

**编译原理课程设计报告**

**设计题目** 36.LR(0)分析器自动构造程序的实现

**学生姓名**  祁明

**学 号**  2020216774

**专业班级**  计算机20-2班

**指导教师**

**完成日期**

**一、设计目的及设计要求**

**设计目的：**LR(0)分析器自动构造程序的实现

**设计内容及要求：**对任意给定的文法G，构造识别文法活前缀的DFA，输出DFA的状态转化矩阵及LR(0)项目集规范族；实现LR(0)分析表构造算法；实现LR分析器总控程序。程序输出一个完整的LR(0)分析器源程序，可输出到指定文件中。

**二、开发环境描述**

开发环境为Windows10下的Microsoft Visio studio 2022，使用C# .NET Core6.0和WPF完成项目开发。



图 1 Windows开发环境



图 2 C# .NET Core6.0

**三、设计内容、主要算法描述**

3.1 设计内容及主要方法

LR分析法：是一种有效的自底向上的分析方法，是一种“移进-归约”分析方法， 1965年 由Knuth提出。L表示从左向右扫描输入串（read from Left to right），R表示构造最右推导的逆过程（for constructing a Rightmost derivation in reverse）。大多数用上下文无关文法描述的高级语言的语法成分可以用LR分析器来识别。LR(0)分析根据当前分析栈中的符号串和向右顺序查看输入串就可以确定分析的动作是移进还是归约。

自底向上分析法的关键问题是如何确定可归约的串。LR(0)分析方法是通过求句柄逐步归约来进行语法分析。



图 3 产生分析表



图 4 LR(0)分析器工作原理

在完成LR(0)分析器自动构造程序的实现的过程中，将实现方式分为以下步骤：



图 5 LR(0)分析器自动构造程序的实现步骤

3.2 各个步骤的具体算法及实现

3.2.1**LR(0)项目集族的构造**

对于一个文法G，我们可以构造一个有限自动机，它能识别G的所有活前缀。由于产生式右部的符号就是句柄，若这些符号串都已进栈，则表示它已处于“归态活前缀”，若只有部分进栈，则表示它处于“非归态活前缀”，要想知道活前缀有多大部分进栈了，可以为每个产生式构造一个自动机，由它的状态来记住这些情况，此“状态”称为“项目”。这些自动机的全体就是识别所有活前缀的有限自动机。文法G的每个产生式的右部添加一个圆点称为G的LR(0)项目。构成识别一个文法活前缀的DFA的项目集(状态)的全体称为文法的LR(0)项目集规范族。若项目A→α .Bβ对活前缀η=δα是有效的且B → γ是一个产生式，则项目B → .γ对η=δα也是有效的。假定文法G是一个以S为开始符号的文法，我们构造一个G’，它包含了整个G，但它引进了一个不出现在G中的非终结符S’，并加进一个新产生式S’→S，而这个S’是G’的开始符号。那么，我们称G’是G的拓广文法。这样，便会有一个仅含项目S'→S·的状态，这就是唯一的“接受”态。



图 6 项目集规范族的构造过程

|  |
| --- |
| **构造项目集规范族的算法** |
| **PROCEDURE ITEMSETS(G′)；**  **BEGIN**  **C:={CLOSURE({S′→·S})}；**  **REPEAT**  **FOR C中每个项目集I和G′的每个符号X DO**  **IF GO(I，X)非空且不属于C THEN**  **把GO(I，X)放入C族中;**  **UNTIL C 不再增大**  **END** |

**具体实现如下：**

|  |
| --- |
| **构造项目集规范族** |
| private void CreateProjectSet()  {  produceFormula[0] = "S'->" + vn[0];//拓展文法开始符号为S'  C.Add(new List<string>());//初始状态加入  C[0].Add("S'->·" + vn[0]);  C[0] = Closure(C[0]);  while (true)  {  int oldCLength = C.Count;  for (int i = 0; i < C.Count; i++)//遍历C中的每一个项目集I  {  List<string> I = C[i];  foreach (string X in symbol)  {  List<string> tmp = GO(I, X);  if (tmp != null && !isExistedInC(C, tmp)) C.Add(tmp);  }  }  if (C.Count == oldCLength) break;  }  } |

为了识别活前缀，我们定义一个状态转换函数GO是一个状态转换函数。I是一个项目集，X是一个文法符号。函数值GO(I，X)定义为：

GO(I，X)＝CLOSURE(J)

其中

J＝{任何形如A→αX·β的项目| A→α · Xβ属于I}。

|  |
| --- |
| **状态转换函数GO** |
| private List<string>? GO(List<string> I, string X)  {  List<string> J = new List<string>();  foreach (string projectStr in I)  {  int pointIndex = projectStr.IndexOf('·');  if (pointIndex < projectStr.Length - 1 && (projectStr[pointIndex + 1] + "").Equals(X)) //找到形如 A->α·Xβ 的项目  J.Add(projectStr.Substring(0, pointIndex) + X + '·' + projectStr.Substring(pointIndex + 2));  }  J = Closure(J);  return J;  } |

|  |
| --- |
| **Closure** |
| private List<string>? Closure(List<string> J)  {  if (J == null || J.Count == 0) return null;  List<string> result = new List<string>(J);  foreach (string productionStr in J)//遍历J中的所有产生式  {  for (int i = 0; i < productionStr.Length; i++)  {  if (productionStr[i] == '·' && i != productionStr.Length - 1 && vn.Contains(productionStr[i + 1] + ""))//找到 A->α·Bβ 属于CLOUSURE(J)  {  List<string>? productionRightList = processedProduction[productionStr[i + 1] + ""] as List<string>;  if (productionRightList == null) continue;  foreach (string rightStr in productionRightList)  result.Add(productionStr[i + 1] + "->·" + rightStr);  }  }  }  return result;  } |

3.2.2**构造LR(0)分析表**

|  |
| --- |
| **分析表的ACTION和GOTO子表构造方法** |
| **1. 若项目A→α·aβ属于Ik且GO(Ik, a)＝Ij，a为终结符，则置ACTION[k,a] 为“sj”。**  **2. 若项目A→α·属于Ik，那么，对任何终结符a(或结束符#)，置ACTION[k,a]为 “rj”(假定产生式A→α是文法G′的第j个产生式)。**  **3. 若项目S′→S·属于Ik，则置ACTION[k,#]为 “acc”。**  **4. 若GO(Ik,A)＝Ij，A为非终结符，则置GOTO[k,A]=j。**  **5. 分析表中凡不能用规则1至4填入信息的空白格均置上“报错标志”。** |

3.2.3**判断文法是否符合****LR(0)文法**

a) 规约-规约冲突

若该文法中有两条产生式的右部相同，则会产生“规约-规约”冲突，不符合LR0文法。

b) 移进-规约冲突

i. 若该文法中含有产生式能产生空，则会产生移进-规约，不符合LR(0)文法。

ii. 若分析该文法时，构建自动机过程中，有某个节点中的所有产生式中，有任意两条产生式，一条需要移进，一条需要规约，则不符合LR(0)文法。

**四、设计的输入和输出形式**

程序读取输入串，将输入串按照LR(0)的分析方法进行分析，并在屏幕上输出分析过程和结果。

**五、程序运行**

**程序读取输入串为bcccd，以下为分析过程和结果：**

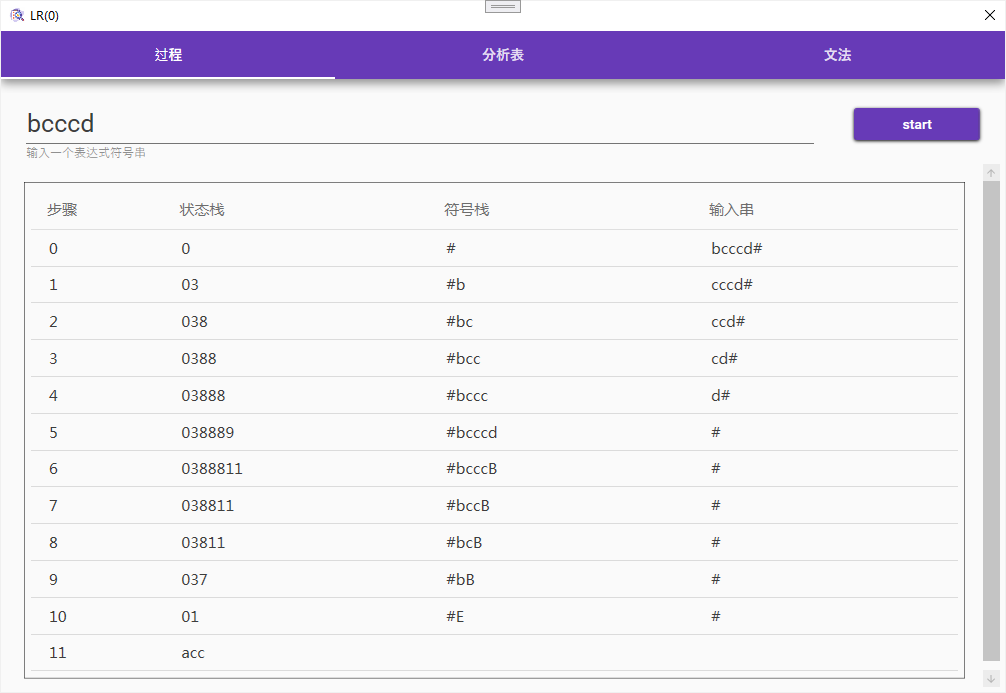


图 7 分析输入串bcccd

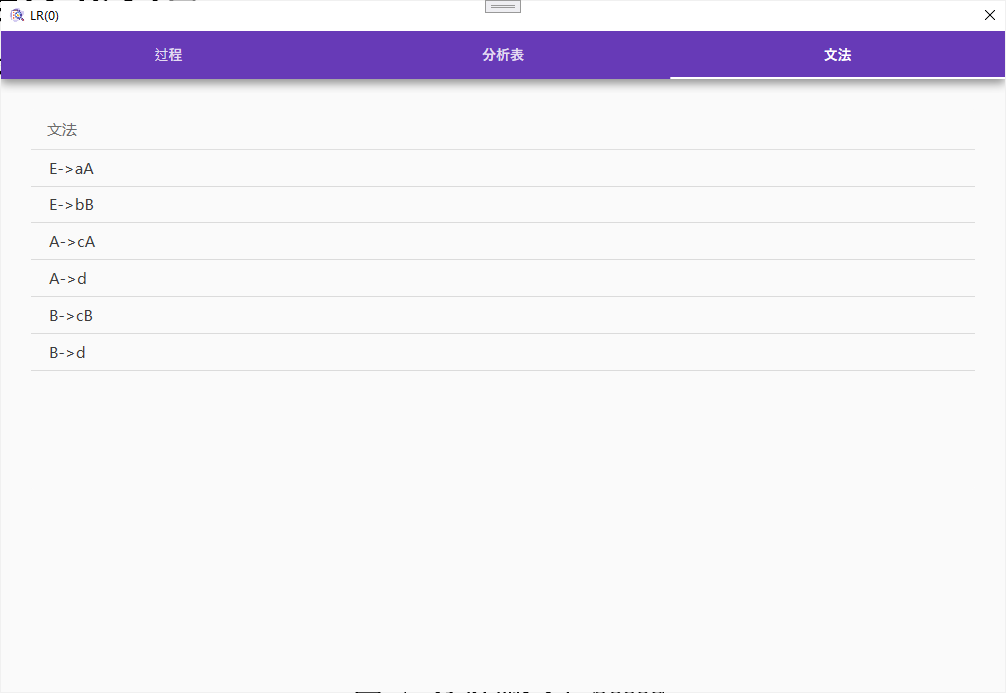


图 8 展示文法



图 9 显示分析表

**六、总结**

语法分析的本质就是是否可以生成一颗语法树，它的叶子节点及终结符与我们程序的符号流一致，存在的正确，不存在的错误。可以这样思考，将程序的符号表看作是语法树的叶子节点，按照文法进行反向推到，就是根据叶子节点向上推到根节点。如果我们能成功推到出根节点的话，那么从上往下推导肯定也能推导出来。我们要按照产生式来推导，就是根据知道了产生式的体，就可以确定它的头，在语法树上就是知道了所有的孩子节点，就可以反推出父节点。这里的关键在于如何知道有哪些子节点，以及这些孩子节点的顺序。这个需要状态机来帮忙了。从初始状态开始，一个特定状态和一个特定输入决定了要到达的下一个状态。反过来如果知道现在的状态了，那么之前一定经过了特定的状态，进而可以确定，之前一定按照特定的顺序输入了特定的字符。

在LR(0)分析器自动构造程序的实现的过程中，我们可以实现知识与控制的分离，可以在不改变程序结构的基础上仅仅替换掉知识部分便可以完成另外一个语言的分析和处理。

**七、源程序清单**

|  |
| --- |
| **LR0.cs** |
| using System.Collections;  using System.Collections.Generic;  namespace Course\_design\_of\_compiling\_principles  {      internal class LR0      {          public List<string> produceFormula = new List<string> { "", "E->aA", "E->bB", "A->cA", "A->d", "B->cB", "B->d" };          public List<string> vt = new List<string>(); //终结符集          public List<string> vn = new List<string>(); //非终结符集          private List<string> symbol = new List<string>();          public List<List<string>> C = new List<List<string>>();          public string[,]? action; //action表          public string[,]? GOTO; //goto表          private Hashtable projectLink = new Hashtable();          private Hashtable processedProduction = new Hashtable();          public List<Data> dataList = new List<Data>();          public string inStr = "bcd";          private void ProduceFormulaHandle()          {              foreach (string s in vn)                  processedProduction.Add(s, new List<string>());              foreach (string productionStr in produceFormula)              {                  string[] splitedStr = productionStr.Split("->");                  if (splitedStr.Length > 1)                  {                      List<string>? valueList = processedProduction[splitedStr[0]] as List<string>;                      string[] splitedRight = splitedStr[1].Split('|');                      foreach (string s in splitedRight)                      {                          if (valueList != null) valueList.Add(s);                      }                  }              }          }          //构造非终结符集          private void Getvn()          {              foreach (string str in produceFormula)              {                  for (int i = 0; i < str.Length; i++)                  {                      char c = str[i];                      if (c >= 'A' && c <= 'Z')                      {                          if (i < str.Length - 1 && str[i + 1] == '\'')                          {                              if (!vn.Contains(c + "'")) vn.Add(c + "'");                          }                          else                          {                              if (!vn.Contains(c + "")) vn.Add(c + "");                          }                      }                  }              }          }          private void Getvt()          {              foreach (string str in vn)              {                  List<string> rightList = (List<string>)processedProduction[str];                  foreach (string s in rightList)                  {                      foreach (char c in s)                      {                          if (c < 'A' || c > 'Z')                          {                              if (!vt.Contains(c + "")) vt.Add(c + "");                          }                      }                  }              }          }          public void Init()          {              Getvn();              ProduceFormulaHandle();              Getvt();              foreach (string str in vn) symbol.Add(str);              foreach (string str in vt) symbol.Add(str);              CreateProjectSet();              GetLink();              CreateAnalysisTable();          }          /\*\*           \* Closure 求闭包CLOUSURE(I)：           \* I的任何项目都属于CLOUSURE(I)           \* 若 A->α·Bβ 属于CLOUSURE(I),那么对任何关于B的产生式 B->γ ，项目 B->·γ 也属于CLOUSURE(I)           \*/          private List<string>? Closure(List<string> I)          {              if (I == null || I.Count == 0) return null;              List<string> result = new List<string>(I);              foreach (string productionStr in I)//遍历I中的所有产生式              {                  for (int i = 0; i < productionStr.Length; i++)                  {                      if (productionStr[i] == '·' && i != productionStr.Length - 1 && vn.Contains(productionStr[i + 1] + ""))//找到 A->α·Bβ 属于CLOUSURE(I)                      {                          List<string>? productionRightList = processedProduction[productionStr[i + 1] + ""] as List<string>;                          if (productionRightList == null) continue;                          foreach (string rightStr in productionRightList)                              result.Add(productionStr[i + 1] + "->·" + rightStr);                      }                  }              }              return result;          }          /\*\*           \* GO(I, X) = CLOSURE(J)           \* 其中 J={任何形如 A->αX·β 的项目 | A->α·Xβ 属于I}           \*/          private List<string>? GO(List<string> I, string X)          {              List<string> J = new List<string>();              foreach (string projectStr in I)              {                  int pointIndex = projectStr.IndexOf('·');                  if (pointIndex < projectStr.Length - 1 && (projectStr[pointIndex + 1] + "").Equals(X))  //找到形如 A->α·Xβ 的项目                  {                      J.Add(projectStr.Substring(0, pointIndex) + X + '·' + projectStr.Substring(pointIndex + 2));                  }              }              J = Closure(J);              return J;          }          private bool isExistedInC(List<List<string>> C, List<string> tmp)          {              foreach (List<string> List in C)              {                  if (List[0].Equals(tmp[0])) return true;              }              return false;          }          //构造项目集规范族          private void CreateProjectSet()          {              //拓展文法开始符号为S'              produceFormula[0] = "S'->" + vn[0];              //vn.Add("S'");              C.Add(new List<string>());  //初始状态加入              C[0].Add("S'->·" + vn[0]);              C[0] = Closure(C[0]);              while (true)              {                  int oldCLength = C.Count;                  for (int i = 0; i < C.Count; i++)   //遍历C中的每一个项目集I                  {                      List<string> I = C[i];                      foreach (string X in symbol)                      {                          List<string> tmp = GO(I, X);                          if (tmp != null && !isExistedInC(C, tmp))                          {                              C.Add(tmp);                              //projectLink.Add(i + "," + X, C.Count-1);                          }                      }                  }                  if (C.Count == oldCLength) break;              }          }          private int FindLinkNum(string targetFindStr)          {              for (int i = 0; i < C.Count; i++)              {                  List<string> I = C[i];                  foreach (string str in I)                  {                      if (str.Equals(targetFindStr)) return i;                  }              }              return -1;          }          private void GetLink()          {              for (int i = 0; i < C.Count; i++)              {                  List<string> I = C[i];                  foreach (string str in I)                  {                      int pointIndex = str.IndexOf('·');                      if (pointIndex < str.Length - 1)  //找到形如 A->α·Xβ 的项目                      {                          string X = "" + str[pointIndex + 1];                          string targetFindStr = str.Substring(0, pointIndex) + X + '·' + str.Substring(pointIndex + 2); //A->αX·β                          int index = FindLinkNum(targetFindStr);                          if (index >= 0)                          {                              if (!projectLink.ContainsKey(i + "," + X))                                  projectLink.Add(i + "," + X, index);                          }                      }                  }              }          }          private void CreateAnalysisTable()          {              vt.Add("#"); // 加入#              action = new string[C.Count, vt.Count];              GOTO = new string[C.Count, vn.Count];              for (int i = 0; i < C.Count; i++)              {                  for (int j = 0; j < vt.Count; j++)                  {                      string vtStr = vt[j];                      if (projectLink.ContainsKey(i + "," + vtStr))                      {                          action[i, j] = "s" + projectLink[i + "," + vtStr];                      }                  }                  for (int j = 0; j < vn.Count; j++)                  {                      string vnStr = vn[j];                      if (projectLink.ContainsKey(i + "," + vnStr))                      {                          GOTO[i, j] = "" + projectLink[i + "," + vnStr];                      }                  }                  if (C[i].Count == 1 && C[i][0].IndexOf('·') == C[i][0].Length - 1)                  {                      if (C[i][0].Equals(produceFormula[0] = "S'->" + vn[0] + "·"))                      {                          action[i, vt.Count - 1] = "acc";                      }                      else                      {                          int index = 0;                          for (index = 0; index < produceFormula.Count; index++)                          {                              if (produceFormula[index].Equals(C[i][0].Replace("·", ""))) break;                          }                          for (int j = 0; j < vt.Count; j++)                          {                              action[i, j] = "r" + index;                          }                      }                  }              }          }          public void Analysis()          {              dataList.Clear();              int step = 0;              Stack<int> stateStack = new Stack<int>(); //状态栈              Stack<string> symbolStack = new Stack<string>(); //符号栈              stateStack.Push(0);//将0压入状态栈              symbolStack.Push("#");//将符号#压入符号栈              //string inputStr = "bcd";              string inputStr = inStr;              inputStr = inputStr + "#";//输入串              string a; //输入符号a              int aIndex = 0; //指向输入串的头              int s = 0; //栈顶状态s              dataList.Add(new Data() { step = "" + step, state = "0", symbol = "#", inStr = inputStr.Substring(aIndex) });              while (true)              {                  string stateStr = "";                  string symbolStr = "";                  step++;                  a = inputStr[aIndex] + "";                  s = stateStack.Peek();                  string todo = null;                  if (vt.Contains(a))                  {                      todo = action[s, vt.IndexOf(a)];                  }                  if (todo == null)                  {                      //error                      dataList.Add(new Data() { step = "" + step, state = "ERROR", symbol = "ERROR", inStr = "ERROR" });                      return;                  }                  if (todo[0] == 's')                  {                      stateStack.Push(int.Parse(todo.Replace("s", "")));                      symbolStack.Push(a);                      aIndex++;                  }                  else if (todo[0] == 'r')                  {                      int length = produceFormula[int.Parse(todo.Replace("r", ""))].Split("->")[1].Length;                      for (int i = 0; i < length; i++)                      {                          stateStack.Pop();                          symbolStack.Pop();                      }                      symbolStack.Push(produceFormula[int.Parse(todo.Replace("r", ""))].Split("->")[0]);                      stateStack.Push(int.Parse(GOTO[stateStack.Peek(), vn.IndexOf(symbolStack.Peek())]));                  }                  else if (todo.Equals("acc"))                  {                      //acc                      dataList.Add(new Data() { step = "" + step, state = "acc", symbol = "", inStr = "" });                      return;                  }                  foreach (int i in stateStack)                  {                      stateStr = i + stateStr;                  }                  foreach (string str in symbolStack)                  {                      symbolStr = str + symbolStr;                  }                  dataList.Add(new Data() { step = "" + step, state = stateStr, symbol = symbolStr, inStr = inputStr.Substring(aIndex) });              }          }      }      class Data      {          public string step { get; set; }          public string state { get; set; }          public string symbol { get; set; }          public string inStr { get; set; }      }  } |

|  |
| --- |
| **MainWindow.xaml.cs** |
| using System.Collections.Generic;  using System.Text;  using System.Windows;  namespace Course\_design\_of\_compiling\_principles  {  public partial class MainWindow : Window  {  List<grammar> grammarList = new List<grammar>();  LR0 lr0;  public static string inputstr;  public MainWindow()  {  WindowStartupLocation = WindowStartupLocation.CenterScreen;  lr0 = new LR0();  lr0.Init();  InitializeComponent();  showTable();  showGrammar();  }  private void startbtn\_Click(object sender, RoutedEventArgs e)  {  lr0.inStr = inputArea.Text;  showArea.ItemsSource = null;  lr0.Analysis();  showArea.AutoGenerateColumns = false;  showArea.ItemsSource = lr0.dataList;  }  private void showTable()  {  AnalysisTable.Text = "";  StringBuilder sb = new StringBuilder();  sb.AppendLine("ACTION" + '\t' + '\t' + '\t' + '\t' + '\t' + '\t' + '\t' + "GOTO");  sb.Append("状态" + '\t');  for (int i = 0; i < lr0.vt.Count; i++)  sb.Append(lr0.vt[i] + '\t');  for (int i = 0; i < lr0.vn.Count; i++)  sb.Append(lr0.vn[i] + '\t');  sb.Append('\n');  for (int i = 0; i < lr0.C.Count; i++)  {  sb.Append(i + "" + '\t');  for (int j = 0; j < lr0.vt.Count; j++)  sb.Append(lr0.action[i, j] + '\t');  for (int j = 0; j < lr0.vn.Count; j++)  sb.Append(lr0.GOTO[i, j] + '\t');  sb.Append('\n');  }  string str = sb.ToString();  AnalysisTable.Text = str;  }  private void showGrammar()  {  grammarInput.AutoGenerateColumns = false;  for (int i = 1; i < lr0.produceFormula.Count; i++)  grammarList.Add(new grammar() { GrammarStr = lr0.produceFormula[i] });  grammarInput.ItemsSource = grammarList;  }  }  public class grammar  {  public string GrammarStr { get; set; }  }  } |

|  |
| --- |
| **App.xaml** |
| <Application x:Class="Course\_design\_of\_compiling\_principles.App"  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  xmlns:local="clr-namespace:Course\_design\_of\_compiling\_principles"  StartupUri="MainWindow.xaml"  xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes">  <Application.Resources>  <ResourceDictionary>  <ResourceDictionary.MergedDictionaries>  <materialDesign:BundledTheme BaseTheme="Light" PrimaryColor="DeepPurple" SecondaryColor="Lime" />  <ResourceDictionary Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.Defaults.xaml" />  </ResourceDictionary.MergedDictionaries>  </ResourceDictionary>  </Application.Resources>  </Application> |

|  |
| --- |
| **MainWindow.xaml** |
| <Window x:Class="Course\_design\_of\_compiling\_principles.MainWindow"  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  xmlns:local="clr-namespace:Course\_design\_of\_compiling\_principles"  mc:Ignorable="d"  xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"  TextElement.FontWeight="Regular"  TextElement.FontSize="13"  TextOptions.TextFormattingMode="Ideal"  TextOptions.TextRenderingMode="Auto"  TextElement.Foreground="{DynamicResource MaterialDesignBody}"  Background="{DynamicResource MaterialDesignPaper}"  FontFamily="{DynamicResource MaterialDesignFont}"  Title="LR(0)" Height="700" Width="1020" ResizeMode="NoResize" WindowStyle="SingleBorderWindow" Icon="/图标.ico">  <Grid>  <Grid.ColumnDefinitions>  <ColumnDefinition/>  <ColumnDefinition Width="84\*"/>  </Grid.ColumnDefinitions>  <TabControl Grid.ColumnSpan="2">  <TabItem Header="过程">  <Border Margin="5">  <Grid>  <Grid.RowDefinitions>  <RowDefinition Height="\*" MinHeight="70" MaxHeight="80"/>  <RowDefinition Height="5\*" MinHeight="300"/>  </Grid.RowDefinitions>  <Grid>  <Grid.ColumnDefinitions>  <ColumnDefinition/>  <ColumnDefinition/>  <ColumnDefinition/>  <ColumnDefinition/>  <ColumnDefinition/>  <ColumnDefinition/>  </Grid.ColumnDefinitions>  <TextBox Grid.ColumnSpan="5" Margin="20" Name="inputArea" Text="bcd" FontSize="25" materialDesign:HintAssist.HelperText="输入一个表达式符号串" materialDesign:HintAssist.HelperTextFontSize="12"/>  <Button Grid.Column="5" Name="startbtn" Content="start" Margin="20" Click="startbtn\_Click"/>  </Grid>  <ScrollViewer Grid.Row="1" Grid.ColumnSpan="4">  <Grid>  <DataGrid Margin="25" Name="showArea" FontSize="15" IsReadOnly="True" FontFamily="微软雅黑">  <DataGrid.Columns>  <DataGridTextColumn Width="\*" Header="步骤" Binding="{Binding step}" />  <DataGridTextColumn Width="2\*" Header="状态栈" Binding="{Binding state}"/>  <DataGridTextColumn Width="2\*" Header="符号栈" Binding="{Binding symbol}"/>  <DataGridTextColumn Width="2\*" Header="输入串" Binding="{Binding inStr}"/>  </DataGrid.Columns>  </DataGrid>  <Border BorderBrush="Black" BorderThickness="0.5" Margin="18">  </Border>  </Grid>  </ScrollViewer>  </Grid>  </Border>  </TabItem>  <TabItem Header="分析表">  <Grid>  <Grid.RowDefinitions>  <RowDefinition Height="9\*"/>  <RowDefinition Height="\*"/>  </Grid.RowDefinitions>  <Border BorderBrush="Black" BorderThickness="0.5" Margin="30">  <TextBlock x:Name="AnalysisTable" FontSize="15" FontFamily="微软雅黑" TextAlignment="Center"/>  </Border>  </Grid>  </TabItem>  <TabItem Header="文法">  <Border Margin="5">  <Grid>  <Grid.RowDefinitions>  <RowDefinition Height="5\*"/>  <RowDefinition Height="\*"/>  <RowDefinition Height="\*"/>  </Grid.RowDefinitions>  <Grid.ColumnDefinitions>  <ColumnDefinition/>  <ColumnDefinition/>  </Grid.ColumnDefinitions>  <DataGrid Margin="25" Name="grammarInput" FontSize="15" IsReadOnly="True" FontFamily="微软雅黑" Grid.ColumnSpan="2" >  <DataGrid.Columns>  <DataGridTextColumn Width="\*" Header="文法" Binding="{Binding GrammarStr}" IsReadOnly="False" CanUserReorder="True" />  </DataGrid.Columns>  </DataGrid>  </Grid>  </Border>  </TabItem>  </TabControl>  </Grid>  </Window> |