Лабораторная работа № 2.3 «Синтаксический анализатор на основе предсказывающего анализа»

15 апреля 2024 г.

Сергей Виленский, ИУ9-62Б

Цель работы

Целью данной работы является изучение алгоритма построения таблиц предсказывающего анализатора.

Индивидуальный вариант

```
' аксиома
<axiom <E>>
' правила грамматики
<E    <T E'>>
<E'   <+ T E'> <>>
<T    <F T'>>
<T'   <* F T'> <>>
<F    <n> <( E )>>
```

Реализация

Неформальное описание синтаксиса входного языка

Язык представления правил грамматики, в записи которых правила и альтернативы правил обернуты в угловые скобки.

Лексическая структура

Лекисческие домены в порядке возрастания приоритета.

```
Term    ::= [^a-zA-Z<>\s]
Nterm   ::= [a-zA-Z][^<>\s]*
```

```
Space   ::= \s
Comment ::= \'.*\n
```

Грамматика языка

```
Rules     ::= Rule Rules | ε.
Rule      ::= '<' Nterm '<' Altrule '>' Altrules '>'.
Altrules  ::= '<' Altrule '>' Altrules | ε.
Altrule   ::= Term Altrule | Nterm Altrule | ε.
```

Таблица предсказывающего разбора

	NTerm	Term	‘<’	‘>’	$
Rules	ERROR	ERROR	Rule Rules	ERROR	ε
Rule	ERROR	ERROR	‘<’ Nterm ‘<’ Altrule ‘>’ Altrules ‘>’	ERROR	ERROR
Altrules	ERROR	ERROR	‘<’ Altrule ‘>’ Altrules	ε	ERROR
Altrule	Nterm Altrule	Term Altrule	ERROR	ε	ERROR

Программная реализация

```
#include <iostream>
#include <fstream>
#include <vector>
#include <unordered_set>
#include <unordered_map>
#include <stack>
#include <string>
#include <cassert>

#include "lib/Compiler/Compiler.cpp"


struct TabelLexem
{
    std::string nonTerm;
    std::string domen;
    std::shared_ptr<Compiler::Token> token;
};
```

```
auto errorLexem(Compiler::MessageList& messageList,
    const TabelLexem& errorLex, std::string expected) -> void
{
    messageList.AddError(errorLex.token->Coords.Strarting,
     "Unexpected lexem '" + errorLex.token->str + "', expected '" + expected + "'.");
}

auto errorEarlyFinish(Compiler::MessageList& messageList,
    const TabelLexem& errorLex) -> void
{
    messageList.AddError(errorLex.token->Coords.Strarting,
        "Text parsing was not completed.");
}

auto errorUnrecognizedTail(Compiler::MessageList& messageList,
    const TabelLexem& errorLex) -> void
{
    messageList.AddError(errorLex.token->Coords.Strarting,
        "Unreconized tail of text.");
}

auto readNewLexem(Compiler::Scanner& scanner) -> TabelLexem
{
    auto nextToken = scanner.nextToken();
   auto nextLexem = nextToken ? std::make_shared<Compiler::Token>(*nextToken) : nullptr;
    if (!nextLexem)
    {
        return {"", "$", std::make_shared<Compiler::Token>()};
    }
    if (nextLexem->Tag == Compiler::Token::DomainTag::NIL)
    {
        return {"", "$", nextLexem};
    }
    if (nextLexem->Tag == Compiler::Token::DomainTag::TERM)
    {
        return {"", "Term", nextLexem};
    }
    if (nextLexem->Tag == Compiler::Token::DomainTag::NTERM)
    {
        return {"", "Nterm", nextLexem};
    }
    if (nextLexem->Tag == Compiler::Token::DomainTag::KEYWORD)
    {
        if (nextLexem->str == "<")
        {
            return {"", "<", nextLexem};
```

```
        }
        else
        {
            return {"", ">", nextLexem};
        }
    }
    return {"", "$", std::make_shared<Compiler::Token>()};
}

auto TopDownParse(
    const std::unordered_set<std::string>& nonTerminals,
    const std::string& startTerm,
    const std::unordered_map<std::string,
                  std::unordered_map<std::string, std::vector<std::string>>>&
                          predictTable,
    Compiler::Scanner& scanner,
    Compiler::MessageList& messageList
) -> std::vector<std::pair<TabelLexem, std::vector<std::string>>>
{

   std::vector<std::pair<TabelLexem, std::vector<std::string>>> result{};
    std::stack<std::string> magazine{};
    magazine.push("$");
    magazine.push(startTerm);
    auto alpha = readNewLexem(scanner);
    std::string topSym;

    do
    {
        topSym = magazine.top();
        while (topSym == "ε")
        {
            magazine.pop();
            topSym = magazine.top();
        result.push_back({{"ε", "ε", std::make_unique<Compiler::Token>()},
                std::vector<std::string>{"ε"}});
        }
        alpha.nonTerm = topSym;
        if (!nonTerminals.contains(topSym))
        {
            if (topSym == alpha.domen)
            {
                magazine.pop();
                result.push_back({alpha,
                    std::vector<std::string>{alpha.token->str}});
            }
```

```
            else if (topSym == "$")
            {
                errorUnrecognizedTail(messageList, alpha);
                break;
            }
            else if (alpha.domen == "$")
            {
                std::cout << topSym << '\n';
                errorEarlyFinish(messageList, alpha);
                break;
            }
            else
            {
                errorLexem(messageList, alpha, topSym);
            }
            alpha = readNewLexem(scanner);
        }
     else if (predictTable.at(alpha.domen).at(topSym).size() != 1 ||
           predictTable.at(alpha.domen).at(topSym).at(0) != "ERROR")
        {
            magazine.pop();
        const auto& rule = predictTable.at(alpha.domen).at(topSym);
            for (auto it = rule.rbegin(); it != rule.rend(); ++it)
            {
                magazine.push(*it);
            }
            result.push_back({alpha, rule});
        }
        else if (alpha.domen == "$")
        {
            std::cout << topSym << '\n';
            errorEarlyFinish(messageList, alpha);
            break;
        }
        else
        {
            errorLexem(messageList, alpha, topSym);
            alpha = readNewLexem(scanner);
        }
    }
    while (topSym != "$");

    return result;
}

int main()
```

```
{
    std::unordered_set<std::string> nonTerminals{
        "Rules",
        "Rule",
        "Altrules",
        "Altrule"
    };
    std::string startTerm = "Rules";
    std::unordered_map<std::string,
        std::unordered_map<std::string, std::vector<std::string>>>
        predictTable{};
    std::ifstream inputChain{"example.txt"};


    predictTable.insert({"Nterm", {
        {"Rules", {"ERROR"}},
        {"Rule", {"ERROR"}},
        {"Altrules", {"ERROR"}},
        {"Altrule", {"Nterm", "Altrule"}}}});


    predictTable.insert({"Term", {
        {"Rules", {"ERROR"}},
        {"Rule", {"ERROR"}},
        {"Altrules", {"ERROR"}},
        {"Altrule", {"Term", "Altrule"}}}});

    predictTable.insert({"<", {
        {"Rules", {"Rule", "Rules"}},
      {"Rule", {"<", "Nterm", "<", "Altrule", ">", "Altrules", ">"}},
        {"Altrules", {"<", "Altrule", ">", "Altrules"}},
        {"Altrule", {"ERROR"}}}});

    predictTable.insert({">", {
        {"Rules", {"ERROR"}},
        {"Rule", {"ERROR"}},
        {"Altrules", {"ε"}},
        {"Altrule", {"ε"}}}});

    predictTable.insert({"$", {
        {"Rules", {"ε"}},
        {"Rule", {"ERROR"}},
        {"Altrules", {"ERROR"}},
        {"Altrule", {"ERROR"}}}});

    Compiler::Compiler compiler{};
```

```
 auto scanner = compiler.GetScanner(inputChain);

auto res = TopDownParse(nonTerminals, startTerm, predictTable, scanner, compiler.Messages);

 std::stack<std::size_t> magazine{};

 for (const auto& [nonTerm, rule] : res)
 {
     while (!magazine.empty() && magazine.top() == 0)
     {
         magazine.pop();
     }
     for (std::size_t i = 0; i != magazine.size(); ++i)
     {
         std::cout << "  ";
     }
     std::cout << nonTerm.nonTerm << " -> ";
     if (!nonTerminals.contains(nonTerm.nonTerm))
     {
         std::cout << "\'" << nonTerm.domen << "\' ";
         if (nonTerm.nonTerm != "$" && nonTerm.nonTerm != "ε")
         {
             std::cout << nonTerm.token->Coords.Strarting <<
             '-' <<
             nonTerm.token->Coords.Ending;
         }
         if (nonTerm.token->Tag != nonTerm.token->KEYWORD &&
             nonTerm.token->Tag != nonTerm.token->NIL)
         {
             std::cout << " \"" << nonTerm.token->str << "\"";
         }
     }
     else
     {
         for (const auto& ruleTerm : rule)
         {
             std::cout << ruleTerm << ' ';
         }
     }
     std::cout << "\n";

     if (!magazine.empty())
     {
         std::size_t lastTermCount = magazine.top();
         magazine.pop();
         magazine.push(lastTermCount - 1);
```

```
        }

        std::size_t countNonTerms = 0;
        if (nonTerminals.contains(nonTerm.nonTerm))
        {
            countNonTerms = rule.size();
        }
        if (countNonTerms != 0)
        {
            magazine.push(countNonTerms);
        }
    }

    std::cout << "COMMENTS:\n";
    for (const auto& comment : scanner.Comments) {
        std::cout
            << '\t'
            << comment.Strarting
            << '-'
            << comment.Ending
            << '\n';
    }

    std::cout << "MESSAGES:\n";
    for (const auto& message : compiler.Messages.GetSorted()) {
        std::cout
            << '\t'
            << (message.IsError ? "ERROR " : "WRANING ")
            << message.Coord
            << ": "
            << message.Text
            << '\n';
    }

    inputChain.close();

    return 0;
}
```

Тестирование

Входные данные

```
' аксиома
<axiom <E>>
```

```
' правила грамматики
<E    <T E'>>
<E' 123  <+ T E'> <>>
<T    <F T'>>
<T'   <* F T'> <>>
<F    <n> <( E )>>
```

Вывод на stdout

```
Rules -> Rule Rules
  Rule -> < Nterm < Altrule > Altrules >
    < -> '<' (2, 1)-(2, 2)
    Nterm -> 'Nterm' (2, 2)-(2, 7) "axiom"
    < -> '<' (2, 8)-(2, 9)
    Altrule -> Nterm Altrule
      Nterm -> 'Nterm' (2, 9)-(2, 10) "E"
      Altrule -> ε
        ε -> 'ε'
    > -> '>' (2, 10)-(2, 11)
    Altrules -> ε
      ε -> 'ε'
    > -> '>' (2, 11)-(2, 12)
  Rules -> Rule Rules
    Rule -> < Nterm < Altrule > Altrules >
      < -> '<' (4, 1)-(4, 2)
      Nterm -> 'Nterm' (4, 2)-(4, 3) "E"
      < -> '<' (4, 7)-(4, 8)
      Altrule -> Nterm Altrule
        Nterm -> 'Nterm' (4, 8)-(4, 9) "T"
        Altrule -> Nterm Altrule
          Nterm -> 'Nterm' (4, 10)-(4, 12) "E'"
          Altrule -> ε
            ε -> 'ε'
      > -> '>' (4, 12)-(4, 13)
      Altrules -> ε
        ε -> 'ε'
      > -> '>' (4, 13)-(4, 14)
    Rules -> Rule Rules
      Rule -> < Nterm < Altrule > Altrules >
        < -> '<' (5, 1)-(5, 2)
        Nterm -> 'Nterm' (5, 2)-(5, 4) "E'"
        < -> '<' (5, 10)-(5, 11)
        Altrule -> Term Altrule
          Term -> 'Term' (5, 11)-(5, 12) "+"
          Altrule -> Nterm Altrule
            Nterm -> 'Nterm' (5, 13)-(5, 14) "T"
```

```
      Altrule -> Nterm Altrule
        Nterm -> 'Nterm' (5, 15)-(5, 17) "E'"
        Altrule -> ε
          ε -> 'ε'
  > -> '>' (5, 17)-(5, 18)
  Altrules -> < Altrule > Altrules
    < -> '<' (5, 19)-(5, 20)
    Altrule -> ε
      ε -> 'ε'
    > -> '>' (5, 20)-(5, 21)
    Altrules -> ε
      ε -> 'ε'
  > -> '>' (5, 21)-(5, 22)
Rules -> Rule Rules
  Rule -> < Nterm < Altrule > Altrules >
    < -> '<' (6, 1)-(6, 2)
    Nterm -> 'Nterm' (6, 2)-(6, 3) "T"
    < -> '<' (6, 7)-(6, 8)
    Altrule -> Nterm Altrule
      Nterm -> 'Nterm' (6, 8)-(6, 9) "F"
      Altrule -> Nterm Altrule
        Nterm -> 'Nterm' (6, 10)-(6, 12) "T'"
        Altrule -> ε
          ε -> 'ε'
    > -> '>' (6, 12)-(6, 13)
    Altrules -> ε
      ε -> 'ε'
    > -> '>' (6, 13)-(6, 14)
  Rules -> Rule Rules
    Rule -> < Nterm < Altrule > Altrules >
      < -> '<' (7, 1)-(7, 2)
      Nterm -> 'Nterm' (7, 2)-(7, 4) "T'"
      < -> '<' (7, 7)-(7, 8)
      Altrule -> Term Altrule
        Term -> 'Term' (7, 8)-(7, 9) "*"
        Altrule -> Nterm Altrule
          Nterm -> 'Nterm' (7, 10)-(7, 11) "F"
          Altrule -> Nterm Altrule
            Nterm -> 'Nterm' (7, 12)-(7, 14) "T'"
            Altrule -> ε
              ε -> 'ε'
      > -> '>' (7, 14)-(7, 15)
      Altrules -> < Altrule > Altrules
        < -> '<' (7, 16)-(7, 17)
        Altrule -> ε
          ε -> 'ε'
```

```
              > -> '>' (7, 17)-(7, 18)
              Altrules -> ε
                ε -> 'ε'
            > -> '>' (7, 18)-(7, 19)
          Rules -> Rule Rules
            Rule -> < Nterm < Altrule > Altrules >
              < -> '<' (8, 1)-(8, 2)
              Nterm -> 'Nterm' (8, 2)-(8, 3) "F"
              < -> '<' (8, 7)-(8, 8)
              Altrule -> Nterm Altrule
                Nterm -> 'Nterm' (8, 8)-(8, 9) "n"
                Altrule -> ε
                  ε -> 'ε'
              > -> '>' (8, 9)-(8, 10)
              Altrules -> < Altrule > Altrules
                < -> '<' (8, 11)-(8, 12)
                Altrule -> Term Altrule
                  Term -> 'Term' (8, 12)-(8, 13) "("
                  Altrule -> Nterm Altrule
                    Nterm -> 'Nterm' (8, 14)-(8, 15) "E"
                    Altrule -> Term Altrule
                      Term -> 'Term' (8, 16)-(8, 17) ")"
                      Altrule -> ε
                        ε -> 'ε'
                > -> '>' (8, 17)-(8, 18)
                Altrules -> ε
                  ε -> 'ε'
              > -> '>' (8, 18)-(8, 19)
            Rules -> ε
              ε -> 'ε'
$ -> '$'
COMMENTS:
    (1, 1)-(2, 1)
    (3, 1)-(4, 1)
MESSAGES:
    ERROR (5, 5): Unexpected lexem '123', expected '<'.
```

Вывод

В результате выполнения данной работы были изучен алгоритм построения таблиц предсказывающего анализатора.