

ValueRangeAnalyser

潘学海 1500011317

运行环境

程序基于 Python，且所有依赖包均为 Python 内建包。由于所有函数和变量均使用了 Type Hints，故版本要求为 Python version >= 3.5。

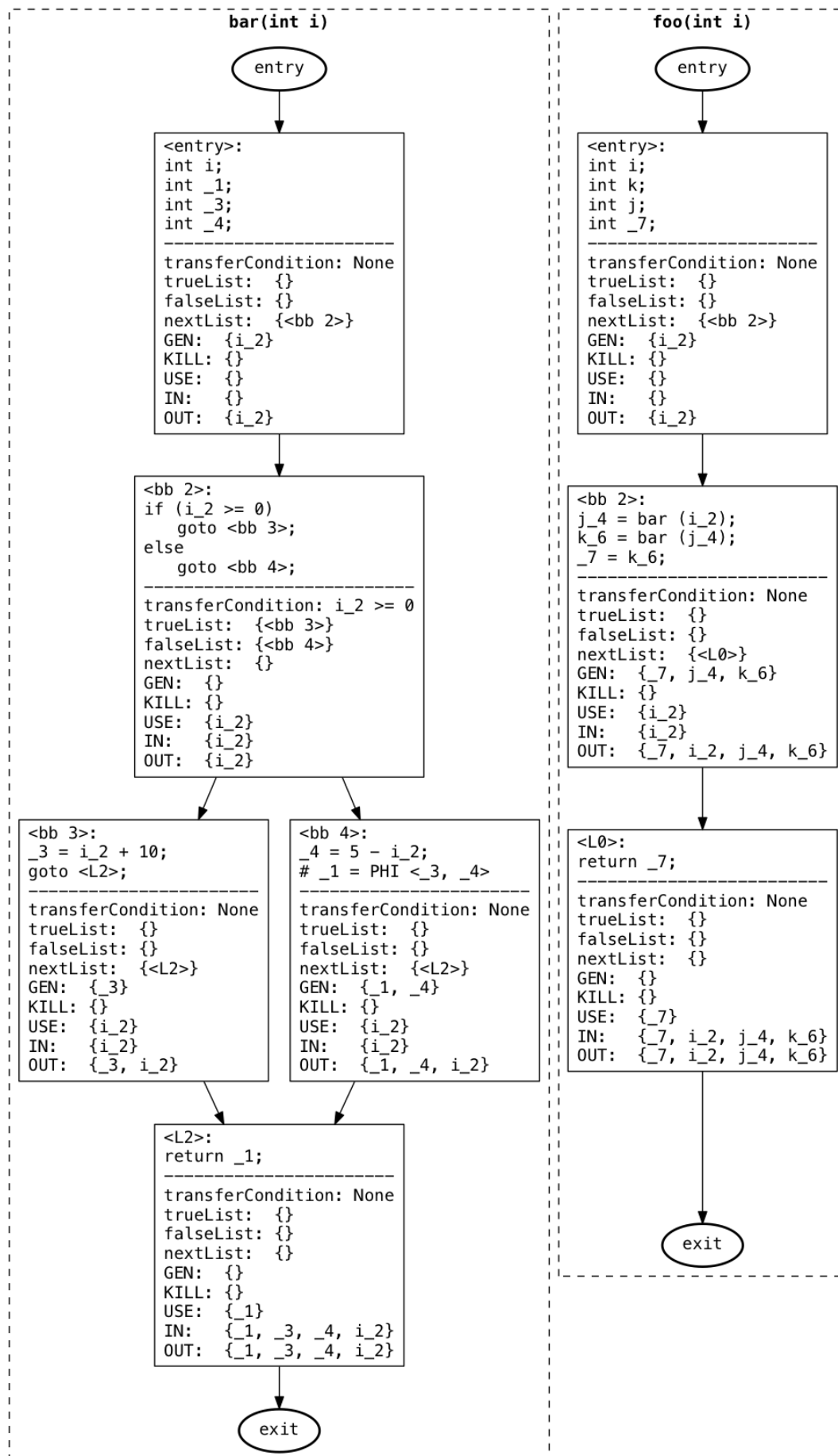
同时为了更直观地显示各个函数及基本块之间的关系，我们使用了 `pygraphviz` 包进行对 Control Flow Graph (CFG) 和 Constraint Graph (CG) 进行可视化。但该包不是必须的，程序会自动尝试导入该包，若导入失败则在测试过程中不会绘制 CFG 和 CG。

```
1  # RangeAnalyser.py
2  # some code
3
4  class RangeAnalyser(object):
5      # some code
6      pass
7
8      drawControlFlowGraph = None
9      drawSimpleControlFlowGraph = None
10     drawConstraintGraph = None
11
12     try:
13         from pygraphviz import AGraph
14
15         def drawControlFlowGraph(self: RangeAnalyser, file: str = None) -> AGraph:
16             # some code
17             pass
18
19         def drawSimpleControlFlowGraph(self: RangeAnalyser, file: str = None) -> AGraph:
20             # some code
21             pass
22
23         def drawConstraintGraph(self: RangeAnalyser, file: str = None) -> AGraph:
24             # some code
25             pass
26
27         RangeAnalyser.drawControlFlowGraph: Callable[[RangeAnalyser, str], AGraph] =
drawControlFlowGraph
28         RangeAnalyser.drawSimpleControlFlowGraph: Callable[[RangeAnalyser, str], AGraph] =
drawSimpleControlFlowGraph
29         RangeAnalyser.drawConstraintGraph: Callable[[RangeAnalyser, str], AGraph] =
drawConstraintGraph
30     except ImportError:
31         RangeAnalyser.drawControlFlowGraph = None
32         RangeAnalyser.drawSimpleControlFlowGraph = None
33         RangeAnalyser.drawConstraintGraph = None
```

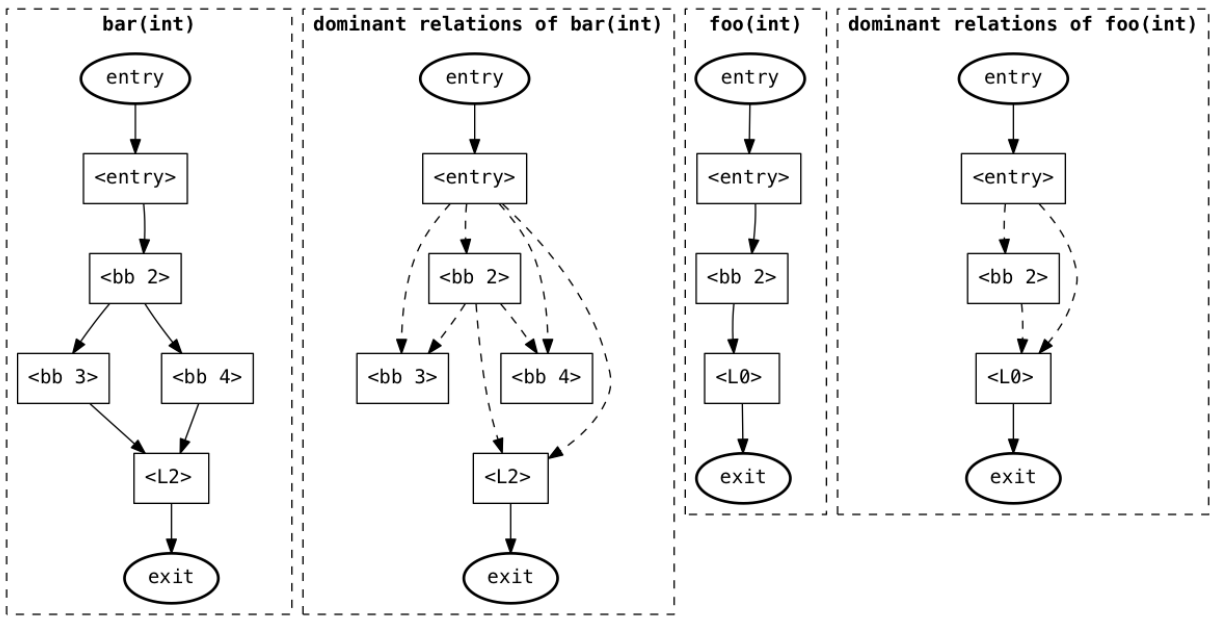
其中 CFG 的可视化结果中除了会显示各个数据块之间的控制关系外，还会包含数据流分析中得到的分析结果，如每个基本块的 `GEN`、`KILL`、`USE`、`IN`、`OUT` 等。

测试示例 `benchmark/t7.ssa` 得到的可视化结果如下：

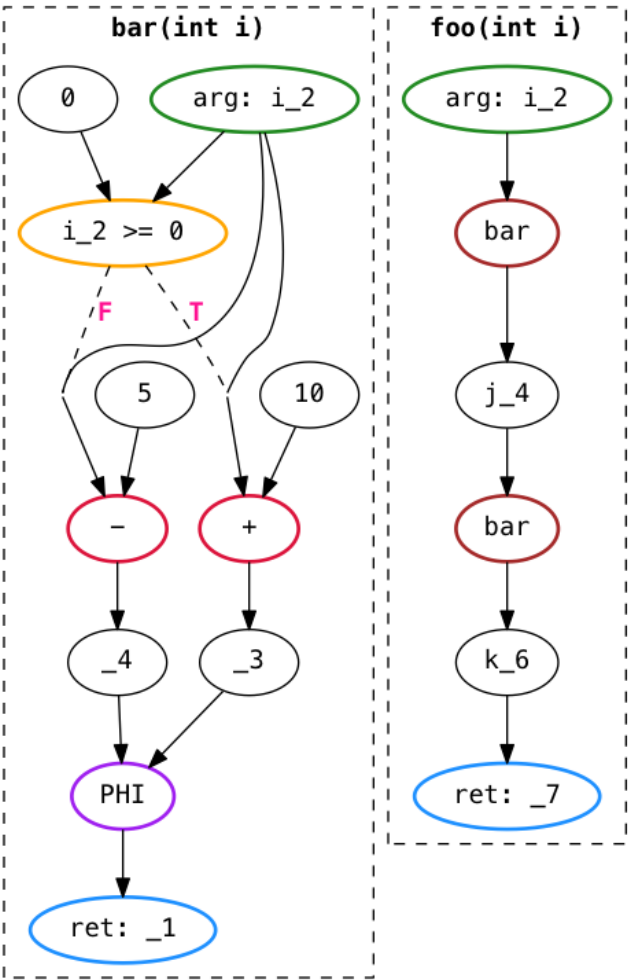
Control Flow Graph of `t7.ssa` :



Simple Control Flow Graph of `t7.ssa` :



Constraint Graph of `t7.ssa` :



```

Run: ValueRangeAnalyser
Run: anaconda3/envs/python36/bin/python3 /Users/PanXuehai/PycharmProjects/ValueRangeAnalyser/main.py

Input the name of the SSA form file (type "quit" to exit): benchmark/t2.ssa
file name: benchmark/t2.ssa
function information:
>>> function: foo(int k)
| identifiers: (int k, int j, int i, int _10)
| variables: (_10, i_2, i_5, i_7, j_3, j_6, k_1, k_4, k_9)
| block labels: ['<entry>', '<bb 2>', '<bb 3>', '<bb 4>', '<bb 5>', '<bb 6>', '<bb 7>', '<bb 8>', '<L6>']
| control flow graph: {'<entry>': {'predecessor': set(), 'successor': {'<bb 2>'}}, '<bb 2>': {'predecessor': {'<entry>'}, 'successor': {'<bb 3>'}}, '<bb 3>': {'predecessor': {'<entry>'}, 'successor': {'<bb 4>'}}, '<bb 4>': {'predecessor': {'<bb 3>'}, 'successor': {'<bb 5>'}}, '<bb 5>': {'predecessor': {'<bb 4>'}, 'successor': {'<bb 6>'}}, '<bb 6>': {'predecessor': {'<bb 5>'}, 'successor': {'<bb 7>'}}, '<bb 7>': {'predecessor': {'<bb 6>'}, 'successor': {'<bb 8>'}}, '<bb 8>': {'predecessor': {'<bb 7>'}, 'successor': {'<L6>'}}, '<L6>': {'predecessor': {'<bb 8>'}, 'successor': {'<bb 3>'}}}
| data flow: {'<entry>': {'IN': set(), 'OUT': {'k_4'}, 'USE': set(), 'GEN': {'k_4'}, 'KILL': set(), '<bb 2>': {'IN': {'k_4'}, 'OUT': {'k_4'}, 'USE': set(), 'GEN': {'k_4'}, 'KILL': set(), '<bb 3>': {'IN': {'k_4'}, 'OUT': {'k_4'}, 'USE': set(), 'GEN': {'k_4'}, 'KILL': set(), '<bb 4>': {'IN': {'k_4'}, 'OUT': {'k_4'}, 'USE': set(), 'GEN': {'k_4'}, 'KILL': set(), '<bb 5>': {'IN': {'k_4'}, 'OUT': {'k_4'}, 'USE': set(), 'GEN': {'k_4'}, 'KILL': set(), '<bb 6>': {'IN': {'k_4'}, 'OUT': {'k_4'}, 'USE': set(), 'GEN': {'k_4'}, 'KILL': set(), '<bb 7>': {'IN': {'k_4'}, 'OUT': {'k_4'}, 'USE': set(), 'GEN': {'k_4'}, 'KILL': set(), '<bb 8>': {'IN': {'k_4'}, 'OUT': {'k_4'}, 'USE': set(), 'GEN': {'k_4'}, 'KILL': set(), '<L6>': {'IN': {'k_4'}, 'OUT': {'k_4'}, 'USE': set(), 'GEN': {'k_4'}, 'KILL': set()}}
| constraints: OrderedDict([('i_5 = 0', {'stmt': 'i_5 = 0', 'type': 'assignment', 'op': 'assign', 'res': 'i_5', 'arg1': '0', 'args': ['0'], 'blockLabel': '<bb 3>'}), ('j_6 = j_3 + 1', {'stmt': 'j_6 = j_3 + 1', 'type': 'assignment', 'op': 'assign', 'res': 'j_6', 'arg1': 'j_3 + 1', 'args': ['j_3', '1'], 'blockLabel': '<bb 4>'}), ('i_2 = PHI <i_5, i_7>', {'stmt': 'i_2 = PHI <i_5, i_7>', 'type': 'phi', 'op': 'phi', 'res': 'i_2', 'arg1': 'i_5', 'args': ['i_5', 'i_7'], 'blockLabel': '<bb 5>'}), ('j_8 = j_3 + -1', {'stmt': 'j_8 = j_3 + -1', 'type': 'assignment', 'op': 'assign', 'res': 'j_8', 'arg1': 'j_3 + -1', 'args': ['j_3', '-1'], 'blockLabel': '<bb 6>'}), ('i_2 = PHI <i_5, i_7>', {'stmt': 'i_2 = PHI <i_5, i_7>', 'type': 'phi', 'op': 'phi', 'res': 'i_2', 'arg1': 'i_5', 'args': ['i_5', 'i_7'], 'blockLabel': '<bb 7>'}), ('j_6 = PHI <j_3, j_8>', {'stmt': 'j_6 = PHI <j_3, j_8>', 'type': 'phi', 'op': 'phi', 'res': 'j_6', 'arg1': 'j_3', 'args': ['j_3', 'j_8'], 'blockLabel': '<bb 8>'}), ('i_7 = PHI <i_2, i_5>', {'stmt': 'i_7 = PHI <i_2, i_5>', 'type': 'phi', 'op': 'phi', 'res': 'i_7', 'arg1': 'i_2', 'args': ['i_2', 'i_5'], 'blockLabel': '<L6>'}), ('k_1 = PHI <k_4, k_9>', {'stmt': 'k_1 = PHI <k_4, k_9>', 'type': 'phi', 'op': 'phi', 'res': 'k_1', 'arg1': 'k_4', 'args': ['k_4', 'k_9'], 'blockLabel': '<L6>'}), ('i_5 = PHI <i_5, i_7>', {'stmt': 'i_5 = PHI <i_5, i_7>', 'type': 'phi', 'op': 'phi', 'res': 'i_5', 'arg1': 'i_5', 'args': ['i_5', 'i_7'], 'blockLabel': '<L6>'}), ('j_6 = PHI <j_3, j_8>', {'stmt': 'j_6 = PHI <j_3, j_8>', 'type': 'phi', 'op': 'phi', 'res': 'j_6', 'arg1': 'j_3', 'args': ['j_3', 'j_8'], 'blockLabel': '<L6>'}), ('i_7 = PHI <i_2, i_5>', {'stmt': 'i_7 = PHI <i_2, i_5>', 'type': 'phi', 'op': 'phi', 'res': 'i_7', 'arg1': 'i_2', 'args': ['i_2', 'i_5'], 'blockLabel': '<L6>'}), ('k_1 = PHI <k_4, k_9>', {'stmt': 'k_1 = PHI <k_4, k_9>', 'type': 'phi', 'op': 'phi', 'res': 'k_1', 'arg1': 'k_4', 'args': ['k_4', 'k_9'], 'blockLabel': '<L6>'}), ('i_5 = PHI <i_5, i_7>', {'stmt': 'i_5 = PHI <i_5, i_7>', 'type': 'phi', 'op': 'phi', 'res': 'i_5', 'arg1': 'i_5', 'args': ['i_5', 'i_7'], 'blockLabel': '<L6>'}), ('j_6 = PHI <j_3, j_8>', {'stmt': 'j_6 = PHI <j_3, j_8>', 'type': 'phi', 'op': 'phi', 'res': 'j_6', 'arg1': 'j_3', 'args': ['j_3', 'j_8'], 'blockLabel': '<L6>'}), ('i_7 = PHI <i_2, i_5>', {'stmt': 'i_7 = PHI <i_2, i_5>', 'type': 'phi', 'op': 'phi', 'res': 'i_7', 'arg1': 'i_2', 'args': ['i_2', 'i_5'], 'blockLabel': '<L6>'}), ('k_1 = PHI <k_4, k_9>', {'stmt': 'k_1 = PHI <k_4, k_9>', 'type': 'phi', 'op': 'phi', 'res': 'k_1', 'arg1': 'k_4', 'args': ['k_4', 'k_9'], 'blockLabel': '<L6>'}), ('i_5 = PHI <i_5, i_7>', {'stmt': 'i_5 = PHI <i_5, i_7>', 'type': 'phi', 'op': 'phi', 'res': 'i_5', 'arg1': 'i_5', 'args': ['i_5', 'i_7'], 'blockLabel': '<L6>'}), ('j_6 = PHI <j_3, j_8>', {'stmt': 'j_6 = PHI <j_3, j_8>', 'type': 'phi', 'op': 'phi', 'res': 'j_6', 'arg1': 'j_3', 'args': ['j_3', 'j_8'], 'blockLabel': '<L6>'}), ('i_7 = PHI <i_2, i_5>', {'stmt': 'i_7 = PHI <i_2, i_5>', 'type': 'phi', 'op': 'phi', 'res': 'i_7', 'arg1': 'i_2', 'args': ['i_2', 'i_5'], 'blockLabel': '<L6>'}), ('k_1 = PHI <k_4, k_9>', {'stmt': 'k_1 = PHI <k_4, k_9>', 'type': 'phi', 'op': 'phi', 'res': 'k_1', 'arg1': 'k_4', 'args': ['k_4', 'k_9'], 'blockLabel': '<L6>'}), ('i_5 = PHI <i_5, i_7>', {'stmt': 'i_5 = PHI <i_5, i_7>', 'type': 'phi', 'op': 'phi', 'res': 'i_5', 'arg1': 'i_5', 'args': ['i_5', 'i_7'], 'blockLabel': '<L6>'}), ('j_6 = PHI <j_3, j_8>', {'stmt': 'j_6 = PHI <j_3, j_8>', 'type': 'phi', 'op': 'phi', 'res': 'j_6', 'arg1': 'j_3', 'args': ['j_3', 'j_8'], 'blockLabel': '<L6>'}), ('i_7 = PHI <i_2, i_5>', {'stmt': 'i_7 = PHI <i_2, i_5>', 'type': 'phi', 'op': 'phi', 'res': 'i_7', 'arg1': 'i_2', 'args': ['i_2', 'i_5'], 'blockLabel': '<L6>'}), ('k_1 = PHI <k_4, k_9>', {'stmt': 'k_1 = PHI <k_4, k_9>', 'type': 'phi', 'op': 'phi', 'res': 'k_1', 'arg1': 'k_4', 'args': ['k_4', 'k_9'], 'blockLabel': '<L6>'}), ('i_5 = PHI <i_5, i_7>', {'stmt': 'i_5 = PHI <i_5, i_7>', 'type': 'phi', 'op': 'phi', 'res': 'i_5', 'arg1': 'i_5', 'args': ['i_5', 'i_7'], 'blockLabel': '<L6>'}), ('j_6 = PHI <j_3, j_8>', {'stmt': 'j_6 = PHI <j_3, j_8>', 'type': 'phi', 'op': 'phi', 'res': 'j_6', 'arg1': 'j_3', 'args': ['j_3', 'j_8'], 'blockLabel': '<L6>'}), ('i_7 = PHI <i_2, i_5>', {'stmt': 'i_7 = PHI <i_2, i_5>', 'type': 'phi', 'op': 'phi', 'res': 'i_7', 'arg1': 'i_2', 'args': ['i_2', 'i_5'], 'blockLabel': '<L6>'}), ('k_1 = PHI <k_4, k_9>', {'stmt': 'k_1 = PHI <k_4, k_9>', 'type': 'phi', 'op': 'phi', 'res': 'k_1', 'arg1': 'k_4', 'args': ['k_4', 'k_9'], 'blockLabel': '<L6>'}), ('i_5 = PHI <i_5, i_7>', {'stmt': 'i_5 = PHI <i_5, i_7>', 'type': 'phi', 'op': 'phi', 'res': 'i_5', 'arg1': 'i_5', 'args': ['i_5', 'i_7'], 'blockLabel': '<L6>'}), ('j_6 = PHI <j_3, j_8>', {'stmt': 'j_6 = PHI <j_3, j_8>', 'type': 'phi', 'op': 'phi', 'res': 'j_6', 'arg1': 'j_3', 'args': ['j_3', 'j_8'], 'blockLabel': '<L6>'}), ('i_7 = PHI <i_2, i_5>', {'stmt': 'i_7 = PHI <i_2, i_5>', 'type': 'phi', 'op': 'phi', 'res': 'i_7', 'arg1': 'i_2', 'args': ['i_2', 'i_5'], 'blockLabel': '<L6>'}), ('k_1 = PHI <k_4, k_9>', {'stmt': 'k_1 = PHI <k_4, k_9>', 'type': 'phi', 'op': 'phi', 'res': 'k_1', 'arg1': 'k_4', 'args': ['k_4', 'k_9'], 'blockLabel': '<L6>'}), ('i_5 = PHI <i_5, i_7>', {'stmt': 'i_5 = PHI <i_5, i_7>', 'type': 'phi', 'op': 'phi', 'res': 'i_5', 'arg1': 'i_5', 'args': ['i_5', 'i_7'], 'blockLabel': '<L6>'}), ('j_6 = PHI <j_3, j_8>', {'stmt': 'j_6 = PHI <j_3, j_8>', 'type': 'phi', 'op': 'phi', 'res': 'j_6', 'arg1': 'j_3', 'args': ['j_3', 'j_8'], 'blockLabel': '<L6>'}), ('i_7 = PHI <i_2, i_5>', {'stmt': 'i_7 = PHI <i_2, i_5>', 'type
```

快速测试

程序提供了快速测试 `benchmark` 文件夹下的 `t1.ssa` - `t10.ssa` 的函数 `benchmark()`，使用时请将变量 `useBenchmark` 设为 `True`（默认为 `False`）。使用快速测试模式时，请确保 `benchmark` 文件夹包含的 `t1.ssa` - `t10.ssa` 这 10 个 SSA 文件，且置于与源程序同一目录下。

```
1  # main.py
2  # imports
3
4  def printSsaInfo(ssaFile: str, analyser: RangeAnalyser) -> None:
5      # some code
6      pass
7
8  def main() -> None:
9      # some code
10     pass
11
12  def benchmark() -> None:
13     testArgs: List[List[ValueRange]] = [
14         [],
15         [ValueRange(200, 300, int)],
16         [ValueRange(0, 10, int), ValueRange(20, 50, int)],
17         [ValueRange(-inf, +inf, int)],
18         [],
19         [ValueRange(-inf, +inf, int)],
20         [ValueRange(-10, 10, int)],
21         [ValueRange(1, 100, int), ValueRange(-2, 2, int)],
22         [],
23         [ValueRange(30, 50, int), ValueRange(90, 100, int)]
24     ]
25     refRanges: List[str] = ['[100, 100]',
26                             '[200, 300]',
27                             '[20, 50]',
28                             '[0, +inf)',
29                             '[210, 210]',
30                             '[-9, 10]',
31                             '[16, 30]',
32                             '[-3.2192308, 5.94230769]',
33                             '[9791, 9791]',
34                             '[-10, 40]']
35     for i, (testArg, refRange) in enumerate(zip(testArgs, refRanges), start = 1):
36         ssaFile = 'benchmark/t{}.ssa'.format(i)
37         # some code
38         pass
39
40  if __name__ == '__main__':
41     useBenchmark: bool = False # 若使用快速测试请改为 True
42     if useBenchmark:
43         benchmark()
44     else:
45         main()
```

快速测试使用的变量范围为 C 源程序的注释中的推荐范围。分析结束后，程序会显示输出范围及函数的实际输出范围。

```
Run - ValueRangeAnalyser
main <
/anaconda3/envs/python36/bin/python3 /Users/PanXuehai/PycharmProjects/ValueRangeAnalyser/main.py

file name: benchmark/t1.ssa
function information:
>>> function: foo()
| Identifiers: {int i, int j, int k, int _10}
| variables: {_10, i_2, i_5, i_7, j_3, j_6, j_8, k_1, k_4, k_9}
| block labels: {'<entry>', '<bb 2>', '<bb 3>', '<bb 4>', '<bb 5>', '<bb 6>', '<bb 7>', '<bb 8>', '<L6>'}
| control flow graph: {'<entry>': {'predecessor': set(), 'successor': {'<bb 2>'}, '<bb 2>': {'predecessor': {'<entry>', 'successor': {'<bb 7>'}}}, '<bb 3>': {'pr
| data flow: {'<entry>': {'IN': set(), 'OUT': set(), 'USE': set(), 'KILL': set(), '<bb 2>': {'IN': set(), 'OUT': {'k_4'}, 'USE': set(), 'GEN': {'k_
| constraints: OrderedDict([('k_4 = 0', {'stmt': 'k_4 = 0', 'type': 'assignment', 'op': 'assign', 'res': 'k_4', 'arg1': '0', 'args': ['0'], 'blockLabel': '<bb 2>'
| def statement of variables: {'k_4': 'k_4 = 0', 'i_5': 'i_5 = 0', 'j_6': 'j_6 = k_1', 'i_7': 'i_7 = i_2 + 1', 'j_8': 'j_8 = j_3 + 1', 'i_2': '# i_2 = PHI <i_5,
| use statements of variables: {'k_4': ['# k_1 = PHI <k_4, k_9>'], 'i_5': ['# i_2 = PHI <i_5, i_7>'], 'j_6': ['# j_3 = PHI <j_6, j_8>'], 'i_7': ['# i_2 = PHI <i_5,

#####
analyse foo()
<entry>:
| <bb 2>:
| | int k_4: { 0 }
| <bb 3>:
| | int i_5: { 0 }
| | int j_6: [0, 99]
| <bb 4>:
| | int i_7: [1, +inf)
| | int j_8: [0, +inf)
| <bb 5>:
| | int i_2: [0, +inf)
| | int j_3: [1, +inf)
| <bb 6>:
| | int k_9: [1, 100]
| <bb 7>:
| | int k_1: [0, 100]
| <bb 8>:
| | int _10: { 100 }
| <L6>:
| | int _10: { 100 }
foo() returns { 100 }
reference range: [100, 100]

#####

file name: benchmark/t2.ssa
function information:
>>> function: foo(int k)
| identifiers: {int k, int j, int i, int _10}
| variables: {_10, i_2, i_5, i_7, j_3, j_6, j_8, k_1, k_4, k_9}
| block labels: {'<entry>', '<bb 2>', '<bb 3>', '<bb 4>', '<bb 5>', '<bb 6>', '<bb 7>', '<bb 8>', '<L6>'}
```