编译大作业 第二部分 报告

小组分工与自动求导技术设计

小组分工

Json文件处理与构造Json语法树 王嘉睿

构造IR求导语法树 王嘉睿 侯锐泽 刘诗逸

构造IRPrinter 余忠蔚

算法设计 余忠蔚 王嘉睿

自动求导技术设计

在本次lab中，我们设计的是一个仅对多项式有效的自动求导工具。假设有等式 $A = P(B)$，那么设多项式$P(B)$中B第i次出现在项$f_i(B)$，都会为 dB贡献$f_i(B) * dA/B$的大小，具体解释会在例子解释中介绍。因此，我们需要先遍历一遍输入的式子，找出所有要被求导的变量出现的位置，然后，对于每一个要被求导的变量，我们再搜索一遍，以找到该变量所在项对该变量导数的影响，然后输出所有结果。由于结果中需要多次对结果进行累加，所以我们在最后输出时会将每个赋值的=变成+=。

实现流程与实验结果

Json文件处理与构造Json语法树

这一部分基本与我们第一部分的实现一致。由于这次需要维护需要被求导的变量，因此我们会将Json文件中grads行另存入一个set中，以方便后面的处理。

IR语法树构造

第一次遍历 第一次遍历时，我们会为每个TRef构造它自己应该被转化为的Expr类示例，然后将其存入lazy成员中。这里构造Expr类的方法与我们第一部分的实现一致。然后，对于每个要求导的变量的出现，都会为其分配一个ID，这个ID将在第二次遍历时用于指示当前在处理哪一次出现。由于最后的结果左边的下标不能够出现运算，因此我们会将下标中出现的每个运算转化成一个temp变量，并将该变量参与循环，并判断该变量是否与其原来的值大小相等。

第二次遍历 第二次遍历发生在对每个语句遍历完成后（即Json语法树中表示S的结点）。这里我们已经为每个要求导的变量的每一次出现都分别维护了1个ID，那么接下来我们会对每个ID分别遍历。这里的遍历只会从RHS结点开始遍历，且会在TRef处停止。对于RHS的每种不同子结点，我们的处理如下：

TRef结点：如果该结点的ID与此次遍历的ID相同，那么我们会返回立即数1，否则会返回第一次遍历时存入其lazy成员的Expr类结果。

加法/减法结点：如果这个表达式的左边或是右边的式子的ID与当前ID相等，那么当前结点的ID就被设为与当前ID相等，然后返回与当前ID相等的一边的结果。显然，只可能有一边的ID与当前ID相等。另外，如果与当前ID一样的式子出现在减数一侧，那么我们返回的结果需要乘以-1。如果都不相等，则会将该表达式的ID设为0，然后返回表达式本身。

乘法/除法结点：如果这个表达式的左边或是右边的式子的ID与当前ID相等，那么当前结点的ID就被设为与当前ID相等，否则该表达式的ID会被设为0，然后依然会返回表达式两端相乘或相除的结果（不考虑待求导变量在除数的情况）。

括号表达式结点：该表达式的结果、ID与其括号内表达式一致。

我们每得到一个结果，就会将该结果加上我们所有的条件判断存入一个vector中，最后将vector中的所有语句构造成一个LoopNest来输出。

IR语法树输出

基本上与我们的第一部分一致，但我们将输出的=变成了+=。

实验结果

经多次实验，我们通过了所有测试样例，测试器输出如下：

```
xwg@DESKTOP-Q7HJSIT:/mnt/c/Users/WangJR/Documents/b编译原理/CompilerProject-
2020Spring-Part1-master/CompilerProject-2020Spring-Part1-master/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.
```

样例解释

假设我们要求导的表达式为`A<4, 4>[i, j] = B<4,4>[i, j] *B<4, 4>[i, j] + B<4, 4>[i, j]`，对`B`求导。结果应该为（省略循环）：

```
dB[i][j] = 2 * dA[i][j] * B[i][j] + dA[i][j];
```

那么在这个表达式中，B共出现了3次，从左到右依次编ID为1，2，3。

对于第一次出现，会为结果贡献`dA[i,j] * 1.0 * B[i,j]`,对于第二次出现，会为结果贡献`dA[i,j] * B[i,j] * 1.0`,对于第三次出现，会为结果贡献`dA[i,j] * 1.0`.因此，我们输出的结果为（省略循环、判断和产生的括号等）：

```
dB[i][j] += dA[i][j] * 1.0 * B[i][j];
dB[i][j] += dA[i][j] * B[i][j] * 1.0;
dB[i][j] += dA[i][j] * 1.0
```

由于dB一开始会被初始化为0，因此结果正确。

用到的编译知识

在构造Json语法树时，我们使用的是LR自底向下的语法分析方法。

在构造IR语法树和输出结果时，涉及到SDT的构造。

实现方法见实现过程节和第一部分报告。