SLR(0)分析算法

实验内容

给定下面的文法

```
(1) E -> E + T
(2) E -> T
(3) T -> T * F
(4) T -> F
(5) F -> ( E )
(6) F -> id
```

输出字符串 id + (id + id) * id的语法树

实验步骤

1. 构建拓广文法

read_grams()：从文件中读入初始文法，并将它转化为**拓广文法**

```
grams = read_grams()
for idx in range(len(grams)):
    print(str(idx)+":",grams[idx])
```

运行结果：

```
0: S->E
1: E->E+T
2: E->T
3: T->T*F
4: T->F
5: F->(E)
6: F->i
```

2. 使用SLR(0)分析算法生成可行前缀DFA

```
items, goto, chars = get_items()

for idx in range(len(items)):
    print("I"+str(idx)+":",items[idx])
print("goto:")
print("chars:",chars)
for idx in range(len(goto)):
    print("I"+str(idx)+":",goto[idx])
```

运行结果：

```
I0: ['T->.F', 'E->.T', 'F->.i', 'S->.E', 'F->.(E)', 'E->.E+T', 'T->.T*F']
I1: ['T->.F', 'E->.T', 'F->.i', 'F->(.E)', 'F->.(E)', 'E->.E+T', 'T->.T*F']
I2: ['F->i.']
I3: ['E->E.+T', 'S->E.']
I4: ['T->F.']
I5: ['T->T.*F', 'E->T.']
I6: ['E->E.+T', 'F->(E.)']
I7: ['T->.F', 'F->.i', 'F->.(E)', 'T->.T*F', 'E->E+.T']
I8: ['F->.i', 'T->T*.F', 'F->.(E)']
I9: ['F->(E).']
I10: ['T->T.*F', 'E->E+T.']
I11: ['T->T*F.']
```

```
goto:
chars: ['+', '*', '(', ')', 'i', '$', 'T', 'E', 'S', 'F']
I0: [None, None, 1, None, 2, None, 3, 4, None, 5]
I1: [None, None, 1, None, 2, None, 3, 6, None, 5]
I2: [None, None, None, None, None, None, None, None, None, None]
I3: [None, 7, None, None, None, None, None, None, None, None]
I4: [8, None, None, None, None, None, None, None, None, None]
I5: [None, None, None, None, None, None, None, None, None, None]
I6: [8, None, None, 9, None, None, None, None, None, None]
I7: [None, None, 1, None, 2, None, None, None, None, 10]
I8: [None, None, 1, None, 2, None, 11, None, None, 5]
I9: [None, None, None, None, None, None, None, None, None, None]
I10: [None, None, None, None, None, None, None, None, None, None]
I11: [None, 7, None, None, None, None, None, None, None, None]
```

具体分析：

```python
def get_items():
    items = [get_closure(["S->.E"])]
    chars = get_chars()
    goto = []
    idx = -1
    for i in items:
        idx += 1
        goto.append([None for _ in range(len(chars))])
        for c_idx in range(len(chars)):
            next = get_next(i, chars[c_idx])
            if len(next) == 0:
                continue
            # add next
            flag = True
            for check_idx in range(len(items)):
                if set(items[check_idx]) == set(next):
                    flag = False
                    goto[idx][c_idx] = check_idx
                    break
            if flag:
                items.append(next)
                goto[idx][c_idx] = len(items) - 1
    return items, goto, chars
```

1. 手动生成I0状态（S->.E和根据此条语法求出的闭包）

2. 不断枚举**items**状态列表中的状态**i**直到枚举完：

（1） 对于所有的非终结符和终结符，判断是否可以产生状态转化

（2） 如果可以，判断新状态是否已经再items状态列表中。如果不在，那么append到列表尾部，用于后续**i**的循环，并在goto表中添加相应标记；如果已经存在此状态，那么只需要在goto表中添加标记即可

（3）如果不可以，那么continue，枚举下一个状态**i**

3. 生成SLR(0)状态转换表

```
action = get_table(items, goto, chars)

print("action:")
print("chars:", chars)
for idx in range(len(action)):
    print("I" + str(idx) + ":", action[idx])
```

运行结果：

```
action:
chars: ['+', '*', '(', ')', 'i', '$', 'T', 'E', 'F', 'S']
I0: [None, None, 's1', None, 's2', None, '3', '4', '5', None]
I1: [None, None, 's1', None, 's2', None, '3', '6', '5', None]
I2: ['r6', 'r6', None, 'r6', None, 'r6', None, None, None, None]
I3: ['r2', 's7', None, 'r2', None, 'r2', None, None, None, None]
I4: ['s8', None, None, None, None, 'acc', None, None, None, None]
I5: ['r4', 'r4', None, 'r4', None, 'r4', None, None, None, None]
I6: ['s8', None, None, 's9', None, None, None, None, None, None]
I7: [None, None, 's1', None, 's2', None, None, None, '10', None]
I8: [None, None, 's1', None, 's2', None, '11', None, '5', None]
I9: ['r5', 'r5', None, 'r5', None, 'r5', None, None, None, None]
I10: ['r3', 'r3', None, 'r3', None, 'r3', None, None, None, None]
I11: ['r1', 's7', None, 'r1', None, 'r1', None, None, None, None]
```

具体分析：

1. 首先获取**first**集合和**follow**集合

求解规则如下：

- FIRST
  - 计算X的FIRST(X)时，不断运用以下规则，直到没有新的终结符或ε可以被加入到FIRST(X)
    - 如果X是终结符，FIRST(X)=X
    - 如果$X \rightarrow Y_1Y_2\ldots Y_k$，且a在FIRST($Y_i$)集合中，并且$Y_1 \Rightarrow^* \varepsilon, Y_2 \Rightarrow^* \varepsilon, \ldots, Y_{i-1} \Rightarrow^* \varepsilon$，则将a插入到FIRST(X)中。
    - 如果$X \rightarrow \varepsilon$是一个产生式，则将ε插入到FIRST($X$)中。

- FOLLOW
  - 计算非终结符 A 的 FOLLOW(A) 时，不断运用以下规则，直到没有新的终结符可以被加入到 FOLLOW(A)
    - 将 $ 放到 FOLLOW(S) 中，S 是开始符，$ 是输入右端的结束标记。
    - 如果存在 $A \rightarrow \alpha B \beta$，那么 FIRST(β) 中非 ε 的所有符号都在 FOLLOW(B) 中。
    - 如果存在 $A \rightarrow \alpha B$ 或 $A \rightarrow \alpha B \beta$ 且 FIRST(β) 包含 ε，则 FOLLOW(A) 中的所有符号都在 FOLLOW(B) 中。

代码如下：

```
def get_first():
    first = {}
    for gram in grams:
        x, y = gram.split("->")
        if not y[0].isupper():
            first[x] = first.get(x, "") + y[0]
    while True:
        cpy = copy.deepcopy(first)
        for gram in grams:
            x, y = gram.split("->")
            if y[0].isupper():
                first[x] = first.get(x, "") + first.get(y[0], "")
        same_flag = True
        for x, y in first.items():
            new_set, old_set = set(list(y)), set(list(cpy.get(x, "")))
            if new_set != old_set:
                same_flag = False
            first[x] = "".join(new_set)
        if same_flag:
            break
    return first
```

```
def get_follow(first):
    follow = {"S": "$"}
    while True:
        cpy = copy.deepcopy(follow)
        for gram in grams:
            x, y = gram.split("->")
            for idx in range(len(y) - 1):
                # A->…Ba…
                if y[idx].isupper() and not y[idx + 1].isupper():
                    follow[y[idx]] = follow.get(y[idx], "") + y[idx + 1]
                # A->…BC…
                if y[idx].isupper() and y[idx + 1].isupper():
                    follow[y[idx]] = follow.get(y[idx], "") + first.get(y[idx +
1], "").replace("e", "")
            # A->…B
            if y[-1].isupper():
                follow[y[-1]] = follow.get(y[-1], "") + follow.get(x[0], "")
            # A->…BC
```

```
            if len(y) >= 2 and y[-1].isupper() and y[-2].isupper():
                if first.get(y[-1], "").find("e") != -1:
                    follow[y[-2]] = follow.get(y[-2], "") + follow.get(x[0], "")
        # 去重
        same_flag = True
        for x, y in follow.items():
            new_set, old_set = set(list(y)), set(list(cpy.get(x, "")))
            if new_set != old_set:
                same_flag = False
            follow[x] = "".join(new_set)
        if same_flag:
            break
    return follow
```

2. 根据如下规则，利用goto表和first，follow集合求解出action表：

- 从DFA构造SLR分析表
  - 状态 i 从 I_i 构造，它的action函数如下确定：
    - 如果$[A\to\alpha\cdot a\beta]$在I_i中，并且$goto(I_i, a)=I_j$，那么置$action[i, a]$为sj
    - 如果$[A\to\alpha\cdot]$在I_i中，那么对FOLLOW(A)中的所有a，置$action[i, a]$为rj，j是产生式$A\to\alpha$的编号
    - 如果$[S'\to S\cdot]$在I_i中，那么置$action[i, \$]$为接受$acc$
  - 使用下面规则构造状态i的goto函数：
    - 对所有的非终结符A，如果$goto(I_i, A)=I_j$，那么$goto[i, A]=j$

代码如下：

```
def get_table(items, goto, chars):
    action = [[None for _ in range(len(chars))] for _ in range(len(items))]
    first = get_first()
    follow = get_follow(first)
    final_idx = list_find(chars, "$")
    for idx in range(len(items)):
        # 如果[A->a.]在Ii中，那么对FOLLOW(A)中的所有b，置action[i, b]为rj，j是产生式A->a的编号
        for gram in items[idx]:
            if gram[-1] == ".":
                for b in follow[gram[0]]:
                    action[idx][list_find(chars, b)] = "r" + str(list_find(grams, gram[:-1]))
                continue
            # 如果[A->a□ab]在Ii中，并且goto(Ii, a ) = Ij，那么置action[i, a]为sj
            next_char_idx = list_find(chars, gram.split(".")[1][0])
            if goto[idx][next_char_idx] != None:
                action[idx][next_char_idx] = "s" + str(goto[idx][next_char_idx])
        # 如果[S->E.]在Ii中，那么置action[i, $] = acc
        if "S->E." in items[idx]:
            action[idx][final_idx] = "acc"
    # 对所有的非终结符A，如果goto(Ii, A) = Ij，那么goto[i, A] = j
    for i in action:
        for j in range(len(chars)):
            if chars[j].isupper() and i[j] is not None:
                i[j] = i[j][1:]
    return action
```

4. 根据得到的action表对sentence进行SLR(0)分析

步骤：

1. 定义一个Node类，表示语法树上的点，其中**ch**表示当前点的符号，**child**存储子节点：

```
class Node:
    def __init__(self, ch=None):
        self.ch = ch
        self.child = []

    def to_tree(self):
        if len(self.child) != 0:
            res = self.ch
            for i in self.child:
                res += i.to_tree()
            return "[" + res + "]"
        else:
            return "[" + self.ch + "]"
```

2. 定义一个栈stack，并不断地取出待处理sentence的第一位，在action表中找到栈顶行，sentence的第一位列，将其中存储的字符串定义为act

   如果act的第一位是**s**，代表移进，那么将sentence的第一位移到栈顶，继续读取sentence的下一位

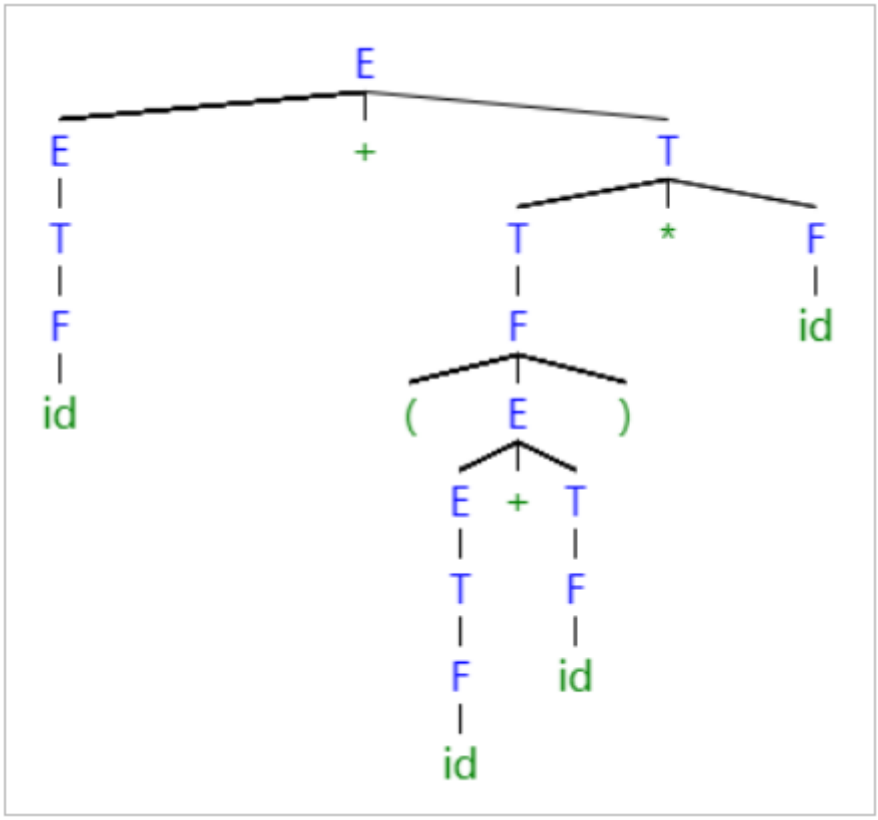
   如果act的第一位是**r**，代表规约，则按照r后面的数字到grams中寻找对应语法规则进行规约

   如果act的第一位是**a**，代表结束，那么跳出循环并返回语法树

```
def analyse(action, sentence):
    sentence = sentence.replace("id", "i")
    sentence = sentence + "$"
    stack = [(None, 0, None)]
    while (True):
        act = action[stack[-1][1]][list_find(chars, sentence[0])]
        if act[0] == "s":
            stack.append((sentence[0], int(act[1:]), Node(sentence[0])))
            sentence = sentence[1:]
        if act[0] == "r":
            gram = grams[int(act[1:])]
            new_node = Node(gram[0])
            for i in range(len(gram.split("->")[1])):
                temp_node = stack.pop()[2]
                new_node.child.insert(0, temp_node)
            stack.append((gram[0], int(action[stack[-1][1]][list_find(chars,
gram[0])]), new_node))
        if act[0] == "a":
            break
    return stack[1][2].to_tree().replace("i", "id")
```

实验结果

```
[E[E[T[F[id]]]][+][T[T[F[(][E[E[T[F[id]]]][+][T[F[id]]]][)]]][*][F[id]]]]
```


E
E
+
T
T
*
F
T
F
id
F
id
(
E
)
E
+
T
T
F
F
id
id