编译技术Project1报告 - 第26组

设计思路

将实现流程分成三部分：分析输入json文件、IR树生成、C代码生成

组内分工

宋苑铭：分析输入json文件

朱立人：C代码生成

魏龙：IR树生成

实现方法

一、分析输入json文件

1. 使用jsoncpp对测试数据进行解析：

   利用jsoncpp解析输入json文件，得到函数名称、输入和输出参数、数据类型、表达式等信息。进行一些必要的预处理，如过滤掉已经在输入中出现的输出参数等。

2. 使用Lex (flex) 对表达式进行词法分析：

   编写一个.lex文件，识别输入中的Id, FloatV, IntV以及特殊符号，并返回识别到的token和字符串。在词法分析阶段难以区分二元操作符减和常量的负号，我们将这一操作延迟到语法分析阶段进行。

3. 使用Yacc (bison) 对表达式进行语法分析：

   编写一个.y文件，在声明部分声明使用的token、结合性和优先级，在文法部分按照编译大作业第一部分-补充内容中的输入文法编写文法，实现基本的语法分析功能。

4. 将Json解析器、词法分析器、语法分析器组合起来：

   分别编译.lex文件和.y文件。在solution.cc中使用jsoncpp解析数据，调用lexer的yy_scan_string函数创建kernel表达式的buffer用于扫描，然后调用yyparse进行语法分析。Yacc会试图调用yylex，从词法分析器中得到标记流。在yyparse完成后，释放lexer使用的buffer。

二、 IR树生成

1. 在对kernel表达式进行语法分析的同时构建对应的IR树：

   在一个栈中存放当前已规约表达式对应IR树的节点。每通过一个新的产生式进行规约，就结合栈中已有的元素构造新的节点，并将其压入栈中

2. 得到表达式对应IR树后，依照求和规范将表达式拆成若干求和部分：

   遍历IR树，遇到+,-节点则说明当前节点需要继续拆分，否则该节点对应一个独立求和部分

3. 对循环变量进行边界推断：

语法分析过程中，每得到一个TRef项，对于其每个下标项导出两个不等式，将这些不等式单独保存起来。使用这些不等式便可以粗略推出各个循环变量的上下界。

4. 生成外层求和循环语句：

   扫描kernel表达式左侧出现的变量，将其作为最外层的循环变量。对于每个求和部分，找出未出现在表达式左侧的变量，构建相应循环语句，结合该部分的表达式得到这一求和部分的语句。将所有部分的语句组合，得到完整的循环体。

   由于边界推断的结果不能准确限定循环范围，故在生成循环语句过程中的每一处，都会维护"已出现的循环变量列表"，并扫描语法分析得出的不等式列表，只要不等式中所有的变量都已出现过，就在当前位置放置该不等式对应的if语句，尽可能早地终止不必要的循环。

三、C代码生成

根据已生成的IR树，修改提供文件中的IRPrinter.cc生成C代码。实际上，对于给出IR代码采用的visitor模式对代码进行生成，每一个结点有visitor接口。

1. IntImm, UIntImm, FloatImm, StringImm在C代码没有必要输出op->type()，只需要输出op->value()
2. 表达式结点visitor不变
3. Var中对于多维数组的输出修改为C的格式
4. 重点是对于for循环的修改，我们需要知道Dom域的begin和extent，这里我们可以使用IR提供的as进行指针的强制转换，如果该节点不能转换会返回空指针，所以我们可以在Index生成for循环代码。由于循环变量均为int型，这里reduce，spatial都可以以int来做。
5. 赋值语句，不需要赋值类型，注释即可
6. 对于一维数组和单变量的分辨，由于单变量分析容易被误认为一维数组，我们需要在使用时判断数组是否是一维的，而且长度是否为1
7. 一些符合C代码的细节