NULL checks in C

1 Introduction

The goal of this assignment is to detect array accesses with `NULL` bases at runtime.

2 Data flow analysis [6 Marks]

In this assignment, you are to implement a data-flow analysis to identify array accesses where the array base is guaranteed to be not `NULL`. Let's call these non `NULL` bases as safe pointers. Pointer arithmetic to safe pointers always yields safe pointers. The data-flow analysis tracks all arguments and local variables that are pointers. Because a function can be called from multiple places, and it is valid to pass a `NULL` value as an argument, all pointer arguments are treated as unsafe. Similarly, the return value of a function is always unsafe. However, there is one exception to the above rule. A pointer can be initialized using "mymalloc" routine. You can assume that "mymalloc" always returns a valid memory location.

The local variables and arguments are stored on the stack. LLVM uses "alloca" instruction to allocate space on the stack. It is always safe to load or store from a stack-allocated memory operand. When the analysis pass encounters a use of an unsafe pointer in the address computation for a load or store, or the unsafe pointer is used directly in a load or store, the data-flow analysis marks the corresponding instruction (which uses the unsafe pointer) for the dynamic check insertion.

After insertion of the dynamic check, an unsafe pointer becomes safe in that basic block because the check ensures the termination of the program if the pointer value is `NULL` during execution. Similarly, when a safe operand is getting stored in a stack location, the corresponding stack location is also marked as safe (assuming the program can not overwrite the stack location indirectly). Future loads from a safe stack location in that basic block yield the safe value unless we store an unsafe value in that stack location.

Your goal is to implement an LLVM pass to identify instructions that are unsafe and need dynamic checks. For this, you need to define transfer func-

```
int max(int *arr1, int *arr2) {
  if (arr1[0] > arr2[0]) {
    return arr1[0];
  }
  return arr2[0];
}
```

(a) Without checks

```
int max(int *arr1, int *arr2) {
  if (arr1 == NULL) {
    goto out;
  }
  if (arr2 == NULL) {
    goto out;
  }
  if (arr1[0] > arr2[0]) {
    return arr1[0];
  }
  return arr2[0];
out:
  abort();
  return 0;
}
```

(b) With checks

Figure 1: Dynamic checks for `NULL` detection

tions for every LLVM instruction and a meet operator to merge results from predecessors.

3 Dynamic checks [4 Marks]

After you identify the instructions that needed dynamic checks, your next goal is to insert the actual checks before them. The dynamic check compares the value of the memory operand with `NULL` and calls abort if the memory operand is `NULL`. For example, Figure 1 shows a transformation with dynamic checks.

4 Environment

For this assignment, you need to clone the project repository.
To clone, run, git clone `https://github.com/Systems-IIITD/CSE601`.
If you have already cloned, sync your local repository by running `git pull origin master`. To build the project, follow the steps in the `README.md` file.
You have to make all the changes in the `‘‘llvm/lib/CodeGen/SafeC/NullChecks.cpp’’` file. The name of this pass is `nullcheck`. You can use the `opt` tool to run `nullcheck` pass. The makefile in the tests folder is already running `nullcheck` pass using the `opt` tool. `Opt` tool allows you to run a custom sequence of LLVM passes. You can run `ninja` in the build folder to compile LLVM after making changes to the “nullcheck.cpp”.

5 Test cases

``tests/PA1'' folder contains few test cases. You are encouraged to add more test cases. Run "make" in the "tests/PA1" folder to run the test cases. The makefile uses the `llvm-dis` tool to print the `LLVM IR` in a file. `nullcheck(*)_opt.ll` files contain the `LLVM IR` after the `nullcheck` pass. `nullcheck(*).ll` files contain the original `LLVM IR` before the `nullcheck` pass. The "make" command also generates an executable for each test case, if your implementation doesn't throw any error. You can execute them to test your implementation.

6 Tools

You can use cscope, ctags, and vim to navigate the source code. "Sublime text-3" also works well with LLVM.

7 Other resources

You can refer to "`https://llvm.org/doxygen/`" for quick reference to the `classes` in LLVM. E.g., to search all the public functions in the LLVM `Function` class, type "llvm Function" in the google search bar for a doxygen page related to the Function class in LLVM.

8 Other LLVM details

You can refer to "`http://llvm.org/docs/WritingAnLLVMPass.html`" to read about an LLVM pass. A function in LLVM is represented using ``class Function''. In an LLVM function pass, `runOnFunction` routine is called for every function with a handle to ``class Function'' of that routine.

```
To iterate all the basic blocks in a function F,
for (BasicBlock &BB : F)

To iterate all the instructions in a basic block BB,
for (Instruction &I : BB)

To check if an instruction is Alloca, Call, Load, Store, GetElementPtr or Cast:

AllocaInst *AI = dyn_cast<AllocaInst>(&I); // where I is of type Instruction
if (AI != NULL) {
  // This means that the instruction is Alloca instruction.
}
CallInst *CI = dyn_cast<CallInst>(&I);
LoadInst *LI = dyn_cast<LoadInst>(&I);
StoreInst *SI = dyn_cast<StoreInst>(&I);
GetElementPtrInst *GEP = dyn_cast<GetElementPtrInst>(&I);
CastInst *CAI = dyn_cast<CastInst>(&I);
```

To find the allocation type of a AllocaInst(AI):

```
Type *Ty = AI->getAllocationType();
```

To check if a Type(Ty) is pointer:

```
Ty->isPointerTy()
```

To check if a CallInst(CI) is indirect:

```
CI->isIndirectCall()
```

To get the name of the called function (in case of direct call):

```
CI->getCalledFunction()->getName();
```

To get the pointer operand of LoadInst/StoreInst/GetElementPtrInst:

```
Value *PtrOp = (LI/SI/GEP)->getPointerOperand();
```

To get the value of a StoreInst:

```
Value *Val = SI->getValueOperand();
```

To check if a value is Constant:

```
isa<Constant>(Val)
```

To check is a constant value is Null:

```
dyn_cast<Constant>(Val)->isNullValue()
```

To add a new basic block (with name newBB) in function (F):

```
BasicBlock::Create(F.getContext(), "newBB", &F)
```

To get a handle of the given function foo:

```
getOrInsertFunction("foo", ...);
```

To add new instructions at the end of a basic block.

```
IRBuilder<> IRB(BB);
IRB.CreateCall(...);
IRB.CreateBr(...);
```

The above statements will append a call and branch instruction to the basic block BB.

To split a basic block at Instruction I:

```
SplitBlock(BB, I);
```

To get the next instruction of instruction I:

```
I->getNextNode();
```

To add a conditional branch to a basic block BB:

```
IRBuilder<> IRB(BB);
auto Cmp = IRB.CreateICmp(...,..., ...);
IRB.CreateCondBr(Cmp, ..., ...);
```

To add a return statement:

```
IRBuilder<> IRB(BB);
IRB1.CreateRetVoid(); // return value is void
IRB1.CreateRet(...);  // takes return value as an argument

To create a input map for all basic blocks in a function.
Study std::map in c++.
std::map<BasicBlock*, std::map<Value*, bool>>

To walk all the successors of basic block BB:
for (BasicBlock *SuccBB : successors(BB))

To walk all the predecessors of basic block BB:
for (BasicBlock *PredBB : predecessors(BB))

To implement DFS or BFS you can use std::vector or std::deque.
```

9 How to submit.

Submit the "nullcheck.cpp" file and design documentation at the submission link.

Answer the following questions in your design documentation.

1. What is the transfer function for alloca?
2. What is the transfer function for store?
3. What is the transfer function for load?
4. What is the transfer function for getelementptr?
5. What is the transfer function for icmp?
6. What is the transfer function for call?
7. What is the transfer function for bitcast?
8. What is the transfer function for br?
9. What is the transfer function for ret?
10. Write transfer functions of other instructions if needed in your scheme.
11. What is the meet operator?
12. How do you initialize OUT[ENTRY]?
13. How do you initialize the OUTs of basic blocks other than ENTRY?
14. For each test case, what are the IR instructions that require `NULL` checks?

If you want to suggest a test case, please send a pull request to the project repository. You can do this assignment in a group of up to two students. Your submission will be graded if and only if your design documentation is complete.