1202
20123852

January 27, 2016

1

2

3

1.

2.

4

LR(1)LL(1)LR(1)LL(1)

`SyntaxAnalyzer` 1

```
class SyntaxAnalyzer
{
public:
  SyntaxAnalyzer(const string& filePath);

  Grammar G;

  void display(const LR0Item& item);
  void display(const LR1Item& item);
  void display(const set<LR0Item>& I);
  void display(const set<LR1Item>& I);

```

```
13
14    vector<set<LR1Item>> lr1ItemSetFamily;
15    vector<vector<LR1ActionTabItem>> lr1ActionTab;
16    vector<vector<unsigned int>> lr1GoToTab;
17    vector<unsigned int> lr1Stack;
18    string lr1Input;
19
20    set<LR0Item> LR0Closure(const set<LR0Item>& I);
21    set<LR1Item> LR1Closure(const set<LR1Item>& I);
22    set<LR1Item> LR1GoTo(const set<LR1Item>& I, char X);
23    void calLR1ItemSetFamily();
24    void buildLR1ParseTab();
25    bool lr1SyntaxAnalyze(const string& myLR1Input);
26
27
28    vector<vector<LL1Item>> ll1ParseTab;
29    vector<char> ll1Stack;
30    string ll1Input;
31
32    void buildLL1ParseTab();
33    bool ll1SyntaxAnalyze(const string& myLL1Input);
34
35  private:
36
37  };
```

Listing 1: SyntaxAnalyzer

display LR(1)LL(1)

4.1 LR(1)

4.1.1 LR1

2

```
1  class LR1Item
2  {
3  public:
4    LR1Item() = default;
5    LR1Item(const LR0Item& myLR0Item, char
       myLookaheadSymbol);
6    LR1Item(unsigned int myLProductionRuleIdx, unsigned
       int myRProductionRuleIdx, unsigned int myDotPos,
       char myLookaheadSymbol);
7
```

```
8   unsigned int lProductionRuleIdx;
9   unsigned int rProductionRuleIdx;
10  unsigned int dotPos;
11  char lookaheadSymbol;
12
13  friend bool operator < (const LR1Item& i1, const
     LR1Item& i2);
14  friend bool operator ==(const LR1Item& i1, const
     LR1Item& i2);
15  friend bool operator > (const LR1Item& i1, const
     LR1Item& i2);
16  friend bool operator !=(const LR1Item& i1, const
     LR1Item& i2);
17
18 private:
19
20 };
21
22 bool operator <(const LR1Item& i1, const LR1Item& i2)
23 {
24   return i1.lProductionRuleIdx < i2.lProductionRuleIdx
      ?
25     true: (i1.lProductionRuleIdx > i2.
     lProductionRuleIdx?
26       false: (i1.rProductionRuleIdx < i2.
     rProductionRuleIdx?
27         true: (i1.rProductionRuleIdx > i2.
     rProductionRuleIdx?
28           false: (i1.dotPos < i2.dotPos?
29             true: (i1.lookaheadSymbol < i2.
     lookaheadSymbol)))));
30 }
31
32 bool operator ==(const LR1Item& i1, const LR1Item& i2)
33 {
34   return i1.lProductionRuleIdx == i2.
     lProductionRuleIdx
35     && i1.rProductionRuleIdx == i2.rProductionRuleIdx
36     && i1.dotPos == i2.dotPos
37     && i1.lookaheadSymbol == i2.lookaheadSymbol;
38 }
39
40 bool operator >(const LR1Item& i1, const LR1Item& i2)
41 {
42   return !(i1 < i2) && !(i1 == i2);
43 }
```

```
44
45 bool operator !=(const LR1Item& i1, const LR1Item& i2)
46 {
47   return !(i1 == i2);
48 }
49
50 LR1Item::LR1Item(const LR0Item& myLR0Item, char
       myLookaheadSymbol):
51   lProductionRuleIdx(myLR0Item.lProductionRuleIdx),
       rProductionRuleIdx(myLR0Item.rProductionRuleIdx),
       dotPos(myLR0Item.dotPos),
52   lookaheadSymbol(myLookaheadSymbol)
53 {
54
55 }
56
57 LR1Item::LR1Item(unsigned int myLProductionRuleIdx,
       unsigned int myRProductionRuleIdx, unsigned int
       myDotPos, char myLookaheadSymbol):
58   lProductionRuleIdx(myLProductionRuleIdx),
       rProductionRuleIdx(myRProductionRuleIdx), dotPos(
       myDotPos), lookaheadSymbol(myLookaheadSymbol)
59 {
60
61 }
```

Listing 2: LR1

4.1.2 LR1Closure

LR1Closure 3

```
1 set<LR1Item> SyntaxAnalyzer::LR1Closure(const set<
      LR1Item>& I)
2 {
3   set<LR1Item> ret;
4   for(auto ptr = I.begin(); ptr != I.end(); ++ptr)
5   {
6     ret.insert(*ptr);
7   }
8
9   bool updated = true;
10  while(updated)
11  {
12    updated = false;
13
```

```
14      for(auto ptr = ret.begin(); ptr != ret.end(); ++
    ptr)
15      {
16        if(ptr->dotPos < G.P[ptr->lProductionRuleIdx].
    rPartSet[ptr->rProductionRuleIdx].size()
17           && G.isNonTerminal(G.P[ptr->lProductionRuleIdx
    ].rPartSet[ptr->rProductionRuleIdx][ptr->dotPos]))
18        {
19          ProductionRule pr;
20          int prIdx = G.getProductionRuleByLeft(pr, G.P[
    ptr->lProductionRuleIdx].rPartSet[ptr->
    rProductionRuleIdx][ptr->dotPos]);
21          for(size_t i = 0; i < pr.rPartSet.size(); ++i)
22          {
23            string str;
24            if(ptr->dotPos + 1 < G.P[ptr->
    lProductionRuleIdx].rPartSet[ptr->
    rProductionRuleIdx].size())
25            {
26              str = G.P[ptr->lProductionRuleIdx].
    rPartSet[ptr->rProductionRuleIdx][ptr->dotPos + 1]
    + ch2str(ptr->lookaheadSymbol);
27            }
28            else
29            {
30              str = ch2str(ptr->lookaheadSymbol);
31            }
32            set<char> s = G.getFirst(str);
33            int oldSize = ret.size();
34            for(auto& sPtr: s)
35            {
36              if(sPtr != 'e')
37              {
38                ret.insert(LR1Item(prIdx, i, 0, sPtr));
39              }
40            }
41            int newSize = ret.size();
42            if(oldSize != newSize)
43            {
44              updated = true;
45            }
46          }
47        }
48      }
49    }
50
```

```
51    // cout << "LR1Closure:\n"; display(ret); cout <<
      endl;
52
53    return ret;
54  }
```

Listing 3: LR1Closure

4.1.3 LR1GoTo

4

```
1   set<LR1Item> SyntaxAnalyzer::LR1GoTo(const set<LR1Item
      >& I, char X)
2   {
3     if(find(G.VN.begin(), G.VT.begin(), X) != G.VN.end()
4       || find(G.VT.begin(), G.VT.end(), X) != G.VT.end()
      )
5     {
6       set<LR1Item> ret;
7
8       for(auto ptr = I.begin(); ptr != I.end(); ++ptr)
9       {
10        if(ptr->dotPos < G.P[ptr->lProductionRuleIdx].
      rPartSet[ptr->rProductionRuleIdx].size()
11            && G.P[ptr->lProductionRuleIdx].rPartSet[ptr->
      rProductionRuleIdx][ptr->dotPos] == X)
12          {
13            ret.insert(LR1Item(ptr->lProductionRuleIdx,
      ptr->rProductionRuleIdx, ptr->dotPos + 1, ptr->
      lookaheadSymbol));
14          }
15        }
16
17      // cout << "LR1GoTo:\n"; display(LR1Closure(ret));
        cout << endl;
18
19      return LR1Closure(ret);
20    }
21    else
22    {
23      return set<LR1Item>();
24    }
25  }
```

Listing 4: LR1GoTo

4.1.4 calLR1ItemSetFamily

5

```
void SyntaxAnalyzer::calLR1ItemSetFamily()
{
  G.P.push_back(ProductionRule('S', vector<string>({
   ch2str(G.S)})));
  sort(G.P.begin(), G.P.end(),
    [](const ProductionRule& pr1, const ProductionRule
   & pr2)
    {
      return pr1.lPart < pr2.lPart?
        true: (pr1.lPart > pr2.lPart?
          false: pr1.rPartSet[0] < pr2.rPartSet[0]);
    });

  G.VN.push_back('S');
  sort(G.VN.begin(), G.VN.end());

  G.S = 'S';

  G.nullable.clear();
  G.first.clear();
  G.follow.clear();
  G.nullable.assign(G.VN.size(), false);
  G.first.assign(G.VN.size() + G.VT.size(), set<char
   >());
  G.follow.assign(G.VN.size(), set<char>());

  G.display();

  G.calNullableFirstFollow();

  lr1ItemSetFamily.push_back(
    LR1Closure(
      set<LR1Item>({
        LR1Item(
          find_if(
            G.P.begin(), G.P.end(),
            [](const ProductionRule& pr)
            {
              return pr.lPart == 'S';
            }) - G.P.begin(),
          0, 0, '$'
          )
```

```
40          })
41        )
42      );
43
44    bool updated = true;
45    while(updated)
46    {
47      updated = false;
48
49      for(auto& itemSetPtr: lr1ItemSetFamily)
50      {
51        for(auto& symbolPtr: G.VN)
52        {
53          set<LR1Item> itemSet = LR1GoTo(itemSetPtr,
    symbolPtr);
54          if(!itemSet.empty()
55            && find_if(lr1ItemSetFamily.begin(),
    lr1ItemSetFamily.end(),
56              [itemSet](const set<LR1Item>& s)
57              {
58                if(s.size() != itemSet.size())
59                {
60                  return false;
61                }
62                else
63                {
64                  auto ptr1 = itemSet.begin();
65                  auto ptr2 = s.begin();
66                  for(; ptr1 != itemSet.end(); ++ptr1,
    ++ptr2)
67                  {
68                    if(!(*ptr1 == *ptr2))
69                    {
70                      return false;
71                    }
72                  }
73                  return true;
74                }
75              }) == lr1ItemSetFamily.end())
76          {
77            lr1ItemSetFamily.push_back(itemSet);
78            updated = true;
79          }
80        }
81
82        for(auto& symbolPtr: G.VT)
```

```
83        {
84          set<LR1Item> itemSet = LR1GoTo(itemSetPtr,
    symbolPtr);
85          if(!itemSet.empty()
86            && find_if(lr1ItemSetFamily.begin(),
    lr1ItemSetFamily.end(),
87                [itemSet](const set<LR1Item>& s)
88                {
89                  if(s.size() != itemSet.size())
90                  {
91                    return false;
92                  }
93                  else
94                  {
95                    auto ptr1 = itemSet.begin();
96                    auto ptr2 = s.begin();
97                    for(; ptr1 != itemSet.end(); ++ptr1,
    ++ptr2)
98                    {
99                      if(!(*ptr1 == *ptr2))
100                     {
101                       return false;
102                     }
103                   }
104                   return true;
105                 }
106               }) == lr1ItemSetFamily.end())
107         {
108           lr1ItemSetFamily.push_back(itemSet);
109           updated = true;
110         }
111       }
112     }
113   }
114
115   // sort lr1ItemSetFamily
116   sort(lr1ItemSetFamily.begin(), lr1ItemSetFamily.end
    (),
117     [](const set<LR1Item>& itemSet1, const set<LR1Item
    >& itemSet2)
118     {
119       auto ptr1 = itemSet1.begin();
120       auto ptr2 = itemSet2.begin();
121       for(; ptr1 != itemSet1.end() && ptr2 != itemSet2
    .end(); ++ptr1, ++ptr2)
122       {
```

```
123           if(*ptr1 > *ptr2)
124           {
125             return false;
126           }
127           else if(*ptr1 < *ptr2)
128           {
129             return true;
130           }
131         }
132         if(ptr1 != itemSet1.end() && ptr2 == itemSet2.
    end())
133         {
134           return false;
135         }
136         else
137         {
138           return true;
139         }
140       });
141
142     cout << "item set family" << endl;
143     for(size_t i = 0; i < lr1ItemSetFamily.size(); ++i)
144     {
145       cout << "I" << i << endl;
146       display(lr1ItemSetFamily[i]);
147       cout << endl;
148     }
149
150     cout << "Goto graph" << endl;
151     for(size_t i = 0; i < lr1ItemSetFamily.size(); ++i)
152     {
153       for(size_t j = 0; j < G.VT.size(); ++j)
154       {
155         set<LR1Item> Ij = LR1GoTo(lr1ItemSetFamily[i], G
    .VT[j]);
156         if(!Ij.empty())
157         {
158           cout << "I" << i << endl;
159           display(lr1ItemSetFamily[i]);
160           cout << "receives " << G.VT[j] << ", goes to"
    << endl;
161           display(Ij);
162           cout << endl;
163         }
164       }
165
```

```
166     for(size_t j = 0; j < G.VN.size(); ++j)
167     {
168       set<LR1Item> Ij = LR1GoTo(lr1ItemSetFamily[i], G
    .VN[j]);
169       if(!Ij.empty())
170       {
171         cout << "I" << i << endl;
172         display(lr1ItemSetFamily[i]);
173         cout << "receives " << G.VN[j] << ", goes to"
    << endl;
174         display(Ij);
175         cout << endl;
176       }
177     }
178   }
179 }
```

Listing 5: calLR1ItemSetFamily

$S' \rightarrow S$nullable first follow 7

4.1.5 buildLR1ParseTab

6

```
1  void SyntaxAnalyzer::buildLR1ParseTab()
2  // notice that the lookahead symbol may be e!!!
3  {
4    lr1ActionTab.assign(lr1ItemSetFamily.size(), vector<
      LR1ActionTabItem>(G.VT.size() + 1, LR1ActionTabItem
      (ERR, 0, 0, 0)));
5    lr1GoToTab.assign(lr1ItemSetFamily.size(), vector<
      unsigned int>(G.VN.size(), lr1ItemSetFamily.size())
      );
6    // if an item in lr1GoToTab equals to the value of
      lr1ItemSetFamily.size(), it means this item is
      invalid
7
8    for(size_t i = 0; i < lr1ItemSetFamily.size(); ++i)
9    {
10     for(auto itemSetPtr = lr1ItemSetFamily[i].begin();
       itemSetPtr != lr1ItemSetFamily[i].end(); ++
      itemSetPtr)
11     {
12       auto VTPtr = find(G.VT.begin(), G.VT.end(),
      itemSetPtr->lookaheadSymbol);
```

```
13        auto VNPtr = find(G.VN.begin(), G.VN.end(),
    itemSetPtr->lookaheadSymbol);
14        if(itemSetPtr->dotPos < G.P[itemSetPtr->
    lProductionRuleIdx].rPartSet[itemSetPtr->
    rProductionRuleIdx].size())
15        {
16          if(G.P[itemSetPtr->lProductionRuleIdx].
    rPartSet[itemSetPtr->rProductionRuleIdx] == "e")
17          {
18            lr1ActionTab[i][(VTPtr == G.VT.end())? G.VT.
    size(): (VTPtr - G.VT.begin())] =
19              LR1ActionTabItem(REDUCE, itemSetPtr->
    lProductionRuleIdx, itemSetPtr->rProductionRuleIdx,
     0);
20          }
21
22          auto ptr = find(G.VT.begin(), G.VT.end(),
23            G.P[itemSetPtr->lProductionRuleIdx].rPartSet
    [itemSetPtr->rProductionRuleIdx][itemSetPtr->dotPos
    ]);
24          if(ptr != G.VT.end())
25          {
26            set<LR1Item> Ij = LR1GoTo(lr1ItemSetFamily[i
    ],
27              G.P[itemSetPtr->lProductionRuleIdx].
    rPartSet[itemSetPtr->rProductionRuleIdx][itemSetPtr
    ->dotPos]);
28            int lr1ActionTabCol = ptr - G.VT.begin();
29            lr1ActionTab[i][lr1ActionTabCol] =
    LR1ActionTabItem(
30              SHIFT, 0, 0,
31              find_if(lr1ItemSetFamily.begin(),
    lr1ItemSetFamily.end(),
32                [Ij](const set<LR1Item>& s)
33                {
34                  if(s.size() != Ij.size())
35                  {
36                    return false;
37                  }
38                  else
39                  {
40                    auto ptr1 = Ij.begin();
41                    auto ptr2 = s.begin();
42                    for(; ptr1 != Ij.end(); ++ptr1, ++
    ptr2)
43                    {
```

```
                    if(!(*ptr1 == *ptr2))
                    {
                      return false;
                    }
                  }
                  return true;
                }
              }) - lr1ItemSetFamily.begin()
            );
        }
      }
      else
      {
        if(G.P[itemSetPtr->lProductionRuleIdx].lPart
  != 'S')
        {
          int lr1ActionTabCol = (VTPtr == G.VT.end())?
   G.VT.size(): (VTPtr - G.VT.begin());
          lr1ActionTab[i][lr1ActionTabCol] =
  LR1ActionTabItem(REDUCE, itemSetPtr->
  lProductionRuleIdx, itemSetPtr->rProductionRuleIdx,
   0);
        }
        else
        {
          lr1ActionTab[i][G.VT.size()] =
  LR1ActionTabItem(ACCEPT, 0, 0, 0);
        }
      }
    }
  }

  for(size_t i = 0; i < lr1ItemSetFamily.size(); ++i)
  {
    for(size_t A = 0; A < G.VN.size(); ++A)
    {
      set<LR1Item> Ij = LR1GoTo(lr1ItemSetFamily[i], G
  .VN[A]);
      lr1GoToTab[i][A] = find_if(lr1ItemSetFamily.
  begin(), lr1ItemSetFamily.end(),
        [Ij](const set<LR1Item>& s)
        {
          if(s.size() != Ij.size())
          {
            return false;
          }
```

```
82            else
83            {
84              auto ptr1 = Ij.begin();
85              auto ptr2 = s.begin();
86              for(; ptr1 != Ij.end(); ++ptr1, ++ptr2)
87              {
88                if(!(*ptr1 == *ptr2))
89                {
90                  return false;
91                }
92              }
93              return true;
94            }
95          }) - lr1ItemSetFamily.begin();
96      }
97    }
98
99    cout << "lr1ActionTab" << endl;
100   for(size_t i = 0; i < lr1ActionTab.size(); ++i)
101   {
102     for(size_t j = 0; j < lr1ActionTab[i].size(); ++j)
103     {
104       cout << "lr1ActionTab[" << i << "," << ((j == G.
      VT.size())? '$': G.VT[j]) << "] = ";
105       switch(lr1ActionTab[i][j].action)
106       {
107         case SHIFT: cout << "SHIFT " << lr1ActionTab[i
      ][j].itemSetIdx << endl; break;
108         case REDUCE: cout << "REDUCE "; G.P[
      lr1ActionTab[i][j].lProductionRuleIdx].display();
      cout << endl; break;
109         case ACCEPT: cout << "ACCEPT" << endl; break;
110         default: cout << "ERR" << endl;
111       }
112     }
113   }
114   cout << endl;
115
116   cout << "lr1GoToTab" << endl;
117   for(size_t i = 0; i < lr1GoToTab.size(); ++i)
118   {
119     for(size_t j = 0; j < lr1GoToTab[i].size(); ++j)
120     {
121       cout << "lr1GoToTab[" << i << "," << G.VN[j] <<
      "] = "
122            << ((lr1GoToTab[i][j] == lr1ItemSetFamily.
```

```
    size())? " ": to_string(lr1GoToTab[i][j])) << endl;
123      }
124   }
125 }
```

Listing 6: buildLR1ParseTab

4.1.6 lr1SyntaxAnalyze

7

```
1  bool SyntaxAnalyzer::lr1SyntaxAnalyze(const string&
       myLR1Input)
2  {
3    if(myLR1Input.size() == 0)
4    {
5      cout << "Input is empty!" << endl;
6      return false;
7    }
8
9    cout << "Parsing " << myLR1Input << "..." << endl;
10
11   lr1Stack.push_back(
12     find_if(
13       lr1ItemSetFamily.begin(), lr1ItemSetFamily.end()
       ,
14       [this](const set<LR1Item>& s)
15       {
16         return find_if(s.begin(), s.end(),
17           [this](const LR1Item& i)
18           {
19             return G.P[i.lProductionRuleIdx].lPart ==
       'S';
20           }) != s.end();
21       }
22       ) - lr1ItemSetFamily.begin()
23     );
24
25   string str = myLR1Input;
26   reverse(str.begin(), str.end());
27   lr1Input = "$" + str;
28
29   int idx = lr1Input.size() - 1;
30   char a = lr1Input[idx];
31   for(;;)
32   {
```

```
33      unsigned int s = lr1Stack[lr1Stack.size() - 1];
34      auto ptr = find(G.VT.begin(), G.VT.end(), a);
35      int aIdx = (ptr == G.VT.end())? G.VT.size(): ptr -
        G.VT.begin();
36      LR1ActionTabItem curAction = lr1ActionTab[s][aIdx
        ];
37      cout << "Begin a new pass..." << endl;
38      cout << "Current state(top of the stack): "
39           << lr1Stack[lr1Stack.size() - 1] << endl;
40      display(lr1ItemSetFamily[lr1Stack[lr1Stack.size()
        - 1]]);
41      cout << "Current input symbol: " << a << endl <<
        endl;
42      if(curAction.action == SHIFT)
43      {
44        lr1Stack.push_back(curAction.itemSetIdx);
45        cout << "Push back state " << curAction.
        itemSetIdx << endl;
46        display(lr1ItemSetFamily[curAction.itemSetIdx]);
47        --idx;
48        a = lr1Input[idx];
49        cout << "Shift and now the input symbol is " <<
        a << endl << endl;
50      }
51      else if(curAction.action == REDUCE)
52      {
53        cout << "Reduce using production rule "
54             << G.P[curAction.lProductionRuleIdx].lPart
        << "->"
55             << G.P[curAction.lProductionRuleIdx].
        rPartSet[curAction.rProductionRuleIdx] << endl;
56        int num = (G.P[curAction.lProductionRuleIdx].
        rPartSet[curAction.rProductionRuleIdx] == "e")?
57          0: G.P[curAction.lProductionRuleIdx].rPartSet[
        curAction.rProductionRuleIdx].size();
58        for(int i = 0; i < num; ++i)
59        {
60          lr1Stack.erase(lr1Stack.end() - 1);
61        }
62        cout << "Pop " << num << " state(s) out of the
        stack" << endl;
63        cout << "Now the stack top is state " <<
        lr1Stack[lr1Stack.size() - 1] << endl;
64        display(lr1ItemSetFamily[lr1Stack[lr1Stack.size
        () - 1]]);
65        int t = lr1Stack[lr1Stack.size() - 1];
```

```
66        lr1Stack.push_back(lr1GoToTab[t][find(G.VN.begin
    (), G.VN.end(), G.P[curAction.lProductionRuleIdx].
    lPart) - G.VN.begin()]);
67        cout << "lr1GoToTab[" << t << "," << G.P[
    curAction.lProductionRuleIdx].lPart << "] = "
68              << ((lr1GoToTab[t][find(G.VN.begin(), G.VN.
    end(), G.P[curAction.lProductionRuleIdx].lPart) - G
    .VN.begin()] == lr1ItemSetFamily.size())?
69                    " ": to_string(lr1GoToTab[t][find(G.VN
    .begin(), G.VN.end(), G.P[curAction.
    lProductionRuleIdx].lPart) - G.VN.begin()]));
70        cout << ", so we push back state "
71              << lr1GoToTab[t][find(G.VN.begin(), G.VN.
    end(), G.P[curAction.lProductionRuleIdx].lPart) - G
    .VN.begin()] << endl;
72        display(lr1ItemSetFamily[lr1GoToTab[t][find(G.VN
    .begin(), G.VN.end(), G.P[curAction.
    lProductionRuleIdx].lPart) - G.VN.begin()]]);
73        cout << endl;
74      }
75      else if(curAction.action == ACCEPT)
76      {
77        cout << "Parsing succeed!" << endl << endl;
78        return true;
79      }
80      else
81      {
82        cout << "Parsing error!" << endl;
83        cout << "In state " << lr1Stack[lr1Stack.size()
    - 1] << endl;
84        display(lr1ItemSetFamily[lr1Stack[lr1Stack.size
    () - 1]]); cout << endl;
85        return false;
86      }
87    }
88  }
```

Listing 7: `lr1SyntaxAnalyze`

$

5 LL(1)

5.0.7 `buildLL1ParseTab`

8

```
void SyntaxAnalyzer::buildLL1ParseTab()
{
  ll1ParseTab.assign(G.VN.size(), vector<LL1Item>(G.VT
   .size() + 1, LL1Item(G.P.size(), 0)));

  for(size_t i = 0; i < G.P.size(); ++i)
  {
    unsigned int AIdx = find(G.VN.begin(), G.VN.end(),
    G.P[i].lPart) - G.VN.begin();

    for(size_t j = 0; j < G.P[i].rPartSet.size(); ++j)
    {
      set<char> firstAlpha = G.getFirst(G.P[i].
   rPartSet[j]);

      for(auto& p: firstAlpha)
      {
        unsigned int aIdx = find(G.VT.begin(), G.VT.
   end(), p) - G.VT.begin();
        if(aIdx < G.VT.size())
        {
          ll1ParseTab[AIdx][aIdx] = LL1Item(i, j);
        }
      }

      if(find(firstAlpha.begin(), firstAlpha.end(), 'e
   ') != firstAlpha.end())
      {
        set<char> followA = G.getFollow(G.P[i].lPart);

        for(auto& p: followA)
        {
          unsigned int bIdx = find(G.VT.begin(), G.VT.
   end(), p) - G.VT.begin();
          ll1ParseTab[AIdx][bIdx] = LL1Item(i, j);
        }

        if(find(followA.begin(), followA.end(), '$')
   != followA.end())
        {
          ll1ParseTab[AIdx][G.VT.size()] = LL1Item(i,
   j);
        }
      }
    }
```

```
38    }
39
40    for(size_t i = 0; i < ll1ParseTab.size(); ++i)
41    {
42      for(size_t j = 0; j < ll1ParseTab[i].size(); ++j)
43      {
44        cout << "ll1ParseTab[" << G.VN[i] << ", " << ((j
      == G.VT.size())? '$': G.VT[j]) << "] = ";
45        if(ll1ParseTab[i][j].i < G.P.size())
46        {
47          cout << G.P[ll1ParseTab[i][j].i].lPart << "->"
      << G.P[ll1ParseTab[i][j].i].rPartSet[ll1ParseTab[i
     ][j].j];
48        }
49        cout << endl;
50      }
51    }
52  }
```

Listing 8: buildLL1ParseTab

5.0.8 ll1SyntaxAnalyze

9

```
1  bool SyntaxAnalyzer::ll1SyntaxAnalyze(const string&
       myLL1Input)
2  {
3    if(myLL1Input.size() == 0)
4    {
5      cout << "Input is empty!" << endl;
6      return false;
7    }
8
9    cout << "Parsing " << myLL1Input << "..." << endl;
10
11   string str = myLL1Input;
12   reverse(str.begin(), str.end());
13   ll1Input = '$' + str;
14   ll1Stack.push_back('$');
15   ll1Stack.push_back(G.S);
16   int ip = ll1Input.size() - 1;
17   char x = ll1Stack[ll1Stack.size() - 1];
18
19   while(x != '$')
20   {
```

```
21      if(x == ll1Input[ip])
22      {
23        ll1Stack.erase(ll1Stack.end() - 1);
24        --ip;
25      }
26      else if(find(G.VT.begin(), G.VT.end(), x) != G.VT.
        end())
27      {
28        cout << "Parsing error!" << endl;
29        return false;
30      }
31      else
32      {
33        unsigned int row = find(G.VN.begin(), G.VN.end()
        , x) - G.VN.begin();
34        unsigned int col = (ll1Input[ip] == '$')? G.VT.
        size(): (find(G.VT.begin(), G.VT.end(), ll1Input[ip
        ]) - G.VT.begin());
35
36        if(ll1ParseTab[row][col].i >= G.P.size())
37        {
38          cout << "Parsing error!" << endl;
39          return false;
40        }
41        else
42        {
43          cout << G.P[ll1ParseTab[row][col].i].lPart <<
        "->" << G.P[ll1ParseTab[row][col].i].rPartSet[
        ll1ParseTab[row][col].j] << endl;
44          ll1Stack.erase(ll1Stack.end() - 1);
45          if(!(G.P[ll1ParseTab[row][col].i].rPartSet[
        ll1ParseTab[row][col].j].size() == 1 &&
46            G.P[ll1ParseTab[row][col].i].rPartSet[
        ll1ParseTab[row][col].j][0] == 'e'))
47          {
48            for(int cnt = G.P[ll1ParseTab[row][col].i].
        rPartSet[ll1ParseTab[row][col].j].size() - 1; cnt
        >= 0; --cnt)
49            {
50              ll1Stack.push_back(G.P[ll1ParseTab[row][
        col].i].rPartSet[ll1ParseTab[row][col].j][cnt]);
51            }
52          }
53        }
54      }
55      x = ll1Stack[ll1Stack.size() - 1];
```

```
56    }
57
58    cout << "Parsing succeed!" << endl;
59    return true;
60 }
```

Listing 9: `ll1SyntaxAnalyze`

$

6

3

1

$$
\begin{aligned}
&E \to TA \\
&A \to +TA|-TA|\varepsilon \\
&T \to FB \\
&B \to *FB|/FB|\varepsilon \\
&F \to (E)|0|1|2|3|4|5|6|7|8|9
\end{aligned} \tag{1}
$$

2

$$
\begin{aligned}
&E \to E+T|E-T|T \\
&T \to T*F|T/F|F \\
&F \to (E)|0|1|2|3|4|5|6|7|8|9
\end{aligned} \tag{2}
$$

3

$$
\begin{aligned}
&Z \to AA \\
&A \to aA|b
\end{aligned} \tag{3}
$$

3 2LR(1) 1LL(1)LR(1)LL(1)

`1+(1-3*(2-(8+9-7*6/2)/3+9+(1+3)-2)+1)/4`

4

`1+(1-3*(2-(8+9-7*6/2)/3+9+(1+3)-2)+1)4`

7

7.1

10

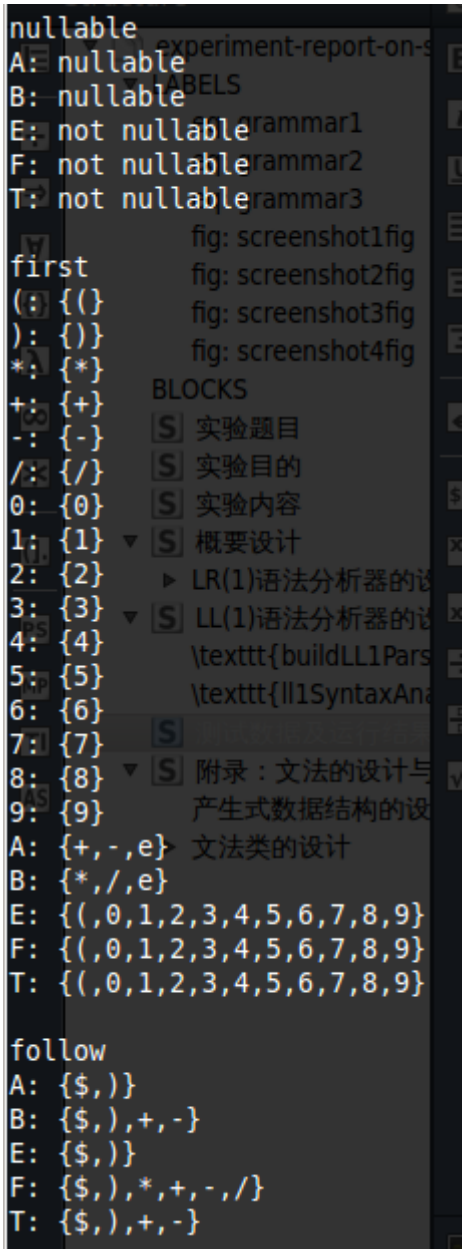
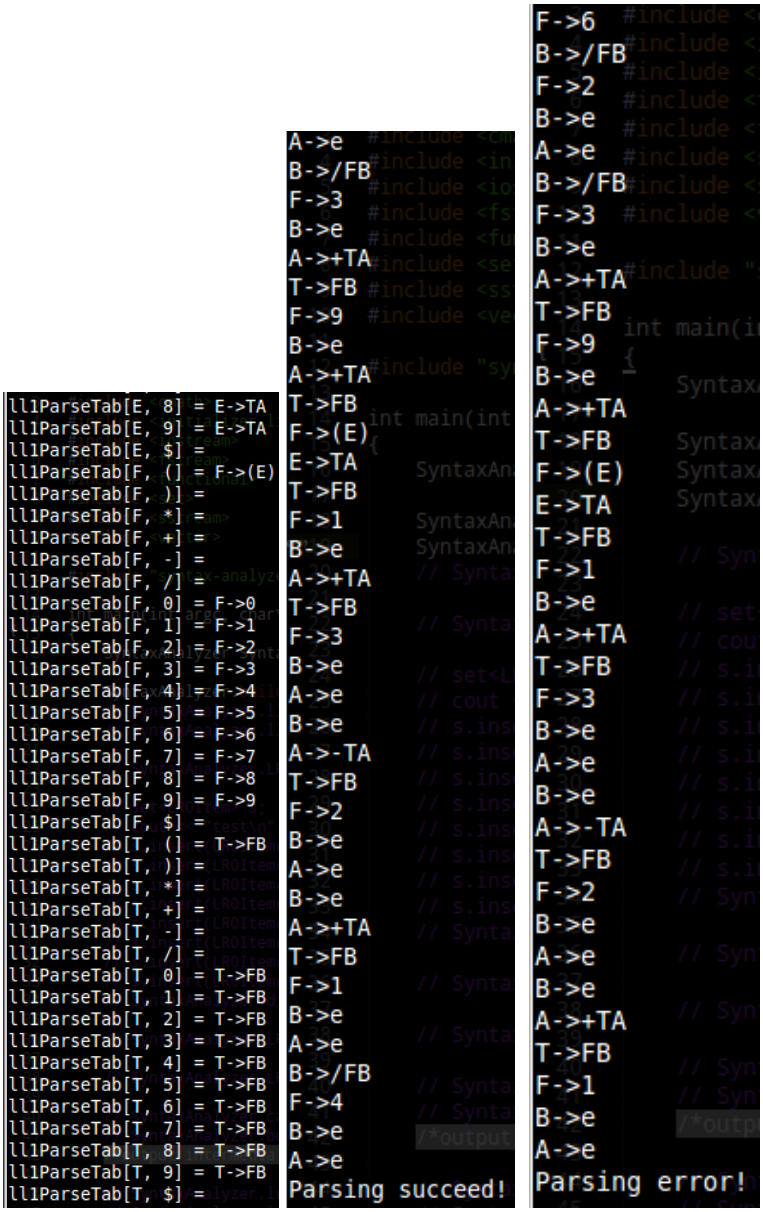
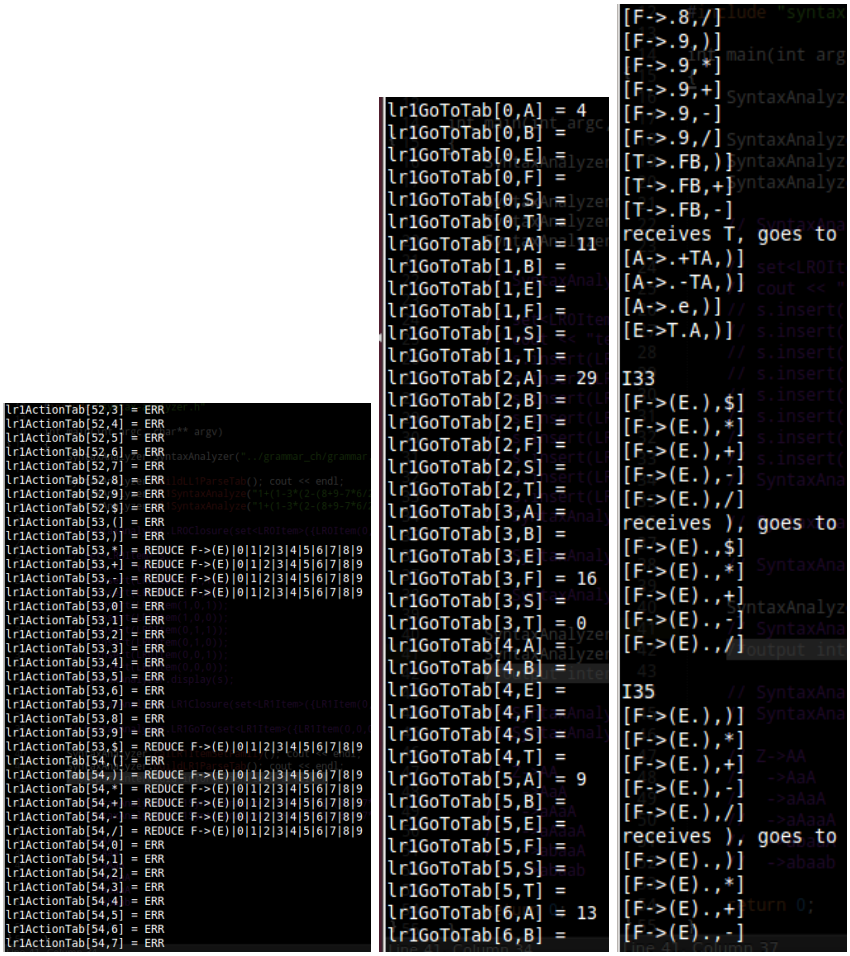

nullable
A: nullable
B: nullable
E: not nullable
F: not nullable
T: not nullable
first
(: {(}
): {)}
: {}
+: {+}
-: {-}
/: {/}
0: {0}
1: {1}
2: {2}
3: {3}
4: {4}
5: {5}
6: {6}
7: {7}
8: {8}
9: {9}
A: {+,-,e}
B: {*,/,e}
E: {(,0,1,2,3,4,5,6,7,8,9}
F: {(,0,1,2,3,4,5,6,7,8,9}
T: {(,0,1,2,3,4,5,6,7,8,9}
follow
A: {$,)}
B: {$,),+,-}
E: {$,)}
F: {$,),*,+,-,/}
T: {$,),+,-}
LABELS
fig: screenshot1fig
fig: screenshot2fig
fig: screenshot3fig
fig: screenshot4fig
BLOCKS
实验题目
实验目的
实验内容
概要设计
文法类的设计

Figure 1:



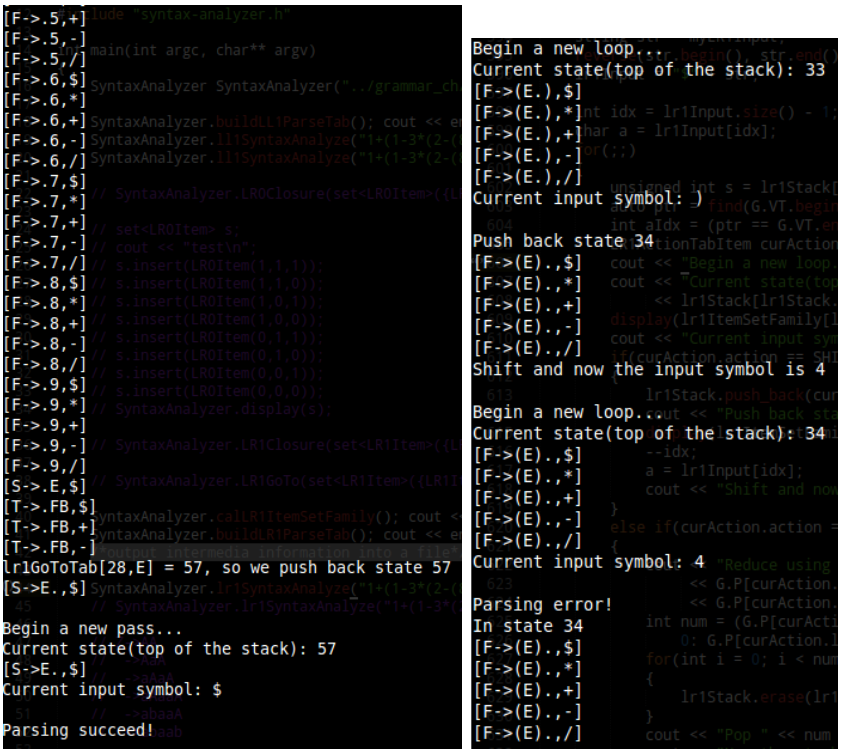
(a) LL(1) (b) LL(1) (c) LL(1)

Figure 2: LL(1)



(a) LR(1)Action (b) LR(1)Goto (c) LR(1)

Figure 3: LR(1)1



(a) LR(1) (b) LR(1)

Figure 4: LR(1)2

```
class ProductionRule
{
public:
  ProductionRule();
  ProductionRule(const char& l, const vector<string>&
   r);

  void display();

  char lPart;
  vector<string> rPartSet;

private:

};

ProductionRule::ProductionRule()
{

}

ProductionRule::ProductionRule(const char& l, const
   vector<string>& r): lPart(l), rPartSet(r)
{

}

void ProductionRule::display()
{
  cout << lPart << "->";
  for(size_t i = 0; i < rPartSet.size(); ++i)
  {
    cout << rPartSet[i];
    if(i != rPartSet.size() - 1)
    {
      cout << "|";
    }
  }
}
```

Listing 10:

7.2

11

```
class Grammar
{
public:
  Grammar(const string& filePath);

  void calNullableFirstFollow();

  void display();
  bool isNonTerminal(char ch);
  int getProductionRuleByLeft(ProductionRule& pr, char
    lPart); // return index of production rule which
   has lPart as left part
  set<char> getFirst(char ch);
  set<char> getFirst(string str);
  set<char> getFollow(char ch);

  vector<char> VN;
  vector<char> VT;
  vector<ProductionRule> P;
  char S;

  vector<bool> nullable; // for symbols in VN and VT,
   elements in VT are followed by elements in VN
  vector<set<char>> first; // for symbols in VN and VT
   , elements in VT are followed by elements in VN
  vector<set<char>> follow; // only for symbols in VN

private:

};
```

Listing 11:

calNullableFirstFollow

7.2.1

12

```
Grammar::Grammar(const string& filePath)
{
  ifstream f(filePath);
  assert(f);

  // file format
```

```
7   // VNSize VN
8   // VTSize VT
9   // PSize
10  // P
11  // S
12
13  int VNSize;
14  f >> VNSize;
15  VN.resize(VNSize);
16  for(int i = 0; i < VNSize; ++i)
17  {
18    f >> VN[i];
19  }
20
21  int VTSize;
22  f >> VTSize;
23  VT.resize(VTSize);
24  for(int i = 0; i < VTSize; ++i)
25  {
26    f >> VT[i];
27  }
28
29  struct functor: public binary_function<
     ProductionRule, char, bool>{
30  public:
31    bool operator ()(const ProductionRule pr, const
     char ch) const {
32      return (pr.lPart == ch);
33    }
34  };
35
36  int PSize;
37  f >> PSize;
38  string str;
39  for(int i = 0; i < PSize; ++i)
40  {
41    f >> str;
42    vector<string> v_pr = split(str, '>');
43    v_pr[0].erase(v_pr[0].end() - 1);
44    vector<string> v_pr_r = split(v_pr[1], '|');
45
46    functor myFunctor;
47    auto itr = find_if(P.begin(), P.end(), bind2nd(
     myFunctor, *(v_pr[0].begin())));
48    if(itr == P.end())
49    {
```

```
50          ProductionRule pr;
51          pr.lPart = *(v_pr[0].begin());
52          for(size_t j = 0; j < v_pr_r.size(); ++j)
53          {
54            pr.rPartSet.push_back(v_pr_r[j]);
55          }
56          P.push_back(pr);
57        }
58        else
59        {
60          for(size_t j = 0; j < v_pr_r.size(); ++j)
61          {
62            P[itr - P.begin()].rPartSet.push_back(v_pr_r[j
      ]);
63          }
64        }
65
66      }
67
68      f >> S;
69
70      f.close();
71
72      sort(VN.begin(), VN.end());
73      sort(VT.begin(), VT.end());
74      for(size_t i = 0; i < P.size(); ++i)
75      {
76        sort(P[i].rPartSet.begin(), P[i].rPartSet.end());
77      }
78      sort(P.begin(), P.end(),
79        [](ProductionRule pr1, ProductionRule pr2)
80        {
81          return pr1.lPart < pr2.lPart?
82            true: (pr1.lPart > pr2.lPart?
83              false: pr1.rPartSet[0] < pr2.rPartSet[0]);
84        });
85
86      nullable.assign(VN.size(), false);
87      first.assign(VN.size() + VT.size(), set<char>());
88      follow.assign(VN.size(), set<char>());
89
90      display();
91
92      calNullableFirstFollow();
93  }
```

Listing 12:

7.2.2 calNullableFirstFollow

13

```
void Grammar::calNullableFirstFollow()
{
  // calculate nullable

  for(size_t i = 0; i < P.size(); ++i)
  {
    for(size_t j = 0; j < P[i].rPartSet.size(); ++j)
    {
      if(P[i].rPartSet[j].size() == 1 && *(P[i].
   rPartSet[j].begin()) == 'e')
      {
        nullable[i] = true;
        break;
      }
    }
  }

  bool updated = true;
  while(updated)
  {
    updated = false;

    for(size_t i = 0; i < P.size(); ++i)
    {
      for(size_t j = 0; j < P[i].rPartSet.size(); ++j)
      {
        bool allNullable = true;
        for(size_t k = 0; k < P[i].rPartSet[j].size();
    ++k)
        {
          auto ptr = find(VN.begin(), VN.end(), P[i].
   rPartSet[j][k]);
          if(ptr == VN.end() || !nullable[ptr - VN.
   begin()])
          {
            allNullable = false;
```

```
33              break;
34            }
35          }
36
37          if(allNullable && !nullable[i])
38          {
39            updated = true;
40            nullable[i] = true;
41            break;
42          }
43        }
44      }
45    }
46
47    // calculate first
48
49    for(size_t i = 0; i < VT.size(); ++i)
50    {
51      first[i].insert(VT[i]);
52    }
53
54    updated = true;
55    while(updated)
56    {
57      updated = false;
58
59      for(size_t i = 0; i < P.size(); ++i)
60      {
61        for(size_t j = 0; j < P[i].rPartSet.size(); ++j)
62        {
63          int idx = find(VN.begin(), VN.end(), P[i].
    lPart) - VN.begin() + VT.size();
64
65          int l = 0;
66          for(size_t k = 0; k < P[i].rPartSet[j].size();
     ++k)
67          {
68            auto ptr = find(VN.begin(), VN.end(), P[i].
    rPartSet[j][k]);
69            if(ptr == VN.end() || !nullable[ptr - VN.
    begin()])
70            {
71              break;
72            }
73            else
74            {
```

```
75              ++l;
76
77              int oldSize = first[idx].size();
78              int newSize = 0;
79              auto ptr = find(VN.begin(), VN.end(), P[i
    ].rPartSet[j][k]);
80              setUnion(first[idx], first[ptr - VN.begin
    () + VT.size()]);
81              newSize = first[idx].size();
82              if(oldSize != newSize)
83              {
84                updated = true;
85              }
86            }
87          }
88
89          ////////////////////////////////
90          // cout << P[i].lPart << "->" << P[i].rPartSet
    [j] << " " << l << endl;
91
92          int oldSize = first[idx].size();
93          int newSize = 0;
94          if(l >= P[i].rPartSet[j].size())
95          {
96            first[idx].insert('e');
97            newSize = first[idx].size();
98          }
99          else
100         {
101           auto ptr = find(VN.begin(), VN.end(), P[i].
    rPartSet[j][l]);
102           if(ptr == VN.end())
103           {
104             if(P[i].rPartSet[j][l] != 'e')
105             {
106               setUnion(first[idx], first[find(VT.begin
    (), VT.end(), P[i].rPartSet[j][l]) - VT.begin()]);
107
108               ////////////////////////////////
109               // cout << i << " " << idx << " ";
110               // ::display(first[find(VT.begin(), VT.
    end(), P[i].rPartSet[j][l]) - VT.begin()]); cout <<
     endl << endl;
111             }
112             else
113             {
```

```
114                 setUnion(first[idx], set<char>({'e'}));
115               }
116             }
117             else
118             {
119               setUnion(first[idx], first[ptr - VN.begin
    () + VT.size()]);
120             }
121
122             newSize = first[idx].size();
123           }
124
125           if(oldSize != newSize)
126           {
127             updated = true;
128           }
129         }
130       }
131     }
132
133     // sth about $ and e
134
135     follow[find(VN.begin(), VN.end(), S) - VN.begin()].
    insert('$');
136
137     updated = true;
138     while(updated)
139     {
140       updated = false;
141
142       for(size_t i = 0; i < P.size(); ++i)
143       {
144         for(size_t j = 0; j < P[i].rPartSet.size(); ++j)
145         {
146           int idx = find(VN.begin(), VN.end(), P[i].
    lPart) - VN.begin();
147
148           for(size_t k = 0; k < P[i].rPartSet[j].size();
    ++k)
149           {
150             bool allNullable1 = true;
151             for(size_t cnt = k + 1; cnt < P[i].rPartSet[
    j].size(); ++cnt)
152             {
153               if(P[i].rPartSet[j][cnt] != 'e')
154               {
```

```
155             auto cntPtr = find(VN.begin(), VN.end(),
     P[i].rPartSet[j][cnt]);
156             if(cntPtr == VN.end() || !nullable[
    cntPtr - VN.begin()])
157             {
158               allNullable1 = false;
159               break;
160             }
161           }
162           else
163           {
164             break;
165           }
166         }
167         if(allNullable1)
168         {
169           auto kPtr = find(VN.begin(), VN.end(), P[i
    ].rPartSet[j][k]);
170           if(kPtr != VN.end())
171           {
172             int oldSize = follow[kPtr - VN.begin()].
    size();
173             setUnion(follow[kPtr - VN.begin()],
    follow[idx]);
174             int newSize = follow[kPtr - VN.begin()].
    size();
175             if(oldSize != newSize)
176             {
177               updated = true;
178             }
179           }
180         }
181
182         for(size_t l = k + 1; l < P[i].rPartSet[j].
    size(); ++l)
183         {
184           bool allNullable2 = true;
185           for(int cnt = k + 1; cnt < l - 1; ++cnt)
186           {
187             if(P[i].rPartSet[j][cnt] != 'e')
188             {
189               auto cntPtr = find(VN.begin(), VN.end
    (), P[i].rPartSet[j][cnt]);
190               if(cntPtr == VN.end() || !nullable[
    cntPtr - VN.begin()])
191               {
```

```
192                     allNullable2 = false;
193                     break;
194                   }
195                 }
196                 else
197                 {
198                   break;
199                 }
200               }
201               if(allNullable2)
202               {
203                 auto kPtr = find(VN.begin(), VN.end(), P
    [i].rPartSet[j][k]);
204                 if(kPtr != VN.end())
205                 {
206                   auto lPtr = find(VN.begin(), VN.end(),
     P[i].rPartSet[j][l]);
207                   int lIdx = (lPtr == VN.end())?
208                     (find(VT.begin(), VT.end(), P[i].
    rPartSet[j][l]) - VT.begin()):
209                     (lPtr - VN.begin() + VT.size());
210                   int oldSize = follow[kPtr - VN.begin()
    ].size();
211                   set<char> __first__ = first[lIdx];
212                   __first__.erase('e');
213                   setUnion(follow[kPtr - VN.begin()],
    __first__);
214                   int newSize = follow[kPtr - VN.begin()
    ].size();
215                   if(oldSize != newSize)
216                   {
217                     updated = true;
218                   }
219                 }
220               }
221             }
222           }
223         }
224       }
225     }
226
227     // display nullable
228     cout << "nullable" << endl;
229     for(size_t i = 0; i < nullable.size(); ++i)
230     {
231       cout << VN[i] << ": " << ((nullable[i])? "nullable
```

```
    ": "not nullable") << endl;
  }
  cout << endl;

  // display first
  cout << "first" << endl;
  for(size_t i = 0; i < first.size(); ++i)
  {
    if(i < VT.size())
    {
      cout << VT[i] << ": ";
      ::display(first[i]);
      cout << endl;
    }
    else
    {
      cout << VN[i - VT.size()] << ": ";
      ::display(first[i]);
      cout << endl;
    }
  }
  cout << endl;

  // display follow
  cout << "follow" << endl;
  for(size_t i = 0; i < follow.size(); ++i)
  {
    cout << VN[i] << ": ";
    ::display(follow[i]);
    cout << endl;
  }
  cout << endl;
}
```

Listing 13: `calNullableFirstFollow`

```
nullable first follow e
```