

实践二说明_通过约束生成测试用例输入

🔍 勘误：实践一说明_Py用例转Cpp须知中使用的样例文件名并非 `000_has_close_elements.py/cpp`，而为 `008_sum_product.py/cpp`，本说明会继续使用该样例代码进行说明。

⚠️ 注意：本次实践需要在**实践一完成并上交之后**方可进行（原因在于上交后助教会反馈给各位同学约束，本实践需要基于约束进一步实施），请各位同学将实践一的成果打包（按照原先命名方式命名的转换后的 Cpp 代码文件，以及实验报告一并打包），压缩包名称与邮件主题名称均为“**实践一 <学号> <姓名>**”，通过邮箱发送至助教邮箱 sakiyary@smail.nju.edu.cn，实践二所需约束将在成功上交实践一成果后通过邮件分发(回复)，请注意查收。

约束（Constraint）是程序静态分析的一项重要产物，关于约束求解的简介，可以查看以下链接：<http://bbs.huaweicloud.com/blogs/detail/229334>

主要任务

本次实践的主要任务是将实践一转换得到的 CPP 代码通过约束求解得到的约束，**借助大语言模型得到相应约束的测试用例输入**，一是进一步评估大语言模型对于抽象表示的识别能力，二是探索生成测试用例输入的新方法，在下述样例中会详细阐述。

在上交实践一成果后，助教会生成好对应代码的约束，分发给各位同学作为实践二的原始材料，**不需要大家去仔细学习约束求解的过程**，大概了解约束是怎样的东西即可，想了解的同学也可以查看附录内容自行探索。

举例

以下仍旧以 `008_sum_product.cpp` 举例（下简称 008），我们聚焦于主函数（有略微改动，会解释原因）：

```
// other codes ...

int main() {
    std::vector<int> numbers = {};
    std::tuple<int, int> result = sum_product(numbers);
    int sum = std::get<0>(result);
    int product = std::get<1>(result);
    return 0;
}
```

在主函数中主要是对功能函数 `sum_product()` 的调用与测试，我们的约束则会聚焦于功能函数的入参，即 `numbers`。（相较于实践一说明中的代码，这里将入参变为变量，方便后续操作）

`numbers` 的类型为 `vector<int>`，对于该容器来说，约束求解会聚焦于其长度 `size` 以及其中的元素 `elements`。在 008 样例中，容器元素的变化并不会对约束产生影响（对于其他的代码，可能会产生影响），所以 008 样例求解生成的约束主要针对 `size`。

简单理解为，在 008 程序中，当 `size` 有不同的取值，程序就会运行不同的分支。抽象表示 `size` 的取值范围，就是所谓的**约束**。

此处仅为举例，在其他程序中，约束可能会同时针对多个变量，如 `vector<>` 的长度与元素都会有约束。

为了约束不那么冗长，助教在生成约束前会将类似 `size` 的变量限定在一个较小的范围内，如 `[0,3)`。以下是 008 样例根据约束求解的要求更改后的代码：（这一步不需要大家完成，由助教来做，此处仅为示意，以解释约束是怎么得来的，是长什么样的）

```
// other codes ...

int main() {
    std::vector<int> numbers = {};

    int size;
    klee_make_symbolic(&size, sizeof(size), "size"); // 这是约束求解用到的符号生成函数
    // 约束求解时会使 size 变为不同的值，所以此处没有初始化

    int elements[3] = {1, 2, 3}; // 该程序中元素不进入约束表达式
    // 对于其他程序，也会为每个元素生成符号，加入约束求解中

    if (size >= 0) {
        // 限定 numbers 的实际 size 在 [0,3) 范围内
        for (int i = 0; i < size && i < 3; i++) {
            numbers.push_back(elements[i]);
        }
    }

    std::tuple<int, int> result = sum_product(numbers);
    // 由于只需要约束求解，所以不需要运行后续测试
    return 0;
}
```

以下即为上述更改后的 008 样例求解得到的约束：

```
Constraints 1:
(Eq false
  (Sle 0
    (ReadLSB w32 0 size)))
-----
Constraints 2:
(Sle 0
  (ReadLSB w32 0 size))
(Eq false
  (Slt 0
    (ReadLSB w32 0 size)))
-----
Constraints 3:
(Sle 0
  (ReadLSB w32 0 size))
(Slt 0
  (ReadLSB w32 0 size))
(Eq false
  (Slt 1
    (ReadLSB w32 0 size)))
-----
Constraints 4:
(Sle 0
  (ReadLSB w32 0 size))
(Slt 0
```

```

        (ReadLSB w32 0 size))
(Slt 1
  (ReadLSB w32 0 size))
(Eq false
  (Slt 2
    (ReadLSB w32 0 size)))
-----
Constraints 5:
(Sle 0
  (ReadLSB w32 0 size))
(Slt 0
  (ReadLSB w32 0 size))
(Slt 1
  (ReadLSB w32 0 size))
(Slt 2
  (ReadLSB w32 0 size))
-----


```

由于 008 代码十分简单，所以这些约束也相对规整。因为在 `size` 取不同的值时，程序都会运行到不同的分支（最终影响就是 `numbers` 的不同，即功能函数运行的分支不同），所以上述约束看起来就是枚举 `size` 的不同取值，用合取范式表示。

解释一下，约束一表示的意思是 `(0 <= size) == false`，

约束二表示的意思是 `0 <= size && ((0 < size) == false)`，以此类推。

这样的约束文本就是大家在上交实践一成果后会得到的实践二原材料。请大家设计相应的提示语（prompt），让大语言模型理解这样的约束，并生成相应的测试用例输入。

 注意，如 008 样例，约束中仅含有 `size` 变量，所以大语言模型可能只会生成不同的 `size` 变量作为测试用例输入，但其实 008 需要的测试用例输入是一个 `vector<int>` 类型的变量，不止是它的 `size`，所以请大家自行想办法生成合法的测试用例输入（即完整的 `numbers`），例如使用随机数去填充容器，让大语言模型额外生成容器的元素等等。

实践二 提交要求

请将每份代码的测试用例输入与实验报告打包，压缩包名称与邮件主题名称均为“实践二 <学号> <姓名>”，通过邮箱发送至 sakiyary@smail.nju.edu.cn。

对于每份代码的测试用例输入的格式，请在实践一成果的代码上更改主函数，将大语言模型依据约束生成的且补全的测试用例输入包含在其中。以 008 为例，如下：

```

// other codes ...

int main() {
    std::vector<std::vector<int>> cases = {{}, {1}, {1, 2}};
    std::tuple<int, int> result;
    for (auto numbers : cases) {
        result = sum_product(numbers);
        // TODO: check if the result is right
    }
}

```

实验报告中需包含你所设计的提示语（prompt）以及与大语言模型对话的截图（两三张即可），可以将遇到的困难与你的解决办法也阐述出来。

附录 如何自行生成约束

助教使用了更改了部分代码的开源程序分析工具 [KLEE](#) 来进行约束的生成，各位同学有兴趣的可以自行尝试，也可通过邮件向助教寻求帮助，不作要求。