;
            var dict = stack.Peek().Dict;
            dict.TryGetValue(words[2], out val);
            dict[words[2]] = val + 1;
        }
        var t = new ParserTreeToken(parent, words[2], val, pos,
        ↪  stack.Count);
        state.Add(t);
        stack.Push(t);
    } else {
        var t = stack.Pop();
        t.EndPos = pos;
        if (words[1] != "rule.match") {
            t.EndPos = -2;
            t.Parent.ChildCount--;
```

```
                    var i = state.IndexOf(t);
                    state.RemoveRange(i, state.Count - i);
                }
            }

            SaveState();
        }

        public IEnumerator<HistoryEntry> GetEnumerator() =>
        ↪ history.GetEnumerator();
        IEnumerator IEnumerable.GetEnumerator() => history.GetEnumerator();
        public HistoryEntry this[int i] => history[i];
    }
}
```

1.7 ParserLib/ParserSpawner.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Diagnostics;

namespace ParserLib {
    public class ParserSpawner : Parser {
        private string name;
        public ParserSpawner(string name) {
            this.name = name;
        }

        public override ParserHistory Run(string input) {
            var rtf = File.ReadAllText("parsers/" + this.name + ".rtf");
            var tree = new ParserHistory(rtf, input);

            var process = new Process();
            process.StartInfo.CreateNoWindow = true;
            process.StartInfo.UseShellExecute = false;
            process.StartInfo.RedirectStandardInput = true;
            process.StartInfo.RedirectStandardOutput = true;
            process.OutputDataReceived += (sender, args) => {
                tree.Add(args.Data);
            };
            process.StartInfo.FileName = "parsers/" + this.name + ".exe";
            process.Start();
            process.BeginOutputReadLine();

            process.StandardInput.Write(input);
```

```
            process.StandardInput.Close();

            process.WaitForExit();
            return tree;
        }
    }
}
```

1.8 ParserLib/ParserTreeToken.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Diagnostics;

namespace ParserLib {
    internal class ParserTreeToken {
        public ParserTreeToken Parent { get; }
        public string Name { get; }
        public int Index { get; }
        public int StartPos { get; }
        public int RecLevel { get; }
        public int EndPos { get; set; }
        public int ChildCount { get; set; }
        public Dictionary<string, int> Dict { get; }

        public ParserTreeToken(ParserTreeToken parent, string name, int
        ↪  index, int startPos, int recLevel) {
            if (name.StartsWith('_') && name.EndsWith('_') && name.Length >
            ↪  1) {
                var chars = Enumerable.Range(0, name.Length / 2 - 1)
                    .Select(i => (char)Convert.ToUInt16(name.Substring(i * 2
                    ↪  + 1, 2), 16));
                name = '"' + string.Join("", chars) + '"';
            }
            this.Parent = parent;
            this.Name = name;
            this.Index = index;
            this.StartPos = startPos;
            this.RecLevel = recLevel;
            this.EndPos = -1;
            this.Dict = new Dictionary<string, int>();
        }

        public override string ToString() {
            return $"{StartPos}:{EndPos}({Name}, {Index})";
```

```
        }

        public HistoryToken Clone() {
            return new HistoryToken(this);
        }
    }
}
```

1.9 ParserLib/Program.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ParserLib {
    class Program {
        static void Main() {
            Console.WriteLine("Main");

            ↪  Directory.SetCurrentDirectory(AppDomain.CurrentDomain.BaseDirectory);
            Parser parser = new ParserSpawner("simple");
            var sw = new Stopwatch();
            sw.Start();
            var tree = parser.Run("1+1");
            sw.Stop();
            foreach (var item in tree) {
                Console.WriteLine(item);
            }
            Console.WriteLine(sw.Elapsed);
        }
    }
}
```

1.10 ParserLib/RtfBuilder.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.IO;
using System.Diagnostics;

namespace ParserLib {
```

```
    internal class RtfBuilder {
        private string[] lines;

        private RtfBuilder(string rtf) {
            lines = rtf.Split('\n');
             ↪  //.Where(e=>e.StartsWith("\\cf")).ToArray();
        }

        private void HighlightIdentifier(ParserTreeToken t) {
            if (t.Parent == null) return;
            var lineIndex = Array.FindIndex(lines, e => e.StartsWith("\\cf2
             ↪  " + t.Parent.Name));
            var line = lines[lineIndex];

            var i = 0;
            var regex = @"\\b\{\}(\\cf[0-9] )" + Regex.Escape(t.Name);
            line = Regex.Replace(line, regex, m => {
                if (i++ == t.Index) return @"\b " + m.Groups[1].Value +
                 ↪  t.Name;
                return m.Groups[1].Value + t.Name;
            });
            line = line.Replace("\\b{}", "");

            lines[lineIndex] = line;
        }

        private string End() {
            return string.Join('\n', lines).Replace("{}", "0");
        }

        public static string Build(string rtf, IEnumerable<ParserTreeToken>
         ↪  tokens) {
            var builder = new RtfBuilder(rtf);
            foreach (var tok in tokens) {
                builder.HighlightIdentifier(tok);
            }
            return builder.End();
        }

        public static IEnumerable<string> GetNames(string rtf) {
            return rtf.Split('\n')
                .Where(e => e.StartsWith("\\cf"))
                .Select(e => e.Split("\\cf4")[0].Split(' ')[1]);
        }
    }
}
```

2 Лист регистрации изменений

Изм.	Номера листов				Всего листов в документе	№ документа	Входящий № сопроводит. докум. и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					