# Lab3 Week1

王宁森 周子轩
22307130058 22307130401

## 截图

| | operation | parent_stmt_id | stmt_id | attrs | data_type | name | body | unit_id | target | operand | positional_args | condition | then_body | array | index | source |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | method_decl | 0 | 10 | | | aaa | 11.0 | 1 | | | | | | | | |
| 1 | block_start | 10 | 11 | | | | | 1 | | | | | | | | |
| 2 | assign_stmt | 11 | 12 | | | | | 1 | a | 1 | | | | | | |
| 3 | assign_stmt | 11 | 13 | | | | | 1 | b | 2 | | | | | | |
| 4 | assign_stmt | 11 | 14 | | | | | 1 | b | a | | | | | | |
| 5 | call_stmt | 11 | 15 | | | func1 | | 1 | %v0 | | ['a'] | | | | | |
| 6 | if_stmt | 11 | 16 | | | | | 1 | | | | c | 17.0 | | | |
| 7 | block_start | 16 | 17 | | | | | 1 | | | | | | | | |
| 8 | assign_stmt | 17 | 18 | | | | | 1 | e | 3 | | | | | | |
| 9 | block_end | 16 | 17 | | | | | 1 | | | | | | | | |
| 10 | array_read | 11 | 19 | | | | | 1 | %v1 | | | | | arr | 0 | |
| 11 | assign_stmt | 11 | 20 | | | | | 1 | c | %v1 | | | | | | |
| 12 | array_read | 11 | 21 | | | | | 1 | %v1 | | | | | arr | a | |
| 13 | assign_stmt | 11 | 22 | | | | | 1 | d | %v1 | | | | | | |
| 14 | array_write | 11 | 23 | | | | | 1 | | | | | | arr | b | c |
| 15 | block_end | 10 | 11 | | | | | 1 | | | | | | | | |

| | unit_id | stmt_id | parent_stmt_id | scope_kind | package_stmt | import_stmt | variable_decl | method_decl | class_decl |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 3 | | | | (10, 'aaa') | |
| 1 | 1 | 10 | 0 | 1 | | | | | |

| | unit_id | method_id | stmt_id | defined_symbol | used_symbols | field | operation | in_bits | out_bits |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 10 | 12 | 1 | [0] | | 2 | 0 | 0 |
| 1 | 1 | 10 | 13 | 3 | [2] | | 2 | 0 | 0 |
| 2 | 1 | 10 | 14 | 5 | [4] | | 2 | 0 | 0 |
| 3 | 1 | 10 | 15 | 8 | [6, 7] | | 2 | 0 | 0 |
| 4 | 1 | 10 | 16 | -1 | [9] | | 2 | 0 | 0 |
| 5 | 1 | 10 | 18 | 11 | [10] | | 2 | 0 | 0 |
| 6 | 1 | 10 | 19 | 14 | [12, 13] | | 2 | 0 | 0 |
| 7 | 1 | 10 | 20 | 16 | [15] | | 2 | 0 | 0 |
| 8 | 1 | 10 | 21 | 19 | [17, 18] | | 2 | 0 | 0 |
| 9 | 1 | 10 | 22 | 21 | [20] | | 2 | 0 | 0 |
| 10 | 1 | 10 | 23 | 25 | [22, 23, 24] | | 2 | 0 | 0 |

| | unit_id | method_id | stmt_id | index | symbol_or_state | symbol_id | name | states | default_data_type | state_id | state_type | data_type | array | array_tangping_flag | fields | value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 10 | 12 | 0 | 1 | | | | | 1 | 1 | int | [] | False | {} | 1 |
| 1 | 1 | 10 | 12 | 1 | 0 | 2 | a | set() | | -1 | 0 | | | | | |
| 2 | 1 | 10 | 13 | 2 | 1 | | | | | 3 | 1 | int | [] | False | {} | 2 |
| 3 | 1 | 10 | 13 | 3 | 0 | 4 | b | set() | | -1 | 0 | | | | | |
| 4 | 1 | 10 | 14 | 4 | 0 | 5 | a | set() | | -1 | 0 | | | | | |
| 5 | 1 | 10 | 14 | 5 | 0 | 6 | b | set() | | -1 | 0 | | | | | |
| 6 | 1 | 10 | 15 | 6 | 0 | 7 | func1 | set() | | -1 | 0 | | | | | |
| 7 | 1 | 10 | 15 | 7 | 0 | 8 | a | set() | | -1 | 0 | | | | | |
| 8 | 1 | 10 | 15 | 8 | 0 | 9 | %v0 | set() | | -1 | 0 | | | | | |
| 9 | 1 | 10 | 16 | 9 | 0 | 10 | c | set() | | -1 | 0 | | | | | |
| 10 | 1 | 10 | 18 | 10 | 1 | | | | | 11 | 1 | int | [] | False | {} | 3 |
| 11 | 1 | 10 | 18 | 11 | 0 | 12 | e | set() | | -1 | 0 | | | | | |
| 12 | 1 | 10 | 19 | 12 | 0 | 13 | arr | set() | | -1 | 0 | | | | | |
| 13 | 1 | 10 | 19 | 13 | 1 | | | | | 14 | 1 | int | [] | False | {} | 0 |
| 14 | 1 | 10 | 19 | 14 | 0 | 15 | %v1 | set() | | -1 | 0 | | | | | |
| 15 | 1 | 10 | 20 | 15 | 0 | 16 | %v1 | set() | | -1 | 0 | | | | | |
| 16 | 1 | 10 | 20 | 16 | 0 | 17 | c | set() | | -1 | 0 | | | | | |
| 17 | 1 | 10 | 21 | 17 | 0 | 18 | arr | set() | | -1 | 0 | | | | | |
| 18 | 1 | 10 | 21 | 18 | 0 | 19 | a | set() | | -1 | 0 | | | | | |
| 19 | 1 | 10 | 21 | 19 | 0 | 20 | %v1 | set() | | -1 | 0 | | | | | |
| 20 | 1 | 10 | 22 | 20 | 0 | 21 | %v1 | set() | | -1 | 0 | | | | | |
| 21 | 1 | 10 | 22 | 21 | 0 | 22 | d | set() | | -1 | 0 | | | | | |
| 22 | 1 | 10 | 23 | 22 | 0 | 23 | arr | set() | | -1 | 0 | | | | | |
| 23 | 1 | 10 | 23 | 23 | 0 | 24 | b | set() | | -1 | 0 | | | | | |
| 24 | 1 | 10 | 23 | 24 | 0 | 25 | c | set() | | -1 | 0 | | | | | |
| 25 | 1 | 10 | 23 | 25 | 0 | 26 | arr | set() | | -1 | 0 | | | | | |

## 解析call_expression

```
    def call_stmt_def_use(self, stmt_id, stmt):
        # convert stmt.args(str) to list
        args_list = []
```

```
        positional_args = ast.literal_eval(stmt.positional_args)
        #lab3week1 work

        # The function name is used
        used_symbols = [stmt.name]

        # All positional arguments are used
        for arg in positional_args:
            if not util.isna(arg):
                used_symbols.append(arg)

        # If there's a target, it's defined
        defined_symbol = stmt.target

        self.add_def_use_symbols(
            stmt_id,
            def_symbol=defined_symbol,
            used_symbols=used_symbols,
            op=ComputeOperation.CALL
        )
```

`call_stmt`需要解析函数调用时发生的`def-use`，在这里`positional_args`存储有函数调用时的所有实参，有效实参和被调用函数本身`stmt.name`均为发生了`use`。在添加实参到`used_symbols`时会检查当前参数是否不是一个“不可用”（NA - Not Available）的值，如果不是则添加到`used_symbols`。如果函数调用时，函数返回值被赋值给了一个新的对象即`target`字段的内容，则`target`字段部分对应了`def`。

在我们的测试样例中涉及到`call_stmt_def_use`的为：

```
func1(a)
```

它在GIR中的`stmt_id`为15。

| operation | stmt_id | name | target | positional_args |
|---|---|---|---|---|
| call_stmt | 15 | func1 | %v0 | ['a'] |

去找对应的`def-use`：

| stmt_id | defined_symbol | used_symbols |
|---|---|---|
| 15 | 8 | [6, 7] |

可以看到被`def`的`index`为8，`used`的`index`为6，7。

| stmt_id | index | name |
|---|---|---|
| 15 | 8 | %v0 |
| 15 | 6 | func1 |

| stmt_id | index | name |
|---|---|---|
| 15 | 7 | a |

对应地，在这里找到index为6，7，8的三行，可以看到函数func1被调用时，used的是这个函数本身func1和它的参数a，def的是中间变量%v0，这与我们解析得到的GIR保持一致。

## 解析if_stmt

```
    def if_stmt_def_use(self, stmt_id, stmt):
        #lab3week1 work
        # Only the condition is used
        used_symbols = [stmt.condition]

        self.add_def_use_symbols(
            stmt_id,
            def_symbol=None,  # if statement doesn't define anything
            used_symbols=used_symbols,
            op=ComputeOperation.IF
        )
```

if_stmt就要简单很多，整个过程中只有条件变量condition字段部分是used，而if_stmt本身**不会定义任何内容。**

在测试中涉及if_stmt的是：

```
    if (c){
        ...
    }
```

它在GIR中的stmt_id为16。

| operation | stmt_id | condition |
|---|---|---|
| if_stmt | 16 | c |

去找对应的def-use：

| stmt_id | defined_symbol | used_symbols |
|---|---|---|
| 16 | -1 | [9] |

可以看到被def的index没有，used的index为9。

| stmt_id | index | name |
|---|---|---|
| 16 | 9 | c |

对应地，在这里找到index为9的，可以看到这个if_stmt只会涉及use条件变量c。

## 解析array_wrirte

```
    def array_write_def_use(self, stmt_id, stmt):
        # The array variable, index, and value are used
        used_symbols = [stmt.array, stmt.index, stmt.source]

        # The array element is defined (array_name is both used and defined)
        defined_symbol = stmt.array

        self.add_def_use_symbols(
            stmt_id,
            def_symbol=defined_symbol,
            used_symbols=used_symbols,
            op=ComputeOperation.ARRAY_WRITE
        )
```

array_write需要解析数组写入时的def-use，在array_write_def_use()方法中完成。在写入数组时，例如语句arr[b] = c，被define的量是数组变量名arr，被use的量有数组变量名arr、数组索引b和写入的数据c。其中，数组变量名arr即被define也被use是因为arr既要用于提供计算内存地址的起始位置，其数组值也会被修改。 在上述代码实现中，used_symbols = [stmt.array, stmt.index, stmt.source]表示被使用的值有数组变量名stmt.array、数组索引stmt.index和右式中写入数组的值stmt.source。
defined_symbol = stmt.array表示被定义的值是数组变量名stmt.array。

在"截图"中可以看到，上述例子arr[b]=c对应的def-use关系如下：

| stmt_id | defined_symbol | used_symbols |
|---|---|---|
| 23 | 25 | [22, 23, 24] |

| stmt_id | index | name |
|---|---|---|
| 23 | 22 | arr |
| 23 | 23 | b |
| 23 | 24 | c |
| 23 | 25 | arr |

可以看出，解析结果于之前的分析一致。

## 解析array_read

```
    def array_read_def_use(self, stmt_id, stmt):
        # The array variable and index are used
        used_symbols = [stmt.array, stmt.index]

        # The target is defined
```

```
            defined_symbol = stmt.target

            self.add_def_use_symbols(
                stmt_id,
                def_symbol=defined_symbol,
                used_symbols=used_symbols,
                op=ComputeOperation.ARRAY_READ
            )
```

array_read需要解析数组读取的def-use，在array_read_def_use()方法中完成。在读取数组时，例如语句d=arr[a]，被define的量是d，被use的量有该数组的变量名arr和访问数组的索引a。在上述代码实现中，defined_symbol = stmt.target，即语句的左式stmt.targrt被定义；used_symbols = [stmt.array, stmt.index]，即数组变量名stmt.array和数组索引stmt.index被使用。

在"截图"中可以看到，上述例子d=arr[a]对应的def-use关系如下：

| stmt_id | defined_symbol | used_symbols |
|---|---|---|
| 21 | 19 | [17, 18] |
| 22 | 21 | [20] |

| stmt_id | index | name |
|---|---|---|
| 21 | 17 | arr |
| 21 | 18 | a |
| 21 | 19 | %v2 |
| 22 | 20 | %v2 |
| 22 | 21 | d |

在上表中，该赋值语句被拆解成两条语句，并由中间临时变量%v2存储读取的值，再赋给d。可以看出，解析结果于之前的分析一致。

## 遇到的问题

- 一开始在写array_write_def_use时，以为数组名只会被define不会被use，所以漏掉了一个使用值。后来对照给的参考答案和一些资料了解到数组名会同时被定义和使用。
- 在写if_stmt_def_use时产生了一个奇思妙想，想到C语言中可以这么写：

```
#include <stdio.h>

bool isTrue(bool a){
    return a;
}

int main() {
```

```
    bool a = 1;
    if(bool condition = isTrue(a)){
        printf("Yes, it is legal to write a def in if statement for C.\n");
    }

    return 0;
}
```

也就是说在c语言中可以在`if_stmt`中进行`def`操作的。我开始担忧ts中是否也有类似做法。经过资料查阅后发现ts里严格限制了`if_stmt`中不可以进行变量声明，于是就不担心了。