NET LL(1) grammar parser

(Započtový program C#(zima) a Advanced .NET (leto). Pavel Vigilev)

Uživatelska dokumentace.

1.C#-ova čast

Aby mohl využit framework, uživatel musi přidat odkaz na assembly frameworku (LL1_Parser.dll).
Třida *LL1GrammarParser* (namespace *LL1_Parser*) je hlavni třidou, s kterou uživatel pracuje.

Obsah namespacu *LL1_Parser*:

1. Namespace obsahuje definici rozhraní *IToken*, *ITokenWithValue* a *ILexer*.
   Uživatelske datove typy tokenů a lexeru musí se dědit z těchto rozhrani.
2. Instance třidy *LL1GrammarParser<T, UToken>* je recursive descent parser pro LL(1) gramatiky.
   Generic-parameter *T* definuje typ hodnoty, kterou framework parsuje; *UToken* - typ datove struktury popisujuci uživatlske tokeny.

Třida *LL1GrammarParser* má nasledujici konstruktor:
LL1GrammarParser(string grammar, ILexer<UToken> lexer, IDictionary<string, UToken> NameTerminalTable, Assembly[] assemblies):

- *grammar* - obsah souboru s gramatikou
- *lexer* - objekt uživatelske třidy lexeru, ktera tvoři ze vstupniho řetězcu, posloupnost tokenu typu UToken.
- *NameTerminalTable* - tabulka jmen terminalu, aby framework věděl, ktery symbol je v popisu gramatiky terminal.
- *assemblies* - pole assembly, kde framework hleda typy a metody s kterymi pracuje v ramci parsovani

Metody *T Parse(string str)* a *bool TryParse(string str, out T result)* maji stejnou semantiku, jakou maji *int.Parse* a *int.TryParse* resp.

Rozhrani *IToken*, *ITokenWithValue* a *ILexer*:

```
interface IToken
{
    bool IsCompatible(IToken other);
}

interface ITokenWithValue<out T> : IToken where T : class
{
    T Value { get; }
}

interface ILexer<T> where T : IToken
{
    IList<T> Tokenize(string str);
}
```

2.Soubor s gramatikou

Gramatika se popisuje nasledujicim zpusobem:

```
nonterminal_name : symbol1 symbol2 ... symboln { expression; expression; ... expression; }
```

kde *symbol* je jmeno terminalu nebo neterminalu.
Nove neterminaly se definuji automaticky. Pokud na prave strane pravidla vyskytuje symbol "symbol" a neexistuje terminal s takovem jmenem, registruje se novy neterminal s timto jmenem.
Jmeno terminalu je jedneho z dvou typů:

Vyruzy v '{' a '}'

Složite zavorky naproti každomu pravidlu obsahuji neprazdnou posloupnost vyrazu. Každy vyraz podporuje zavolani instančni metody, staticke metody, kostruktoru, využiti literalu int, double, string a čislo vysledku parsingu (*$uint*).
int, string, double lze využit jako podobne literaly v C#.
$i odkazuje na rozparsovany symbol v pravidle, kde vyskytuje.

Zavolani metod maji nasledujici syntax. Jako parametr lze využit nějaky jiny libovolny vyraz.

```
// Instance-method invokation
FullMethodName(instance, arg1, arg2, ..., argn);

// static-method invokation
static FullMethodName(arg1, ... argn);

// constructor invokation
new FullTypeName(arg1, ..., argn);
```

FullMethodName (podobně i *FullMethodType*) je uplně jmeno metody včetně všech jmenovych prostoru a tříd, kde se nachazi tato metoda.

Příklad:
grammar file:

```
Expr : var { new Lambda.LVar($0); }
Expr : '(' Expr1 ')' { $1; }
Expr1 : Expr Expr { new Lambda.LApplication($0, $1); }
Expr1 : '\' var '.' Expr { new Lambda.LAbstraction(new Lambda.LVar($1), $3); }
```

'(', ')', '\', '.', var jsou terminaly a musi byt definovany v *NameTerminalTable* (v přikladu je uvnitř lexeru - *lexer.NameTokenTBL*).

data structure:

```
    public abstract class LExprBasic{}

    public class LVar :LExprBasic
    {
        public string Value { get; }
        public LVar(string val)
        {
            Value = val ?? throw new ArgumentNullException();
        }
        public override string ToString()
        {
            return Value;
        }
    }


    public class LApplication : LExprBasic
```

```
    {
        public LExprBasic What { get; }
        public LExprBasic To { get; }


        public LApplication(LExprBasic w, LExprBasic t)
        {
            What = w ?? throw new ArgumentNullException();
            To = t ?? throw new ArgumentNullException();
        }
        public override string ToString()
        {
            return $"({What.ToString()} {To.ToString()})";
        }
    }

    public class LAbstraction : LExprBasic
    {
        public LVar Variable { get; }
        public LExprBasic Expression { get; }
        public LAbstraction(LVar var, LExprBasic expr)
        {
            Variable = var ?? throw new ArgumentNullException();
            Expression = expr ?? throw new ArgumentNullException();
        }
        public override string ToString()
        {
            return $"(\\{Variable}.{Expression.ToString()})";
        }
    }
```

parser-creating:

```
var parser = new LL1GrammarParser<LExprBasic, AbstractLToken>(grammar, lexer, lexer.NameTokenTBL, new
System.Reflection.Assembly[] { typeof(LExprBasic).Assembly });
```

Kde *AbstractLToken* je uživatelska implementace rozhrani *IToken*; lexer je instance nějake třidy *Lexer*, ktera je implementace rozhrani *ILexer*<*AbstractLToken*>.

Tedy objekt parser lze využit na parsovani lambda-vyrazu.

3.Chyby a vyjimky

Pokud v průběhu parsovani došlo k typove chybě (např. jako argument metody byla předana hodnota nějakého špatného typu), vznika vyjimka **ParsingTypeErrorException**
Pokud nenalezena metoda, ktera by vychovovala podminkam (typy a pořadi parametrů) **ArgumentErrorException**
Chyba hledani třidy - **TypeNotFoundException**, chyba hledani metody v třidě - **MetodNotFoundException.**
V případě nalezení dvou uspěšných derivaci pro jeden řetězec, vzniká vyjimka **AmbiguousGrammarException.**
V případě nalezeni leve rekurze - **LeftRecursionGrammarException.**
Vyjimky vyšši se dědi z **ParsingErrorException**, ktera se dědi z **System.Exception**.
Pokud vstupní řetězec nedá se rozparsovat gramatikou, tak vznika **FormatException** (pro kompatibilitu s *int.Parse*, *uint.Parse,* atd. metodami).