第三次Lab：

Dataflow analysis

本次任务

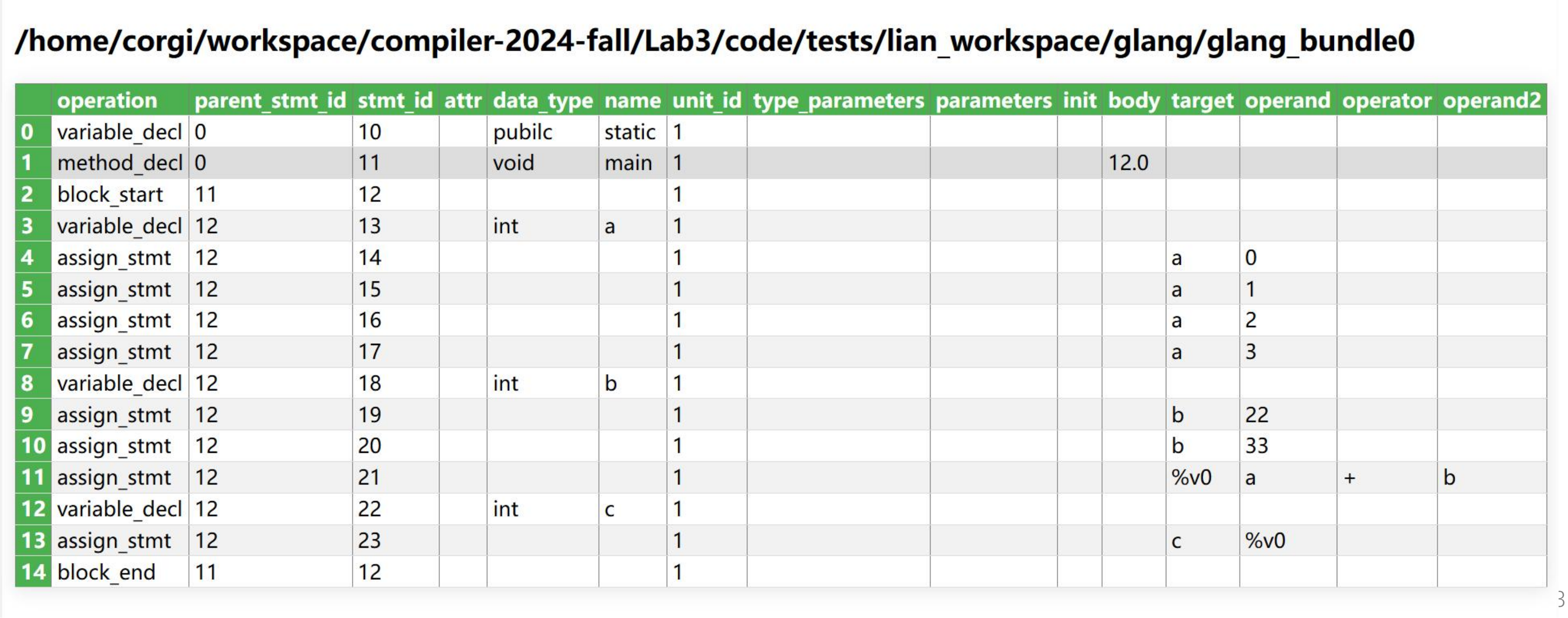
1. 补充完成reaching_symbol_analysis，包括三个部分
   a. 根据cfg准备并设置status.in_bits
   b. 根据当前defined_symbol的all_def_stmts,通过self.bit_vector_manager,应用kill-gen算法对status.out_bits进行更新
   c. 通过判断out_bits是否变化来判断是否到达不动点
2. 在bit_vectcor_manager中补充完成对bit_vector的操作，包括kill和gen操作。

请参考《6.中端代码分析和优化》课件中数据流算法描述

[1]

环境搭建

- 从gitee仓库中下载代码，在/script/目录下运行./lian.sh处理typescript文件，若dataframe.html网页中显示了正确的glang ir，则说明环境搭建正确.



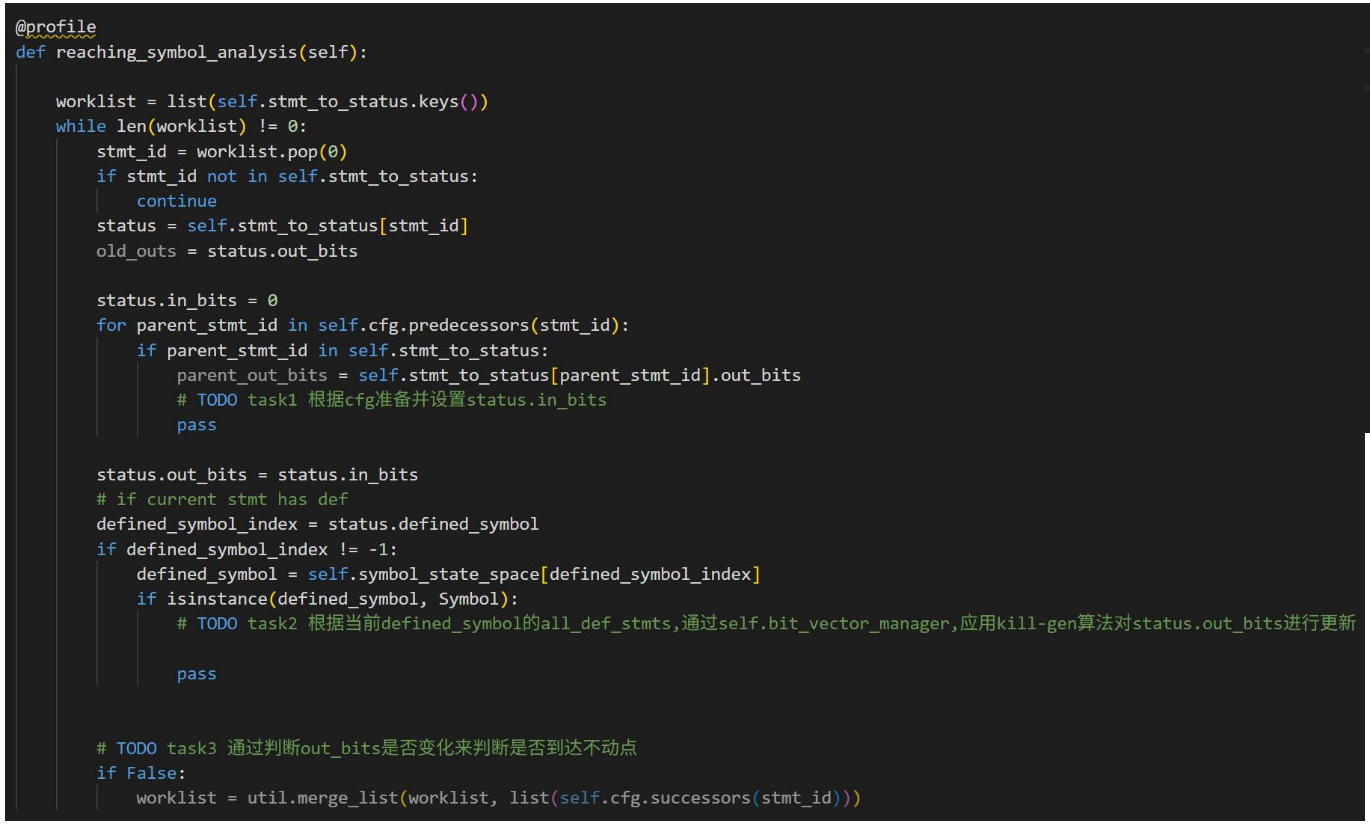
/home/corgi/workspace/compiler-2024-fall/Lab3/code/tests/lian_workspace/glang/glang_bundle0

	operation	parent_stmt_id	stmt_id	attr	data_type	name	unit_id	type_parameters	parameters	init	body	target	operand	operator	operand2
0	variable_decl	0	10		pubilc	static	1								
1	method_decl	0	11		void	main	1				12.0				
2	block_start	11	12				1								
3	variable_decl	12	13		int	a	1								
4	assign_stmt	12	14				1					a	0		
5	assign_stmt	12	15				1					a	1		
6	assign_stmt	12	16				1					a	2		
7	assign_stmt	12	17				1					a	3		
8	variable_decl	12	18		int	b	1								
9	assign_stmt	12	19				1					b	22		
10	assign_stmt	12	20				1					b	33		
11	assign_stmt	12	21				1					%v0	a	+	b
12	variable_decl	12	22		int	c	1								
13	assign_stmt	12	23				1					c	%v0		
14	block_end	11	12				1								

本次任务

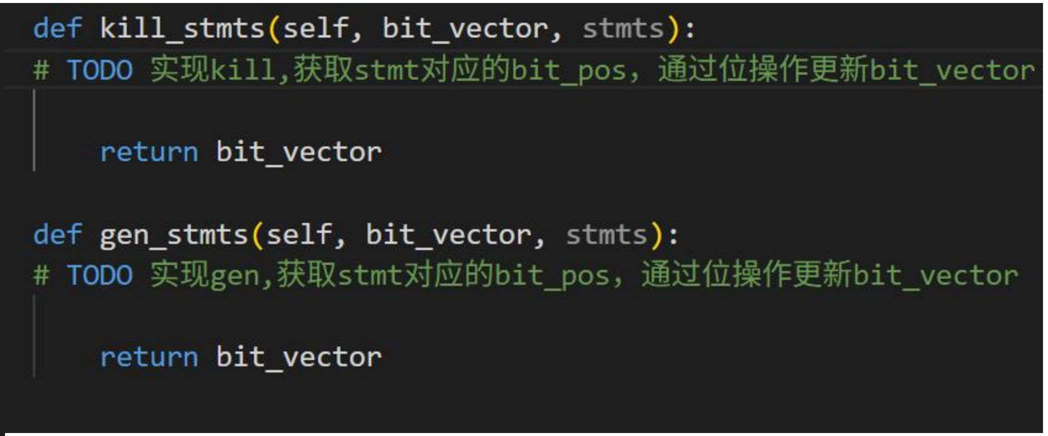
- 需要完成两个文件，共三个函数

在state_flow.py中，需要完成：



```python
@profile
def reaching_symbol_analysis(self):

    worklist = list(self.stmt_to_status.keys())
    while len(worklist) != 0:
        stmt_id = worklist.pop(0)
        if stmt_id not in self.stmt_to_status:
            continue
        status = self.stmt_to_status[stmt_id]
        old_outs = status.out_bits

        status.in_bits = 0
        for parent_stmt_id in self.cfg.predecessors(stmt_id):
            if parent_stmt_id in self.stmt_to_status:
                parent_out_bits = self.stmt_to_status[parent_stmt_id].out_bits
                # TODO task1 根据cfg准备并设置status.in_bits
                pass

        status.out_bits = status.in_bits
        # if current stmt has def
        defined_symbol_index = status.defined_symbol
        if defined_symbol_index != -1:
            defined_symbol = self.symbol_state_space[defined_symbol_index]
            if isinstance(defined_symbol, Symbol):
                # TODO task2 根据当前defined_symbol的all_def_stmts,通过self.bit_vector_manager,应用kill-gen算法对status.out_bits进行更新

                pass


        # TODO task3 通过判断out_bits是否变化来判断是否到达不动点
        if False:
            worklist = util.merge_list(worklist, list(self.cfg.successors(stmt_id)))
```

在internal_structure.py中，需要完成：



```python
def kill_stmts(self, bit_vector, stmts):
# TODO 实现kill,获取stmt对应的bit_pos, 通过位操作更新bit_vector

    return bit_vector

def gen_stmts(self, bit_vector, stmts):
# TODO 实现gen,获取stmt对应的bit_pos, 通过位操作更新bit_vector

    return bit_vector
```

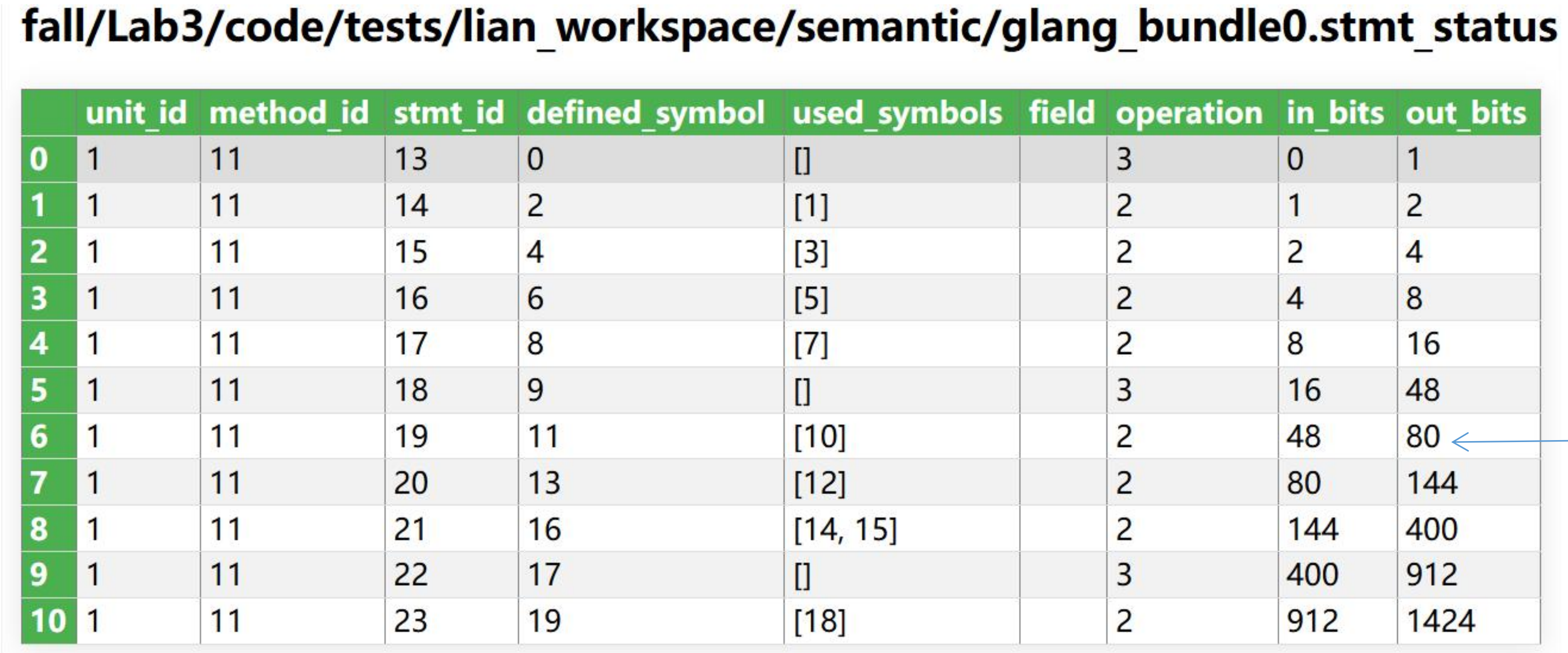
补充知识

- status是什么?

  status记录了每条语句的信息，包括语句的stmt_id, bit_vector等信息

- bit_vector是什么样的?

  给glang指令中每个symbol的define分配一个bit，利用int(整数)存储bit_vector，例如(103)D=(01100111)B



fall/Lab3/code/tests/lian_workspace/semantic/glang_bundle0.stmt_status

	unit_id	method_id	stmt_id	defined_symbol	used_symbols	field	operation	in_bits	out_bits
0	1	11	13	0	[]		3	0	1
1	1	11	14	2	[1]		2	1	2
2	1	11	15	4	[3]		2	2	4
3	1	11	16	6	[5]		2	4	8
4	1	11	17	8	[7]		2	8	16
5	1	11	18	9	[]		3	16	48
6	1	11	19	11	[10]		2	48	80
7	1	11	20	13	[12]		2	80	144
8	1	11	21	16	[14, 15]		2	144	400
9	1	11	22	17	[]		3	400	912
10	1	11	23	19	[18]		2	912	1424

bit_vector

补充知识

- 如何通过symbol的name拿到一个symbol所有define它的stmt?

  all_def_stmts = self.symbol_to_def_stmts[symbol]

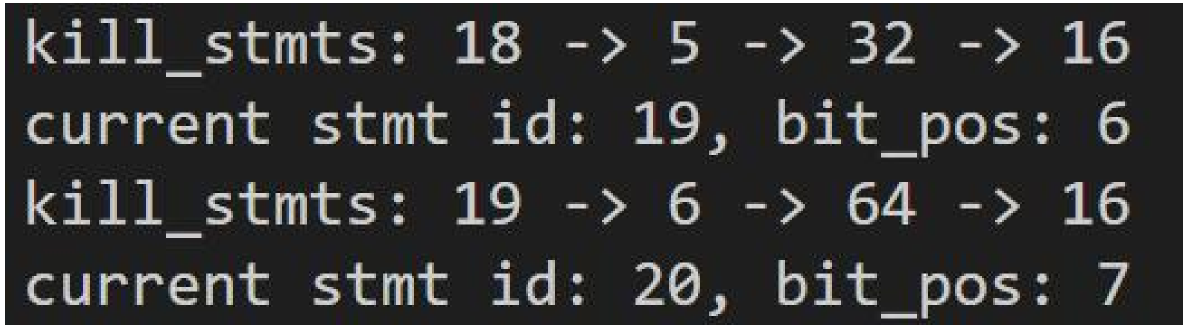
- 如何通过kill和gen获取current_bits?

  current_bits = self.bit_vector_manager.kill_stmts(current_bits, all_def_stmts), 这里的kill_stmts需要自己在bit_vector_manager中实现。

- 如何通过当前stmt_id，得到它在bit_vector中对应的bit在哪一位?

  在BitVectorManager中，stmt_to_bit_pos的get()方法

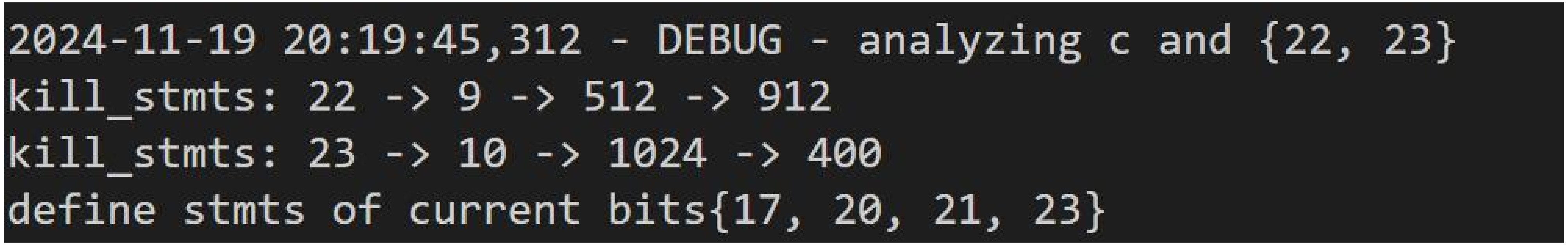
  bit_pos = self.stmt_to_bit_pos.get(stmt_id)



```
kill_stmts: 18 -> 5 -> 32 -> 16
current stmt id: 19, bit_pos: 6
kill_stmts: 19 -> 6 -> 64 -> 16
current stmt id: 20, bit_pos: 7
```

补充知识

- 可以使用bit_vector_manager的explain来翻译当前bit_vector，得到bit_vector所对应的define的stmt。



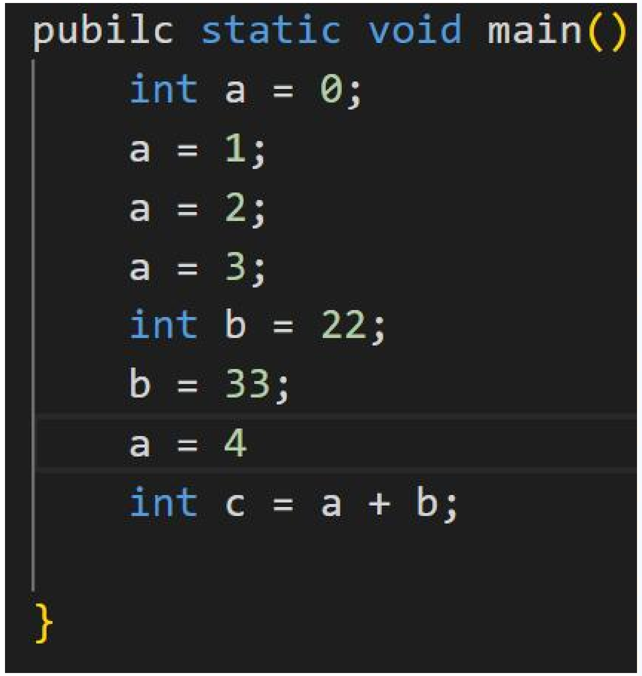
```
2024-11-19 20:19:45,312 - DEBUG - analyzing c and {22, 23}
kill_stmts: 22 -> 9 -> 512 -> 912
kill_stmts: 23 -> 10 -> 1024 -> 400
define stmts of current bits{17, 20, 21, 23}
```

注意，图中“17，20，21，23”为glang ir的stmt_id，并非源代码行号, 也不是index。

[1]

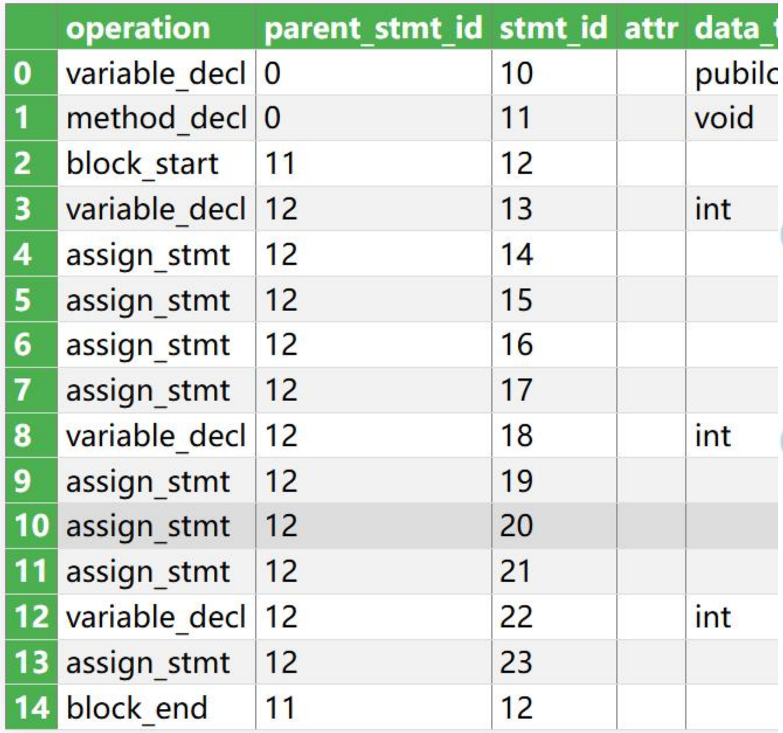
补充知识： cfg in lian

源代码:



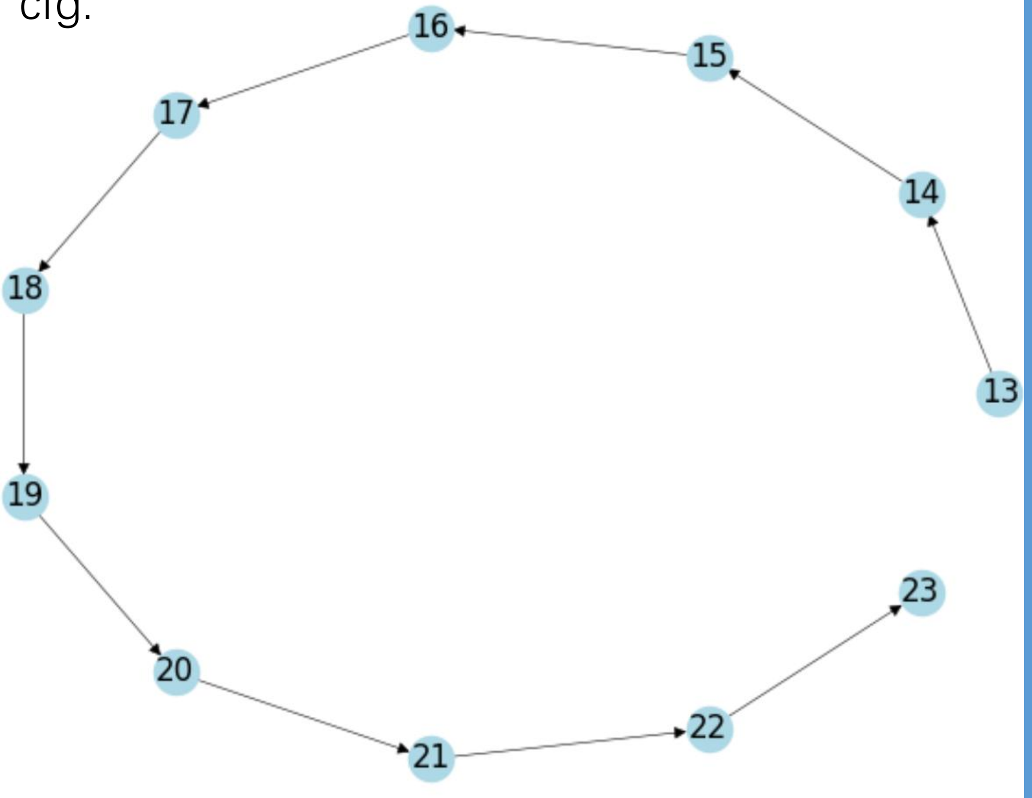
```
pubilc static void main(){
    int a = 0;
    a = 1;
    a = 2;
    a = 3;
    int b = 22;
    b = 33;
    a = 4
    int c = a + b;

}
```

glang:



	operation	parent_stmt_id	stmt_id	attr	data_
0	variable_decl	0	10		pubilc
1	method_decl	0	11		void
2	block_start	11	12		
3	variable_decl	12	13		int
4	assign_stmt	12	14		
5	assign_stmt	12	15		
6	assign_stmt	12	16		
7	assign_stmt	12	17		
8	variable_decl	12	18		int
9	assign_stmt	12	19		
10	assign_stmt	12	20		
11	assign_stmt	12	21		
12	variable_decl	12	22		int
13	assign_stmt	12	23		
14	block_end	11	12		

cfg:

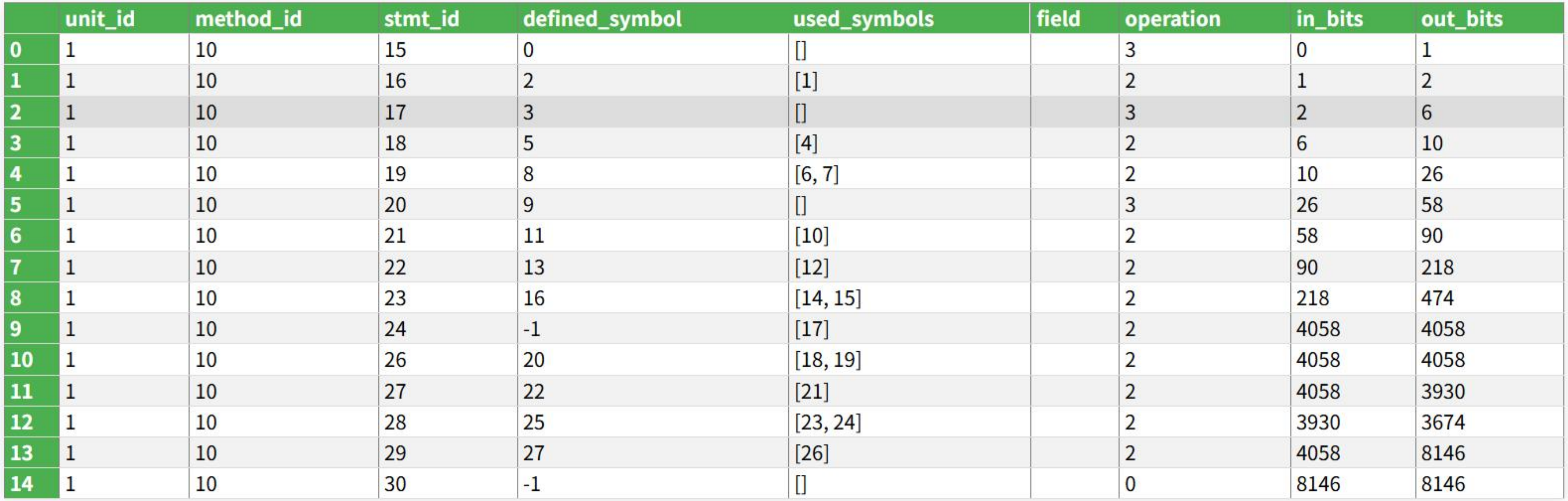


源代码在转换成glang ir后，根据glang ir生成cfg，每一个节点对应一条stmt，节点的编号是stmt_id.
cfg在/code/tests/lian_workspace/semantic/glang_bundle0_cfg.png

参考结果

根据测试代码，得出的bit——vector结果正确即可

/home/corgi/workspace/compiler2025spring/lab3/code/tests/lian_workspace/semantic/glang_bundle0.stmt_status



	unit_id	method_id	stmt_id	defined_symbol	used_symbols	field	operation	in_bits	out_bits
0	1	10	15	0	[]		3	0	1
1	1	10	16	2	[1]		2	1	2
2	1	10	17	3	[]		3	2	6
3	1	10	18	5	[4]		2	6	10
4	1	10	19	8	[6, 7]		2	10	26
5	1	10	20	9	[]		3	26	58
6	1	10	21	11	[10]		2	58	90
7	1	10	22	13	[12]		2	90	218
8	1	10	23	16	[14, 15]		2	218	474
9	1	10	24	-1	[17]		2	4058	4058
10	1	10	26	20	[18, 19]		2	4058	4058
11	1	10	27	22	[21]		2	4058	3930
12	1	10	28	25	[23, 24]		2	3930	3674
13	1	10	29	27	[26]		2	4058	8146
14	1	10	30	-1	[]		0	8146	8146

[1]

TIPS

- 多打印调试可以快速熟悉api的用法
- 先从简单的例子开始测试

[1]