

# 编译project2报告

---

## 一、实验内容

## 二、设计思路

## 三、具体实现

### 1.解析json文件

### 2. 求导表达式生成

#### 2.1 主要变量

#### 2.2 autodiff函数

#### 2.3 change\_token函数

#### 2.4 其他细节处理

## 四、编译知识

## 五、实验结果

## 六、小组分工

## 一、实验内容

本次实验在之前实现的project1的IR文法基础上，做一些进一步的探索。project1的目标是，给定一个矩阵计算的表达式，需要通过对该表达式的结构进行词法、文法解析生成对应的语法树，再遍历该语法树，以c++的形式输出内容。在project2的话，内容相应的变成，给定一个矩阵计算的表达式，我们已知对output项的导数dOutput，去求解各个input的导数dInput。仅仅得出dInput的表达式是不够的，还需要转化成c++的代码生成dInput的值。

## 二、设计思路

由于在project1里我们已经实现了从给定计算表达式生成c++代码的过程，所以一个很自然的想法就是，为了给出对input项求导的c++代码，先实现由给定表达式生成dInput的求导表达式，然后调用project1里的成果生成最后的c++代码。生成求导表达式的过程主要在src/dx.cc中。在dx.cc中，整体上我们从给定的json里的in、out生成新的求导表达式中的in、out，以及最后新的求导表达式字符串，生成一个新的json，传递给project1生成c++代码。

## 三、具体实现

### 1.解析json文件

和project1类似处理。只不过这里json类中多了一项grad\_to，只需类似地增加该项的解析即可。最终得到一个json类，进行下一步处理：

```
1 class json{
2     public:
3     string name;
4     vector<string> ins;
5     vector<string> outs;
6     string data_type;
7     string kernel;
8     vector<string> grad_to;
9 };
```

之后我们用引用的形式定义了新的in和out，这样在求解新的in和out之后，就可以顺带生成一个新的json config了。

## 2. 求导表达式生成

整个project最关键的部分就是在求导表达式的生成，基本都写在dx.cc中。其接口是te函数。首先仍然调用project1使用的词法分析工具mylex解析kernel，得到一个token流，token类的定义也和之前一样。然后主要的过程是调用P()开始进行语法分析。这里仍然运用的是递归下降子程序法解析kernel，其中P代表的是最原始的产生式，先得到一个merge\_result。因为最终表达式有一些约束，比如左端的下标不能存在表达式，所以用change\_token函数做一些转换，得到最终需要的求导表达式final\_result。更新in和out，返回最终的结果。下面介绍其中几个关键的部分：

### 2.1 主要变量

```
1 namespace dx{
2     std::vector<std::string> ins;
3     std::vector<std::string> outs;
4     std::vector<std::string> grad_tos;
5     std::vector<Token> tokens;
6     std::vector<Token> need_to_handle;
7     std::vector<Token> handle_result;
8     std::vector<Token> left_row;
9     std::vector<Token> merge_result;
10    std::vector<Token> leftdiff;
11    Token nowdiff;
12    int token_index;
13    Token look;
```

```

14     bool flag1 = 0;
15 }

```

- ins, outs和grad\_tos顾名思义用来存储求导表达式里新的in和out等。
- tokens列表存储由词法分析得到的token流。
- token\_index是当前处理token流的索引，look是下一个待处理的token
- need\_to\_handle每次存储求导的一个最小单元(以+和-划分)，flag1用来在每次匹配一个token的时候判断是否需要放进need\_to\_handle数组，比如如果是'+'和'-'就不需要
- handle\_result存储处理一个计算单元的导数结果，merge\_result用来合并多个handle\_result作为最后传给change\_token的结果。
- left\_row和leftdiff用来表示等号左边的token以及左值求导的结果。

## 2.2 autodiff函数

autodiff函数用来对表达式中，被+和-分割开来的项进行求导。并且，这里的不可分割项都是进行了分配律之后的，所以不可分割项里只会出现乘法和除法。乘除法的求导就直接用导数的求导法则即可。当然这里为了方便，就没有形如 $1 * 1 + 0 * x$ 了，直接是对待求导项求导。

## 2.3 change\_token函数

由于在最后的c++代码中等式左边的下标不允许出现计算表达式，所以对之前得到的merge\_result形式上进行一下修改。比如对于case6，我们将等号左边里面的下标替换成tmp，具体方法如下：

引入标记flag用来标记现在是在等号左边还是右边；Change数组里记录了一系列需要发生变化的标识符，Change\_to标记了它们会被替换成什么token流。比如对于索引a+b可以用如下代码实现：用一个新的变量tmp2替换a+b，然后在后面需要用到a的时候替换成tmp2-b即可。其中count用来计数，给tmp2命名；merge\_result[i-1]就表示a即将被替换成tmp.Change\_to中的token流：tmp2,tmp3加上merge\_result[i+1]，也就是新生成的token："tmp2-b"。之后的表达式里如果遇到a，遍历Change数组看看是否发生过变化，如果有过那么就进行替换。示例代码如下：

```

1  if (merge_result[i].type == '+') {
2      count++;
3      change tmp;
4      Token tmp2;
5      Token tmp3;
6      tmp3.type = '-';
7      tmp2.type = 0;
8      tmp.need_tochange = merge_result[i - 1];
9      change_result.pop_back();
10     std::string temp_name = "tmp";
11     temp_name += std::to_string(count);
12     tmp2.stringValue = temp_name;
13     change_result.push_back(tmp2);

```

```

14     tmp.Change_to.push_back(tmp2);
15     tmp.Change_to.push_back(tmp3);
16     tmp.Change_to.push_back(merge_result[i + 1]);
17     Change.push_back(tmp);
18     i += 2;
19 }

```

其中change\_result会记录转化之后的token流。最后再由change\_result逐个生成最后的表达式字符串final\_result，作为返回值返回。

这里以case6为例子具体阐释一下求导技术实现的过程。

1. 用json.parse解析json源文件，解析出最开始的in、out以及kernel表达式。
2. 用mylex函数进行词法分析，返回kernel里包含的token流。
3. 用递归下降子程序法进行语法分析，从P()开始，每一个不可分割项的求导结果放在handle\_result中，合并到merge\_result里。
4. 现在merge\_result里包含了对左值索引问题转化之前的token流，对于case6大概是这样：

```

dA
dB
dB<2,16,7,7>[n,c,p+r,q+s]=dA<2,8,5,5>[n,k,p,q]*C<8,16,3,3>[k,c,r,s];

```

用change\_token函数转化之后会变成如下的字符串final\_result：

```

successfully match ,
P is success
dB<2,16,7,7>[n,c,tmp1,tmp2]=dA<2,8,5,5>[n,k,tmp1-r,tmp2-s]*C<8,16,3,3>[k,c,r,s];
successfully match 8
successfully match ,
successfully match 5
successfully match ,

```

5. 将转化得到的求导表达式交给project1中实现的分析器处理，构建相应的语法分析树，输出相应的c++代码。对于case6，转化得到的代码如下：

```

1 void grad_case6(float (&C)[8][16][3][3], float (&dA)[2][8][5][5], fl
    oat (&dB)[2][16][7][7]) {
2 float temp1[2][16][7][7];
3 float temp2[2][16][7][7];
4 for (int n = 0;n < 2;n++) {
5     for (int c = 0;c < 16;c++) {
6         for (int tmp1 = 0;tmp1 < 7;tmp1++) {
7             for (int tmp2 = 0;tmp2 < 7;tmp2++) {
8                 temp1[n][c][tmp1][tmp2] = 0;
9                 temp2[n][c][tmp1][tmp2] = 0;
10                for (int k = 0;k < 8;k++) {

```

```

11         for (int r = 0; r < 3; r++) {
12             for (int s = 0; s < 3; s++) {
13                 if ((tmp2 - s) >= 0) {
14                     if ((tmp2 - s) < 5) {
15                         if ((tmp1 - r) >= 0) {
16                             if ((tmp1 - r) < 5) {
17                                 temp2[n][c][tmp1][tmp2] = (temp2[n][c][tmp1]
18 [tmp2] + (dA[n][k][(tmp1 - r)][(tmp2 - s)] * C[k][c][r][s]));
19                                 }
20                             }
21                         }
22                     }
23                 }
24             }
25             temp1[n][c][tmp1][tmp2] = temp2[n][c][tmp1][tmp2];
26         }
27     }
28 }
29 }
30 for (int n = 0; n < 2; n++) {
31     for (int c = 0; c < 16; c++) {
32         for (int tmp1 = 0; tmp1 < 7; tmp1++) {
33             for (int tmp2 = 0; tmp2 < 7; tmp2++) {
34                 dB[n][c][tmp1][tmp2] = temp1[n][c][tmp1][tmp2];
35             }
36         }
37     }
38 }
39 }

```

## 2.4 其他细节处理

- case4有多个矩阵需要求导的情况，同时需要对B和C进行求导，我们选择在solution2.cc主函数里进行判断，如果求导的矩阵个数有两个，那么就修改相应的grad\_to，再调用一次te函数，然后将两次te的结果合并得到最后的in、out和kernel。

```

1 if (grad_to.size() == 2)
2     {

```

```

3         std::vector<string> tmpvec;
4         tmpvec.push_back(grad_to[grad_to.size() - 1]);
5         std::string tmpstr = Boost::Internal::dx::te(kernel, second_in, second_out, tmpvec);
6         json_config.kernel += tmpstr;
7
8         in = original_in;
9         bool flag;
10        for (int i = 0; i < second_out.size(); i++)
11        {
12            flag = true;
13            for (int j = 0; j < out.size(); j++)
14            {
15                if (second_out[i] == out[j])
16                    flag = false;
17            }
18            if (flag)
19                out.push_back(second_out[i]);
20        }
21
22    }

```

- case10里涉及到三个项的运算，并且范围各不相同，如果放在一起处理会比较繁琐，这里选择将等式右边括号里相加的三项分离开来，逐个求导，然后再把导数累加，这样就避免了使用select语句。事实上，我们在处理表达式的时候，不可分割项都是进行过分配律之后的。

## 四、编译知识

1. 词法分析。整个project里重要的一个部分就是token，用来表示一个词法单元，分成了标识符、整数、浮点数以及一系列符号四个类型。在token类中，type表示token的类型，0代表标识符，1和2代表整数和浮点数，3代表'/'，其余特殊符号用ascii码代表。另外各个token的值也存储在对应类型的变量value中。project中我们使用了开源的词法分析工具flex，编写语法文件lex.l可以自动生成词法分析的代码lex.cc，以及mylex函数可以读取字符串，解析生成vector<token>的token流。
2. 句法分析。在词法分析的基础上，句法分析读取词法单元，分析出词法单元之间的结构关系。这里主要沿用了project1的语法分析方法，用look(当前需要处理的词法单元)和match(匹配词法单元)预处理表达式，构建IR树，然后再用之前实现的printer遍历树节点输出相应的c++代码。
3. 语法制导翻译SDT相关内容。分析表达式结构，我们得出从 $P \rightarrow S$ 作为开始的一系列语法制导定义。然后采用递归下降子程序法，对每一个子结构定义一个处理函数，在其中完成综合属性、继承属性的转化，并完成相应代码的输出。

## 五、实验结果

按照测试命令，在build文件夹下输入命令 `cmake ..` 以及 `make -j 4`，成功build。刚刚已经展示过 case 6转化之后的c++代码，这里再举例case10的代码：

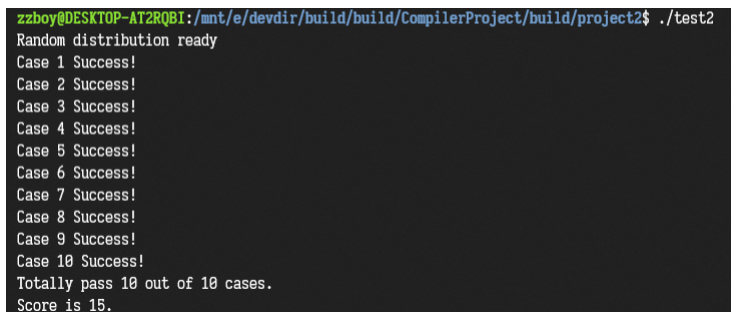
```
1 void grad_case10(float (&dA)[8][8], float (&dB)[10][8]) {
2     float temp1[10][8];
3     float temp2[10][8];
4     float temp3[10][8];
5     float temp4[10][8];
6     float temp5[10][8];
7     float temp6[10][8];
8     float temp7[10][8];
9     float temp8[10][8];
10    for (int i = 0; i < 10; i++) {
11        for (int j = 0; j < 8; j++) {
12            temp1[i][j] = 0;
13            temp2[i][j] = 0;
14            temp2[i][j] = (temp2[i][j] + (dA[i][j] / 3));
15            temp1[i][j] = temp2[i][j];
16        }
17    }
18    for (int i = 0; i < 10; i++) {
19        for (int j = 0; j < 8; j++) {
20            dB[i][j] = temp1[i][j];
21        }
22    }
23    for (int tmp1 = 0; tmp1 < 10; tmp1++) {
24        for (int j = 0; j < 8; j++) {
25            temp3[tmp1][j] = 0;
26            temp4[tmp1][j] = 0;
27            temp5[tmp1][j] = 0;
28            temp4[tmp1][j] = (temp4[tmp1][j] + dB[tmp1][j]);
29            if ((tmp1 - 1) >= 0) {
30                if ((tmp1 - 1) < 8) {
31                    temp5[tmp1][j] = (temp5[tmp1][j] + (dA[(tmp1 - 1)][j] / 3)
32                );
33            }
34            temp3[tmp1][j] = (temp4[tmp1][j] + temp5[tmp1][j]);
35        }
36    }
```

```

36 }
37 for (int tmp1 = 0;tmp1 < 10;tmp1++) {
38     for (int j = 0;j < 8;j++) {
39         dB[tmp1][j] = temp3[tmp1][j];
40     }
41 }
42 for (int tmp2 = 0;tmp2 < 10;tmp2++) {
43     for (int j = 0;j < 8;j++) {
44         temp6[tmp2][j] = 0;
45         temp7[tmp2][j] = 0;
46         temp8[tmp2][j] = 0;
47         temp7[tmp2][j] = (temp7[tmp2][j] + dB[tmp2][j]);
48         if ((tmp2 - 2) >= 0) {
49             if ((tmp2 - 2) < 8) {
50                 temp8[tmp2][j] = (temp8[tmp2][j] + (dA[(tmp2 - 2)][j] / 3)
51             );
52         }
53         temp6[tmp2][j] = (temp7[tmp2][j] + temp8[tmp2][j]);
54     }
55 }
56 for (int tmp2 = 0;tmp2 < 10;tmp2++) {
57     for (int j = 0;j < 8;j++) {
58         dB[tmp2][j] = temp6[tmp2][j];
59     }
60 }
61 }

```

输入命令 `cd project2`，运行test文件：`./test2`，结果截图如下：



```

zzboy@DESKTOP-AT2RQBI:/mnt/e/devdir/build/build/CompilerProject/build/project2$ ./test2
Random distribution ready
Case 1 Success!
Case 2 Success!
Case 3 Success!
Case 4 Success!
Case 5 Success!
Case 6 Success!
Case 7 Success!
Case 8 Success!
Case 9 Success!
Case 10 Success!
Totally pass 10 out of 10 cases.
Score is 15.

```

## 六、小组分工



判断索引和自动求导：宁淳-1600011009

bug调试和代码整理：邢博威-1700012884

token流访问和括号拆开处理：李智超-1700012911

工作整理和报告编写：季陆炀-1700012929