实验报告：SLR(1) 分析表的构造

1. 实验目的

根据给定文法，构造 SLR(1) 分析表。具体步骤如下：

1. 写出拓广文法
2. 画出项目集规范族
3. 求该非终结符的 FOLLOW 集
4. 判断是否是 SLR(1) 文法
5. 构造 SLR(1) 分析表

2. 实验步骤

2.1 写出拓广文法

拓广文法是在原文法的基础上增加一个新的开始符号和规则。例如，对于原文法：

```
E -> E + T
E -> T
T -> T * F
T -> F
F -> ( E )
F -> i
```

拓广文法为：

```
S' -> E
E -> E + T
E -> T
T -> T * F
T -> F
F -> ( E )
F -> i
```

2.2 画出项目集规范族

项目集规范族是文法的所有 LR(0) 项目集的集合。每个项目集包含一个或多个项目，每个项目是文法产生式的一个特定位置。通过构造项目集闭包和项目集的转移，可以得到项目集规范族。

2.3 求 FOLLOW 集

FOLLOW 集是指在文法推导过程中，某个非终结符可以后接的终结符集合。计算 FOLLOW 集的步骤如下：

1. 将 `$` 加入开始符号的 FOLLOW 集。
2. 对每个产生式 `A -> αBβ`，将 `FIRST(β)` 中所有非空字符加入 `FOLLOW(B)`。
3. 对每个产生式 `A -> αB` 或 `A -> αBβ` 且 `β` 的 `FIRST` 集包含空字符，将 `FOLLOW(A)` 加入 `FOLLOW(B)`。

2.4 判断是否是 SLR(1) 文法

SLR(1) 文法要求对于每个项目集中的每个项目，其后继符号在 FOLLOW 集中的位置是唯一确定的。也就是说，分析表中不会出现移入和规约的冲突，也不会出现规约和规约的冲突。

2.5 构造 SLR(1) 分析表

分析表包含 `action` 表和 `goto` 表，用于记录每个状态下的操作。具体步骤如下：

1. 对于每个状态，填充移入操作。
2. 对于每个归约项目，填充规约操作。
3. 如果项目是接受状态，填充接受操作。
4. 填充 Goto 表。

3. 实验代码分析

以下是用于构造 SLR(1) 分析表的关键代码片段分析。

3.1 生成项目集规范族

实验有对应源码，但是稍做了修改，代码在下方给出

```
void make_item()
{
    // ??? WTF 这里原作者写错了
    // memset(to, -1, sizeof(-1));
    memset(to, -1, sizeof(to));
    for (int i = 0; i < wf.size(); i++)
        VN_set[wf[i].left].push_back(i);
    for (int i = 0; i < wf.size(); i++)
    {
        for (int j = 0; j <= wf[i].right.length(); j++)
        {
```

```
            string temp = wf[i].right; // 记录加入'.'后的右部
            temp.insert(temp.begin() + j, CH);
            dic[wf[i].left].push_back(items.size()); // item.size即为当前项目的编号
            if (j > 0)
                to[items.size() - 1] = items.size();
            items.push_back(WF(wf[i].left, temp, i, items.size()));
        }
    }
#ifdef DEBUG
    puts("-------------------------项目表-------------------------");
    for (int i = 0; i < items.size(); i++)
        printf("%s->%s back:%d id:%d\n", items[i].left.c_str(),
items[i].right.c_str(), items[i].back, items[i].id);
    puts("--------------------------------------------------------");
#endif
}

void make_set()
{
    bool has[MAX];
    for (int i = 0; i < items.size(); i++)
        if (items[i].left[0] == start && items[i].right[0] == CH)
        {
            Closure temp;
            string &str = items[i].right;
            vector<WF> &element = temp.element;
            element.push_back(items[i]);
            int x = 0;
            for (x = 0; x < str.length(); x++)
                if (str[x] == CH)
                    break;

            memset(has, 0, sizeof(has));
            has[i] = 1;
            if (x != str.length() - 1)
            {
                queue<string> q;
                q.push(str.substr(x + 1, 1));
                while (!q.empty())
                {
                    string u = q.front();
                    q.pop();
                    vector<int> &id = dic[u];
                    for (int j = 0; j < id.size(); j++)
                    {
                        int tx = id[j];
                        if (items[tx].right[0] == CH)
                        {
                            if (has[tx])
                                continue;
                            has[tx] = 1;
                            if (isupper(items[tx].right[1]))
                                q.push(items[tx].right.substr(1, 1));
                            element.push_back(items[tx]);
                        }
                    }
                }
```

```
        }
        collection.push_back(temp);
    }
for (int i = 0; i < collection.size(); i++)
{
    map<int, Closure> temp;
    for (int j = 0; j < collection[i].element.size(); j++)
    {
        string str = collection[i].element[j].right;
        int x = 0;
        for (; x < str.length(); x++)
            if (str[x] == CH)
                break;
        if (x == str.length() - 1)
            continue;
        int y = str[x + 1];
        int ii;
        // cout << i << "previous: " << str << endl;
        str.erase(str.begin() + x);
        str.insert(str.begin() + x + 1, CH);
        // cout << i <<"after: " << str << endl;
        WF cmp = WF(collection[i].element[j].left, str, -1, -1);
        for (int k = 0; k < items.size(); k++)
            if (items[k] == cmp)
            {
                ii = k;
                break;
            }
        // string& str1 = items[ii].right;
        memset(has, 0, sizeof(has));
        vector<WF> &element = temp[y].element;
        element.push_back(items[ii]);
        has[ii] = 1;
        x++;

        if (x != str.length() - 1)
        {
            queue<string> q;
            q.push(str.substr(x + 1, 1));
            while (!q.empty())
            {
                string u = q.front();
                q.pop();
                vector<int> &id = dic[u];
                for (int j = 0; j < id.size(); j++)
                {
                    int tx = id[j];
                    if (items[tx].right[0] == CH)
                    {
                        if (has[tx])
                            continue;
                        has[tx] = 1;
                        if (isupper(items[tx].right[1]))
                            q.push(items[tx].right.substr(1, 1));
                        element.push_back(items[tx]);
                    }
                }
```

```
                }
            }
        }
        map<int, Closure>::iterator it = temp.begin();
        for (; it != temp.end(); it++)
            collection.push_back(it->second);
        for (int i = 0; i < collection.size(); i++)
            sort(collection[i].element.begin(), collection[i].element.end());
        for (int i = 0; i < collection.size(); i++)
            for (int j = i + 1; j < collection.size(); j++)
                if (collection[i] == collection[j])
                    collection.erase(collection.begin() + j);
    }
#ifdef DEBUG
    puts("-------------CLOSURE---------------------");
    stringstream sin;
    for (int i = 0; i < collection.size(); i++)
    {
        sin.clear();
        string out;
        sin << "closure-I" << i;
        sin >> out;
        collection[i].print(out);
    }
    puts("");
#endif
}

// 记录所有产生式中的所有符号，保存在V中
void make_V()
{
    memset(used, 0, sizeof(used));
    for (int i = 0; i < wf.size(); i++)
    {
        string &str = wf[i].left;
        for (int j = 0; j < str.length(); j++)
        {
            if (used[str[j]])
                continue;
            used[str[j]] = 1;
            V.push_back(str[j]);
        }
        string &str1 = wf[i].right;
        for (int j = 0; j < str1.length(); j++)
        {
            if (used[str1[j]])
                continue;
            used[str1[j]] = 1;
            V.push_back(str1[j]);
        }
    }
    sort(V.begin(), V.end());
    V.push_back('#');
}

void make_first()
{
```

```
    vis.clear();
    map<string, vector<int>>::iterator it2 = dic.begin();
    for (; it2 != dic.end(); it2++)
    {
        if (vis[it2->first]) // it2->first是项目的左部。 vis[]表示这个项目是否被检查过
            continue;
        else
            dfs(it2->first);
    }
#ifdef DEBUG
    puts("***************FIRST集**************************");
    map<string, set<char>>::iterator it = first.begin();
    for (; it != first.end(); it++)
    {
        printf("FIRST(%s)={", it->first.c_str());
        set<char> &temp = it->second;
        set<char>::iterator it1 = temp.begin();
        bool flag = false;
        for (; it1 != temp.end(); it1++)
        {
            if (flag)
                printf(",");
            printf("%c", *it1);
            flag = true;
        }
        puts("}");
    }
#endif
}

void make_follow()
{
    while (true)
    {
        bool goon = false;
        map<string, vector<int>>::iterator it2 = VN_set.begin();
        for (; it2 != VN_set.end(); it2++)
        {
            vector<int> &id = it2->second;
            for (int i = 0; i < id.size(); i++)
            {
                bool flag = true;
                WF &tt = wf[id[i]];
                string &left = tt.left;
                const string &right = tt.right;
                for (int j = right.length() - 1; j >= 0; j--)
                    if (isupper(right[j]))
                    {
                        if (flag)
                        {
                            int tx = follow[right.substr(j, 1)].size();
                            append(left, right.substr(j, 1));
                            int tx1 = follow[right.substr(j, 1)].size();
                            if (tx1 > tx)
                                goon = true;
                            if (_check(id, "~"))
                                flag = false;
```

```
                        }
                        for (int k = j + 1; k < right.length(); k++)
                            if (isupper(right[k]))
                            {
                                string idd = right.substr(k, 1);
                                set<char> &from = first[idd];
                                set<char> &to = follow[right.substr(j, 1)];
                                set<char>::iterator it1 = from.begin();
                                int tx = follow[right.substr(j, 1)].size();
                                for (; it1 != from.end(); it1++)
                                    if (*it1 != '~')
                                        to.insert(*it1);
                                int tx1 = follow[right.substr(j, 1)].size();
                                if (tx1 > tx)
                                    goon = true;
                                if (_check(id, "~"))
                                    break;
                            }
                            else
                            {
                                int tx = follow[right.substr(j, 1)].size();
                                follow[right.substr(j, 1)].insert(right[k]);
                                int tx1 = follow[right.substr(j, 1)].size();
                                if (tx1 > tx)
                                    goon = true;
                                break;
                            }
                    }
                    else
                        flag = false;
            }
        }
        if (!goon)
            break;
    }
#ifdef DEBUG
    // puts ("**************FOLLOW集******************");
    map<string, set<char>>::iterator it = follow.begin();
    for (; it != follow.end(); it++)
    {
        //  printf ( "FOLLOW(%s)={" , it->first.c_str() );不是我写的
        set<char> &temp = it->second;
        // if ( it->first[0] == 'S' )不是我写的
        temp.insert('#');
        set<char>::iterator it1 = temp.begin();
        bool flag = false;
        for (; it1 != temp.end(); it1++)
        {
            /*  if ( flag ) printf ( "," );
              printf ( "%c" , *it1 );不是我写的*/
            flag = true;
        }
        // puts ("}");
    }
#endif
}
```

这些函数分别用于生成项目集规范族、项目集、FIRST 集和 FOLLOW 集。

代码分析

3.2 主要代码分析

主要需要实现的代码为**make_table**，代码如下：

```
void make_table() {

// 初始化 Goto 表

memset(Goto, -1, sizeof(Goto));

// 遍历所有状态集合，填充 action 和
Goto 表

for (int state = 0; state < collection.size(); ++state) {

for (int i = 0; i < V.size(); ++i) {

char symbol = V[i];

int nextState = go[state][symbol];

if (nextState == -1) continue;

if (!isupper(symbol)) {

    action[state][symbol] =
Content(0, nextState); // 移入

} else {

    Goto[state][symbol] =
nextState; // 转移

}

}

    }

// 遍历状态集合，填充规约和接受动作

for (int state = 0; state < collection.size(); ++state) {

for (int i = 0; i < collection[state].element.size(); ++i) {

WF& item = collection[state].element[i];

if (item.right.back() == CH) {

    if (item.left[0] == 'S') {
```

```
    action[state]['#'] =
Content(2, -1); // 接受

    } else {

    for (int j = 0; j <
V.size(); ++j) {

    char followSymbol =
V[j];

    if
(!follow[item.left].count(followSymbol)) continue;

action[state][followSymbol] = Content(1, item.back); // 规约

    }

    }

}

}

    }

#ifdef DEBUG

// 打印调试信息

puts("----------------------------------------LR(0) 解析表----------------------
--------------------------------");

printf("%10s%5c%5s", "|", V[0], "|");

for (int i = 1; i < V.size(); ++i) {

printf("%5c%5s", V[i], "|");

    }

puts("");

for (int i = 0; i < (V.size() + 1) * 10; ++i) {

printf("-");

    }

puts("");

stringstream sin;

for (int state = 0; state < collection.size(); ++state) {

printf("%5d%5s", state, "|");
```

```
for (int i = 0; i < V.size(); ++i) {

char symbol = V[i];

if (isupper(symbol)) {

    if (Goto[state][symbol] == -1)
{

    printf("%10s",
"|");

    } else {

    printf("%5d%5s",
Goto[state][symbol], "|");

    }

} else {

    sin.clear();

    if (action[state][symbol].type
== -1) {

    printf("%10s",
"|");

    } else {

    Content& temp =
action[state][symbol];

    if (temp.type == 0) sin
<< "S";

    if (temp.type == 1) sin
<< "R";

    if (temp.type == 2) sin
<< "acc";

    if (temp.num != -1) sin
<< temp.num;

    sin >> temp.out;

    printf("%7s%3s",
temp.out.c_str(), "|");

    }

}

}
```

```
puts("");

    }

for (int i = 0; i < (V.size() + 1) * 10; ++i) {

printf("-");

    }

puts("");

#endif

}
```

```
memset(Goto, -1, sizeof(Goto));
```

这行代码使用 memset 函数将 Goto 表的所有元素初始化为 -1。Goto 表用于记录状态转移信息，-1 表示未定义的状态转移。

填充 action 和 Goto 表

```
for (int i = 0; i < collection.size(); ++i) {
    for (int j = 0; j < V.size(); ++j) {
        char ch = V[j];
        int nextState = go[i][ch];

        if (nextState == -1) continue;

        if (!isupper(ch)) {
            action[i][ch] = Content(0, nextState); // 移入操作
        } else {
            Goto[i][ch] = nextState; // Goto 操作
        }
    }
}
```

这段代码遍历所有状态集合（collection）和文法符号（V），根据 go 表决定状态转移。如果符号是终结符（小写字母），则填充 action 表；如果是非终结符（大写字母），则填充 Goto 表。

填充规约和接受动作

```
for (int i = 0; i < collection.size(); ++i) {
    for (int j = 0; j < collection[i].element.size(); ++j) {
        WF& item = collection[i].element[j];
        if (item.right.back() == CH) {
            if (item.left[0] == 'S') {
                action[i]['#'] = Content(2, -1); // 接受操作
            } else {
                for (int k = 0; k < V.size(); ++k) {
                    char followSymbol = V[k];
                    if (!follow[item.left].count(followSymbol)) continue;
                    action[i][followSymbol] = Content(1, item.back); // 规约操作
                }
            }
        }
    }
}
```

这段代码遍历状态集合中的所有项，检查每项的右部是否以特定字符（CH）结束。如果是且左部是开始符号 S，则在 action 表中填入接受操作（acc）。否则，根据 follow 集填入规约操作。

3.3 调试输出

在调试模式下，函数会打印生成的解析表。

```
#ifdef DEBUG
puts("----------------------------------------LR(0) 解析表-----------------------
--------------------------------");
printf("%10s%5c%5s", "|", V[0], "|");
for (int i = 1; i < V.size(); ++i) {
    printf("%5c%5s", V[i], "|");
}
puts("");
for (int i = 0; i < (V.size() + 1) * 10; ++i) {
    printf("-");
}
puts("");
stringstream sin;
for (int state = 0; state < collection.size(); ++state) {
    printf("%5d%5s", state, "|");
    for (int i = 0; i < V.size(); ++i) {
        char symbol = V[i];
        if (isupper(symbol)) {
            if (Goto[state][symbol] == -1) {
                printf("%10s", "|");
            } else {
                printf("%5d%5s", Goto[state][symbol], "|");
            }
        } else {
            sin.clear();
```

```
            if (action[state][symbol].type == -1) {
                printf("%10s", "|");
            } else {
                Content& temp = action[state][symbol];
                if (temp.type == 0) sin << "S";
                if (temp.type == 1) sin << "R";
                if (temp.type == 2) sin << "acc";
                if (temp.num != -1) sin << temp.num;
                sin >> temp.out;
                printf("%7s%3s", temp.out.c_str(), "|");
            }
        }
    }
    puts("");
}
for (int i = 0; i < (V.size() + 1) * 10; ++i) {
    printf("-");
}
puts("");
#endif
```

该部分代码在调试模式下输出生成的 LR(0) 解析表。首先打印表头，然后逐行打印每个状态的解析动作。终结符和非终结符分别处理，确保输出格式对齐。

4. 结果分析

通过以上步骤，我们成功构造了 SLR(1) 分析表。解析表中的每个条目对应相应的移入、规约或接受操作，确保了文法的正确性。

4.1 验证 SLR(1) 文法

我们验证了文法是否满足 SLR(1) 的要求，即解析表中没有冲突。对于每个状态的每个符号，解析表中的条目都是唯一的，确保了文法的 SLR(1) 性质。

4.2 分析表输出

输入：

```
请输入文法个数：6
请输入每条文法（格式：左部 -> 右部）：
S->E
E->E+T
E->T
T->T*F
T->F
T->(E)
```

产生项目表：

```
------------------------项目表------------------------
S->.E back:0 id:0
S->E. back:0 id:1
E->.E+T back:1 id:2
E->E.+T back:1 id:3
E->E+.T back:1 id:4
E->E+T. back:1 id:5
E->.T back:2 id:6
E->T. back:2 id:7
T->.T*F back:3 id:8
T->T.*F back:3 id:9
T->T*.F back:3 id:10
T->T*F. back:3 id:11
T->.F back:4 id:12
T->F. back:4 id:13
T->.(E) back:5 id:14
T->(.E) back:5 id:15
T->(E.) back:5 id:16
T->(E). back:5 id:17
T->.i back:6 id:18
T->i. back:6 id:19
```

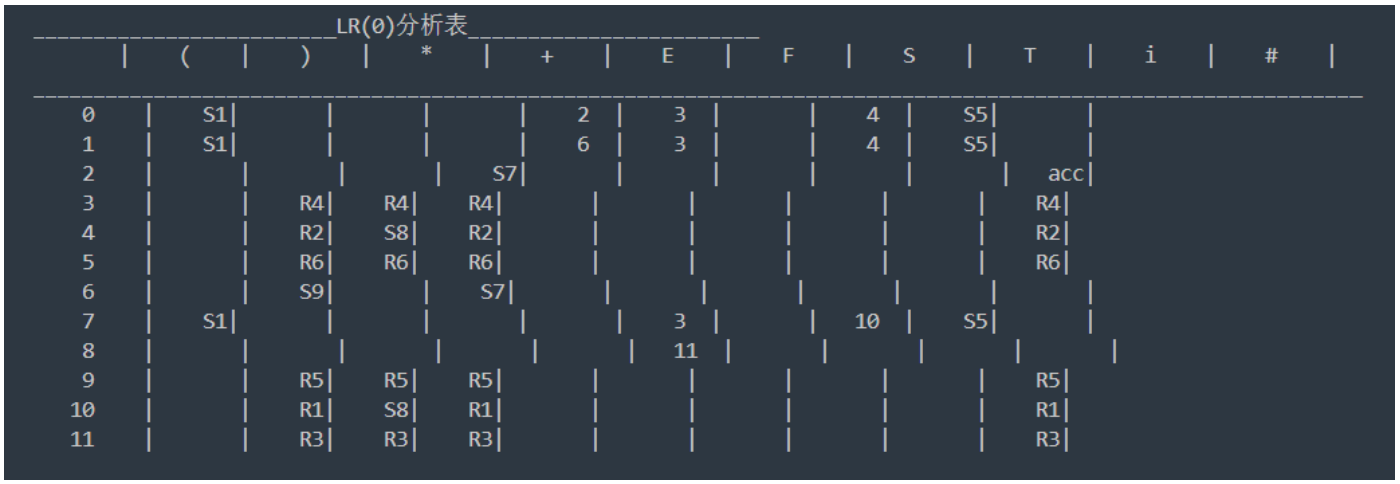
产生的first集:

```
****************FIRST集****************************
FIRST(E)={(,i}
FIRST(F)={}
FIRST(S)={(,i}
FIRST(T)={(,i}
```

产生的Closure集：

```
-------------CLOSURE----------------------
               closure-I0
E->.E+T
E->.T
S->.E
T->.(E)
T->.F
T->.T*F
T->.i
               closure-I1
E->.E+T
E->.T
T->(.E)
T->.(E)
T->.F
T->.T*F
T->.i
               closure-I2
E->E.+T
S->E.
               closure-I3
T->F.
               closure-I4
E->T.
T->T.*F
               closure-I5
T->i.
               closure-I6
E->E.+T
T->(E.)
               closure-I7
E->E+.T
T->.(E)
T->.F
T->.T*F
T->.i
               closure-I8
T->T*.F
               closure-I9
T->(E).
               closure-I10
E->E+T.
T->T.*F
               closure-I11
T->T*F.
```

产生的分析表：



LR(0)分析表

	(	)	*	+	E	F	S	T	i	#
0	S1				2	3		4	S5	
1	S1				6	3		4	S5	
2				S7						acc
3		R4	R4	R4						R4
4		R2	S8	R2						R2
5		R6	R6	R6						R6
6		S9		S7						
7	S1				3		10	S5		
8					11					
9		R5	R5	R5						R5
10		R1	S8	R1						R1
11		R3	R3	R3						R3

5. 总结

通过本实验，我们成功构造了一个 SLR(1) 分析表。该表用于 LR 语法分析器，确保了文法的解析过程没有冲突，验证了文法的 SLR(1) 性质。实验中使用了项目集规范族、FIRST 集和 FOLLOW 集等关键技术，提供了完整的分析过程和结果验证。