ИТМО Кафедра Информатики и прикладной математики

Лабораторная работа №1 «Грамматики простого предшествования» Вариант 7

Выполнил: студент группы P3217
Плюхин Дмитрий
Преподаватель: Лаздин А.В.

2017 год

1. Задание

В качестве исходной выбрать ***приведенную*** грамматику без ***ε-правил*** из домашнего задания №3.

1. Для указанной грамматики построить отношения предшествования.
2. Если отношения построены с конфликтами, то преобразовать исходную грамматику в грамматику простого предшествования.
3. По матрице таблице отношений предшествования реализовать распознаватель для КС грамматики предшествования.

2. Исходная грамматика

$F \rightarrow AB$

$A \rightarrow c$

$B \rightarrow b$

3. Матрица отношений предшествования

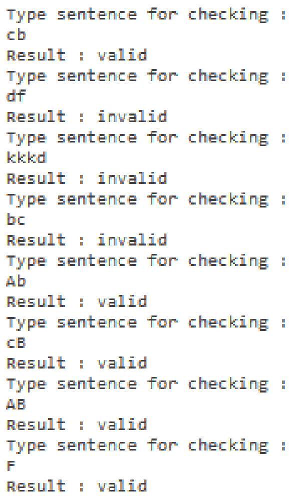
	F	A	B	c	b
F	-	-	-	-	-
A	-	-	=	-	<
B	-	-	-	-	-
c	-	-	>	-	>
b	-	-	-	-	-

4. Распознаватель для грамматики предшествования

```
#include <stdio.h>
#include <string.h>
#include <fstream>
#include <windows.h>
#include <iostream>
#include <cstdlib>
#include <time.h>
#include <sstream>
using namespace std;

#define NO_BASIS 0
#define BASIS 1
#define BEGIN_BASIS 2
#define END_BASIS 3

typedef struct _StackSymbolsNode StackSymbolsNode, *PStackSymbolsNode;
typedef struct _StackRelationsNode StackRelationsNode, *PStackRelationsNode;

typedef struct _PredecessorMatrix{
  unsigned** matrix;
  char* rows;
  char* cols;
  unsigned length;
} PredecessorMatrix, *PPredecessorMatrix;

typedef struct _Grammar{
  char** lefts;
  char** rights;
  unsigned length;
} Grammar, *PGrammar;

typedef struct _StackSymbolsNode{
  PStackSymbolsNode nextNode;
  char symbol;
} StackSymbolsNode, *PStackSymbolsNode;

typedef struct _StackRelationsNode{
  PStackRelationsNode nextNode;
  unsigned relationCode;
} StackRelationsNode, *PStackRelationsNode;
```

```
char pop(PStackSymbolsNode* pHead){
  PStackSymbolsNode head = *pHead;
  if (head == NULL) return 0;
  *pHead = head->nextNode;
  return head->symbol;
}

unsigned pop(PStackRelationsNode* pHead){
  PStackRelationsNode head = *pHead;
  if (head == NULL) return 0;
  *pHead = head->nextNode;
  return head->relationCode;
}

void push(PStackSymbolsNode* pHead, char symbol){
  PStackSymbolsNode pNode = (PStackSymbolsNode)malloc(sizeof(StackSymbolsNode));
  pNode->nextNode = *pHead;
  pNode->symbol = symbol;
  *pHead = pNode;
  return;
}

void push(PStackRelationsNode* pHead, unsigned relationCode){
  PStackRelationsNode pNode = (PStackRelationsNode)malloc(sizeof(StackRelationsNode));
  pNode->nextNode = *pHead;
  pNode->relationCode = relationCode;
  *pHead = pNode;
  return;
}

//predecessorMatrix
// ; ; ; ; ;
// ; ;=; ;<;
// ; ; ; ; ;
// ; ;>; ;>;
// ; ; ; ; ;

PredecessorMatrix getPredecessorMatrix(){
  PredecessorMatrix predecessorMatrix;
  unsigned** matrix = (unsigned**)malloc(5*sizeof(unsigned*));
  for (int i = 0; i < 5; i++)
  {
    matrix[i] = (unsigned*)malloc(5*sizeof(unsigned));
    for (int j = 0; j < 5; j++) matrix[i][j] = NO_BASIS;
  }
  matrix[1][2] = BASIS;
  matrix[1][4] = BEGIN_BASIS;
  matrix[3][2] = END_BASIS;
  matrix[3][4] = END_BASIS;

  char* symbs = (char*)malloc(5*sizeof(char));
  symbs[0] = 'F'; symbs[1] = 'A'; symbs[2] = 'B'; symbs[3] = 'c'; symbs[4] = 'b';
  predecessorMatrix.matrix = matrix;
  predecessorMatrix.rows = symbs;
  predecessorMatrix.cols = symbs;
  predecessorMatrix.length = 5;
  return predecessorMatrix;
}

PGrammar getGrammar(){
  PGrammar grammar = (PGrammar)malloc(sizeof(Grammar));
  char** lefts = (char**)malloc(3*sizeof(char*));
  char** rights = (char**)malloc(3*sizeof(char*));
  for (int i = 0; i < 3; i++){
    lefts[i] = (char*)malloc(5*sizeof(char));
    rights[i] = (char*)malloc(5*sizeof(char));
  }
  strcpy(lefts[0], "F");
  strcpy(lefts[1], "A");
  strcpy(lefts[2], "B");
  strcpy(rights[0], "AB");
  strcpy(rights[1], "c");
  strcpy(rights[2], "b");
  grammar->lefts = lefts;
  cout << "-- " << lefts[0] << endl;
  grammar->rights = rights;
  grammar->length = 3;
  return grammar;
}

unsigned getRelation(char rowSymbol, char colSymbol, PPredecessorMatrix predecessorMatrix){
  char* rowSymbols = predecessorMatrix->rows;
  char* colSymbols = predecessorMatrix->cols;
  unsigned rowNum = 0;
  for (int i = 0; i < predecessorMatrix->length; i++) if (rowSymbols[i] == rowSymbol){
    rowNum = i;
    break;
  }
  unsigned colNum = 0;
  for (int i = 0; i < predecessorMatrix->length; i++) if (colSymbols[i] == colSymbol){
    colNum = i;
    break;
  }
  return predecessorMatrix->matrix[rowNum][colNum];
}

string check(Grammar* grammar, string checked){
  for (int i = 0; i < grammar->length; i++) if (strcmp(grammar->rights[i], checked.c_str()) == 0)
return grammar->lefts[i];
  return "";
}

bool checkSentence(string sentence, PredecessorMatrix predecessorMatrix, PGrammar pGrammar){
  Grammar grammar = *pGrammar;
  PStackSymbolsNode pSymbols = NULL;
  PStackRelationsNode pRelations = NULL;
  PStackSymbolsNode pUnchecked = NULL;
  string sub = "";

  int length = sentence.length();
  bool broken = false;
  unsigned relation = 0;
  char symb;
  unsigned rel;
  string llop;
  string tmp;
  stringstream tmps;
  char currentSymbol;
  for (int i = length-1; i >= 0; i--){
    push(&pUnchecked, sentence[i]);
  }
  while (true){
    if (pUnchecked == NULL) {
      sub = "";
      relation = END_BASIS;
      while ((pSymbols != NULL) && (relation != BEGIN_BASIS)){
        tmp = "";
        tmps.clear();
        tmps << pop(&pSymbols);
        tmps >> tmp;
        sub.insert(0,tmp);
        relation = pop(&pRelations);
      }
      llop = check(&grammar, sub);
      if (llop.empty()) return false;
      for (int i = llop.length() - 1; i >= 0; i--){
        push(&pUnchecked, llop[i]);
      }
    }
    currentSymbol = pop(&pUnchecked);
    if (currentSymbol == 'F') return true;
    if (pSymbols == NULL){
      push(&pSymbols, currentSymbol);
      continue;
    }
    relation = getRelation(pSymbols->symbol, currentSymbol, &predecessorMatrix);
    if ((relation == BASIS) || (relation == BEGIN_BASIS)){
      push(&pSymbols, currentSymbol);
      push(&pRelations, relation);
      continue;
    }
    if (relation == END_BASIS){
      push(&pUnchecked, currentSymbol);
      sub = "";
      while ((pSymbols != NULL) && (relation != BEGIN_BASIS)){
        tmp = "";
        tmps.clear();
        tmps << pop(&pSymbols);
        tmps >> tmp;
        sub.insert(0,tmp);
        relation = pop(&pRelations);
      }
      llop = check(&grammar, sub);
      if (llop.empty()) return false;
      for (int i = llop.length() - 1; i >= 0; i--){
        push(&pUnchecked, llop[i]);
      }
    }
  }
  return true;
}

int main(int argc, char* argv[]){
  PredecessorMatrix predecessorMatrix = getPredecessorMatrix();
  PGrammar grammar = getGrammar();
  cout << "-- " << grammar->lefts[0] << endl;
  char identifier[256];
  string sentence = "";
  while(true){
    cout << "Type sentence for checking : " << endl;
    getline(cin, sentence);
    cout << "Result : ";
    if (checkSentence(sentence, predecessorMatrix, grammar)){
      cout << "valid";
    } else {
      cout << "invalid";
    }
    cout << endl;
  }
}
```

5. Результаты тестирования



```
Type sentence for checking :
cb
Result : valid
Type sentence for checking :
df
Result : invalid
Type sentence for checking :
kkkd
Result : invalid
Type sentence for checking :
bc
Result : invalid
Type sentence for checking :
Ab
Result : valid
Type sentence for checking :
cB
Result : valid
Type sentence for checking :
AB
Result : valid
Type sentence for checking :
F
Result : valid
```