Bonus Task: Escape Detection

Chew Yung Chung	Tan Qiu Hao, Joel	Juliana Seng
a0133662J	a0125473H	a0126332R
a0133662@u.nus.edu	a0125473@u.nus.edu	a0126332@u.nus.edu

Implementation and Algorithm

The algorithm described here is an extension of the one in task 2. In addition to the original algorithm, we now maintain a global data structure **`globalPtrVarMap`** that is populated with information about all the global pointer variables that exist in the program.

For each function which is not the main() function, a check is performed. Within each function, the algorithm maintains a list of new pointers allocated just like task 2. For each instruction in the function, if the instruction is an alloca instruction, the variable name is added to the **allocaMap**. Alloca instruction is performed when storage is required on stack.

When a store instruction is encountered, the operands are retrieved for processing. The source operand is checked against entries in allocaMap and the destination operand checked against entries in globalPtrVarMap to verify if they are a local pointer and global pointer respectively. If a match is found in both maps, the program concludes that an attempt is made by a local pointer to escape via a global pointer, and reports an escape violation. The name of the pointer variable and the function it resides in are printed in the warning messages.

Build and Run

Run build.sh to compile the test cases, program and execute the test cases.

```
$ ./build.sh
```

The individual commands can also be found in build.sh.

An expected output of the shell code is as follows, all warnings are truncated.

```
----------COMPILING TESTCASES----------


----------COMPILING PROGRAM----------


----------RUNNING TEST 1----------
WARNING: pointer <local_char> in the function <escape_local> tries to escape through the global pointer <globalptr>!
```

Test Cases

Test case 1: Memory Antipattern 3

```
$ ./asg2-bonus globalescape.ll
```

Test case 1 observes the behaviour of globalescape.c, whereby on line 12, an attempt to store a local pointer reference to a global pointer was made.

```
 1  char *globalptr;
10  void escape_local(void) {
11          char local_char = 'a';
12          globalptr = &local_char;
13  }
```

Code snippet from globalescape.c

Limitations

Escaping local pointers via an input argument pointer

For this bonus task, the algorithm is unable to detect local pointer escape through input argument pointers to a function. An example similar to globalescape.c is as follows

```
char *globalptr;

void escape_local(char **argptr);

int main () {
        char **argptr;
        escape_local(argptr);
        return 0;
}

void escape_local(char **argptr) {
        char local_char = 'a';
        *argptr = &local_char;
}
```

In the function escape_local(), the address of a local char variable **`local_char`** is stored in **`*argptr`**, an input *pointer argument to the function. We were unable to detect an escape of this nature as the address of the input pointer argument **`argptr`** is loaded into a virtual register first, followed by storing the reference of the local variable **`local_char`** into the virtual register. In this way, the name of input argument is difficult to retrieve as there was an intermediate virtual register involved.