**编译原理实验报告**

**实验名称：语法分析2**

**判断文法是否是LL(1)文法**

实验目的和要求

在实验二的基础上，通过设计一个FOLLOW集合、SELECT集合计算器、LL(1)文法判断和冲突求解，帮助理解FOLLOW集合、SELECT集合计算、LL(1)文法判断和冲突求解计算方法。FOLLOW集合、SELECT集、LL(1)文法判断和冲突求解是语法分析中的重要概念，掌握其计算对后续的语法分析有重要意义。

**一、实验内容**

实验内容如下：

1. 在实验二的基础上，进行FOLLOW集合的计算。
2. 进行SELECT集合的计算。
3. 判断是否为LL(1)文法，若不是LL(1)文法时，展示求解有哪些冲突。
4. 结果展示在GUI界面上。

**二、实验步骤**

实验步骤如下：

**1. 界面设计和布局**

1. **创建主窗口**：
   * 使用Tkinter库创建一个主窗口，窗口标题为 "LL(1)文法判断器"。
   * 分为左右两个框架：左侧用于输入文法产生式和触发计算，右侧用于显示计算结果。
2. **左侧框架设计**：
   * 添加一个标签或文本框，提示用户输入文法的产生式。
   * 创建一个多行文本框，用于用户输入文法的产生式。
   * 添加一个按钮，显示文本为“开始计算”或类似，点击按钮时触发计算操作。
3. **右侧框架设计**：
   * 包括多个文本框或标签，用于显示计算得到的结果：
     + FIRST集合的文本框，显示每个非终结符的FIRST集合。
     + FOLLOW集合的文本框，显示每个非终结符的FOLLOW集合。
     + SELECT集合的文本框，显示每个产生式的SELECT集合。
     + LL(1)文法判断结果的文本框，显示文法是否为LL(1)型文法及冲突信息。

**2. 处理用户输入**

1. **解析文法产生式**：
   * 在用户点击“开始计算”按钮后，获取用户在文本框中输入的文法产生式。
   * 解析文法产生式，构建产生式字典 productions，其中键为非终结符，值为产生式列表。

**3. 计算FIRST集合**

1. **定义计算FIRST集合的函数**：
   * 编写一个函数 calculate\_first\_in(productions)，接受文法产生式字典 productions 作为参数。
   * 初始化一个空的FIRST集合字典 firsts，键为非终结符，值为集合。
   * 初始化一个字典 completed，记录每个非终结符是否已计算其FIRST集合。
   * 编写内部函数 calculate\_first(non\_terminal)，用于递归计算给定非终结符的FIRST集合：
     + 如果该非终结符的FIRST集合已计算，则直接返回。
     + 遍历该非终结符的所有产生式，根据产生式的首符号更新FIRST集合，处理终结符、非终结符、ε的情况。
     + 如果产生式的首符号可以推导出ε，则继续递归计算FIRST集合。
   * 返回完整的FIRST集合字典 firsts。

**4. 计算FOLLOW集合**

1. **定义计算FOLLOW集合的函数**：
   * 编写一个函数 calculate\_follow\_in(productions, firsts)，接受文法产生式字典 productions 和计算得到的FIRST集合 firsts 作为参数。
   * 初始化一个空的FOLLOW集合字典 follows，键为非终结符，值为集合。
   * 确定起始符号，并将 '$' 加入起始符号的FOLLOW集合。
   * 使用循环遍历所有产生式，更新每个非终结符的FOLLOW集合，直到不再有更新为止。

**5. 计算SELECT集合**

1. **定义计算SELECT集合的函数**：
   * 编写一个函数 calculate\_select\_in(productions, firsts, follows)，接受文法产生式字典 productions、计算得到的FIRST集合 firsts 和FOLLOW集合 follows作为参数。
   * 初始化一个空的SELECT集合字典 selects，键为产生式，值为集合。
   * 遍历每个产生式，根据产生式右部的首符号计算其SELECT集合：
     + 如果首符号是终结符，则直接将其加入SELECT集合。
     + 如果首符号是非终结符，则将其FIRST集合加入SELECT集合，并考虑是否包含ε来更新SELECT集合。

**6. 判断LL(1)文法和冲突求解**

1. **定义判断LL(1)文法的函数**：
   * 编写一个函数 is\_ll1(selects)，接受计算得到的SELECT集合 selects 作为参数。
   * 检查所有产生式的SELECT集合，判断是否存在冲突：
     + 如果存在冲突，记录冲突的产生式。
     + 如果不存在冲突，文法即为LL(1)型文法。

**7. 结果显示和界面更新**

1. **更新界面显示**：
   * 将计算得到的每个非终结符的FIRST集合、FOLLOW集合、SELECT集合以及LL(1)文法判断结果显示在右侧框架的对应文本框中。
   * 使用界面文本框显示每个非终结符的集合，格式如 FIRST(A) = { a, b, c }，FOLLOW(A) = { $, a, b }，SELECT(A -> α) = { a, b, $ }。
   * 如果存在冲突，使用特殊颜色或标记方式显示冲突信息。

**8. 主程序入口**

1. **程序的主入口函数**：
   * 定义主函数或主入口函数，用于创建并显示图形用户界面。
   * 在主函数中调用 create\_window() 或类似函数启动程序。

实验代码如下：

import tkinter as tk

from tkinter import scrolledtext

def create\_window():

root = tk.Tk()

root.title("LL(1)文法判断器")

# 创建左侧框架

left\_frame = tk.Frame(root)

left\_frame.pack(side="left", fill="both", expand=True, padx=10, pady=10)

input\_label = tk.Label(left\_frame,

text="请输入文法的产生式(每行一个产生式，格式:非终结符->产生式|产生式，点击按钮进行计算)：",

fg="blue", font=("Arial", 12))

input\_label.pack()

global text

text = scrolledtext.ScrolledText(left\_frame, height=10, width=15, font=("Verdana", 12))

text.pack(expand=True, fill="both")

calculate\_button = tk.Button(left\_frame, text="开始判断", command=process\_input, fg="blue", font=("Arial", 12))

calculate\_button.pack(pady=10)

# 创建右侧框架

right\_frame = tk.Frame(root)

right\_frame.pack(side="right", fill="both", expand=True, padx=(5, 10), pady=10, ipadx=5)

# 创建右侧框架中的文本结果显示区域

result\_frame = tk.Frame(right\_frame)

result\_frame.pack(expand=True, fill="both")

global first\_result

first\_label = tk.Label(result\_frame, text="FIRST 集合：", fg="red", font=("Arial", 12))

first\_label.pack()

first\_result = scrolledtext.ScrolledText(result\_frame, height=5, width=40, state="disabled", font=("Verdana", 12))

first\_result.pack(expand=True, fill="both")

global follow\_result

follow\_label = tk.Label(result\_frame, text="FOLLOW 集合：", fg="red", font=("Arial", 12))

follow\_label.pack()

follow\_result = scrolledtext.ScrolledText(result\_frame, height=5, width=40, state="disabled", font=("Verdana", 12))

follow\_result.pack(expand=True, fill="both")

global select\_result

select\_label = tk.Label(result\_frame, text="SELECT 集合：", fg="red", font=("Arial", 12))

select\_label.pack()

select\_result = scrolledtext.ScrolledText(result\_frame, height=10, width=40, state="disabled", font=("Verdana", 12))

select\_result.pack(expand=True, fill="both")

global conflict\_result

conflict\_label = tk.Label(right\_frame, text="LL(1)判断结果：", fg="red", font=("Arial", 12))

conflict\_label.pack()

conflict\_result = scrolledtext.ScrolledText(right\_frame, height=5, width=40, state="disabled", font=("Verdana", 12))

conflict\_result.pack(expand=True, fill="both", pady=(0, 10))

root.mainloop()

def process\_input():

productions = {}

input\_text = text.get("1.0", tk.END).strip().split("\n")

for line in input\_text:

if line.strip() == '':

continue

left, right = line.split('->')

left = left.strip()

if left not in productions:

productions[left] = []

productions[left].extend([prod.strip() for prod in right.split('|')])

firsts = calculate\_first\_in(productions)

follows = calculate\_follow\_in(productions, firsts)

selects = calculate\_select\_in(productions, firsts, follows)

first\_text = "\n".join(

[f"FIRST({non\_terminal}) = {{ {', '.join(first)} }}" for non\_terminal, first in firsts.items()])

follow\_text = "\n".join(

[f"FOLLOW({non\_terminal}) = {{ {', '.join(follow)} }}" for non\_terminal, follow in follows.items()])

select\_text = "\n".join([f"SELECT({key}) = {{ {', '.join(select)} }}" for key, select in selects.items()])

first\_result.config(state="normal")

first\_result.delete("1.0", tk.END)

first\_result.insert(tk.END, first\_text)

first\_result.config(state="disabled")

follow\_result.config(state="normal")

follow\_result.delete("1.0", tk.END)

follow\_result.insert(tk.END, follow\_text)

follow\_result.config(state="disabled")

select\_result.config(state="normal")

select\_result.delete("1.0", tk.END)

select\_result.insert(tk.END, select\_text)

select\_result.config(state="disabled")

conflict\_productions = is\_ll1(selects)

if conflict\_productions:

conflict\_info = "该文法不是LL(1)型文法，存在以下冲突:\n"

conflict\_text = "\n".join(

[f"\t• 产生式 {conflict[0][0]}->{conflict[0][1]} 与 {conflict[1][0]}->{conflict[1][1]} 冲突" for conflict in

conflict\_productions])

conflict\_info += conflict\_text

# 显示冲突信息

conflict\_result.config(state="normal")

conflict\_result.delete("1.0", tk.END)

conflict\_result.insert(tk.END, conflict\_info)

conflict\_result.tag\_add("red", "1.0", "end")

conflict\_result.tag\_config("red", foreground="red")

conflict\_result.config(state="disabled")

else:

# 如果是LL(1)型文法，清除冲突信息

conflict\_result.config(state="normal")

conflict\_result.delete("1.0", tk.END)

conflict\_result.insert(tk.END, "该文法是LL(1)型文法")

conflict\_result.tag\_add("red", "1.0", "end")

conflict\_result.tag\_config("red", foreground="red")

conflict\_result.config(state="disabled")

def calculate\_first\_in(productions):

firsts = {}

completed = {}

for non\_terminal in productions:

firsts[non\_terminal] = set()

completed[non\_terminal] = False

def calculate\_first(non\_terminal):

if completed[non\_terminal]:

return firsts[non\_terminal]

visited = set()

stack = [non\_terminal]

while stack:

current = stack.pop()

if current in visited:

continue

visited.add(current)

for production in productions[current]:

if production:

double\_char = production[0:3]

if double\_char in terminal\_e:

firsts[non\_terminal].add(double\_char)

else:

first\_char = production[0]

if (first\_char.islower() or first\_char == 'ε' or first\_char.isdigit() or

first\_char in terminal\_b or first\_char in terminal\_c):

firsts[non\_terminal].add(first\_char)

elif first\_char in productions:

stack.append(first\_char)

completed[non\_terminal] = True

return firsts[non\_terminal]

for non\_terminal in productions:

calculate\_first(non\_terminal)

return firsts

def calculate\_follow\_in(productions, firsts):

follows = {}

for non\_terminal in productions:

follows[non\_terminal] = set()

start\_symbol = list(productions.keys())[0]

follows[start\_symbol].add('$')

while True:

updated = False

for non\_terminal in productions:

for production in productions[non\_terminal]:

follow\_temp = follows[non\_terminal].copy()

for symbol in reversed(production):

if symbol in productions: # 非终结符 S->AB

if follows[symbol].union(follow\_temp) != follows[symbol]:

follows[symbol].update(follow\_temp)

updated = True

# 更新副本

if 'ε' in firsts[symbol]: # S->AB

follow\_temp.update(firsts[symbol] - {'ε'})

else:

follow\_temp = firsts[symbol].copy()

elif symbol != 'ε': # 非终结符 S->Ab

follow\_temp = {symbol}

if not updated:

break

return follows

def calculate\_select\_in(productions, firsts, follows):

selects = {}

for non\_terminal in productions:

for production in productions[non\_terminal]:

select\_key = f"{non\_terminal} -> {production}" # SELECT(S->AB)

if select\_key not in selects:

selects[select\_key] = set()

# 求解FIRST集合

first\_set = set()

if production:

if production[0:3] in terminal\_e:

first\_set.add(production[0:3])

else:

for char in production:

if char in terminal\_b or char in terminal\_c or char.islower() or char.isdigit():

first\_set.add(char)

break

elif char in productions:

first\_set.update(firsts[char])

if 'ε' not in firsts[char]:

break

else:

if 'ε' in firsts.get(production[-1], set()):

first\_set.add('ε')

if 'ε' in first\_set:

first\_set.remove('ε')

first\_set.update(follows[non\_terminal]) # 更新，添加FOLLOW集合中的元素

selects[select\_key].update(first\_set)

return selects

def is\_ll1(selects):

# 记录每个非终结符的选择集

non\_terminal\_selects = {}

# 遍历所有产生式的选择集

for key, select\_set in selects.items():

non\_terminal = key.split('->')[0].strip()

if non\_terminal not in non\_terminal\_selects:

non\_terminal\_selects[non\_terminal] = []

# 记录产生式左部非终结符、产生式右部和选择集

non\_terminal\_selects[non\_terminal].append((non\_terminal, key.split('->')[1].strip(), select\_set))

# 记录冲突的产生式

conflict\_productions = []

# 遍历所有非终结符的产生式选择集

for non\_terminal, productions in non\_terminal\_selects.items():

for i in range(len(productions)):

current\_production = productions[i]

for j in range(i + 1, len(productions)):

next\_production = productions[j]

if set(current\_production[2]).intersection(set(next\_production[2])):

conflict\_productions.append((current\_production, next\_production))

return conflict\_productions

if \_\_name\_\_ == "\_\_main\_\_":

terminal\_b = ['+', '-', '\*', '/', '^', '%', '&', '=', '!']

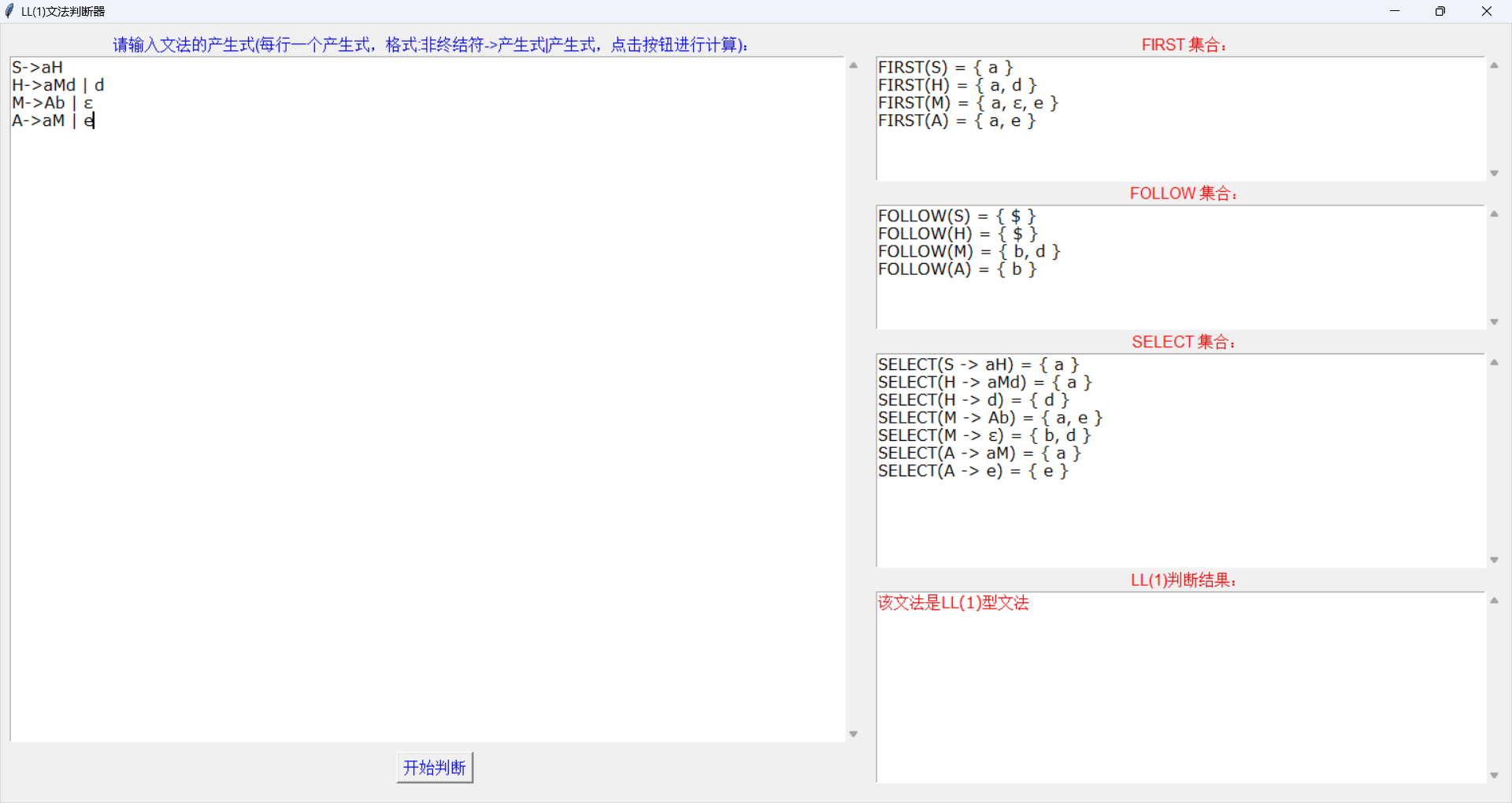
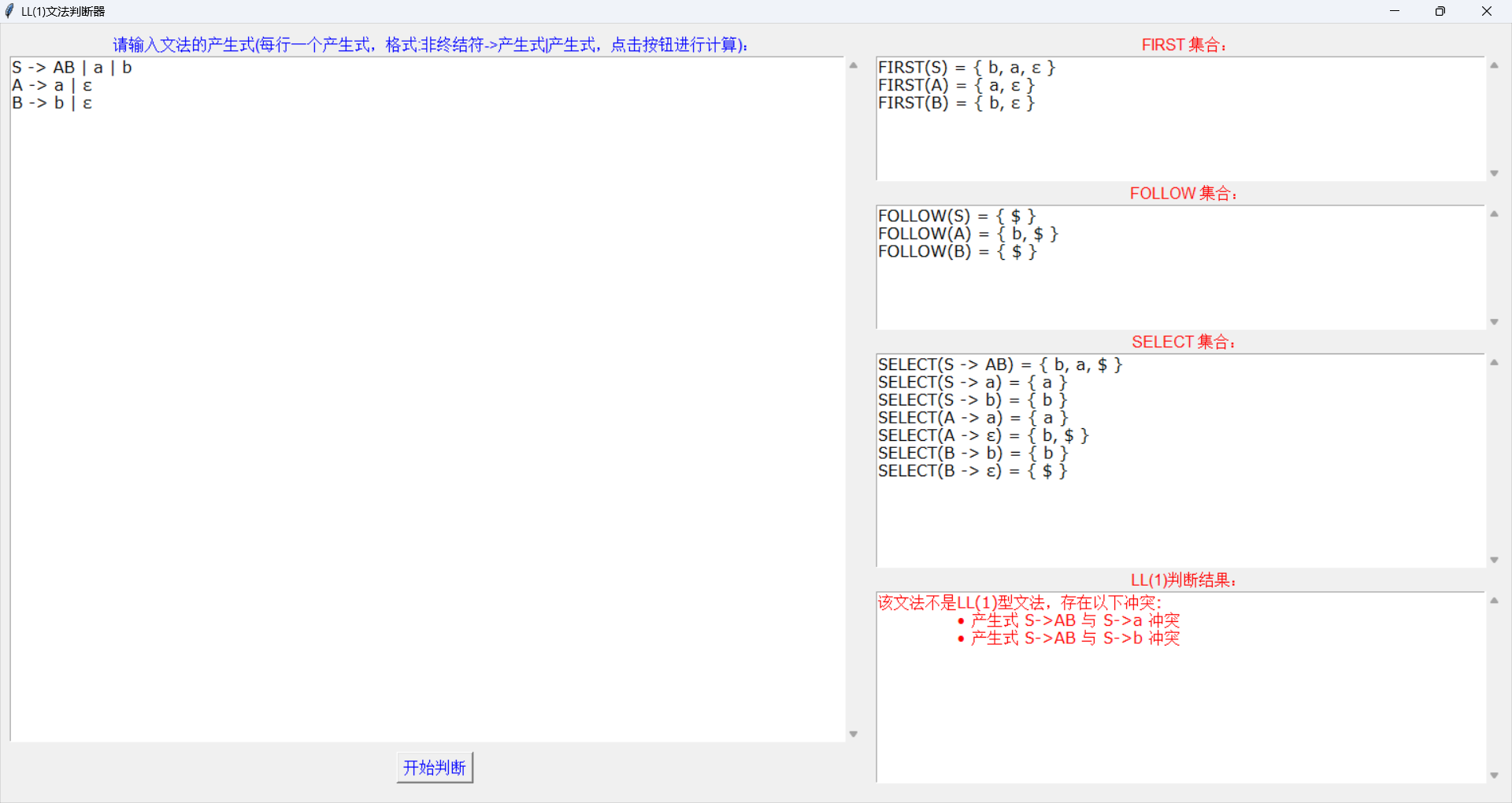
terminal\_c = ['[', ']', '{', '}', '(', ')', ',', '.', '\\', '"', '\'', '#', '@']

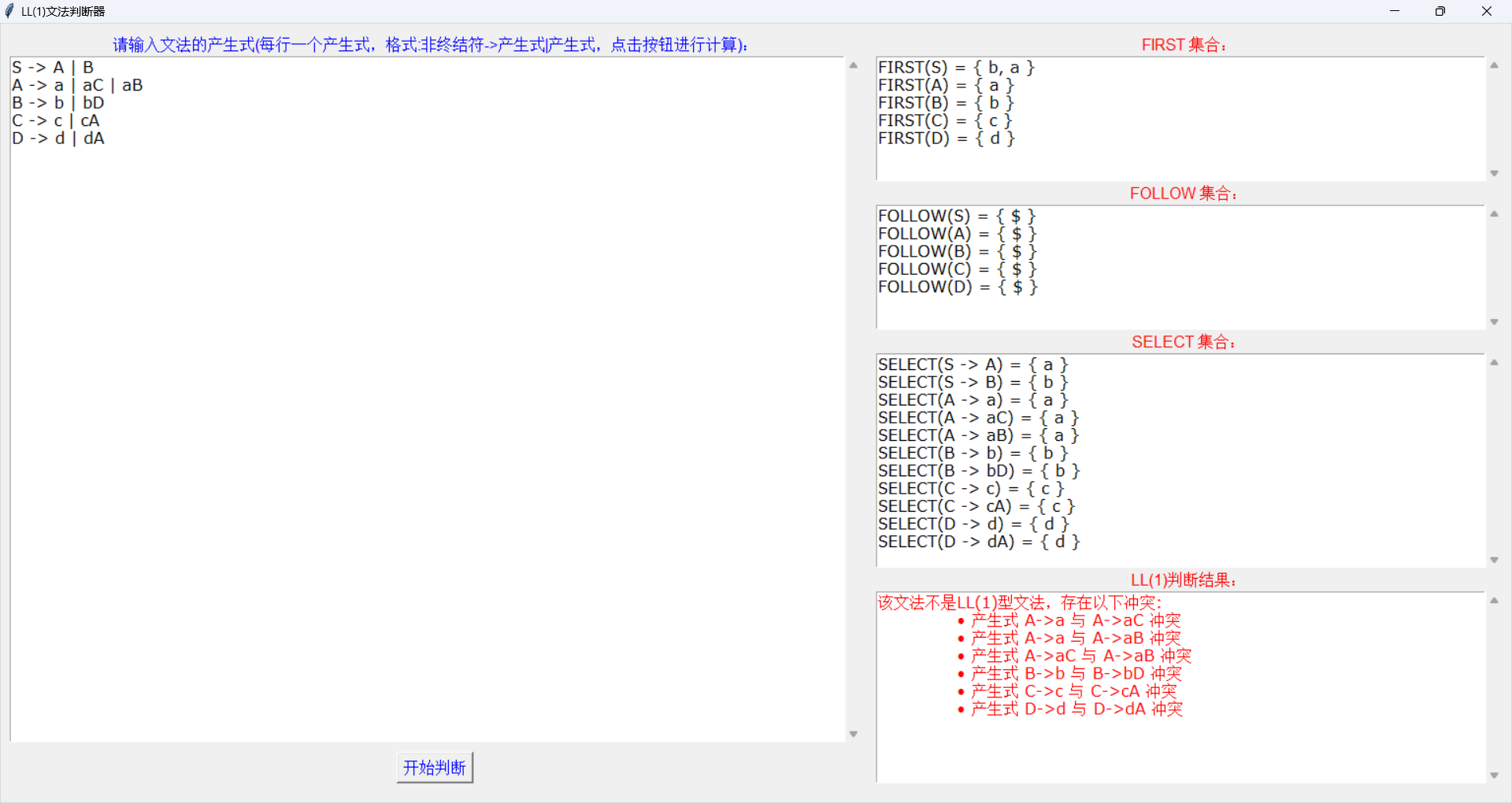
terminal\_e = ['id', 'if']

create\_window()

**三、实验过程记录：**

实验结果截图：





**四、实验总结：**

通过本次实验，设计和实现了一个完整的LL(1)文法判断器。覆盖了界面设计、用户输入处理、FIRST集合计算、FOLLOW集合计算、SELECT集合计算、LL(1)文法判断、冲突显示和结果显示等核心功能。这样的设计能够帮助深入理解和分析文法的结构和特性，同时通过图形界面直观地展示计算结果。

这也加深了我对文法的理解和分析、FOLLOW集合计算、SELECT集合计算、LL(1)文法判断和冲突的求解。